Práctica 1 - Reloj Digital

Alfonso Rodríguez, Andrés Otero y Eduardo de la Torre
Electrónica Digital - E.T.S.I. Industriales UPM

September 30, 2020

La primera práctica de la asignatura de Electrónica Digital tiene por objetivo que el alumno se familiarice con una herramienta profesional de diseño e implementación de sistemas digitales, a la vez que le permitirá probar sobre un circuito real algunos de los conceptos más importantes del lenguaje de descripción hardware VHDL.

La herramienta de diseño que se va a emplear es Vivado Design Suite de Xilinx,[1] mientras que para la validación se empleará un dispositivo lógico programable (FPGA). En particular, se van a utilizar las placas de desarrollo PYNQ-Z1 de la empresa Digilent que se muestran en la Figura 1. La documentación completa y detallada de la placa se puede encontrar en este enlace.[2] Estas placas están dotadas con una FPGA de Xilinx de la familia *Zynq-7000*.[3] Vamos a utilizar además la shield de expansión que se muestra en la Figura 2 y que ha sido especialmente diseñada para esta asignatura.

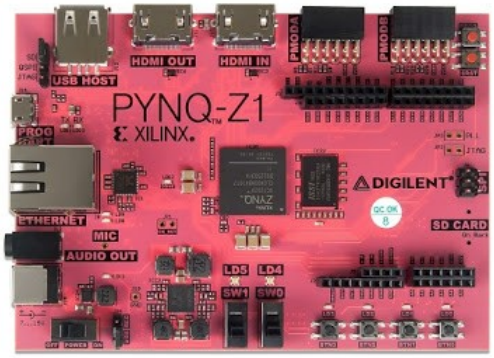


Figure 1: Placa de desarrollo Digilent PYNQ-Z1 que cuenta con una FPGA de la familia Zynq-7000

La práctica se estructura en dos partes. La primera de ellas deberá ser resuelta por el alumno de manera asíncrona. La segunda parte será síncrona, impartida mediante la herramienta TEAMS, en los horarios asignados para ello en el POD. **Es fundamental que**

[1] www.xilinx.com

[2] https://reference.digilentinc.com/reference/programmable-logic/pynq-z1

[3] En realidad, los dispositivos Zynq-7000 son Sistemas en Chip (SoC) e incluyen en su interior, además de una FPGA Artix-7, un procesador ARM Cortex™-A9 dual core. En esta asignatura los emplearemos únicamente como FPGAs.

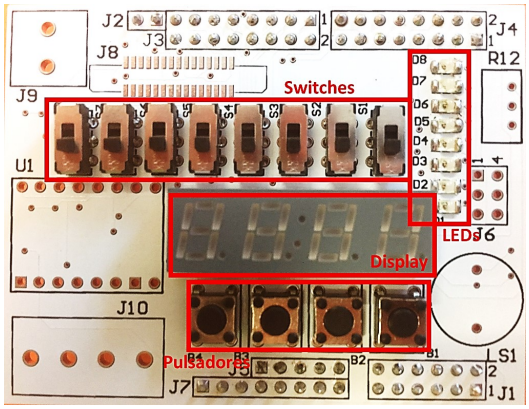


Figure 2: Shield de Expansión con dispositivos de E/S para las prácticas de Electrónica Digital

hayáis resuelto la parte asíncrona antes de acudir al grupo de prácticas que os corresponde. En caso contrario, no podréis seguir la segunda parte de la práctica. Tenéis además a vuestra disposición un vídeo explicativo de la primera parte de la práctica, que complementa a este guión.

La primera parte de la práctica tiene como objetivo que el alumno se familiarice con el manejo de las herramientas propias del desarrollo de sistemas digitales sobre FPGAs. Para ello haremos el diseño de un divisor de frecuencia. A continuación, el alumno deberá implementar un reloj digital, para lo que deberá contará con el divisor de frecuencia como uno de los bloques básicos. En la parte síncrona de la práctica aprenderemos a mostrar en los displays de siete segmentos de la shield la evolución del reloj digital.

1 Implementación de un divisor de frecuencia

El divisor de frecuencia es uno de los bloques básicos que nos podemos encontrar en casi cualquier sistema digital. Su finalidad es la generación de una señal dirigida al control de aquellas tareas periódicas que deben ser realizadas a una frecuencia distinta a la frecuencia de reloj del sistema. El funcionamiento esperado se puede ver en la Figura 3. No se debe confundir la salida de este bloque con una nueva señal de reloj. Es necesario recordar que en los sistemas digitales síncronos todos los elementos secuenciales deben estar gobernados por el mismo reloj. La relación entrada/salida del divisor que se va a implementar en esta primera parte de la práctica debe ser tal que **a su salida se genere un pulso a la frecuencia de 1 Hz, partiendo del reloj de entrada a 125 MHz**.

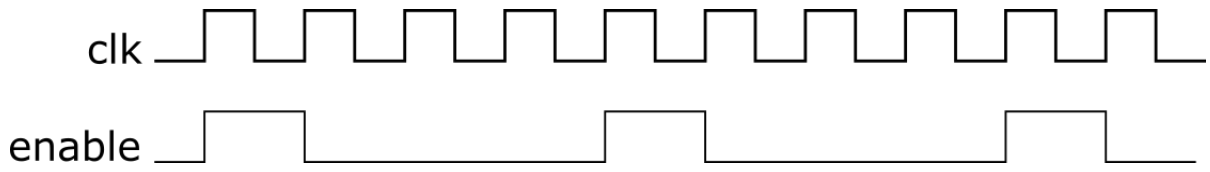
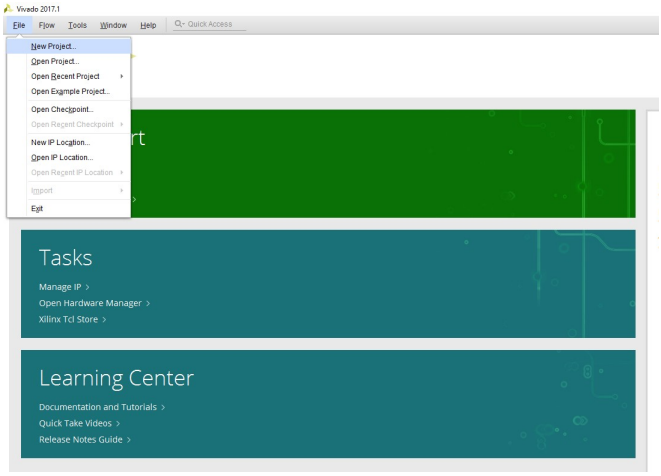


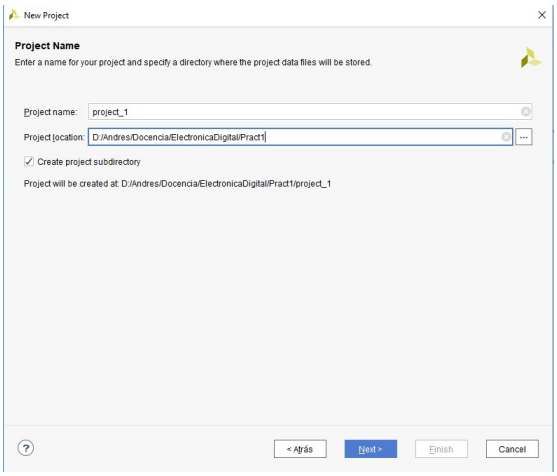
Figure 3: Funcionamiento esperado para el divisor de frecuencia (4 ciclos)

Lo primero que hay que hacer es abrir el entorno de diseño de Vivado, con el que se va

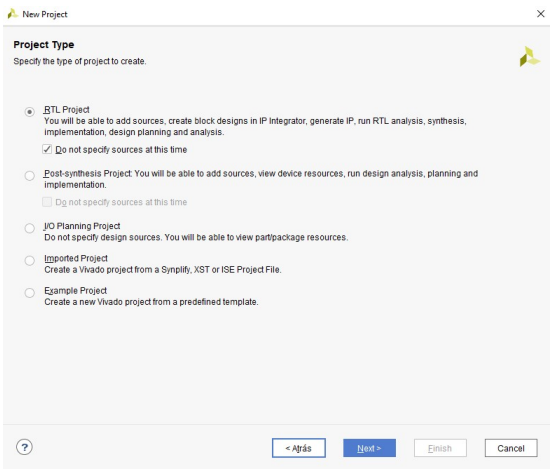
a desarrollar la práctica. Una vez abierto vamos a empezar por crear un nuevo proyecto. **Este proyecto debe encontrarse en una carpeta cuya ruta no contenga espacios, ni eñes, ni tildes en su nombre**. Además de indicar la localización del proyecto, le asignaremos un nombre en el campo Project Name de la ventana de creación de nuevo proyecto. Posteriormente, en la ventana de Tipo de Proyecto indicaremos que es un proyecto de tipo RTL, indicando además que no se especificarán ficheros de diseño en este momento. En la siguiente ventana será necesario indicar el tipo de dispositivo que se va a emplear. El dispositivo a seleccionar es el *xc7z020clg400-1*, de la familia *Zynq-7000*, *Package clg400* y *Speed Grade -1*.



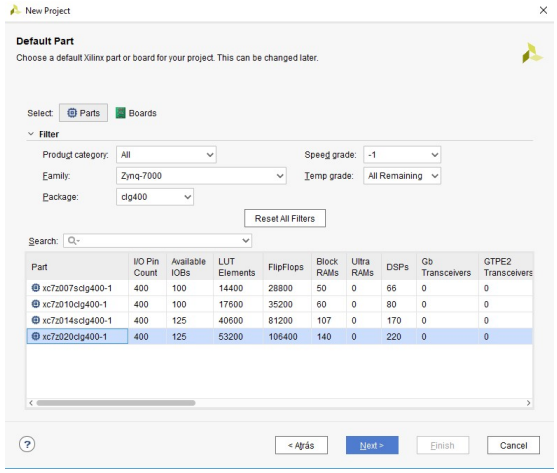
(a) Nuevo Proyecto



(b) Nombre del Proyecto



(c) Tipo de Proyecto

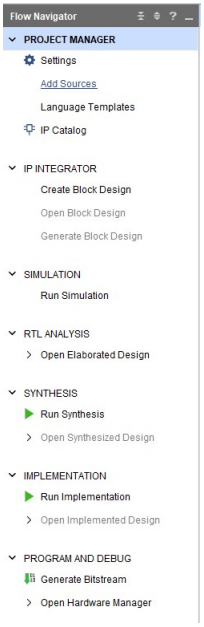


(d) Seleccionar Dispositivo

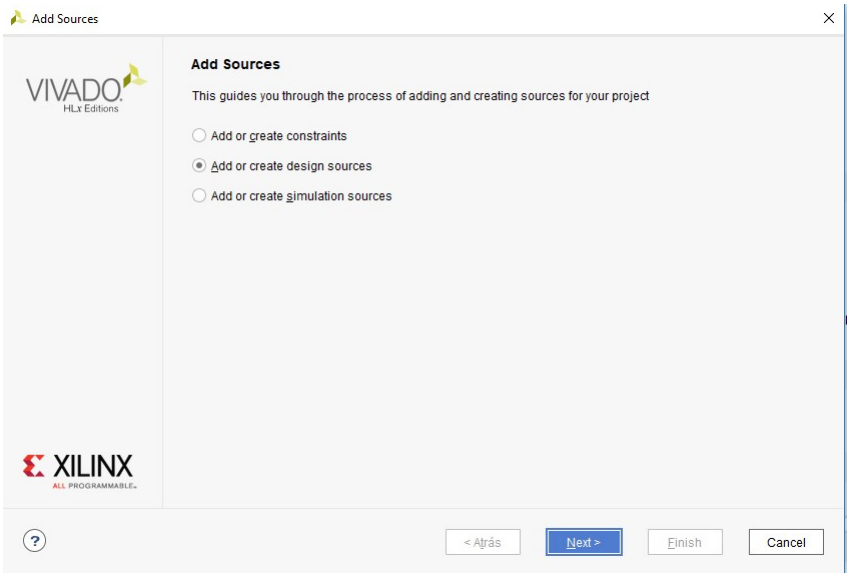
Figure 4: Ventanas para la creación del proyecto

Lo siguiente que haremos será comenzar a realizar el diseño del divisor de frecuencia. Tal y como se muestra en la Figura 5, en el panel de Flow Navigator se seleccionará la opción *Add Sources*. Nos aparecerá la ventana que se muestra a continuación, donde seleccionaremos *Add or create design source* y en la ventana correspondiente seleccionamos la opción *Create*

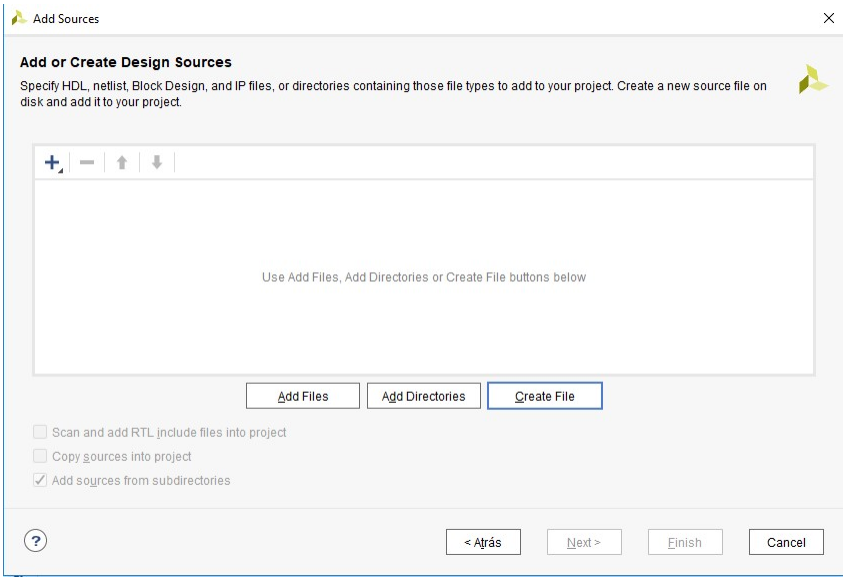
File. A continuación se abrirá la ventana de *Create Source File*, donde podremos indicar que se cree un fichero de tipo VHDL con nombre *freqdiv*.



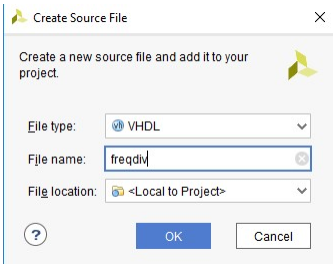
(a) Add Sources



(b) Add Design Sources



(c) Create File



(d) Enter Source File Name

Figure 5: Ventanas añadir un nuevo fichero VHDL al diseño

A partir de aquí se trata de describir, sobre el fichero **freqdiv.vhd** que acabamos de crear, el divisor de frecuencia. Para facilitar su posterior validación, los nombres de las entradas y salidas de la entidad deben ser los que se muestran en Listing 1. En la Listing 2 se recuerda la plantilla de código VHDL correspondiente a la descripción de un proceso síncrono y que se deben seguir en toda la práctica.

Listing 1: Definición de los puertos del divisor de frecuencia

```
entity freqdiv is
  Port (clk           : in std_logic;
        reset         : in std_logic;
        Enable        : in std_logic;
        Output_Signal : out std_logic);
end freqdiv;
```

Listing 2: Esquema de código VHDL de un proceso síncrono.

```
process(clk,reset)
begin
    -- Reset asíncrono por nivel alto
    if reset = '1' then
        -- Inicialización de las señales (= flipflops)

    -- Proceso secuencial síncrono (flanco de subida)
    elsif clk'event and clk = '1' then
        -- Implementación de la lógica secuencial

    end if;
end process;
```

1. Implementación del Divisor de Frecuencia

Debes crear en el fichero *freqdiv.vhd* la arquitectura para el divisor de frecuencia cuya entidad se muestra en el Listing 1. La arquitectura incluirá un contador que se incrementará con cada ciclo de reloj, y que generará un pulso en la señal de salida *enable* cada vez que alcance el límite de cuenta. Dicho límite será tal que se genere un pulso en *enable* cada segundo, teniendo en cuenta que la frecuencia del reloj de entrada es de 125 MHz. La generación de la señal *enable* debe hacerse de forma combinacional (fuera del proceso del contador). Se recomienda usar señales de tipo *integer* para almacenar el valor de cuenta del contador.

Una vez finalicemos con la descripción del divisor de frecuencia, procederemos a simular nuestro diseño para comprobar si funciona como debe. Para ello, deberemos incluir el fichero **Tbfreqdiv.vhd** que os proporcionamos, y que será el encargado de introducir los estímulos al diseño bajo test.

Este fichero lo vamos a añadir al proyecto empleando de nuevo la opción de *Add Sources* del panel lateral. En este caso, se va a seleccionar la opción de *Add or Create Simulation Sources*. Desde el cuadro emergente se selecciona la opción de *Add File*, seleccionando además la opción *Copy Sources into Project*, para permitir que se trabaje con una copia local del fichero, no con la versión original.

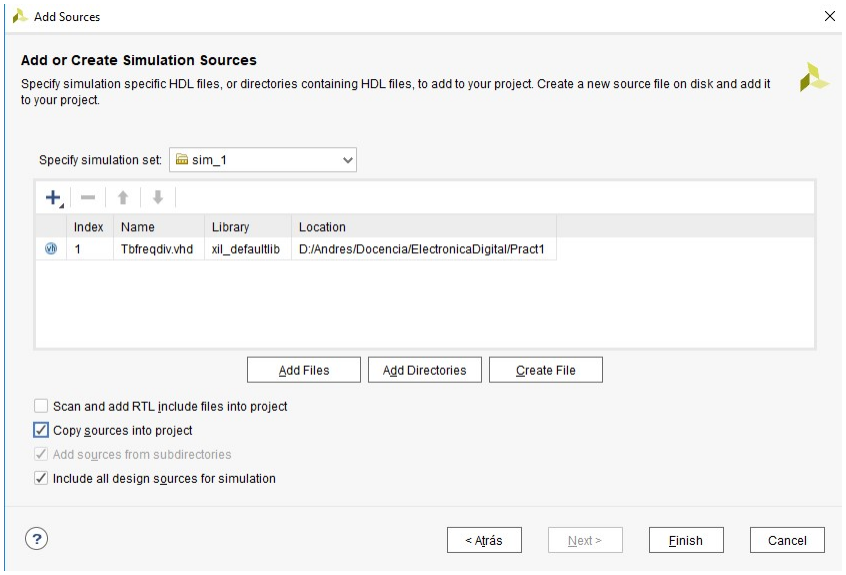


Figure 6: Ventana para añadir el Fichero Testbench de Simulación.

Para arrancar la simulación se debe hacer clic en la opción *Run Simulation* en el panel lateral. A continuación aparecerá la ventana del simulador, en la que podrás ver los cronogramas resultantes de la ejecución de la simulación. Si tienes la impresión de que no ves nada, es que tienes que manejar el Zoom y los controles de los tiempos de la simulación de manera adecuada.

Figure 7: Botones para el control de la simulacion

Las formas de onda que deben que salir como resultado son las siguientes:

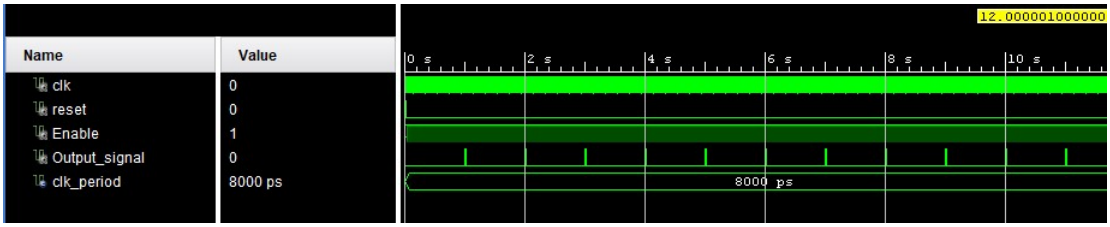


Figure 8: ¿Te ha funcionado el divisor de frecuencia?

Partiendo de este primer diseño digital sencillo, vamos a construir un reloj digital, que podremos simular en VHDL.

2 Desarrollo de un Reloj Digital

Se describe a continuación el funcionamiento esperado para el reloj digital:

- El reloj debe contar internamente los segundos transcurridos desde el reinicio del sistema.
- Se deben mostrar por los displays de siete segmentos los segundos, decenas de segundos, minutos y decenas de minutos, transcurridos desde el reinicio del sistema.
- El reinicio del sistema vendrá provocado por la pulsación del botón de la placa.

La arquitectura internal del reloj que vamos a implementar será la mostrada en la Figura 9.

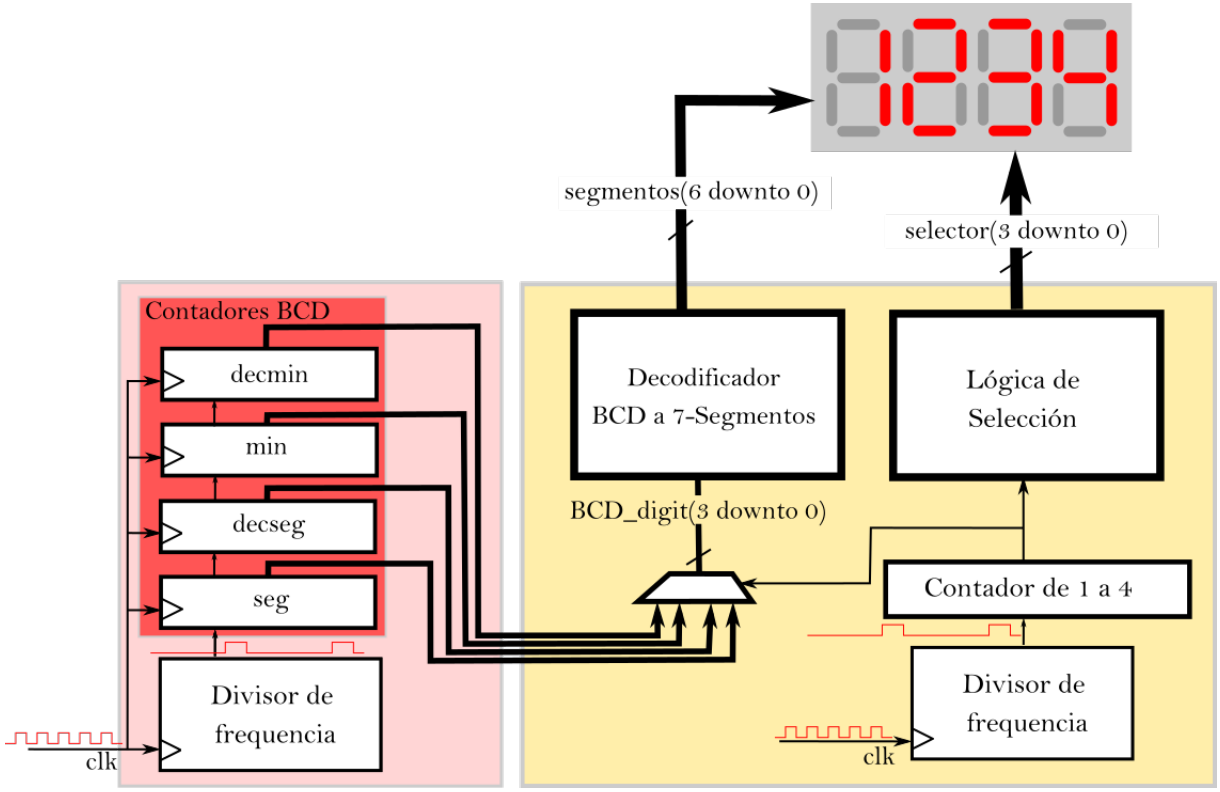


Figure 9: Arquitectura del reloj digital que vamos a implementar

Como se ve en la figura, contará con los siguientes módulos:

- **Divisor de frecuencia**: Deberá generar una señal periódica que contenga un pulso de reloj cada segundo. Se reutilizará el desarrollado en el apartado anterior.
- **Contadores BCD**: Deberán incrementarse con cada pulso generado por el divisor de frecuencia, llevando cuenta de los segundos, decenas de segundo, minutos y decenas de minutos, transcurridos desde el arranque.
- **Controlador de 7 Segmentos**: Se encargará de habilitar los segmentos necesarios para mostrar los dígitos de cada contador, en el display correspondiente.

Una vez explicado el funcionamiento del Reloj, procederemos a su desarrollo, que se hará en dos etapas. En primer lugar se implementarán y simularán los contadores BCD junto con el divisor de frecuencia. **Esta parte la deberéis hacer también de forma asíncrona, antes de la práctica**. Una vez validado este bloque, durante la práctica, se añadirá el control de los displays de siete segmentos, para obtener el diseño final que probaremos en la placa.

El diseño de los contadores se realizará en un nuevo fichero VHDL, que denominaremos **Chrono.vhd**. Tras añadir al proyecto un nuevo fichero con este nombre, definiremos una nueva entidad que tendrá las entradas y salidas necesarias para la simulación del circuito (ver Listing 3). Dentro de este fichero debemos copiar en primer lugar el código VHDL correspondiente al divisor de frecuencia que hemos realizado en el apartado anterior. **Para acelerar la simulación, dividiremos por 1000 el valor máximo de cuenta de los contadores del divisor de frecuencia. Una vez realizada la simulación, volveremos este parámetro a su valor original.**

Listing 3: Descripción de puertos del bloque de contadores BCD

```
entity Chrono is
  Port (clk            : in std_logic;
        reset          : in std_logic;

        -- Outputs (Contadores BCD)
        segundos       : out std_logic_vector(3 downto 0);
        dec_segundos   : out std_logic_vector(3 downto 0);
        minutos        : out std_logic_vector(3 downto 0);
        dec_minutos    : out std_logic_vector(3 downto 0));
end Chrono;
```

2. Implementacion de los contadores BCD del reloj digital

Debes crear en el fichero *Chrono.vhd* la arquitectura para el reloj digital cuya entidad se muestra en el Listing 3. En la arquitectura pegaremos en primer lugar el código del divisor de frecuencia que desarrollamos en el anterior apartado. Dicho divisor generará un pulso cada segundo. A partir de ese pulso se actualizarán cuatro contadores BCD, que debemos implementar en cuatro procesos separados. El primero de ellos contará los segundos transcurridos desde el reset, el segundo contará las decenas de segundo, el tercero los minutos, y el cuarto las decenas de minuto. Teneis que pensar cómo se generan las señales de habilitación de cada uno de los contadores, y cuál es el valor de cuenta máximo en cada caso. Las salidas de cada uno de los contadores se conectan a los puertos de salida de la entidad.

Una vez realizada la implementación de los contadores, procederemos a la simulación del diseño, empleando para ello el fichero de Test **TbChrono.vhd** que se encuentra también disponible en Moodle. Repitiendo los pasos que se han explicado anteriormente para la simulación, procederemos a la validación de los contadores de segundos, decenas de segundos, minutos y decenas de minutos, tal y como se ve en la Figura 10.

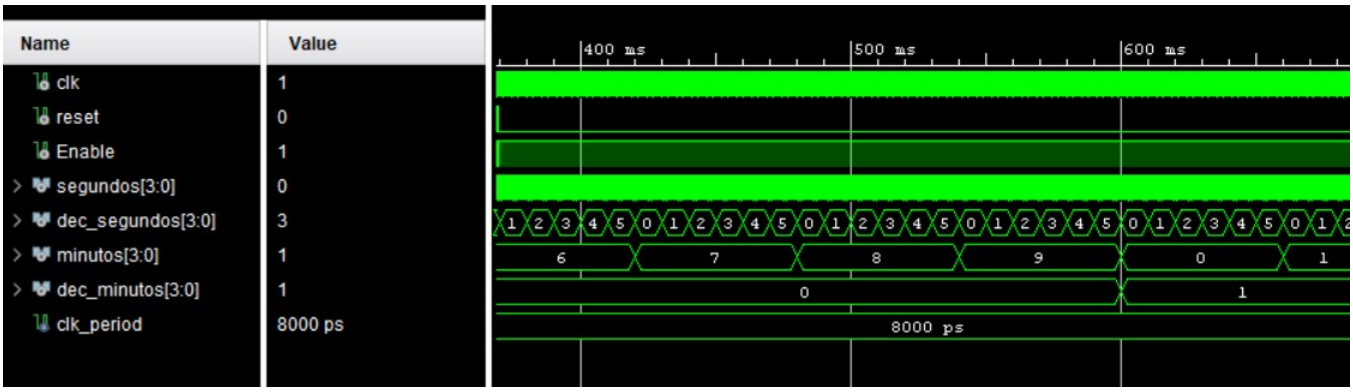


Figure 10: Este es el funcionamiento esperado de los contadores, viendo como se actualizan los segundos, las decenas de segundo, los minutos y las decenas de minuto.

3 Visualización del Reloj Digital con los Displays de Siete Segmentos

Una vez hemos validado los contadores, vamos a completar el diseño con el control de los displays de siete segmentos. El diseño del reloj digital completo se realizará sobre un nuevo fichero VHDL, que denominaremos **Practica1.vhd**. En este fichero definimos la entidad que se muestra en Listing 4 y copiaremos además el código correspondiente al divisor de frecuencia y los contadores BCD, una vez validado en simulación. De esta manera reutilizamos el código realizado en los apartados anteriores.

Listing 4: Descripción de puertos del Reloj Digital (**Practica1.vhd**)

```
entity Practica1 is
  Port (clk       : in std_logic;
        reset     : in std_logic;
        segmentos : out std_Logic_vector(6 downto 0);
        selector  : out std_logic_vector(3 downto 0));
end Practica1;
```

Para el control de los siete segmentos será necesario llevar a cabo la multiplexación temporal de las señales de control, teniendo en cuenta para ello la respuesta del ojo humano. (Frecuencia de refresco de 4 kHz). El control de los segmentos se realiza por nivel bajo mientras que el del selector es por nivel alto. Se debe incluir tambien la lógica para la decodificacion de los Contadores BCD a siete segmentos, y la lógica necesaria para la multiplexación temporal.

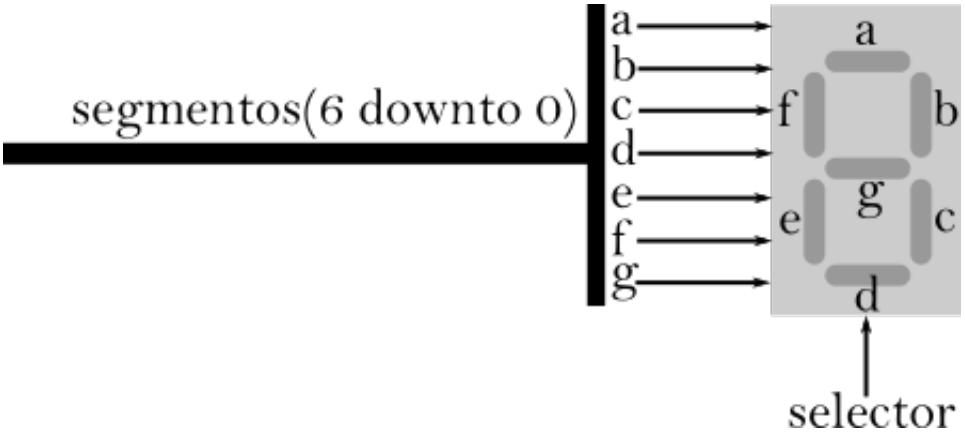


Figure 11: Detalles de la implementación del control de 7 segmentos de la placa.

3. Implementación de la logica de Control de los displays de siete segmentos

Debes crear en el fichero *Practica1.vhd* la arquitectura final del reloj digital. En este caso copiaremos todo el código desarrollado anteriormente y añadiremos la logica para multiplexar los siete segmentos, tal y como se muestra en la Figura 9. Esta logica incluye un contador de 0 a 3 para decidir el segmento que se activa en cada momento, y un decodificador para generar la señal de selección. Se añadirá también la logica para decodificación a siete segmentos.

TODO: generar el sistema para WebLab usando el script automático, y subir el bitstream de configuración a una instancia remota.

Práctica 2 - El Coche Fantástico

Alfonso Rodríguez, Andrés Otero y Eduardo de la Torre
Electrónica Digital - E.T.S.I. Industriales UPM

October 6, 2020

El objetivo de la segunda práctica de Electrónica Digital es que el alumno profundice en el diseño e implementación de alguno de los componentes básicos que podemos encontrar en la mayoría de sistemas digitales, tales como divisores de frecuencia, contadores o decodificadores, entre otros. Procederemos además a la implementación de un fichero de *Test Bench* para realizar la simulación del sistema, antes de su descarga en la FPGA.

Para ello se plantea el diseño de un sistema que emule el funcionamiento de las luces frontales del famoso coche fantástico de *Michael Knight*, que se muestra en la Figura 1.[1] En este caso, emplearemos los LEDS presentes en las placas de FPGA del laboratorio, tal y como se destaca en la Figura 2.

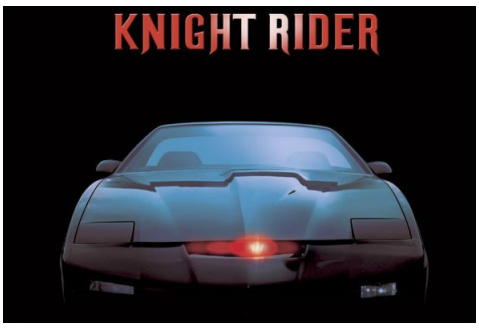


Figure 1: Luces frontales del coche fantástico que se desean emular

El funcionamiento esperado es el siguiente:

- Los LEDs deben encenderse primero de abajo a arriba y después de arriba abajo, sin repetición en los extremos.
- Sólo uno de los LEDs debe estar encendido a la vez.
- El tiempo total para recorrer los 8 LEDs, ida y vuelta, será de 1 segundo.
- El usuario deberá pulsar un botón de la placa (el que se destaca en la Figura 2, etiquetado como *Start*) para indicar al sistema que su funcionamiento debe empezar. Antes de pulsar este botón, los LEDs deberán reflejar el estado de reset (sólo el de abajo estará encendido).

[1] https://www.youtube.com/watch?v=oNyXYPhnUIs

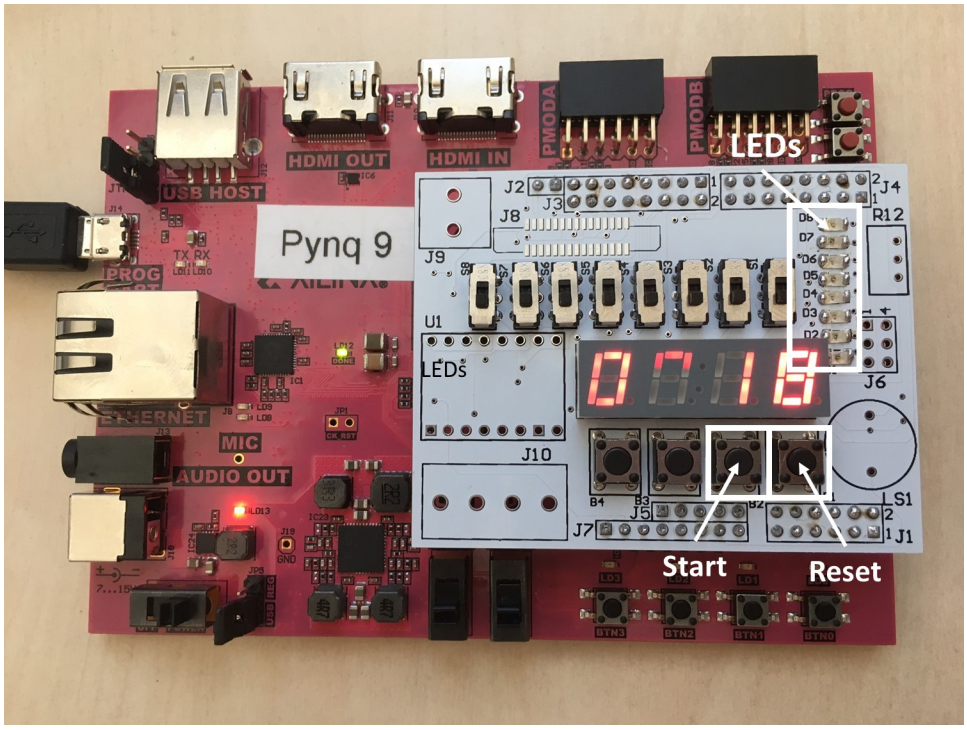


Figure 2: Ubicación de los LEDs en las placas del laboratorio

- El patrón de funcionamiento del circuito se repetirá indefinidamente, hasta que se pulse el botón de reset.

El bloque a diseñar debe tener las siguientes entradas y salidas:

Nombre de la Señal	Dirección	Ancho (bits)
clk	Input	1
reset	Input	1
start	Input	1
leds	Output	8

Para hacerlo, existen varias alternativas de diseño, que deberéis pensar de manera individual. La primera de ellas es mediante contadores, que es la que debéis implementar y simular por vuestra cuenta, antes de acudir a la práctica síncrona. La parte síncrona se impartirá mediante la herramienta TEAMS, en los horarios asignados para ello en el POD.

1 Implementación Mediante Contadores

La implementación mediante contadores puede hacerse al menos de dos maneras. La primera de ellas es empleando un contador ascendente de 0 a 13, y la segunda es mediante un contador de 0 a 7, de tipo ascendente/descendente. En el primero de los casos es necesario añadir un decodificador de 4 a 8 bits para activar los leds en funcion del valor del contador. En el

segundo necesitamos un biestable para almacenar el sentido de cuenta (ascendente o descendente), además de un decodificador de 3 a 8 bits. Ambas implementaciones se muestran en la Figura 3.

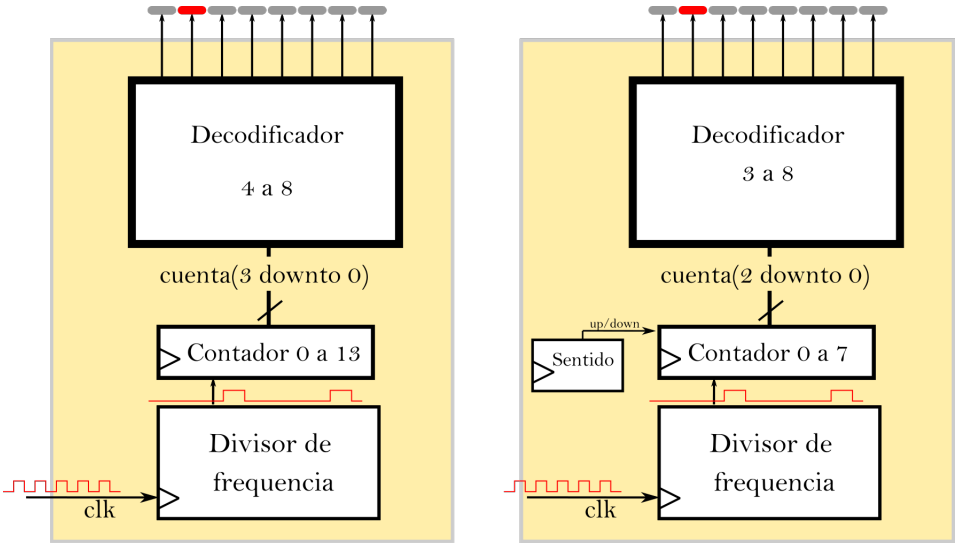


Figure 3: Posibles implementaciones del *coche fantástico* empleando contadores

Para empezar el trabajo, tenemos que abrir un nuevo proyecto tal y como se hizo en la primera práctica. En dicho proyecto añadiremos un fichero de diseño denominado *rider.vhd*, en el que se creará una entidad con los puertos que necesitamos (ver Listing 1):

Listing 1: Definición de los puertos del controlador de los leds

```
-- Declaración de la entidad
entity rider is
  port (
    clk   : in  std_logic;
    reset : in  std_logic;
    boton : in  std_logic;
    leds  : out std_logic_vector(7 downto 0)
  );
end rider;
```

Posteriormente, tenéis que describir la arquitectura del controlador de los LEDs, para que se comporte tal y como se ha detallado antes. Para ello podéis seguir cualquiera de las dos arquitecturas mostradas en la Figura 3.

Una vez se ha finalizado el diseño, procederemos a su simulación. Para ello tenemos que escribir un fichero de Test Bench. La entity de un fichero de Test Bench tiene la particularidad de no tener ni entradas ni salidas:

Listing 2: Entity de un Test Bench, sin entradas ni salidas.

```vhdl
entity rider_tb is
end rider_tb;
```

En su interior instanciaremos el bloque que queremos simular (design under test), tal y como se muestra en la Figura 4.

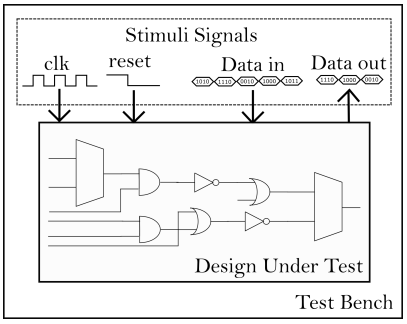


Figure 4: Arquitectura genérica de un fichero de Test Bench

Este fichero deberá contener procesos para simular la generación del reloj y de las señales de control del sistema (señales *reset* y *start*). En Listing 3 se recuerda un ejemplo de proceso para la generación de la señal de reloj.

Listing 3: Ejemplo de un proceso para la generación de la señal de reloj en el *test bench.*

```vhdl
-- Generación del reloj
clk_proc: process
begin
  clk <= '1';
  wait for clk_period/2;
  clk <= '0';
  wait for clk_period/2;
end process;
```

Es importante destacar que los ficheros de *Test Bench* pueden contener procesos dentro de los cuales existen instrucciones VHDL no sintetizables, tales como *wait* o *wait for*. Estos procesos no tienen lista de sensibilidad. Estas instrucciones no se podrán emplear nunca en nuestros diseños digitales.

Cheatsheet sobre testbenches VHDL

Los testbench son artificios que permiten probar (simulando) nuestros diseños, no especificaciones de circuitos digitales. Es en el único archivo de código de la asignatura Electrónica Digital donde se permite no seguir la norma sobre diseño secuencial síncrono (ver template en el guion de la práctica anterior). Hay algunos puntos que no podemos olvidar:

- Entity sin puertos (vacía)
- Procesos sin lista de sensibilidad (se redisparan siempre al llegar al final)
- Construcciones no sintetizables, con concepto de tiempo (wait, wait for)

1. Implementacion mediante contadores

En la primera parte de la práctica debeis implementar en el fichero *rider.vhd* la arquitectura correspondiente al controlador de los LEDs, empleando una de las arquitecturas propuestas mediante contadores. Posteriormente, debéis generar un fichero de Test-bench, en el que se generen los estímulos necesarios para la simulación de vuestro diseño. Finalmente, debéis lanzar la simulación para comprobar que vuestro circuito funciona correctamente.

2 Alternativas de Diseño

En la segunda parte de la práctica, que haremos de forma síncrona, vamos a discutir e implementar algunas alternativas para realizar este tipo de diseños.

2. Implementacion de alternativas de diseño

¿Se os ocurren otras formas de implementar el mismo circuito?, ¿Cuáles son las ventajas e inconvenientes de cada arquitectura? Probad a realizar otras implementaciones y simular su funcionamiento.

2.1 Prueba en hardware remoto

Para probar el diseño que acabamos de hacer vamos a utilizar un servidor remoto que se encuentra en la Escuela. Este laboratorio remoto incluye varias placas Pynq con las *shield* de expansión para las prácticas. El primer paso que tenemos que dar es instanciar el circuito recién diseñado en un *wrapper* que contiene toda la información sobre los periféricos de entrada/salida. Tenéis disponible una plantilla en Moodle bajo el nombre `top_weblab.vhd`. Una vez importado el archivo en nuestro proyecto (Flow Navigator - Add Sources - Add or create design sources - Add files + Copy sources into project), modificaremos el código para que quede como se muestra en el Listing 4.

Listing 4: *Wrapper* para despliegue remoto (**top_weblab.vhd**).

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

entity top_weblab is
    port (
        clk      : in  std_logic;
        reset    : in  std_logic;
        buttons  : in  std_logic_vector(3 downto 0);
        switches : in  std_logic_vector(7 downto 0);
        leds     : out std_logic_vector(7 downto 0);
        segments : out std_logic_vector(7 downto 0);
        selector : out std_logic_vector(3 downto 0)
    );
end top_weblab;

architecture wrapper of top_weblab is
begin

    rider_i: entity work.rider
    port map (
        clk   => clk,
        reset => reset,
        boton => buttons(0),
        leds  => leds
    );

    -- Ponemos el resto de salidas a 0 por si acaso...
    segments <= (others => '0');
    selector <= (others => '0');

end wrapper;
```

Después, utilizaremos un *script* para automatizar el proceso de generación del fichero de configuración, o *bitstream*, para la FPGA. Dicho *script* se encuentra también disponible en Moodle, bajo el nombre `generate_weblab.tcl`. Una vez descargado, podrá ser ejecutado desde la consola Tcl de Vivado (ver Figura 5) escribiendo `source <path>/generate_weblab.tcl`, donde `<path>` es la carpeta en la que nos hemos descargado el *script*. Tras el proceso de implementación, que suele durar varios minutos, el *bitstream* se habrá generado como `weblab/system.bin` dentro de la carpeta del proyecto de Vivado.

Una vez que dicho fichero de configuración ha sido generado, tendréis que ir a la dirección http://138.100.74.2/weblab y conectaros usando como usuario vuestro correo UPM (sin el @alumnos.upm.es) y como contraseña vuestro número de matrícula. **IMPORTANTE: actualmente el servidor no cuenta con certificados SSL válidos, por lo que la comunicación no es segura. Os recomendamos encarecidamente que no uséis contraseñas que incluyan información sensible.** Tras seleccionar el experimento "Pynq", deberemos subir el archivo `system.bin` y darle al botón de reserva. A partir de este momento, y si hay instancias disponibles para vosotros, se habilitará una ventana temporal

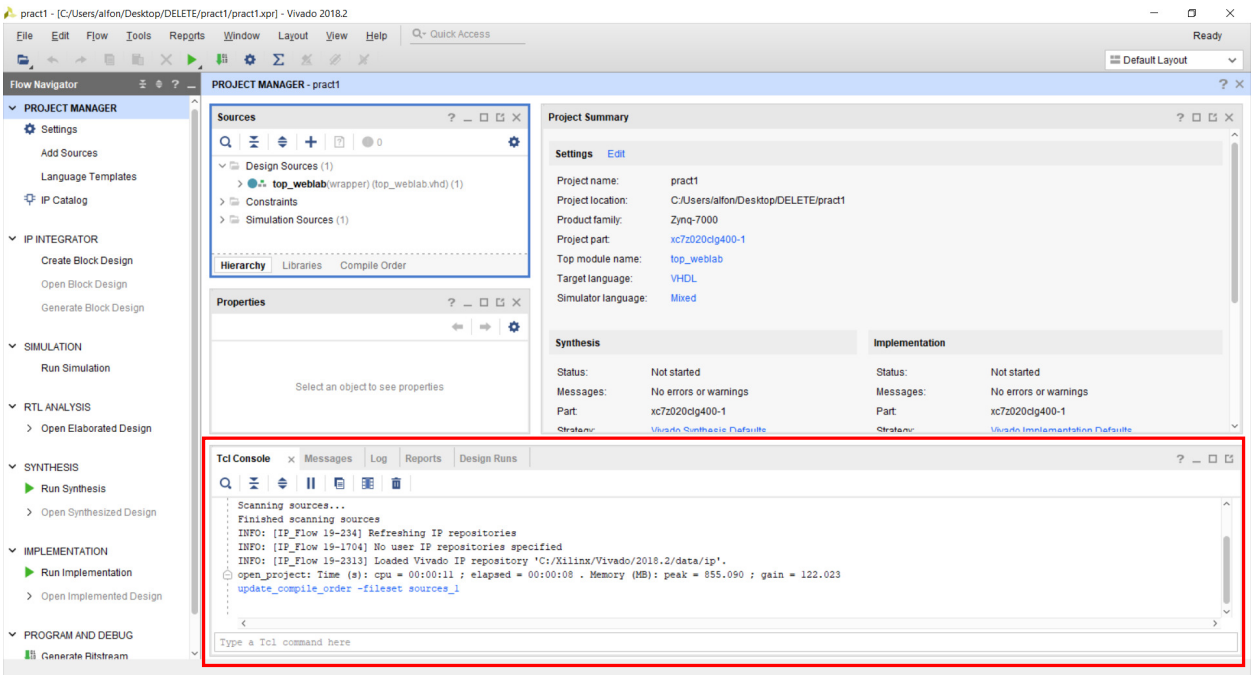


Figure 5: Consola Tcl en Vivado.

de 5 minutos en la que podréis interactuar con la placa de desarrollo. Os pedimos que, una vez terminadas las pruebas que tengáis que hacer, dejéis libre la FPGA (mediante el botón de finalizar) para que el resto de vuestros compañeros pueda usarla también.

Os recordamos que tenéis disponible en Moodle un vídeo con el procedimiento detallado para usar el laboratorio remoto de la asignatura.

Práctica 3 - La Cerradura Electrónica

Alfonso Rodríguez, Andrés Otero y Eduardo de la Torre
Electrónica Digital - E.T.S.I. Industriales UPM

October 6, 2020

La tercera práctica de la asignatura Electrónica Digital tiene por objetivo el diseño de **máquinas de estado síncronas**, así como su implementación sobre una FPGA empleando descripciones en VHDL. En particular, se realizará el diseño de un detector de secuencia que podría formar parte del sistema de control de una cerradura electrónica. Para abrir la cerradura, el usuario deberá introducir la secuencia apropiada mediante dos de los pulsadores de la placa de prácticas. Además de describir la máquina de estados para el control de la cerradura electrónica, y con el objetivo de sincronizar y filtrar las entradas procedentes de los pulsadores, se implementará un **circuito anti-rebotes**.

El diseño de la máquina de estados de la cerradura lo debéis abordar vosotros antes de venir a la práctica, mientras que el circuito anti-rebotes y la integración final las haremos todos juntos en la segunda parte de la práctica, que se celebrará de forma síncrona por TEAMS.

1 La Cerradura Electrónica

El funcionamiento esperado para el sistema de control de la cerradura electrónica es el siguiente: se activará la salida (se encenderá un led) cuando se detecten al menos dos pulsaciones (en cualquier orden) en cada uno de los dos botones. Después, el sistema se bloquea hasta que es reseteado. La máquina de estados que debemos implementar, de acuerdo con la funcionalidad esperada para la cerradura, es la mostrada en la Figura 1. Esta máquina de estados debe implementarse en un nuevo fichero de diseño que llamaremos *FSM.vhd*.

La máquina de estados tendrá las siguientes entradas:

- **A: Entrada filtrada del botón A.** Podemos suponer que es una señal digital de un ciclo de reloj de duración, generada por un circuito secuencial, no es una señal física. Esto se debe a que en la segunda parte de la práctica vamos a conectar esta entrada a un circuito anti-rebotes.
- **B: Entrada filtrada del botón B.** También será una señal digital de un ciclo de reloj de duración, generada por un circuito secuencial, no es una señal física. Procederá también de un circuito anti-rebotes.

A su vez, producirá una salida:

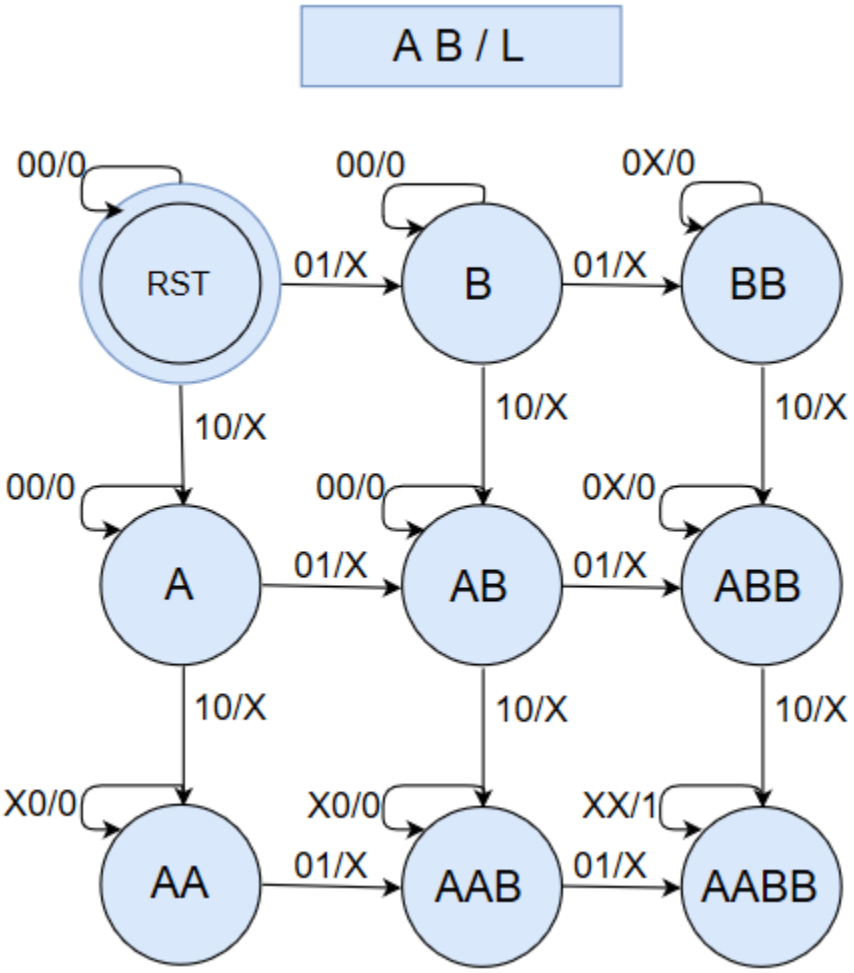


Figure 1: Diagrama de la máquina de estados de control de la cerradura electrónica

- **L: Salida de activación de Led cuando la secuencia es la correcta.** Es una salida física, por lo que su valor durante las transiciones no importa (X).

Una vez detectada una secuencia correcta, la máquina de estados permanecerá en este estado hasta que el usuario presione el botón de reset. La máquina de estados tendrá los puertos que se muestran en Listing 1.

Listing 1: Definición de los puertos de la máquina de estados de control de la cerradura

```
entity Secuencia_FSM is
    Port (clk   : in   STD_LOGIC;
          reset : in   STD_LOGIC;
          A     : in   STD_LOGIC;
          B     : in   STD_LOGIC;
          LED   : out  STD_LOGIC);
end Secuencia_FSM;
```

Una vez finalizado el diseño se procederá a su simulación. Para ello se deberá crear un fichero de *Test Bench* que genere distintas secuencias correctas, viendo si el sistema diseñado es capaz de identificarlas.

1. Implementacion y simulación de la máquina de estados de la cerradura

En la primera parte de la práctica debéis implementar en el fichero *FSM.vhd* la máquina de estados de la Figura 1. Se añade un anexo en este guión con un ejemplo de descripción de máquina de estados en VHDL. Deberéis implementar además un fichero de *Test Bench* que genere distintas secuencias de pulsaciones en los botones A y B, para comprobar que la máquina de estados funciona correctamente.

2 El circuito Antirebotes

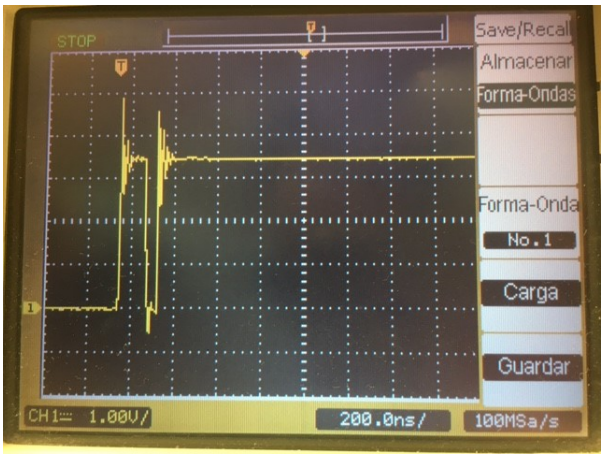
Una vez diseñada la máquina de estados de la cerradura electrónica, procederemos a diseñar un circuito capaz de eliminar los reobotes que se produzcan en las señales de entrada, procedentes de los botones. Se justifica a continuación su necesidad.

El accionamiento de un pulsador, lejos de producir un pulso limpio, generará una señal con múltiples flancos como la que se muestra en la Figura 2. Estos rebotes no deseados pueden suceder tanto al presionar como al soltar el pulsador, por lo que no se puede emplear este tipo de señales en un diseño digital sin antes proceder al filtrado de las mismas. Por otro lado, los cambios en esta señal, que es asíncrona con el reloj de nuestro sistema, se pueden producir en cualquier instante de tiempo, pudiendo coincidir incluso con los flancos de la señal de reloj del circuito que se pretende controlar. Esta situación provocaría que la salida del biestable en el que se registre la señal alcance estados metaestables (indeterminados, entre 0 o 1), que pueden comprometer el correcto funcionamiento del circuito. Por estos motivos, la adaptación de señales asíncronas procedentes del exterior requiere de una electrónica de sincronización y filtrado que garanticen que por cada actuación sobre el pulsador se genera un único pulso que tendrá la duración de un ciclo de reloj.

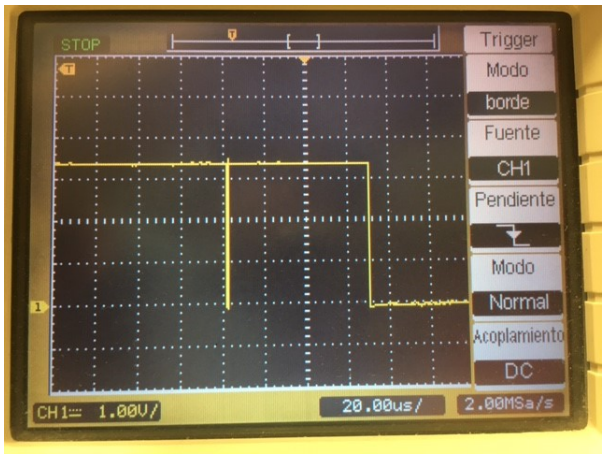
El circuito anti-rebotes que se va a implementar en la práctica tiene por estructura la mostrada en la Figura 3, con tres etapas bien diferenciadas: el sincronizador, el detector de flancos de subida y una pequeña máquina de estados al final. El sincronizador se encargará de eliminar los posibles estados metaestables en los que incurra la señal de entrada; el detector de flancos generará un pulso por cada flanco de subida presente en la señal (cada rebote producirá su propio flanco); finalmente, la máquina de estados se quedará con el primer flanco significativo de todos los generados, tanto al presionar como al soltar el botón.

El funcionamiento del sincronizador es el que se muestra en el cronograma de la Figura 4: si la señal de entrada cambia cerca del flanco de subida de los biestables, la salida del primero (Q1) pasará por un estado metaestable. Sin embargo, gracias a esta estructura la situación de metaestabilidad no se transmite a la salida del segundo biestable (Q2).

Posteriormente, empleando una puerta inversora y otro biestable D, se implementa un detector de flancos de subida, el cual generará un pulso con duración 1 periodo de reloj tras cada flanco presente en la entrada. El funcionamiento de este circuito se puede ver en la



(a) Rebotes de Subida



(b) Rebotes de Bajada

Figure 2: Forma de onda típica de la señal generada por la pulsación de un botón en la placa de prácticas

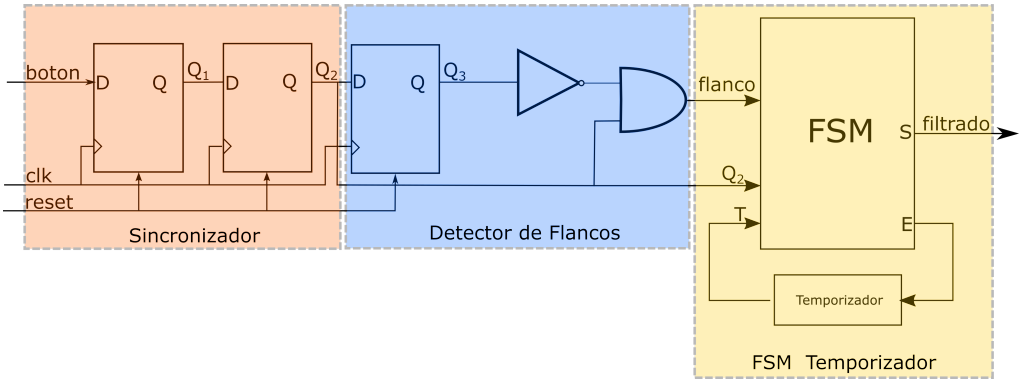


Figure 3: Circuito anti-rebotes a implementar

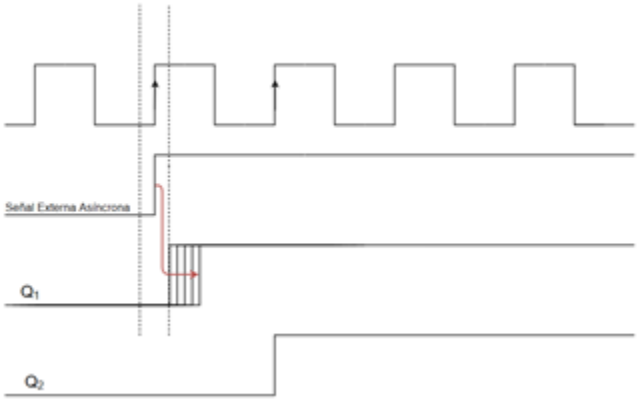


Figure 4: Cronograma del circuito sincronizador

Figura 5.

Finalmente, es necesario implementar una máquina de estados que sea capaz de quedarse

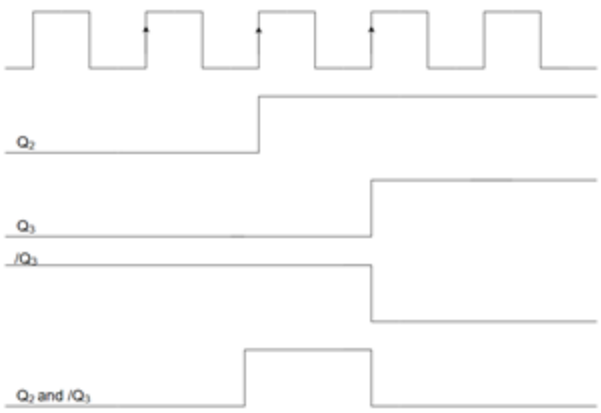


Figure 5: Cronograma del detector de flancos de subida

con el primer flanco significativo de la señal, filtrando el resto de flancos que aparecerán tanto al presionar como al soltar el pulsador, así como los pequeños pulsos debidos al ruido. El diagrama descriptivo de la máquina de estados es el que se muestra en la Figura 6.

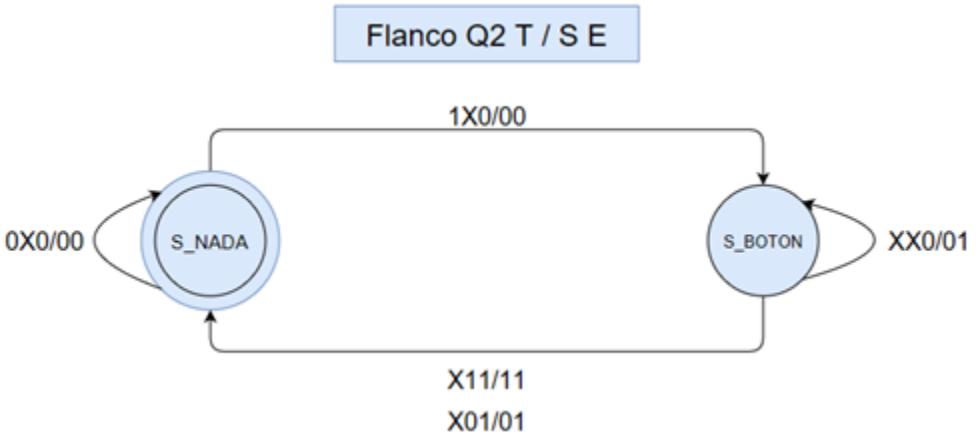


Figure 6: Diagrama de la máquina de estados del detector de pulsos

La máquina de estados tendrá las siguientes entradas:

- **Flanco**: Salida del detector de flancos
- **Q2**: Salida del circuito sincronizador
- **T**: Señal que indica que se ha llegado al valor máximo en la temporización

A su vez, producirá dos salidas:

- **S**: Salida filtrada
- **E**: Señal de habilitación del temporizador.

Como se ve en la figura, la máquina de estados contará con tan sólo dos estados:

- **S_NADA**: La máquina permanece en este estado mientras no se acciona el pulsador. Una vez pulsado pasará a S_BOTON e iniciará un temporizador que va a evitar, durante un tiempo determinado, que se registren los flancos del resto de los rebotes de la señal.
- **S_BOTON**: La máquina permanece en este estado mientras el temporizador no llega a su fin. Una vez recibida la señal que indica que se ha alcanzado el valor máximo de la cuenta, se vuelve a S_NADA. En la transición se activará la señal de salida, si la señal Q2 tiene valor 1. De esta manera se evita considerar los flancos en la señal de bajada.

El valor estimado para el temporizador, considerando el comportamiento de las placas con las que se cuenta en las prácticas, es de aproximadamente 1 ms (deberemos ajustarlo forma manual). El circuito anti-rebotes se debe implementar en una única entidad, que debe tener los puertos que se muestran en Listing 2.

Listing 2: Definición de los puertos del circuito Antirebotes

```
entity Antirebotes is
  Port (clk          : in  std_logic;
        reset        : in  std_logic;
        boton        : in  std_logic;
        filtrado     : out std_logic);
end Antirebotes;
```

Una vez implementado el circuito se procederá a simularlo, para lo que será necesario incluir el fichero de Test Bench que se adjunta en la práctica (*Test_Antirebotes.vhd*). La simulación deberá devolver un comportamiento como el que se muestra en la figura 7. En ella se puede ver como el primer pulso es filtrado (tiene una duración menor de 1 ms), mientras que en el segundo se produce como salida un único pulso de duración un ciclo de reloj, después de la pulsación del botón.

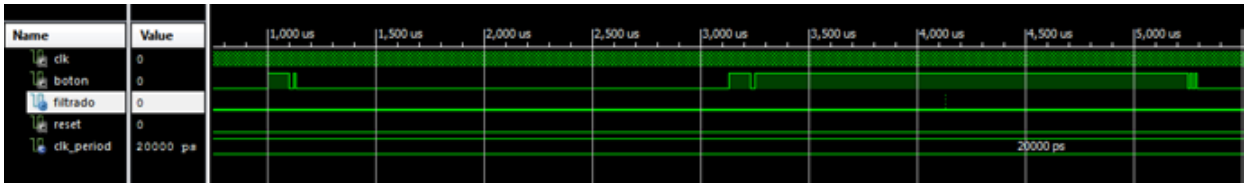


Figure 7: Resultado esperado de la simulación del circuito anti-rebotes.

2. Implementacion y simulación del circuito anti-rebotes

Debéis diseñar el circuito anti-rebotes que acabamos de explicar, incluyendo el sincronizador, el detector de flancos y la máquina de estados del detector de pulsos, en una única entidad denominada *Antirebotes*. Se procederá también a su simulación empleando el fichero *Test_Antirebotes.vhd*, que podéis encontrar en Moodle.

Una vez que hemos diseñado y validado en circuito anti-rebotes, procederemos a integrarlo en el sistema completo, junto con la máquina de estados del control de la cerradura. De esta manera, la maquina de estados recibirá las señales A y B ya filtradas, evitando falsas pulsaciones producidas por los rebotes.

Como se puede ver en la Figura 8, el circuito a diseñar estará compuesto por un anti-rebotes para cada una de las entradas del exterior, seguido por la máquina de estados que implementamos en la primera parte de la práctica. La conexión de estos elementos se realizará empleando una descripción a nivel estructural, en la que se recoge la interconexión entre los componentes que componen el sistema.

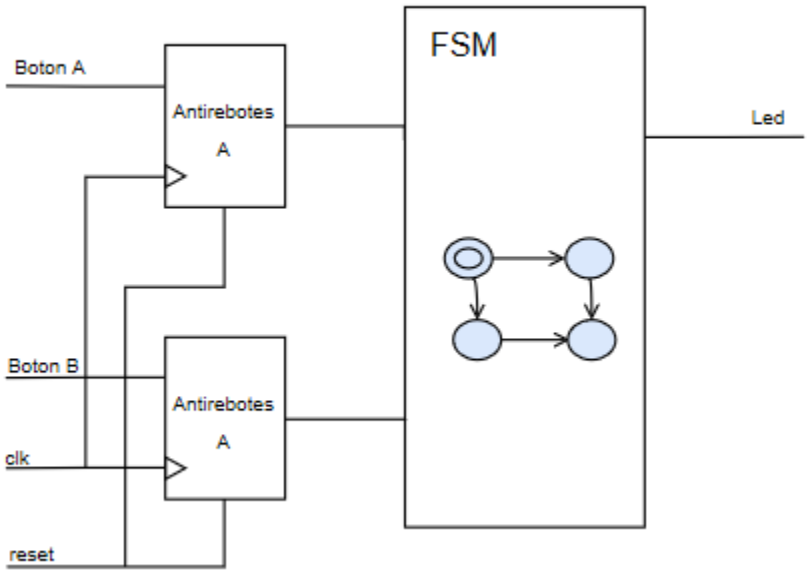


Figure 8: Sistema de control de la cerradura electrónica

El sistema completo debe tener la definición de puertos que se muestra a continuación:

Listing 3: Definición de los puertos del sistema completo

```
entity DetectorDePulsos is
    Port (clk     : in  std_logic;
          reset   : in  std_logic;
          Boton_A : in  std_logic;
          Boton_B : in std_logic;
          Led     : out std_logic);
end DetectorDePulsos;
```

3. Implementacion del circuito completo

Creamos una nueva entidad denominada *DetectorDePulsos*, en cuya arquitectura integraremos dos circutos anti-rebotes y la máquina de estado de control de la cerradura, empleando una descripción estructural.

2.1 Prueba en hardware remoto

Para probar el diseño que acabamos de hacer vamos a utilizar un servidor remoto que se encuentra en la Escuela. Este laboratorio remoto incluye varias placas Pynq con las *shield* de expansión para las prácticas. El primer paso que tenemos que dar es instanciar el circuito recién diseñado en un *wrapper* que contiene toda la información sobre los periféricos de entrada/salida. Tenéis disponible una plantilla en Moodle bajo el nombre `top_weblab.vhd`. Una vez importado el archivo en nuestro proyecto (Flow Navigator - Add Sources - Add or create design sources - Add files + Copy sources into project), modificaremos el código para que quede como se muestra en el Listing 4.

Listing 4: *Wrapper* para despliegue remoto (**top_weblab.vhd**).

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity top_weblab is
6      port (
7          clk      : in  std_logic;
8          reset    : in  std_logic;
9          buttons  : in  std_logic_vector(3 downto 0);
10         switches : in  std_logic_vector(7 downto 0);
11         leds     : out std_logic_vector(7 downto 0);
12         segments : out std_logic_vector(7 downto 0);
13         selector : out std_logic_vector(3 downto 0)
14     );
15 end top_weblab;
16
17 architecture wrapper of top_weblab is
18 begin
19
20     DetectorDePulsos_i: entity work.DetectorDePulsos
21     port map (
22         clk     => clk,
23         reset   => reset,
24         Boton_A => buttons(0),
25         Boton_B => buttons(1),
26         Led     => leds(0)
27     );
28
29     -- Ponemos el resto de salidas a 0 por si acaso...
30     leds(7 downto 1) <= (others => '0');
```

```
31      segments <= (others => '0');
32      selector <= (others => '0');
33
34  end wrapper;
```

Después, utilizaremos un *script* para automatizar el proceso de generación del fichero de configuración, o *bitstream*, para la FPGA. Dicho *script* se encuentra también disponible en Moodle, bajo el nombre `generate_weblab.tcl`. Una vez descargado, podrá ser ejecutado desde la consola Tcl de Vivado (ver Figura 9) escribiendo `source <path>/generate_weblab.tcl`, donde `<path>` es la carpeta en la que nos hemos descargado el *script*. Tras el proceso de implementación, que suele durar varios minutos, el *bitstream* se habrá generado como `weblab/system.bin` dentro de la carpeta del proyecto de Vivado.

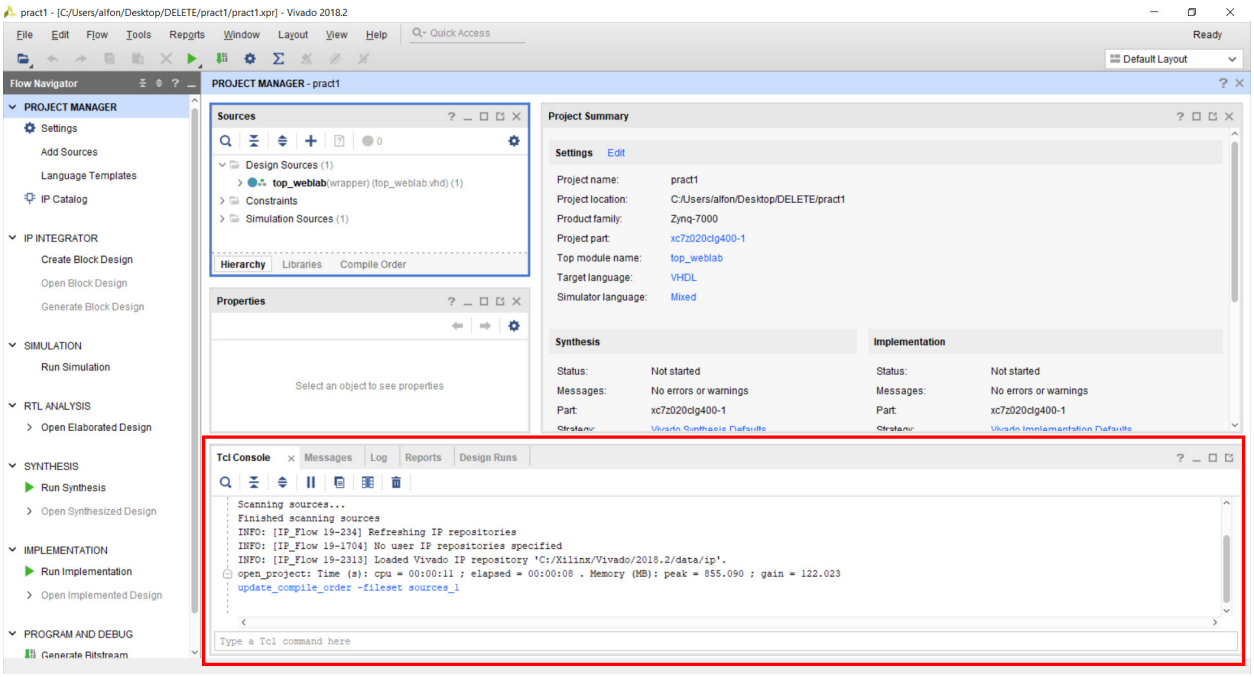


Figure 9: Consola Tcl en Vivado.

Una vez que dicho fichero de configuración ha sido generado, tendréis que ir a la dirección http://138.100.74.2/weblab y conectaros usando como usuario vuestro correo UPM (sin el @alumnos.upm.es) y como contraseña vuestro número de matrícula. **IMPORTANTE: actualmente el servidor no cuenta con certificados SSL válidos, por lo que la comunicación no es segura. Os recomendamos encarecidamente que no uséis contraseñas que incluyan información sensible.** Tras seleccionar el experimento "Pynq", deberemos subir el archivo `system.bin` y darle al botón de reserva. A partir de este momento, y si hay instancias disponibles para vosotros, se habilitará una ventana temporal de 5 minutos en la que podréis interactuar con la placa de desarrollo. Os pedimos que, una vez terminadas las pruebas que tengáis que hacer, dejéis libre la FPGA (mediante el botón de finalizar) para que el resto de vuestros compañeros pueda usarla también.

Os recordamos que tenéis disponible en Moodle un vídeo con el procedimiento detallado para usar el laboratorio remoto de la asignatura.

A Máquinas de Estado en VHDL

Listing 5: Implementación de Máquinas de Estado en VHDL

```
1   ------------------------------------------
2   -- Sección DECLARATIVA de la arquitectura --
3   ------------------------------------------
4
5   -- Declaración del tipo enumerado para describir los distintos estados.
6   type State_t is (S_RESET, S_1, S_2, .... );
7   signal STATE : State_t; -- Definición de una señal de tipo enumerado.
8
9
10  ------------------------------------------------
11  -- Sección de IMPLEMENTACIÓN de la arquitectura --
12  ------------------------------------------------
13
14  -- Proceso de Actualización de Estados, a partir de las entradas y del
15  -- estado actual (ecuaciones de estado).
16  process(clk,reset)
17  begin
18      if (reset = '1') then
19          Status <= S_RESET;
20      elsif (clk'event and clk = '1') then
21          case STATE is
22              when S_RESET =>
23                  if(Entrada_1 = '1') then
24                      STATE <= S_1;
25                  elsif(Entrada_2 = '1') then
26                      STATE <= S_2;
27                  end if;
28
29              when S_1 =>
30                  if(Entrada_2 = '1') then
31                      STATE <= S_2;
32                  elsif(Entrada_3 = '1') then
33                      STATE <= S_N;
34                  end if;
35
36          ... Aquí se definen el resto de estados del sistema ...
37
38              when S_7 =>
39                  if(Entrada_4 = '1') then
40                      STATE <= S_N;
41                  end if;
42
43              when S_N =>
```

```
44
45              end case;
46          end if;
47      end process;
48
49      -- Salidas en función del estado (Moore) o del estado y de alguna señal
50      -- de entrada (Mealy).
51      Led <= '1' when (STATE = ... ) else '0';
```