Circuitos Lógicos: SDI-11322
Práctica 2. BCD a 7 segmentos

Departamento Académico de Sistemas Digitales
Instituto Tecnológico Autónomo de México

Otoño 2018

1. Objetivos

Que el alumno:

- Se familiarice con el uso del software *Quartus* y *Simulink*
- Se familiarice con el uso de circuitos contadores
- Comience a trabajar tanto con esquemáticos como VHDL
- Empiece a utilizar la FPGA para la implementación de circuitos lógicos

2. Problema

Se requiere implementar, simular y alambrar un circuito que reciba cuatro entradas BCD a través de dip switches y despliegue el número correspondiente mediante cuatro *displays* de siete segmentos.

3. Esbozo de la solución

En la Fig. 1 se muestra el esquema general del circuito a implementar. El circuito se va a implementar en *Quartus*, para lo cual es conveniente separarlo en distintos segmentos. La práctica consiste en:

1. Implementar los componentes
   a) Decodificador 2 a 4
   b) Multiplexor 2 a 1
   c) Decodificador BCD a 7 segmentos
2. Simular los componentes
3. Implementar el circuito principal

Entrada		Salida			
S_0	S_1	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Cuadro 1: Tabla de verdad para el decodificador 2 a 4

4. Alambrar el circuito general en la protoboard
5. Programar la FPGA

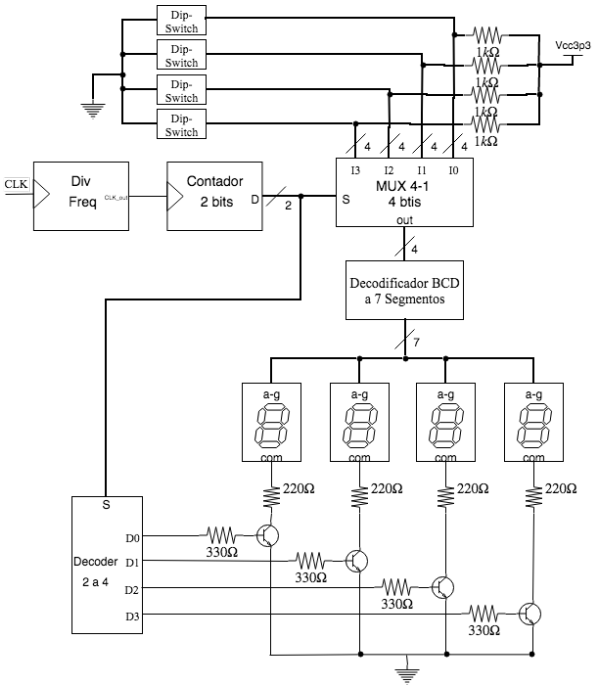

Figura 1: Esquema del circuito a implementar

3.1. Componentes a implementar

3.1.1. Decodificador 2 a 4

Recibe dos bits de entrada y tiene cuatro bits a la salida. De acuerdo a la tabla 2. Implementarlo utilizando VHDL. En la sección 4.2 se muestra como implementar éste componente.

	Entrada				Salida						
HEX	m_3	m_2	m_1	m_0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
A	1	0	1	0	1	1	1	0	1	1	1
B	1	0	1	1	0	0	1	1	1	1	1
C	1	1	0	0	1	0	0	1	1	1	0
D	1	1	0	1	0	1	1	1	1	0	1
E	1	1	1	0	1	0	0	1	1	1	1
F	1	1	1	1	1	0	0	0	1	1	1

Cuadro 2: Tabla de verdad para circuito BCD a 7 segmentos

3.1.2. Multiplexor 4 a 1

Se requiere un multiplexor 4 a 1, donde cada entrada es un vector de 4 bits. Implementarlo utilizando VHDL a partir del ejemplo en la sección 4.2.

3.1.3. Decodificador BCD a 7 segmentos

El decodificador BCD a 7 segmentos recibe de entrada un vector de 4 bits que representa un número BCD y a la salida se tiene la representación en 7 bits necesaria para representar dicho número en un *display* de 7 segmentos. Usualmente la codificación BCD solamente incluye número del 0 al 9, por conveniencia, éste componente puede recibir números del 0 al 15 y la salida se encuentra en su representación hexadecimal en 7 segmentos. En la tabla 2 se muestran las entradas y salidas del circuito. Las ecuaciones características obtenidas se muestran en 1. Implementarlo utilizando VHDL a partir del ejemplo en la sección 4.2.

$$
\begin{aligned}
a &= \overline{m_2 m_0} + m_1 m_2 + \overline{m_3} m_1 + \overline{m_3} m_2 m_0 + m_3 \overline{m_1 m_0} + m_3 \overline{m_2 m_1} \\
b &= \overline{m_3 m_2} + \overline{m_2 m_0} + m_3 \overline{m_1} m_0 + \overline{m_3 m_1 m_0} + \overline{m_3} m_1 m_0 \\
c &= \overline{m_1} m_0 + m_3 \overline{m_2} + \overline{m_3} m_2 + \overline{m_3 m_1} + \overline{m_3} m_0 \\
d &= m_3 \overline{m_1 m_0} + m_2 \overline{m_1} m_0 + m_2 m_1 \overline{m_0} + \overline{m_2} m_1 m_0 + \overline{m_3 m_2 m_0} \\
e &= m_3 m_2 + m_1 \overline{m_0} + m_3 m_1 m_0 + \overline{m_2 m_1 m_0} \\
f &= \overline{m_1 m_0} + m_3 \overline{m_2} + m_3 m_1 + \overline{m_3} m_2 \overline{m_1} + m_2 m_1 \overline{m_0} \\
g &= m_3 \overline{m_2} + m_1 \overline{m_0} + m_3 m_0 + \overline{m_2} m_1 + \overline{m_3} m_2 \overline{m_1}
\end{aligned}
\tag{1}
$$

3.2. Simulación: *ModelSim*

ModelSim es el software mediante el cual se pueden simular los circuitos creados en *Quartus* previo a programarlos a la FPGA. En general, es recomendable simular cada componente que se implementa de manera individual previo a incluirlo en el proyecto general. Para ésta practica, por tratarse de componentes sencillos, es posible simular todas las entradas para verificar todas las posibles salidas. Para componentes más complejos, se pueden simular casos específicos para verificar esas salidas. En 4.3 se describe el uso básico de modelsim.

3.3. Circuito principal

El circuito general es conveniente implementarlo utilizando BDF (ver 4.5) en un nuevo archivo. A éste circuito se importar los componentes implementados (decodificadores, multiplexor) así como el contador y el divisor de frecuencia. Tanto el contador como el divisor de frecuencia **NO** se tienen que implementar, se pueden descargar desde la página del curso.

3.4. Contador

Un circuito contador tiene como entrada una señal de reloj y a la salida se tiene una señal de n bits que incrementa en 1 a cada nuevo ciclo de reloj hasta llegar a $2^n - 1$ y después comienza desde 0.

3.4.1. Divisor de Frecuencia

Un divisor de frecuencia recibe una señal de reloj de entrada y a la salida se tiene una señal que corresponde a un reloj de menor frecuencia.

3.5. Circuito general

El circuito general se puede dividir en las siguiente partes: circuito de entrada, FPGA y circuito de salida/visualización.

3.5.1. Circuito de entrada

El circuito de entrada es la interfaz mediante la cual se van a introducir señales a la FPGA.

3.6. FPGA

Los circuitos implementados en *Quartus*, se programan en la FPGA. Cuando se tenga el circuito completo, conectar la FPGA a la computadora y programarla (ver 4.6).

3.6.1. Circuito de salida/visualización

El circuito de salida consiste en displays de siete segmentos y transistores con sus correspondientes resistencias.

4. Aspectos técnicos de implementación

4.1. Quartus

Quartus es un software para diseño de circuitos lógicos, los cuales se pueden tanto simular como programar en una *FPGA*. Para la *FPGA* que se va a utilizar durante el curso, para cada nuevo proyecto se debe seleccionar como dispositivo Cyclone IV E: EP4CE22F17C6. Un proyecto en *Quartus* puede contener múltiples archivos, en este curso se estará trabajando tanto con archivos tipo *Block Diagram/Schematic File* como archivos *VHDL*. Crear un nuevo proyecto mediante: File $->$ New Proyect Wizard. Se abre una ventana para configurar el proyecto: seleccionar la ubicación del proyecto y su nombre. Seleccionar un proyecto vacio y no se agregan archivos. En la ventana *Family, Device & Board Settings* seleccionar *Family* $->$ *Cyclone IV E*, en *Available devices* $->$ EP4CE22F17C6, el resto se quedan con la opción preestablecida y finalizar la configuración: botón *Finish*.

4.2. VHDL

Algunas tutoriales y referencias de VHDL son:

- M. Morris Mano. *Logic and Computer Design Fundamentals*. Prentice Hall; 4 edition (June 17, 2007), 2007. ISBN: 013198926X, Capítulo 4
- David Jaime González Maxinez. *Programación de sistemas digitales con VHDL*. Grupo Editorial Patria, 2014
- David G Maxinez y Jessica Alcalá Jara. "VHDL: el arte de programar sistemas digitales". En: ()
- Jan Van der Spiegel. *VHDL Tutorial*. 2006. URL: `https://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html`

- Weijun Zhang. *VHDL Tutorial: Learn by Example*. 2001. URL: `http://esd.cs.ucr.edu/labs/tutorial/`
- *Compact Summary of VHDL*. 2009. URL: `https://www.csee.umbc.edu/portal/help/VHDL/summary.html`

Crear un nuevo archivo *VHDL*. A continuación se muestra el código mínimo para implementar el circuito aritmético.

Primero, se importan las librerias necesarias. Particularmente, la librería *ieee.std_logic_arith.all* permite utilizar operadores como +, -, =, etc.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

A continuación se tiene que definir una entidad que será utilizada despues. En éste caso, la entidad se llama *decoder_2_to_4* la cual tiene un vector S de dos bits y cuatro bits de salida.

```
5  entity decoder_2_to_4 is
6     port(
7           S: in std_logic_vector(0 to 1);
8           D0: out std_logic;
9           D1: out std_logic;
10          D2: out std_logic;
11          D3: out std_logic
12          );
13 end decoder_2_to_4;
```

A continuación, se tiene que implementar la *lógica* del circuito. Para circuitos combinacionales la forma general de hacer es:

```
14 architecture arch_name of entity_name is
15     % comment
16     % auxiliary signals
17     signal sig_name: signal_type := init_val;
18         begin
19                 % program
20 end arch_name;
```

Para el caso anterior, *arch_name* representa el nombre dado a la arquitectura. *Entity_name* es el nombre que se utilizó en la parte anterior. A cada señal se le debe dar un nombre único (*sig_name*) y se le debe asignar un tipo, algunos de los más comunes son: *std_logic* y *std_logic_vector*. Se puede asignar un valor inicial.

A continuación se presentan dos versiones distintas que implementan el decodificador 2 a 4. La primera utiliza las ecuaciones obtenidas de la tabla de verdad mientras que la segunda utiliza un tipo de asignación *with/select*.

4.2.1. Ecuaciones:

```
14 architecture function_table of decoder_2_to_4 is
15         begin
16                 D0 <= NOT(S(0)) AND NOT(S(1)) ;
17                 D1 <= NOT(S(0)) AND S(1);
18                 D2 <= S(0) AND NOT(S(1));
19                 D3 <= S(0) AND S(1);
20 end function_table;
```

4.2.2. *with/select*

```
14 architecture function_table of decoder_2_to_4 is
15   signal temp: std_logic_vector(0 to 3) := "0000";
16   begin
17     with S select
18       temp <= "1000" when "00",
19               "0100" when "01",
20               "0010" when "10",
21               "0001" when "11",
22               "XXXX" when others;
23         D0 <= temp(0);
24         D1 <= temp(1);
25         D2 <= temp(2);
26         D3 <= temp(3);
27 end function_table;
```

Cuando se tenga implementado el circuito, utilizar el botón *Start Compilation* (Processing − > Start Compilation) para compilar el proyecto. Si existen errores, revisar el mensaje y corregir lo que sea necesario.

4.3. ModelSim-Altera

Si es la primera vez que se utiliza ModelSim-Altera en una computadora, es necesario configurar el simulador en "Tools − > Options − > EDA Tool Options". En la sección de **Modelsim-Altera** debe de ir la siguiente ruta `/opt/altera/modelsim_ase/linuxaloem`. Despues, checar que la configuración se encuentre igual a la tabla 2 mediante: "Assignments − > Settings − > EDA Tool Settings − > Simulation".

Option	Value
Tool name	**ModelSim-Altera**
Run gate-level simulation automatically after compilation	Disable checkbox
Format for output netlist	**Verilog HDL**
Map illegal VHDL characters	Disable checkbox
Enable glitch filtering	Disable checkbox
Generate Value Change Dump (VCD) file script	Disable checkbox

Figura 2: Configuración

Para abrir model sim: "Tools $->$ Run Simulation Tool $->$ RTL Simulation". Se debe abrir una ventana nueva con ModelSim.

En la parte izquierda de la ventana ModelSim, se encuentra la sección "Library", expandir "Work $->$ *nombre_del_proyecto*". Doble click al proyecto. Las señales del proyecto aparecen en la sección *objects*. Ahora se deben definir los valores que tomarán las señales durante la simulación: click derecho en una señal de entrada $->$ Modify $->$ Apply Wave. Existen 5 opciones:

- Clock - Solamente se puede utilizar con señales de 1 bit.
- Constant - De tratarse de un vector, es posible asignar el valor en formato binario o hexadecimal.
- Random - Para los propósitos del laboratorio, éste tipo de señal no resulta muy util.
- Repeater - Se pueden definir múltiples valores que se van a repetir de acuerdo a los parámetros que se utilicen
- Counter - Similar a *Repeater* pero solamente se define el valor inicial y el final.

Es importante tomar en cuenta la unidad en que se realicen las simulaciones, picosegundos (ps) suele ser una buena opción. Es posible agregar señales por partes variando el tiempo de inicio y fin, por ejemplo, el resultado de una señal *Counter* se puede obtener utilizando segmentos de señales *Constant*.

Para correr la simulación, agregar las señales de salida: Simulate $->$ Run $->$ Run-All. Verificar que las señales de salida tengan los valores correctos.

4.4. Importar archivos

Para agregar un archivo al proyecto, desde la aplicación *Files* (navegador de archivos del sistema operativo) copiar el archivo a la carpeta del proyecto y agregarlo desde *Quartus* al proyecto: Project $->$ Add/Remove Files in Project.

4.5. *Block Diagram/Schematic File*

Crear un nuevo archivo de tipo *Block Diagram/Schematic File*. Una vez creado el archivo, en la parte izquierda de la pantalla en *Proyect Navigator* cambiar la vista de *Hierarchy* a *Files*. Buscar el archivo BDF que se acaba de crear, hacer click derecho y establecerlo como *Top level entity*. **NOTA:** ModelSim no puede simular archivos BDF, si se requiere simular, es necesario convertirlos a VHD. Por esto, es importante simular los componentes de manera individual antes de crear el archivo BDF.

En este tipo de archivo, el circuito se implementa gráficamente. Para seleccionar un componente para ser incluido en el circuito, utilizar el botón que se muestra en la figura 3 que se encuentra en la barra de herramientas de quartus. Mediante ésta herramienta se pueden agregar compuertas lógicas y otros componentes como *vcc* y *gnd*.

Figura 3: Simulación

Utilizando el archivo BDF, implementar el circuito principal. Es conveniente implementar las entradas y salidas mediante buses además de ser necesario crear símbolos para los componentes que se tienen en VHDL.

4.5.1. Crear símbolo

En *Quartus*, la ventana izquierda *Project Navigator* hacer click en *Hierarchy* y seleccionar *Files*. Hacer click derecho al archivo deseado y seleccionar *Create Symbol Files for Current File*. Ahora ya se puede agregar el nuevo símbolo desde el botón *Symbol Tool* (figura 3).

4.5.2. Bus

Para representar un bus en *Quartus*, seleccionar el botón apropiado en la barra de herramientas. Es posible utilizar entradas como buses, para lo cual es necesario modificar el nombre de la entrada: *nombre*[0..*N*] para usar notación *Big Endian* ó *nombre*[N..*0*] para *Little Endian*. En la figura 4 se muestra un ejemplo.

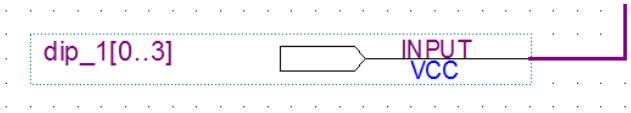

Figura 4: Entrada mediante un *bus*

4.6. FPGA

Cuando se tenga compilado el proyecto, se tienen que asignar las entradas y salidas del circuito implementado a pines de la FPGA. En el manual de la FPGA se puede encontrar la relación de las señales y los pines.

4.6.1. Pin Planner

Se deben asignar los pines de entrada y salida que se van a utilizar en la FPGA. Para el caso de la señal de reloj (la entrada *Clock* del contador) se debe asignar la señal del reloj de la tarjeta, la cual está fija en *PIN_R8*. ¿Cuál es la frecuencia del reloj asociado con éste *PIN*? El resto de las señales se pueden asignar a los pines más convenientes. Para revisar detalles sobre los pines y de la FPGA en general, es necesario consultar el manual de la FPGA el cual se encuentra en la página del curso.

NOTA: Es necesario compilar el proyecto después de la asignación de todos los pines.

4.6.2. Alambrado

Para el alambrado del circuito en la protoboard, conviene dividir el trabajo entre los miembros del equipo. En la *protoboard* se deben alambrar:

- Cuatro dip switches de cuatro salidas cada uno.
- Cuatro displays de siete segmentos de cátodo común. Notar las salidas del circuito BCD a siete segmentos implementado en la FPGA son comúnes a los cuatro displays.
- Transistores asociados a los displays.
- Resistencias asociados a los displays y transistores.
- Vcc a los dip switches
- Tierra a los transistores.

Antes de conectar la FPGA a la computadora, revisar el alambrado.

4.6.3. Programar la FPGA

Para programar la FPGA, buscar el botón de *Programmer* en la barra de herramientas de *Quartus*. Se debería mostrar automáticamente el dispositivo en la computador y solamente es necesario apretar el botón *upload*.

4.6.4. Extra: BDF a VHDL

Si fuera necesario simular un archivo BDF, a continuación se describen los pasos para convertir de BDF a VHDL y poder simular.

La herramienta que permite simular los circuitos es ModelSim-Altera. Esta aplicación utiliza archivos de descripción de lenguaje así como archivos *bech* para simular. Para generar un archivo vhdl sólo dirigete al archivo de esquema de bloques y selecciona la opción "File $->$ create / update $->$ Create HDL Design File from Current file", selecciona la opción VHDL. Finalmente agrega el archivo al proyecto "Assignments $->$ Settings". Esto mostrará una ventana, en la sección "Files" busca el archivo con el boton "..." y agregalo "add", pero retira el archivo de bloques con "remove". Aplica los cambios y cierra la ventana.

5. Validación

Es necesario realizar la simulación de los componentes implementados mediante ModelSim. Verificar que los resultados sean los esperados.

Al alambrar el circuito, las principales fuentes de error son:

- Asignación de pines
- Conexión de los transistores
- Conectar el dipswitch correctamente (revisar las resistencia pullup)

Cuando el circuito se encuentre funcionando, realice un experimento para determinar un rango de frecuencias en las que el circuito funciona correctamente.

¿Porqué si el decodificador a siete segmentos saca la misma señal a los *displays*, se pueden ver números distintos?

textbooks