DEPARTAMENTO DE ELECTRÓNICA Y AUTOMÁTICA

FACULTAD DE INGENIERÍA – UNIVERSIDAD NACIONAL DE SAN JUAN

Informe de Laboratorio Nº 2

Descripción de Hardware con Instrucciones Concurrentes

[Fecha de Presentación: 28 de abril del 2021]

Asignatura: Temas Específicos de Electrónica Digital I

Ingeniería Electrónica

Autor:

Echenique, Leonardo – Registro 28351

1º Semestre

Año 2021

Índice de Contenido

1. Objetivos 1
2. Desarrollo 1
2.1. Decodificador 3 a 8 1
2.2. Decodificador de BCD a 7segmentos 3
2.3. Decodificador de Hexadecimal a 7segmentos 5
2.4. Sistema BCD-7segmento y Multiplexers 4 a 2 7
3. Conclusiones 12
4. Apéndice 13
4.1. Descripciones VHDL 13
4.1.1. Decodificador 3 a 8 13
4.1.2. Decodificador BCD a 7segmento 13
4.1.3. Decodificador Hexadecimal a 7segmento 14
4.1.4. Multiplexer 14
4.1.5. Sistema 15
4.2. Descripciones Test Bench 15
4.2.1. Decodificador 3 a 8 15
4.2.2. Decodificador BCD a 7segmento 16
4.2.3. Decodificador Hexadecimal a 7segmento 16
4.2.4. Multiplexer 17
4.2.5. Sistema 18

Índice de Figuras

Figura 2. 1. 1 - Esquemático de Decodificador 3 a 8 2
Figura 2. 1. 2 – Simulación de Decodificador 3 a 8 2

Figura 2. 2. 1 – Esquematico de Decodificador BCD a 7segmentos 4
Figura 2. 2. 2 - Simulación de Decodificador BCD a 7segmentos 4

Figura 2. 3. 1 - Esquemático de Decodificador Hexadecimal a 7segmentos 6
Figura 2. 3. 2 - Simulación de Decodificador Hexadecimal a 7segmentos 6

Figura 2. 4. 1 - Esquemático de Multiplexer 8
Figura 2. 4. 2 - Simulación de Multiplexer 8
Figura 2. 4. 3 - Esquemático de Sistema 9
Figura 2. 4. 4 - Simulación de Sistema 9
Figura 2. 4. 5 - Caminos Críticos 10
Figura 2. 4. 6 – Información del camino crítico con mayor retardo 11
Figura 2. 4. 7 - Technology View 11
Figura 2. 4. 8 - Chip Planner 11

Índice de Tablas

Tabla 2. 1. 1 - Asignación de Pines Decodificador de 3 a 8 3
Tabla 2. 1. 2 - Reporte de Área de Decodificador de 3 a 8 3

Tabla 2. 2. 1 - Tabla de Verdad de Decodificador BCD a 7segmentos 4
Tabla 2. 2. 2 - Asignación de Pines a Decodificador BCD a 7segmentos..... 5
Tabla 2. 2. 3 - Reporte de Área de Decodificador BCD a 7segmentos..... 5

Tabla 2. 3. 1 - Tabla de Verdad de Decodificador Hexadecimal a 7segmentos..... 6
Tabla 2. 3. 2 - Asignación de Pines de Decodificador Hexadecimal a 7segmentos..... 7
Tabla 2. 3. 3 - Reporte de Área de Decodificador Hexadecimal a 7segmentos..... 7

Tabla 2. 4. 1 - Reporte de Área de Multiplexer..... 8
Tabla 2. 4. 2 - Reporte de Área del Sistema 9
Tabla 2. 4. 3 - Asignación de Pines del Sistema 10

1. Objetivos

En el presente informe se desarrollan los pasos seguidos para la descripción de algunos componentes utilizando únicamente instrucciones concurrentes del lenguaje VHDL. Esto se realizará con la ayuda de las herramientas proporcionadas por el software Quartus, y mediante ModelSim se comprobarán sus funcionamientos. A continuación, se muestran los objetivos complementarios del laboratorio:

- Uso de la guía de usuario del board DE2-115.
- Realizar estudio y determinación de camino crítico combinacional.
- Generar archivo de configuración de FPGA (Cyclone IV EP4CE115F29C7).
- Configuración del FPGA con el código VHDL correspondiente.

2. Desarrollo

Esta sección divide los diferentes procedimientos llevados a cabo para la descripción de cada componente o sistema de ellos (Decodificador 3 a 8, Decodificador BCD a 7segmentos, Decodificador Hexadecimal a 7segmentos).

Pero antes del comienzo de las descripciones de cada componente, se presentaron algunos inconvenientes respecto con la utilización de la herramienta Quartus. Los mismos se muestran a continuación, con sus respectivas soluciones:

a) Error al crear un nuevo proyecto: Se debe modificar la dirección de destino del proyecto que esta por defecto, por otra cualquiera que se desee. Esto se realiza en la ventana de ayuda de creación de un nuevo proyecto.
b) Error al abrir la herramienta ModelSim: Para solucionarlo es necesario dirigirse, dentro de la ventana de Quartus, a la pestaña *Tools>Options>EDA Tools Options>ModelSim-Altera*, y ahí colocar la dirección donde se encuentra la carpeta win32aloem (Esta se ubica en la carpeta donde se instaló Quartus).

También, al realizar las compilaciones correspondientes, en todos los componentes aparecían los siguientes mensajes de *warnings*. A pesar que esto no eran errores como tal, se buscó solucionarlos dentro de las posibilidades.

- [Warning (332068): No clocks defined in design]: este se debe a que, al describir circuitos combinacionales, no es necesario ningún reloj en el diseño.
- [Warning (18236): Number of processors has not been specified which may cause overloading on shared machines. Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an appropriate value for best performance.]: este es debido a que Quartus necesita la especificación de cuantos núcleos puede utilizar, para obtener un mejor rendimiento. Para solucionarlo, se debe abrir el archivo *.qpf* que se encuentra en la misma carpeta que el proyecto. Luego de abrirlo, se debe escribir la siguiente sentencia {set_global_assignment -name NUM_PARALLEL_PROCESSORS n}, donde n corresponde al número de núcleos que podrá usar Quartus. Se guardan los cambios y se vuelve a compilar el proyecto.

2.1. Decodificador 3 a 8

La relación entre las entradas y salidas de un decodificador está dada por $s = 2^e$, donde *"s"* es la cantidad de salidas, y "*e*" la cantidad de entradas. El decodificador descripto en este caso cuenta con $e = 3$ entradas, por lo que se tienen $s = 2^3 = 8$ salidas, y con una entrada de habilitación en bajo. Dependiendo de la combinación de bits presentes en la entrada, se activa (o desactiva) una de las salidas.

Teniendo en cuenta lo anterior, la descripción del mismo en VHDL (Apéndice 4.1.1) presenta una *entidad* con cuatro entradas (vector de 3 bits selectores y 1 bit de habilitación) y una salida (vector de 8 bits). Dentro de la *arquitectura* se declaró una señal auxiliar, la cual estará conformada por la concatenación de los bits selectores y el bit de selección.

Dependiendo del valor de esta señal se realizó la asignación de los bits de salida, utilizando para ello la instrucción With-Select.

El esquemático del diseño obtenido (Figura 2.1.1) muestra la forma en que la herramienta Quartus sintetiza el hardware descripto. A su vez, también se realizó la simulación correspondiente para comprobar el correcto funcionamiento. Para esto último, se generaron ondas acordes a las posibles entradas, realizando el llamado TestBench (Apéndice 4.2.1) con instrucciones Wait. Este se describió justo a continuación del fin de la arquitectura del componente. Utilizando la herramienta ModelSim, se visualizan las ondas de entrada y sus correspondientes salidas (Figura 2.1.2).
El procedimiento llevado a cabo para la obtención de estas figuras (**igual forma para todos los componentes**) es: dentro de la ventana de Quartus, acceder a la pestaña *Tools>Run Simulation Tools>RTL* Simulation. Luego, automáticamente se abre la ventana del software ModelSim, donde se selecciona la pestaña *Compile>Compile…*, se despliega una ventana donde se busca el archivo *.vhd* con el test bench que se necesite, se selecciona el mismo y click en Compile. Después, se selecciona en la pestaña *Simulate>Start Simulation* y se extiende una nueva ventana; dentro de ella elegir *Work>testbenchXXX*. De esta manera, se obtienen las entradas y salidas en la ventana de ModelSim, a las cuales se les aplica click secundario y se elige Add Wave, para cada una. Esta acción conlleva a abrir una ventana donde será posible visualizar las ondas con sus respectivos valores. En la barra de tareas aparece la opción de modificar el tiempo de simulación, y junto ella la opción de simular, Run.

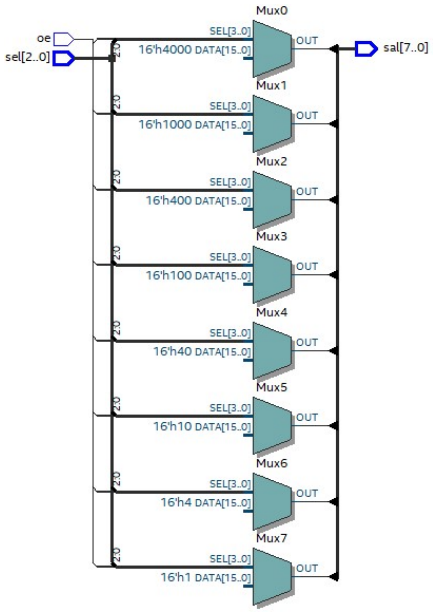

Figura 2. 1. 1 - Esquemático de Decodificador 3 a

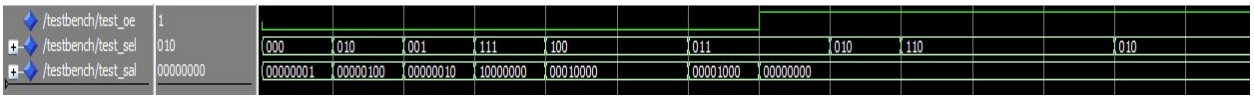

Figura 2. 1. 2 – Simulación de Decodificador 3 a 8

Como se observa, las salidas obtenidas a partir de las entradas dadas son correctas, denotando el correcto funcionamiento del diseño realizado.
Finalmente, se realizó la asignación correspondiente de los pines del FPGA utilizado. Para eso, se utilizó el manual de usuario del board DE2-115. En este componente se dio uso a los switches (SW0 a S23) y Leds (Verdes) del board. La siguiente tabla (Tabla 2.1.1) muestra la relación establecida entre los pines del board y las E/S del diseño realizado.

Tabla 2. 1. 1 - Asignación de Pines Decodificador de 3 a 8

SEÑAL	PIN
oe	PIN_AD27
sal[7]	PIN_G21
sal[6]	PIN_G22
sal[5]	PIN_G20
sal[4]	PIN_H21
sal[3]	PIN_E24
sal[2]	PIN_E25
sal[1]	PIN_E22
sal[0]	PIN_E21
sel[2]	PIN_AC27
sel[1]	PIN_AC28
sel[0]	PIN_AB28

También, se obtuvo el Reporte de Área (Tabla 2.1.2), en el cual se especifica los porcentajes utilizados de los recursos disponibles del FPGA.

Tabla 2. 1. 2 - Reporte de Área de Decodificador de 3 a 8

Recurso	Porcentaje
Total de elementos lógicos	8 / 114,480 (< 1 %)
Total registros	0
Total pins	12 / 529 (2 %)
Total pins virtuales	0
Total bits de memoria	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

2.2. Decodificador de BCD a 7segmentos

Este tipo de decodificador, tal como lo indica su nombre, recibe a su entrada un numero representado en el sistema numérico BCD (Decimal Codificado en Binario) y el cual a la salida es descifrado de tal forma que, pueda representarse en un display de 7 segmentos del tipo ánodo común (cada segmento se activa en bajo).

Con las características mencionadas, la descripción del hardware en VHDL (Apéndice 4.1.2) tiene una *entidad* con una entrada (vector de 4 bits) y una salida (vector de 7 bits para 7segmentos). En la arquitectura se describió el funcionamiento del componente utilizando la instrucción With-Select, por el cual se empleaban los bits de entrada para asignar a la salida los bits correspondientes para formar el respectivo numero decimal. Esto es más visible en la tabla de verdad (Tabla 2.2.1) que se armó para facilitar su codificación.

El esquemático del diseño (Figura 2.2.1) se obtuvo a través de la opción RTL Viewer de la herramienta Quartus, donde se puede observar que no se haya generado ningún Latch erróneo, también se puede asegurar esto leyendo los warnings producidos por la compilación. Por otra parte, se generaron ondas con el objetivo de simular las posibles entradas, y verificar si las salidas obtenidas eran las correctas. Este TestBench (Apéndice 4.2.2) se describió a continuación de la descripción del decodificador, en el cual se utilizaron instrucciones de Wait para la generación de las ondas. Se siguieron los mismos pasos que se explicaron en el anterior decodificador, para poder visualizar las señales de entrada y salida con la herramienta ModelSim (Figura 2.2.2).

Tabla 2. 2. 1 - Tabla de Verdad de Decodificador BCD a 7segmentos

ENTRADA	SALIDA
0000	1000000
0001	1111001
0010	0100100
0011	0110000
0100	0011001
0101	0010010
0110	0000010
0111	1111000
1000	0000000
1001	0010000

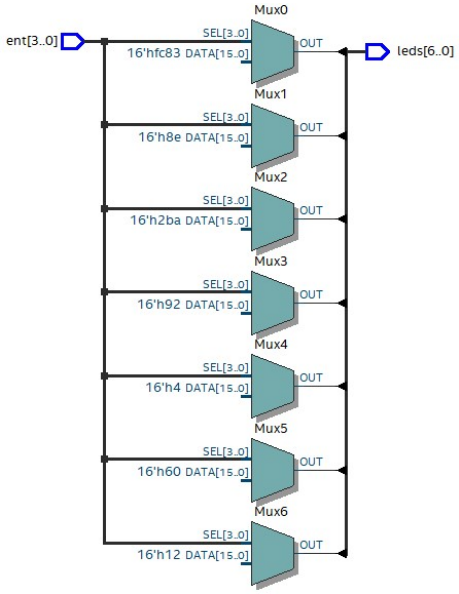

Figura 2. 2. 1 – Esquematico de Decodificador BCD a 7segmentos

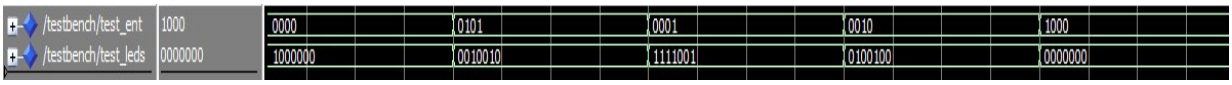

Figura 2. 2. 2 - Simulación de Decodificador BCD a 7segmentos

La simulación realizada muestra resultados acertados al comparar los mismos con la tabla de verdad antes descripta. Por lo tanto, es posible concluir que el hardware descripto es funcional.

Una vez comprobado el funcionamiento del decodificador, se procedió a realizar la asignación correspondiente a los pines del FPGA. Esta asignación se muestra con mayor detalle en la siguiente tabla (Tabla 2.2.2), para la cual se utilizó nuevamente el manual de usuario del board DE2-115. Para este decodificador, las entradas quedaron sin asignar, para su posterior instanciación en un componente de mayor jerarquía, mientras que las salidas fueron destinadas al display 7segmentos (HEX0).

Como información complementaria, se obtuvo también el Reporte de Área (Tabla 2.2.3) con el propósito de mostrar los recursos del FPGA que utiliza este hardware descripto.

Tabla 2. 2. 2 - Asignación de Pines a Decodificador BCD a 7segmentos

SEÑAL	PIN
leds[6]	PIN_H22
leds[5]	PIN_J22
leds[4]	PIN_L25
leds[3]	PIN_L26
leds[2]	PIN_E17
leds[1]	PIN_F22
leds[0]	PIN_G18
ent[3]	-
ent[2]	-
ent[1]	-
ent[0]	-

Tabla 2. 2. 3 - Reporte de Área de Decodificador BCD a 7segmentos

Recurso	Porcentaje
Total de elementos lógicos	7 / 114,480 (< 1 %)
Total registros	0
Total pins	11 / 529 (2 %)
Total pins virtuales	0
Total bits de memoria	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Luego de realizadas todas las simulaciones, es posible generar el archivo de configuración del FPGA. Para ello es necesario tener instalado el driver USB-Blaster. Dentro de la ventana de Quartus, se elige la opción Program Device (Open Programmer), donde se desplegará una nueva ventana. En ella, se debe selecciona el modo JTAG y verificar que en Hardware Setup muestre USB-Blaster. Por otra parte, también comprobar que el archivo *.sof* tenga seleccionada la opción Program/Configure. Una vez verificado todo lo anterior, presionando Start el programa se enviará a la placa FPGA a través de un cable USB, que interconecta la PC host y la placa.

2.3. Decodificador de Hexadecimal a 7segmentos

La diferencia entre el decodificador anterior y el descripto en esta sección, es que este último descifra números entre 0 y 15, por lo que serán necesario dos display de 7segmentos para mostrar el resultado correctamente.

De igual manera, las descripciones en VHDL (Apéndice 4.1.3) son bastantes similares. La *entidad* cuenta con una entrada (vector de 4 bits) y dos salidas (vectores de 7 bits para cada 7segmento). También, en la *arquitectura* se utilizó la instrucción concurrente With-Select, pero en este caso se declaró una señal auxiliar de 13 bits que dio la posibilidad de recibir el valor completo de la salida, y luego ser desglosado para cada display según correspondía. Nuevamente, dependiendo de las entradas recibidas, se asigna cierto valor a la señal de salida. Se genero una tabla de verdad (Tabla 2.3.1) para facilitar el entendimiento entre las entradas y salidas.

El esquemático del diseño (Figura 2.3.1) se obtuvo a través de la opción RTL Viewer de la herramienta Quartus, el cual permite garantizar que no se generó ningún Latch de forma errónea, y a su vez por medio de los warnings también se concluye lo mismo. Por su parte, también se simulo el diseño con la ayuda de la herramienta ModelSim, describiendo el TestBench (Apéndice 4.2.3) correspondiente a continuación de la descripción del

decodificador, utilizando nuevamente las instrucciones Wait. La visualización de las señales (Figura 2.3.2) se realizó siguiendo los pasos descriptos en el decodificador 3 a 8.

Tabla 2. 3. 1 - Tabla de Verdad de Decodificador Hexadecimal a 7segmentos

ENTRADA	SALIDA 1 (Decena)	SALIDA 2 (Unidad)
0000	1000000	1000000
0001	1000000	1111001
0010	1000000	0100100
0011	1000000	0110000
0100	1000000	0011001
0101	1000000	0010010
0110	1000000	0000010
0111	1000000	1111000
1000	1000000	0000000
1001	1000000	0010000
1010	1111001	1000000
1011	1111001	1111001
1100	1111001	0100100
1101	1111001	0110000
1110	1111001	0011001
1111	1111001	0010010

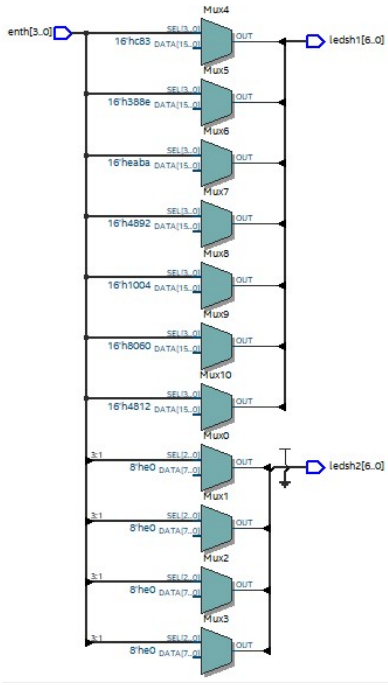

Figura 2. 3. 1 - Esquemático de Decodificador Hexadecimal a 7segmentos

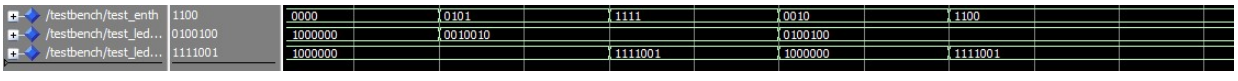

Figura 2. 3. 2 - Simulación de Decodificador Hexadecimal a 7segmentos

La respectiva asignación de los pines del FPGA, mostrada con más detalle en la siguiente tabla (Tabla 2.3.2), los pines de entrada se asignaron a los switches de la placa (SW4 a SW7). A su vez, las salidas fueron asignadas a los displays de 7segmentos HEX6 (Unidad) y HEX7 (Decenas).

Tabla 2. 3. 2 - Asignación de Pines de Decodificador Hexadecimal a 7segmentos

SEÑAL	PIN
ledsh1[6]	PIN_AC17
ledsh1[5]	PIN_AA15
ledsh1[4]	PIN_AB15
ledsh1[3]	PIN_AB17
ledsh1[2]	PIN_AA16
ledsh1[1]	PIN_AB16
ledsh1[0]	PIN_AA17
enth[3]	PIN_AB26
enth[2]	PIN_AD26
enth[1]	PIN_AC26
enth[0]	PIN_AB27
ledsh2[6]	PIN_AA14
ledsh2[5]	PIN_AG18
ledsh2[4]	PIN_AF17
ledsh2[3]	PIN_AH17
ledsh2[2]	PIN_AG17
ledsh2[1]	PIN_AE17
ledsh2[0]	PIN_AD17

Una vez más, se obtuvo el Reporte de Área (Tabla 2.3.3) buscando mostrar los recursos del FPGA utilizados por el hardware descripto.

Tabla 2. 3. 3 - Reporte de Área de Decodificador Hexadecimal a 7segmentos

Recurso	Porcentaje
Total de elementos lógicos	8 / 114,480 (< 1 %)
Total registros	0
Total pins	18 / 529 (3 %)
Total pins virtuales	0
Total bits de memoria	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

2.4. Sistema BCD-7segmento y Multiplexers 4 a 2

Inicialmente, se describió de forma individual el Multiplexer. Este componente consta de 2^n entradas, donde n corresponde al número de entradas de selección que posee. Todos los Multiplexer poseen una salida, a la cual se les asigna una de las posibles entradas, dependiendo del valor de la/s entrada/s de selección. En este caso, se tiene dos entradas de selección y cuatro entradas que se pueden destinar a la salida.

Teniendo como premisa el funcionamiento del componente, la descripción del hardware en VHDL (Apéndice 4.1.4) presenta una *entidad* con cinco entradas (4 bits de entrada individual y un vector de 2 bits de selección) y una salida (1 bit). La *arquitectura* describe el funcionamiento de forma sencilla, utilizando la instrucción concurrente With-Select. De igual manera que los componentes anteriores, se simulo el funcionamiento describiendo las señales con la instrucción Wait. Este TestBench (Apéndice 4.2.4) se describió al finalizar la descripción del Multiplexer.

El esquemático del hardware diseñado (Figura 2.4.1) se obtuvo mediante la opción de Quartus llamada RTL Viewer. Por otra parte, usando la herramienta ModelSim, y siguiendo

los pasos antes descriptos, se observa la evolución de las entradas y salidas en el tiempo (Figura 2.4.2)

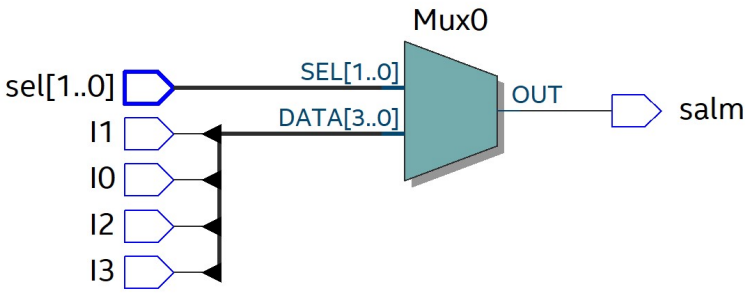

Figura 2. 4. 1 - Esquemático de Multiplexer

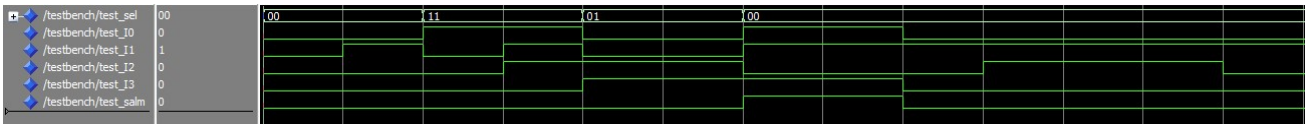

Figura 2. 4. 2 - Simulación de Multiplexer

La señal de salida obtenida en la simulación, corresponde al funcionamiento de un Multiplexer. Por lo tanto, se comprobó que el hardware descripto es correcto.
Para este componente, no se realizó asignación de pines E/S del FPGA, debido a que el mismo será utilizado en un sistema de mayor jerarquía. Sin embargo, si se obtuvo el Reporte de Área (Tabla 2.4.1) correspondiente.

Tabla 2. 4. 1 - Reporte de Área de Multiplexer

Recurso	Porcentaje
Total de elementos lógicos	2 / 114,480 (< 1 %)
Total registros	0
Total pins	7 / 529 (1 %)
Total pins virtuales	0
Total bits de memoria	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Finalmente, mediante un proyecto principal se describió un sistema en VHDL (Apéndice 4.1.5), el cual cuenta con instanciaciones de los componentes: Decodificador BCD a 7segmentos y de cuatro Multiplexers (**también de los demás componentes**). El mismo cuenta con una *entidad* de cuatro entradas (vectores de 4 bits) correspondientes a cuatro contadores BCD externos y otra entrada (vector de 2 bits) de selección, además con tres salidas (vectores de 7 bits) para los display de 7segmentos y otra salida (vector de 8 bits) a los leds verdes de la placa. En la *arquitectura* solo hay instrucciones de instanciación. Para la obtención de los cuatro Multiplexers, se recurrió a la utilización de la instrucción concurrente For-Generate.

Al igual que con el Multiplexer, se obtuvo el esquemático del diseño (Figura 2.4.3) mediante RTL Viewer, mientras que la simulación de entradas y salidas por medio de la herramienta ModelSim (Figura 2.4.4), con el llamado TestBench (Apéndice 4.2.5) utilizando instrucciones Wait.

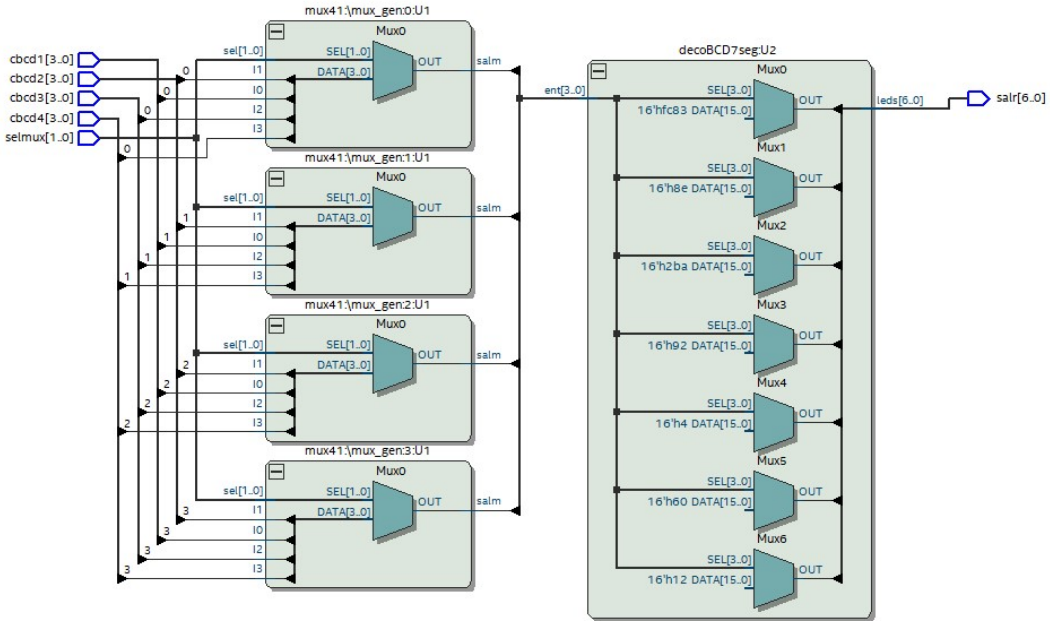

Figura 2. 4. 3 - Esquemático de Sistema

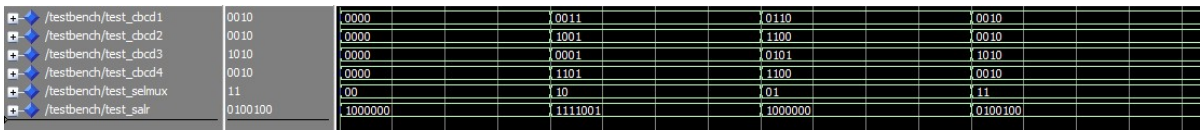

Figura 2. 4. 4 - Simulación de Sistema

Se puede apreciar, que el sistema funciona de forma correcta observando las salidas respecto de cada una de las entradas, para las señales de selección dadas.

Mediante la obtención del Reporte de Área (Tabla 2.4.2) es posible observar los recursos utilizados para la implementación del circuito completo, con todos sus componentes.

Los pines del board asignados se muestran con detalle en la siguiente tabla (Tabla 2.4.3), las entradas a los Multiplexers se asignaron a los switches (SW0 a SW17) para comprobar el funcionamiento del sistema, mientras que la salida del decodificador BCD a 7segmento al display (HEX4).

Tabla 2. 4. 2 - Reporte de Área del Sistema

Recurso	Porcentaje
Total de elementos lógicos	31 / 114,480 (< 1 %)
Total registros	0
Total pins	47 / 529 (9 %)
Total pins virtuales	0
Total bits de memoria	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Tabla 2. 4. 3 - Asignación de Pines del Sistema

SEÑAL	PIN
salr[6]	PIN_AE18
salr[5]	PIN_AF19
salr[4]	PIN_AE19
salr[3]	PIN_AH21
salr[2]	PIN_AG21
salr[1]	PIN_AA19
salr[0]	PIN_AB19
cbcd1[3]	PIN_AD27
cbcd1[2]	PIN_AC27
cbcd1[1]	PIN_AC28
cbcd1[0]	PIN_AB28
cbcd2[3]	PIN_AB26
cbcd2[2]	PIN_AD26
cbcd2[1]	PIN_AC26
cbcd2[0]	PIN_AB27
cbcd3[3]	PIN_AB24
cbcd3[2]	PIN_AC24
cbcd3[1]	PIN_AB25
cbcd3[0]	PIN_AC25
cbcd4[3]	PIN_AA22
cbcd4[2]	PIN_AA23
cbcd4[1]	PIN_AA24
cbcd4[0]	PIN_AB23
selmux[1]	PIN_Y23
selmux[0]	PIN_Y24

Finalmente, para esta última parte se obtuvo el Camino Critico del Sistema Combinacional descripto, siguiendo los pasos explicados en documento de la práctica.
Utilizando la herramienta Timing Analyzer que proporciona Quartus, se pudieron obtener los diez caminos críticos dentro del sistema (Figura 2.4.5). Sim embargo, solo se consideró el primer camino crítico, ya que es el que presentaba mayor retardo. De este último, se pudo obtener la información concreta respecto de los retardos de interconexión (IC) y los retardos lógicos (CELL), incluyendo de forma detallada la locación de las celdas lógicas usadas (Figura 2.4.6).

	Delay	From Node	To Node
1	12.851	cbcd2[3]	salr[2]
2	12.708	cbcd1[2]	salr[2]
3	12.703	selmux[0]	salr[2]
4	12.620	selmux[0]	salr[2]
5	12.617	cbcd2[3]	salr[3]
6	12.612	cbcd1[3]	salr[2]
7	12.524	cbcd1[0]	salr[2]
8	12.469	selmux[0]	salr[3]
9	12.457	cbcd1[1]	salr[2]
10	12.446	selmux[0]	salr[2]

Figura 2. 4. 5 - Caminos Críticos

	Total	Incr	RF	Type	Fanout	Location	Element
1	⌄ 12.851	12.851					data path
1	0.000	0.000			1	PIN_AB26	cbcd2[3]
2	0.000	0.000	FF	IC	1	IOIBUF_X115_Y15_N1	cbcd2[3]~input|i
3	0.796	0.796	FF	CELL	9	IOIBUF_X115_Y15_N1	cbcd2[3]~input|o
4	4.850	4.054	FF	IC	1	LCCOMB_X111_Y7_N20	\mux_gen:3:U1|Mux0~0|datab
5	5.275	0.425	FF	CELL	1	LCCOMB_X111_Y7_N20	\mux_gen:3:U1|Mux0~0|combout
6	5.549	0.274	FF	IC	1	LCCOMB_X111_Y7_N30	\mux_gen:3:U1|Mux0~1|datab
7	5.974	0.425	FF	CELL	7	LCCOMB_X111_Y7_N30	\mux_gen:3:U1|Mux0~1|combout
8	6.343	0.369	FF	IC	1	LCCOMB_X111_Y7_N12	U2|Mux4~0|dataa
9	6.734	0.391	FR	CELL	1	LCCOMB_X111_Y7_N12	U2|Mux4~0|combout
10	9.933	3.199	RR	IC	1	IOOBUF_X74_Y0_N9	salr[2]~output|i
11	12.851	2.918	RR	CELL	1	IOOBUF_X74_Y0_N9	salr[2]~output|o
12	12.851	0.000	RR	CELL	0	PIN_AG21	salr[2]

Figura 2. 4. 6 – Información del camino crítico con mayor

Para una mejor visualización del camino, se acudió a la opción que otorga la herramienta Timing Analyzer, llamada Locate Path para obtener el Technology View of the Data Path (Figura 2.4.7). En esta figura se observan las partes del sistema implementado que inciden en el retado máximo.

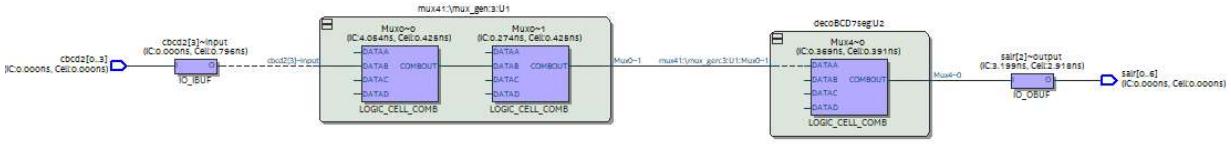

Figura 2. 4. 7 - Technology View

Finalmente, con el fin de obtener una mayor información de la distribución de la lógica y retardos del sistema descripto, usando nuevamente la herramienta Locate Path se obtuvo esta vez el Chip Planner (Figura 2.4.8). Esto proporciona la ubicación física del camino crítico, dentro de una figura real de la placa FPGA. Donde las flechas azules dentro de la figura indican el camino critico en sí.

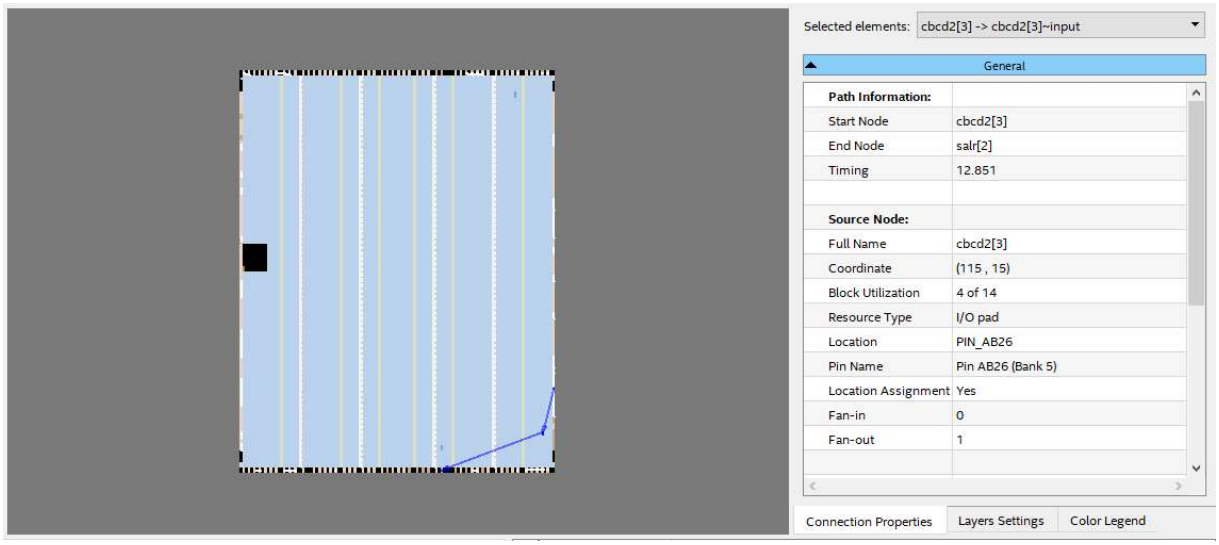

Figura 2. 4. 8 - Chip Planner

3. Conclusiones

El presente informe permitió poner en práctica nuevamente los conocimientos adquiridos en materias anteriores, como así también los obtenidos durante este curso. Respecto a los componentes realizados es posible concluir que los mismos, al ser circuitos lógicos combinacionales, pueden ser descriptos usando únicamente instrucciones concurrentes. A pesar de que esto presento algunos inconvenientes, los mismos fueron resueltos de forma satisfactoria.

4. Apéndice

4.1. Descripciones VHDL

4.1.1. Decodificador 3 a 8

```
library ieee;
use ieee.std_logic_1164.all;
entity deco38 is
  port (oe: in std_logic;
              sel: in std_logic_vector(2 downto 0);
              sal: out std_logic_vector(7 downto 0));
end deco38;
architecture behave of deco38 is
  signal ent: std_logic_vector(3 downto 0); --Señal auxiliar
begin
  ent <= std_logic_vector'(sel(2),sel(1),sel(0), oe);
  with ent select
  --Decodificador 3 a 8
        sal <=       "00000001" when "0000",
                           "00000010" when "0010",
                           "00000100" when "0100",
                           "00001000" when "0110",
                           "00010000" when "1000",
                           "00100000" when "1010",
                           "01000000" when "1100",
                           "10000000" when "1110",
                           "00000000" when others;
end behave;
```

4.1.2. Decodificador BCD a 7segmento

```
library ieee;
use ieee.std_logic_1164.all;
entity decoBCD7seg is
port(ent: in std_logic_vector(3 downto 0);
    leds: out std_logic_vector (6 downto 0));
end decoBCD7seg;

architecture behav of decoBCD7seg is
begin
  with ent select
  --Decodificador BCD a 7seg
        leds <= "1111001" when "0001",--1
                        "0100100" when "0010",--2
                    "0110000" when "0011",--3
                        "0011001" when "0100",--4
                        "0010010" when "0101",--5
                        "0000010" when "0110",--6
                        "1111000" when "0111",--7
                        "0000000" when "1000",--8
                        "0010000" when "1001",--9
                        "1000000" when others;--0
end behav;
```

4.1.3. Decodificador Hexadecimal a 7segmento

```
library ieee;
use ieee.std_logic_1164.all;
entity decoHEX7seg is
port(enth: in std_logic_vector(3 downto 0);
    ledsh1, ledsh2: out std_logic_vector (6 downto 0));
end decoHEX7seg;
architecture behav of decoHEX7seg is
  signal aux: std_logic_vector(13 downto 0);--Señal Auxiliar
begin
  with enth select
  --Decodificador HEX a 7seg
              aux <= "10000001111001" when "0001",--1
                           "10000000100100" when "0010",--2
                           "10000000110000" when "0011",--3
                           "10000000011001" when "0100",--4
                           "10000000010010" when "0101",--5
                           "10000000000010" when "0110",--6
                           "10000001111000" when "0111",--7
                           "10000000000000" when "1000",--8
                           "10000000010000" when "1001",--9
                           "11110011000000" when "1010",--10
                           "11110011111001" when "1011",--11
                           "11110010100100" when "1100",--12
                           "11110010110000" when "1101",--13
                           "11110010011001" when "1110",--14
                           "11110010010010" when "1111",--15
                           "10000001000000" when others;--0

  ledsh1 <= aux(6 downto 0);  --Unidad
  ledsh2 <= aux(13 downto 7); --Decena
end behav;
```

4.1.4. Multiplexer

```
library ieee;
use ieee.std_logic_1164.all;
entity mux41 is
  port(I0, I1, I2, I3: in std_logic;
          sel: in std_logic_vector(1 downto 0);
          salm: out std_logic);
end mux41;
architecture behav of mux41 is
begin
  with sel select
        salm <= I0 when "00",--I0
                        I1 when "01",--I1
                        I2 when "10",--I2
                        I3 when "11",--I3
                        '-' when others;
end behav;
```

4.1.5. Sistema

```
library ieee;
use ieee.std_logic_1164.all;
entity Lab_2 is
  port(cbcd1, cbcd2, cbcd3, cbcd4: in std_logic_vector(3 downto 0);
          --entHex: in std_logic_vector(3 downto 0);
          --ent8: in std_logic_vector(2 downto 0);
          selmux: in std_logic_vector(1 downto 0);
          --oe8: in std_logic;
          sal8: out std_logic_vector(7 downto 0);
          salr, salHex1, salHex2: out std_logic_vector(6 downto 0));
end Lab_2;
architecture behav of Lab_2 is
  signal salmux: std_logic_vector(3 downto 0);
begin
mux_gen: for i in 3 downto 0 generate
  U1: entity work.mux41 port map(I0 => cbcd1(i), I1 => cbcd2(i), I2 =>
cbcd3(i), I3 => cbcd4(i), sel => selmux, salm => salmux(i));
end generate;
U2: entity work.decoBCD7seg port map(ent => salmux, leds => salr);
U3: entity work.deco38 port map(oe => cbcd1(3), sel =>
std_logic_vector'(cbcd1(2),cbcd1(1),cbcd1(0)), sal => sal8);
U4: entity work.decoHEX7seg port map(enth => cbcd2, ledsh1 => salHex1,
ledsh2 => salHex2);
end behav;
```

4.2. Descripciones Test Bench

4.2.1. Decodificador 3 a 8

```
library ieee;
use ieee.std_logic_1164.all;
entity testbench38 is
end;
architecture tb_behave of testbench38 is
  component deco38
        port( oe: in std_logic;
              sel: in std_logic_vector(2 downto 0);
              sal: out std_logic_vector(7 downto 0));
  end component;
  signal test_oe: std_logic;
  signal test_sel: std_logic_vector(2 downto 0);
  signal test_sal: std_logic_vector(7 downto 0);
begin
UUT: deco38 port map(oe => test_oe, sel => test_sel, sal => test_sal);
 test_oe <= '0', '1' after 70ps;
 test_sel(0) <= '0', '1' after 20ps, '0' after 40ps, '1' after 60ps,'0'
 after 80ps, '0' after 100ps;
 test_sel(1) <= '0', '1' after 10ps, '0' after 20ps, '1' after 30ps, '0'
 after 40ps, '0' after 50ps, '1' after 60ps;
 test_sel(2) <= '0', '1' after 30ps, '0' after 60ps, '1' after 90ps, '0'
 after 120ps;
test_signals: process
  begin
```

```
        wait for 35ps;
              assert test_sal = "10000000"
                    Report "salida no es correcta"
                          severity ERROR;
        wait for 55ps;
              assert test_sal = "00000001"
                    Report "salida no es correcta"
                          severity ERROR;
        wait for 100ps;
              assert FALSE
                    Report "FIN"
                          severity FAILURE;
end process test_signals;
end tb_behave;
```

4.2.2. Decodificador BCD a 7segmento

```
library ieee;
use ieee.std_logic_1164.all;
entity testbenchBCD is
end;
architecture tb_behave of testbenchBCD is
  component decoBCD7seg
        port(ent: in std_logic_vector(3 downto 0);
                leds: out std_logic_vector (6 downto 0));
  end component;
  signal test_ent: std_logic_vector(3 downto 0);
  signal test_leds: std_logic_vector(6 downto 0);
begin
UUT: decoBCD7seg port map(ent => test_ent, leds => test_leds);
 test_ent <= "0000", "0101" after 20ps, "0001" after 40ps, "0010" after
 60ps, "1000" after 80ps;
test_signals: process
  begin
        wait for 5ps;
              assert test_leds = "1000000"
                    Report "salida no es correcta"
                          severity ERROR;
        wait for 65ps;
              assert test_leds = "0100100"
                    Report "salida no es correcta"
                          severity ERROR;
        wait for 100ps;
              assert FALSE
                    Report "FIN"
                          severity FAILURE;
end process test_signals;
end tb_behave;
```

4.2.3. Decodificador Hexadecimal a 7segmento

```
library ieee;
use ieee.std_logic_1164.all;
entity testbenchHEX is
end;
```

```
architecture tb_behave of testbenchHEX is
  component decoHEX7seg
        port(enth: in std_logic_vector(3 downto 0);
                ledsh1, ledsh2: out std_logic_vector (6 downto 0));
  end component;
  signal test_enth: std_logic_vector(3 downto 0);
  signal test_ledsh1: std_logic_vector(6 downto 0);
  signal test_ledsh2: std_logic_vector(6 downto 0);
begin
UUT: decoHEX7seg port map(enth => test_enth, ledsh1 => test_ledsh1,
ledsh2 => test_ledsh2);
 test_enth <= "0000", "0101" after 20ps, "1111" after 40ps, "0010" after
 60ps, "1100" after 80ps;
test_signals: process
  begin
        wait for 5ps;
              assert test_ledsh1 = "1000000"
                    Report "salida no es correcta"
                          severity ERROR;
              assert test_ledsh2 = "1000000"
                    Report "salida no es correcta"
                          severity ERROR;
        wait for 45ps;
              assert test_ledsh1 = "0010010"
                    Report "salida no es correcta"
                          severity ERROR;
              assert test_ledsh2 = "1111001"
                    Report "salida no es correcta"
                          severity ERROR;
        wait for 85ps;
              assert test_ledsh1 = "0100100"
                    Report "salida no es correcta"
                          severity ERROR;
              assert test_ledsh2 = "1111001"
                    Report "salida no es correcta"
                          severity ERROR;
        wait for 100ps;
              assert FALSE
                    Report "FIN"
                          severity FAILURE;
end process test_signals;
end tb_behave;
```

4.2.4. Multiplexer

```
--TestBench
library ieee;
use ieee.std_logic_1164.all;
entity testbenchMUX is
end;
architecture tb_behave of testbenchMUX is
  component mux41
        port(I0, I1, I2, I3: in std_logic;
                sel: in std_logic_vector(1 downto 0);
                salm: out std_logic);
```

```
  end component;
  signal test_sel: std_logic_vector(1 downto 0);
  signal test_I0, test_I1, test_I2, test_I3: std_logic;
  signal test_salm: std_logic;
begin
  UUT: mux41 port map(sel => test_sel, I0 => test_I0, I1 => test_I1, I2
  => test_I2, I3 => test_I3, salm => test_salm);
  test_sel <= "00", "11" after 20ps, "01" after 40ps, "00" after 60ps;
  test_I0 <= '0', '1' after 20ps, '0' after 40ps, '1' after 60ps, '0'
  after 80ps, '0' after 100ps;
  test_I1 <= '0', '1' after 10ps, '0' after 20ps, '1' after 30ps, '0'
 after 40ps, '0' after 50ps, '1' after 60ps;
  test_I2 <= '0', '1' after 30ps, '0' after 60ps, '1' after 90ps, '0'
 after 120ps;
  test_I3 <= '0', '1' after 40ps, '0' after 80ps;
test_signals: process
  begin
        wait for 25ps;
              assert test_salm = '0'
                    Report "salida no es correcta"
                          severity ERROR;
        wait for 55ps;
              assert test_salm = '0'
                    Report "salida no es correcta"
                          severity ERROR;
        wait for 100ps;
              assert FALSE
                    Report "FIN"
                          severity FAILURE;
end process test_signals;
end tb_behave;
```

4.2.5. Sistema

```
library ieee;
use ieee.std_logic_1164.all;
entity testbenchLAB2 is
end;
architecture tb_behave of testbenchLAB2 is
  component Lab_2
        port(cbcd1, cbcd2, cbcd3, cbcd4: in std_logic_vector(3 downto
0);
          --entHex: in std_logic_vector(3 downto 0);
          --ent8: in std_logic_vector(2 downto 0);
          selmux: in std_logic_vector(1 downto 0);
          --oe8: in std_logic;
          sal8: out std_logic_vector(7 downto 0);
          salr, salHex1, salHex2: out std_logic_vector(6 downto 0));
  end component;
  signal test_cbcd1, test_cbcd2, test_cbcd3, test_cbcd4:
std_logic_vector(3 downto 0);
  signal test_selmux: std_logic_vector(1 downto 0);
  signal test_salr: std_logic_vector(6 downto 0);
begin
```

```
  UUT: Lab_2 port map(selmux => test_selmux, cbcd1 => test_cbcd1, cbcd2
=> test_cbcd2, cbcd3 => test_cbcd3, cbcd4 => test_cbcd4, salr =>
test_salr, sal8 => open, salHex1 => open, salHex2 => open);
  test_selmux <= "00", "10" after 20ps, "01" after 40ps, "11" after
  60ps;
  test_cbcd1 <= "0000", "0011" after 20ps, "0110" after 40ps, "0010"
  after 60ps;
  test_cbcd2 <= "0000", "1001" after 20ps, "1100" after 40ps, "0010"
  after 60ps;
  test_cbcd3 <= "0000", "0001" after 20ps, "0101" after 40ps, "1010"
  after 60ps;
  test_cbcd4 <= "0000", "1101" after 20ps, "1100" after 40ps, "0010"
  after 60ps;
--      test_cbcd1 <= "0000", "0100" after 20ps, "1000" after 40ps,
"1110" after 60ps, "1100" after 80ps, "1110" after 100ps;
--      test_cbcd2 <= "0000", "1100" after 10ps, "0011" after 20ps,
"1000" after 30ps, "1111" after 40ps, "0001" after 50ps, "0110" after
60ps;
--      test_cbcd3 <= "0000", "0011" after 30ps, "0100" after 60ps,
"0010" after 90ps, "0100" after 120ps;
--      test_cbcd4 <= "0000", "1110" after 40ps, "1001" after 80ps;
test_signals: process
  begin
        wait for 25ps;
              assert test_salr = "1111001"
                    Report "salida no es correcta"
                          severity ERROR;
        wait for 55ps;
              assert test_salr = "0010000"
                    Report "salida no es correcta"
                          severity ERROR;
        wait for 100ps;
              assert FALSE
                    Report "FIN"
                          severity FAILURE;
end process test_signals;
end tb_behave;--
```