DEPARTAMENTO DE ELECTRÓNICA Y AUTOMÁTICA

FACULTAD DE INGENIERÍA – UNIVERSIDAD NACIONAL DE SAN JUAN

Informe de Laboratorio Nº 3

Descripción de Hardware con Instrucciones Secuenciales

[Fecha de Presentación: 4 de junio del 2021]

Asignatura: Temas Específicos de Electrónica Digital I

Ingeniería Electrónica

Autor:

Echenique, Leonardo – Registro 28351

1º Semestre

Año 2021

Índice de Contenido

Índice de Figuras

Índice de Tablas

1. Objetivos

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

- Uso de la guía de usuario del board DE2-115.
- Utilización de las instrucciones secuenciales, concurrentes y paquetes.
- Realizar estudio y determinación de camino crítico secuencial.
- Escritura de restricción de tiempo, frecuencia de trabajo del sistema, en el archivo de restricción.
- Configuración de los PLL disponibles.
- 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.

Pero antes del comienzo de las descripciones de cada componente, se analizaron las compilaciones correspondientes de cada uno, y se analizaron los siguientes mensajes de *warnings*. A pesar que estos no eran errores como tal, se buscó solucionarlos dentro de las posibilidades.

- [Warning (332148): Timing requirements not met]: este se debe a que, es necesario configurar un archivo de restricción de tiempo que permita informar al simulador a que frecuencia funciona el componente descripto y pueda optimizar su funcionamiento, teniendo en cuenta los retardos del sistema.
- [Warning (13046): Tri-state node(s) do not directly drive top-level pin(s)]: esta advertencia se debe a que los dispositivos FPGA no admiten buffers tri-state internos. Lo que realiza el sintetizador es convertir la función tri-state en compuertas OR o Multiplexers, sin embargo, el funcionamiento es el mismo.
- [Warning (332125): Found combinational loop of 2 nodes]: esta advertencia aparece cuando la salida de alguna lógica combinatoria se realimenta a si misma sin ningún registro intermedio. La solución fue crear una variable que funcione como registro, y realimentar la señal de este.
- [Warning (13024): Output pins are stuck at VCC or GND]: este se debe a que al configurar la salida de un display con un decodificador de BCD a 7segmentos para mostrar ciertos valores, coincide en que algunos segmentos siempre están en bajo o alto. La advertencia es válida, sin embargo, es intencional que esto suceda.

También, a modo de aclaración, el procedimiento llevado a cabo para la obtención de las gráficas de simulación de **todos los módulos** 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.

2.1. Circuito Sincronizador

Las señales asincrónicas aparecen de manera inevitable en la mayoría de los sistemas, es por ello que las mismas necesitan sincronizarse para disminuir la probabilidad de fallas, y asegurar un buen funcionamiento de los componentes que las utilicen. El módulo se diseñó utilizando dos flip-flop D en cascada, que usan la misma señal de reloj. El segundo flip-flop se agrega con el objetivo de otorgar más tiempo para que la señal de entrada se estabilice y no ingrese a un estado metaestable. Sin embargo, el diseño esta parametrizado, en caso de que sea necesario la utilización de más de dos registros.

Teniendo en cuenta lo anterior, la descripción del mismo en VHDL (Apéndice 4.1.1) presenta una *entidad* con tres entradas (1 bit para la señal asincrónica, 1 bit del reloj y 1 bit de reset) y una salida (1 bit para la señal sincrónica). Dentro de la *arquitectura* se declaró una señal interna con que permite implementar la cascada de registros. Finalmente, mediante un proceso, se describe el funcionamiento del sincronizador controlado por el flanco positivo del reloj y con un reset activo en bajo.
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 secuenciales, como Wait, y concurrentes. Este se describió en un nuevo archivo .vhd importado en el mismo proyecto. Utilizando la herramienta ModelSim, se visualizan las ondas de entrada y sus correspondientes salidas (Figura 2.1.2).

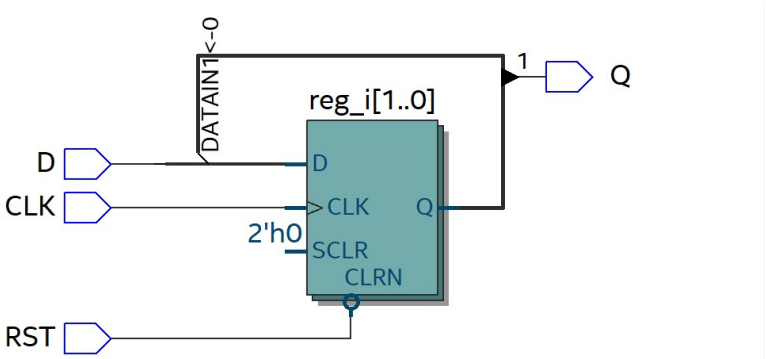

Figura 2. 1. 1 - Esquemático de Sincronizador

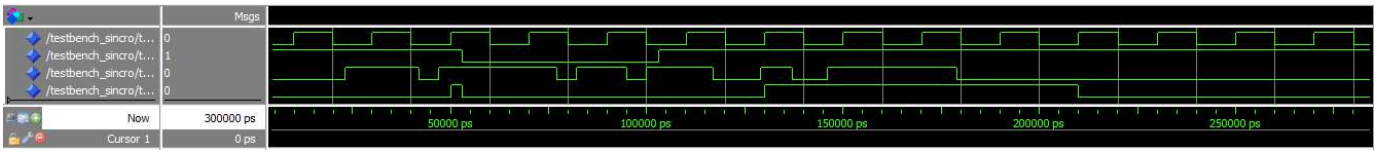

Figura 2. 1. 2 – Simulación de Sincronizador

Como se observa, la entrada asincrónica se sincroniza con el reloj del sistema, y además se puede apreciar como la salida es nula para una señal en bajo en la entrada del reset. Estas graficas denotan el correcto funcionamiento del diseño realizado.
Se asignaron los pines del FPGA (Tabla 2.1.1) con el objetivo de poder generar de forma correcta el bit stream y probar el funcionamiento del módulo en la placa. Se usaron como entradas los switches y los leds verdes como salidas.

Tabla 2. 1. 1 – Asignación de Pines del Sincronizador

SEÑAL	PIN
CLK	PIN_Y2
D	PIN_AB28
RST	PIN_AC28
Q	PIN_E21

Finalmente, 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 del Sincronizador

Recurso	Porcentaje
Total de elementos lógicos	2 / 114,480 (< 1 %)
Total registros	2
Total pins	4 / 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 %)

2.2. Circuito Antirebote

Las llaves mecánicas, como pueden ser interruptores o relés, presentan una serie de pulsos indeseados luego de ser accionadas, estos pulsos son conocidos como rebotes. Los rebotes, por lo general, producen fallas en los componentes donde se aplican debido a que estos los detectan como accionamientos múltiples de la llave. Un circuito antirebote permite eliminar estos pulsos y entregar una señal constante a su salida, brindando seguridad al sistema y evitando cualquier posible daño. La implementación del circuito se basó en un biestable simple, el cual permitió eliminar de manera eficiente los rebotes.

Con las características mencionadas, la descripción del hardware en VHDL (Apéndice 4.1.2) tiene una *entidad* con tres entradas (1 bit para la señal analógica, 1 bit de reloj y 1 bit de reset) y una salida (1 bit para señal estable). En la *arquitectura* se describieron las compuertas NAND interconectadas entre si con instrucciones concurrentes. Mientras que, por medio de un proceso, se estableció el sincronismo de la salida con el reloj del sistema.

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. En este TestBench (Apéndice 4.2.2) se utilizaron instrucciones concurrentes para la generación de las ondas de entrada. Se siguieron los mismos pasos que se explicaron en el anteriormente para poder visualizar las señales de entrada y salida con la herramienta ModelSim (Figura 2.2.2).

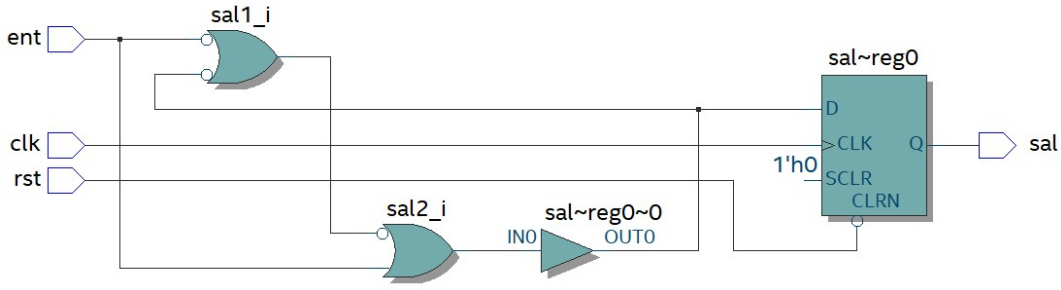

Figura 2. 2. 2 – Esquematico del Circuito Antirebote

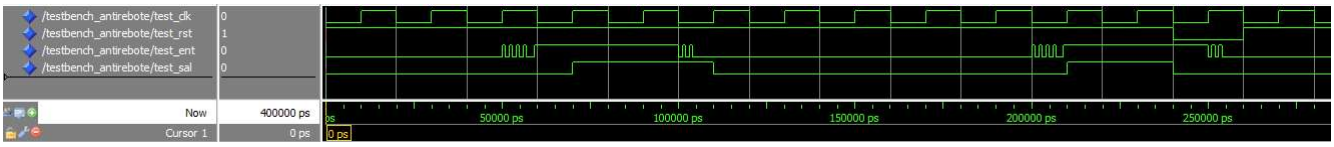

Figura 2. 2. 1 - Simulación del Circuito Antirebote

La simulación realizada muestra resultados acertados respecto a las entradas dadas. Se puede apreciar claramente como los rebotes son eliminados, obteniendo a la salida una señal constante, tanto en el flanco ascendente como en el descendente de la misma. Por lo tanto, es posible concluir que el hardware descripto es funcional.

Para este módulo, al igual que el anterior, se asignaron las E/S a pines específicos del FPGA (Tabla 2.2.1), se utilizaron los switches como entradas y un led verde como salida.

Tabla 2. 2. 1 – Asignación de Pines del Circuito Antirebote

SEÑAL	PIN
clk	PIN_Y2
ent	PIN_AB28
rst	PIN_AC28
sal	PIN_E21

Como información complementaria, se obtuvo también el Reporte de Área (Tabla 2.2.2) con el propósito de mostrar los recursos del FPGA que utiliza este hardware descripto.

Tabla 2. 2. 2 - Reporte de Área del Circuito Antirebote

Recurso	Porcentaje
Total de elementos lógicos	1 / 114,480 (< 1 %)
Total registros	1
Total pins	4 / 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 %)

2.3. Circuito de Reset

El reset de un sistema es muy importante y muchas veces es ignorado en el diseño de componentes. Un mal funcionamiento de este puede decantar en resultados erróneos del sistema. El método utilizado y más aconsejable de uso del reset es una activación asincrónica, para evitar problemas con componentes que funcionan con distintos relojes, y una desactivación sincrónica, con el fin de evitar la violación de los tiempos de recuperación y remoción. Entonces, el sistema descripto se basa en dos flip-flop D en cascada controlados por el mismo reloj, con entradas conectadas en alto constantemente y otra entrada de reset asincrónico. La salida de la cascada permite la desactivación sincrónica del reset, en este caso activo en bajo.

De igual manera que el sincronizador, las descripciones en VHDL (Apéndice 4.1.3) son bastantes similares. La *entidad* cuenta con dos entradas (1 bit de reloj y 1 bit del reset asincrónico) y una salida (1 bit del reset sincrónico). En la *arquitectura* se utilizó un proceso para describir el funcionamiento de la cascada de flip-flop como se explicó en el inciso anterior.

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, utilizando nuevamente las instrucciones concurrentes. La visualización de las señales (Figura 2.3.2) se realizó siguiendo los pasos descriptos al inicio de esta sección.

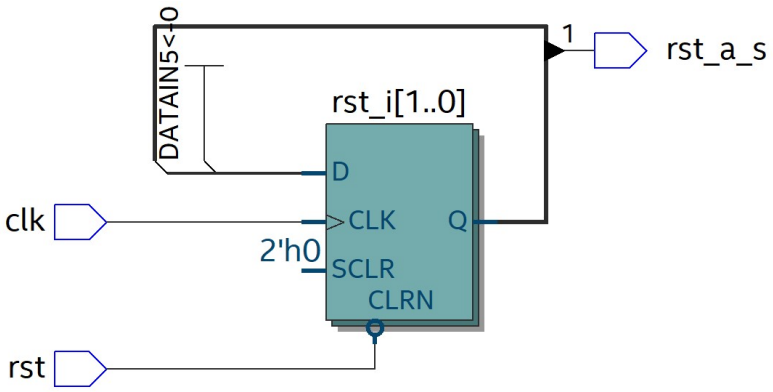

Figura 2. 3. 1 - Esquemático del Circuito de Reset

Figura 2. 3. 2 - Simulación del Circuito de Reset

La simulación demuestra el correcto funcionamiento, donde se puede observar que el reset se activa de manera asincrónica, y se desactiva sincrónicamente evitando siempre que se violen los tiempos antes mencionados.
Como los componentes anteriores, también se realizó la asignación de pines de E/S de la placa FPGA (Tabla 2.3.1). Se uso un switch como entrada y un led verde de salida.

Tabla 2. 3. 1 – Asignación de Pines del Circuito de Reset

SEÑAL	PIN
clk	PIN_Y2
rst	PIN_AC28
sal	PIN_E21

Una vez más, de manera informativa, se obtuvo el Reporte de Área (Tabla 2.3.2) buscando mostrar los recursos del FPGA utilizados por el hardware descripto.

Tabla 2. 3. 2 - Reporte de Área del Circuito de Reset

Recurso	Porcentaje
Total de elementos lógicos	2 / 114,480 (< 1 %)
Total registros	2
Total pins	3 / 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. Contadores LFSR

Los contadores binarios de secuencia pseudoaleatoria son muy útiles para la realización de test y desarrollos de diferentes componentes. Se denominan pseudoaleatorio debido a que a pesar de que no sigue un patrón determinado, este se repite cada $2^n - 1$ ciclos de reloj, donde n es la cantidad de registros del contador. A pesar de que es llamado contador, es un registro de desplazamiento con realimentación lineal con una entrada, por la cual ingresa un valor que resulta de realizar la operación lógica XOR entre ciertos bits del registro de desplazamiento. El registro de desplazamiento descripto cuenta con la facultad de otorgar diferentes largos de

secuencia ($n = 4, 8, 16, 32\ bits$), los cuales se eligen por medio de dos entradas de selección. Para el caso de $n = 4, 8, 16$, los registros no utilizados ponen sus salidas en alta impedancia para evitar cualquier tipo de error en la salida.

Teniendo como premisa el funcionamiento explicado, la descripción del hardware en VHDL (Apéndice 4.1.4) presenta una *entidad* con tres entradas (1 bit de reloj, 1 bit de reset y un vector de 2 bits de selección) y una salida (Un vector de 32 bits del registro de desplazamiento). La *arquitectura* describe el funcionamiento utilizando dos procesos principales, uno establece la ecuación de realimentación con la compuerta XOR y el otro el desplazamiento de los bits junto con el valor de reset de los registros. De igual manera que los componentes anteriores, se simulo el funcionamiento describiendo las señales con instrucciones concurrentes. Este TestBench (Apéndice 4.2.4) se describió al finalizar la descripción de la arquitectura principal.

El esquemático del hardware diseñado (Figura 2.4.1) se obtuvo mediante la opción de Quartus llamada RTL Viewer, pero al ser muy extenso (por la cantidad componentes usados) se amplió la imagen del sistema principal para una mejor visualización. 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). Como en el esquemático, también se debió hacer algunas modificaciones en las gráficas extraídas para poder visualizar el caso de cada contador.

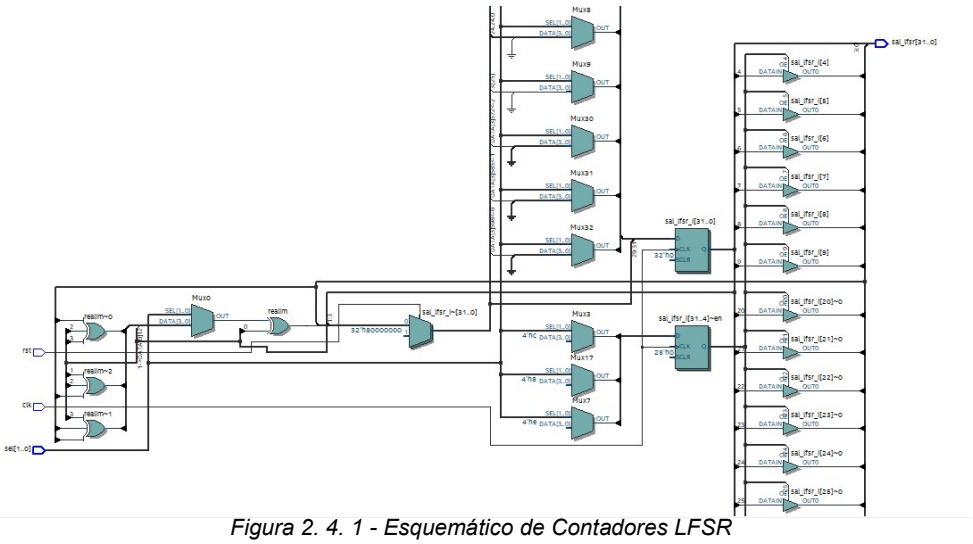

Figura 2. 4. 1 - Esquemático de Contadores LFSR

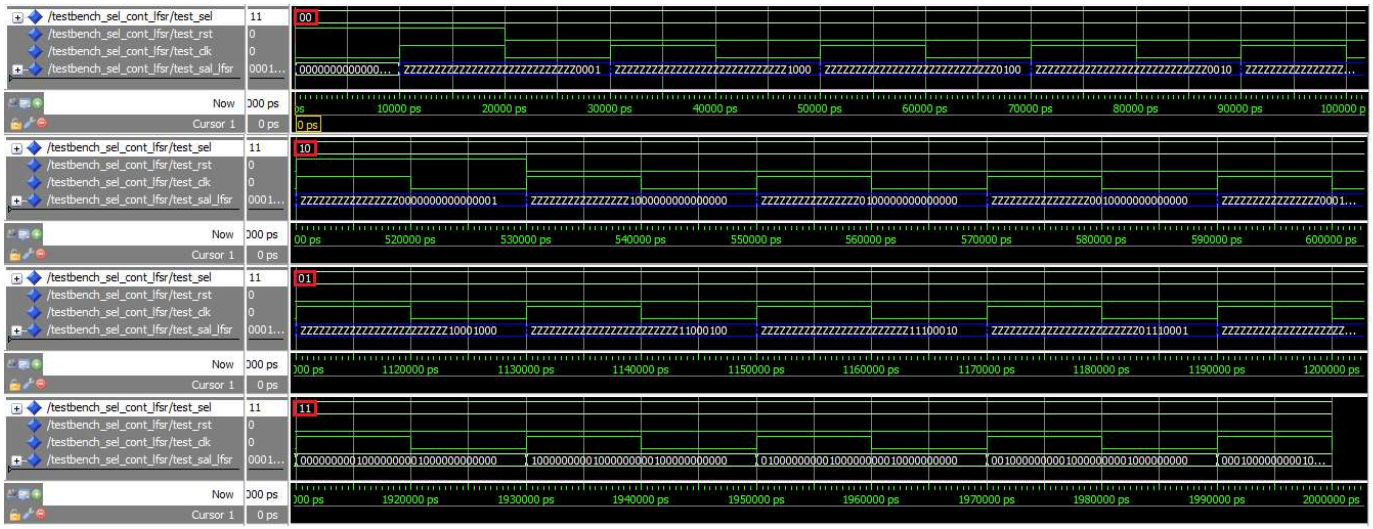

Figura 2. 4. 2 - Simulación de Contadores LFSR

En la grafica de simulacion se puede observar recuadrado en rojo las diferentes selecciones, indicando a que contador corresponde las señales de salidas($00 \rightarrow 4bits, 01 \rightarrow 8bits, 10 \rightarrow 16bits, 11 \rightarrow 32bits$). También, se puede apreciar como se colocan en alta impedancia las salidas de los registros no utilizados, el desplazamiento de los bits y como la realimentacion puede cambiar dependiendo el valor que entrega la operación logica XOR. Es por ello que se puede concluir que el sistema descripto es correcto.

Para este componente, al igual que los demas, se realizó asignación de pines E/S del FPGA (Tabla 2.4.1). Sin embargo, para su mejor visualizacion se modifico un poco el diseño original (Apéndice 4.1.5), al cual se le agrego un divisor de frecuencia y decodificadores a 7 segmentos, para mostrar los resultados a traves de los display que tiene la placa.

Tabla 2. 4. 1 - Tabla de Asignación de los Contadores LFSR

SEÑAL	PIN	SEÑAL	PIN
clk	PIN_Y2	bcd5[0]	PIN_AB19
rst	PIN_M23	bcd5[1]	PIN_AA19
bcd1[0]	PIN_G18	bcd5[2]	PIN_AG21
bcd1[1]	PIN_F22	bcd5[3]	PIN_AH21
bcd1[2]	PIN_E17	bcd5[4]	PIN_AE19
bcd1[3]	PIN_L26	bcd5[5]	PIN_AF19
bcd1[4]	PIN_L25	bcd5[6]	PIN_AE18
bcd1[5]	PIN_J22	bcd6[0]	PIN_AD18
bcd1[6]	PIN_H22	bcd6[1]	PIN_AC18
bcd2[0]	PIN_M24	bcd6[2]	PIN_AB18
bcd2[1]	PIN_Y22	bcd6[3]	PIN_AH19
bcd2[2]	PIN_W21	bcd6[4]	PIN_AG19
bcd2[3]	PIN_W22	bcd6[5]	PIN_AF18
bcd2[4]	PIN_W25	bcd6[6]	PIN_AH18
bcd2[5]	PIN_U23	bcd7[0]	PIN_AA17
bcd2[6]	PIN_U24	bcd7[1]	PIN_AB16
bcd3[0]	PIN_AA25	bcd7[2]	PIN_AA16
bcd3[1]	PIN_AA26	bcd7[3]	PIN_AB17
bcd3[2]	PIN_Y25	bcd7[4]	PIN_AB15
bcd3[3]	PIN_W26	bcd7[5]	PIN_AA15
bcd3[4]	PIN_Y26	bcd7[6]	PIN_AC17
bcd3[5]	PIN_W27	bcd8[0]	PIN_AD17
bcd3[6]	PIN_W28	bcd8[1]	PIN_AE17
bcd4[0]	PIN_V21	bcd8[2]	PIN_AG17
bcd4[1]	PIN_U21	bcd8[3]	PIN_AH17
bcd4[2]	PIN_AB20	bcd8[4]	PIN_AF17
bcd4[3]	PIN_AA21	bcd8[5]	PIN_AG18
bcd4[4]	PIN_AD24	bcd8[6]	PIN_AA14
bcd4[5]	PIN_AF23	sel[1]	PIN_AC28
bcd4[6]	PIN_Y19	sel[0]	PIN_AB28

Tambien, se obtuvo la tabla con el Reporte de Área (Tabla 2.4.2), en donde se puede apreciar un aumento en la cantidad de recursos utilizados, comparado con los componentes anteriores.

Tabla 2. 4. 2 - Reporte de Área de Contadores LFSR

Recurso	Porcentaje
Total de elementos lógicos	64 / 114,480 (< 1 %)
Total registros	60
Total pins	36 / 529 (7 %)
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 nuevo donde se describió en VHDL un contador LFSR de 12 bits (Apéndice 4.1.6), se realizó un análisis del Camino Crítico del Sistema Secuencial con la ayuda de la herramienta "Timing Analyzer" de Quartus. Se pudieron obtener los diez caminos críticos dentro del sistema (Figura 2.4.3), sin embargo, solo se consideró el primero, debido a que es el que presenta mayor retardo.

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	18.766	q_lfsr_12b_i[1]	q_lfsr_...b_i[11]	clk	clk	20.000	-0.079	1.173
2	18.774	q_lfsr_12b_i[6]	q_lfsr_...b_i[11]	clk	clk	20.000	-0.079	1.165
3	18.948	q_lfsr_12b_i[1]	q_lfsr_12b_i[0]	clk	clk	20.000	-0.079	0.991
4	18.960	q_lfsr_12b_i[0]	q_lfsr_...b_i[11]	clk	clk	20.000	-0.079	0.979
5	19.002	q_lfsr_...b_i[11]	q_lfsr_...b_i[10]	clk	clk	20.000	-0.079	0.937
6	19.103	q_lfsr_12b_i[4]	q_lfsr_...b_i[11]	clk	clk	20.000	-0.079	0.836
7	19.111	q_lfsr_12b_i[6]	q_lfsr_12b_i[5]	clk	clk	20.000	-0.079	0.828
8	19.114	q_lfsr_12b_i[4]	q_lfsr_12b_i[3]	clk	clk	20.000	-0.079	0.825
9	19.125	q_lfsr_12b_i[7]	q_lfsr_12b_i[6]	clk	clk	20.000	-0.079	0.814
10	19.125	q_lfsr_...b_i[10]	q_lfsr_12b_i[9]	clk	clk	20.000	-0.079	0.814

Figura 2. 4. 3 – Caminos Críticos del Contador LFSR 12 bits

Podemos ver en la columna Slack el tiempo que "sobra" en cada camino respecto al configurado ($20ns$). El primer camino es quien presenta menor cantidad de tiempo ($18.76ns$), por lo tanto, es el camino más crítico. De este último camino, 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 2.4.4). Tener en cuenta que los retardos se conforman también por retardos de setup y hold de cada registro.

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	0.000	0.000					launch edge time
2	⌄ 3.044	3.044					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_Y2	clk
3	0.000	0.000	RR	IC	1	IOIBUF_X0_Y36_N15	clk~input|i
4	0.720	0.720	RR	CELL	1	IOIBUF_X0_Y36_N15	clk~input|o
5	0.907	0.187	RR	IC	1	CLKCTRL_G4	clk~inputclkctrl|inclk[0]
6	0.907	0.000	RR	CELL	12	CLKCTRL_G4	clk~inputclkctrl|outclk
7	2.446	1.539	RR	IC	1	FF_X84_Y72_N27	q_lfsr_12b_i[1]|clk
8	3.044	0.598	RR	CELL	1	FF_X84_Y72_N27	q_lfsr_12b_i[1]
3	⌄ 4.217	1.173					data path
1	3.276	0.232		uTco	1	FF_X84_Y72_N27	q_lfsr_12b_i[1]
2	3.276	0.000	FF	CELL	3	FF_X84_Y72_N27	q_lfsr_12b_i[1]|q
3	3.689	0.413	FF	IC	1	LCCOMB_X84_Y72_N30	q_lfsr_12b_i~0|dataa
4	4.113	0.424	FF	CELL	1	LCCOMB_X84_Y72_N30	q_lfsr_12b_i~0|combout
5	4.113	0.000	FF	IC	1	FF_X84_Y72_N31	q_lfsr_12b_i[11]|d
6	4.217	0.104	FF	CELL	1	FF_X84_Y72_N31	q_lfsr_12b_i[11]

Figura 2. 4. 4 – Información del Camino Critico con mayor retardo

La herramienta proporciona también otra forma de visualizar la información mostrada en las figuras anteriores, esto es a través de formas de ondas (Figura 2.4.5), donde se ve claramente los retardos y sus valores.

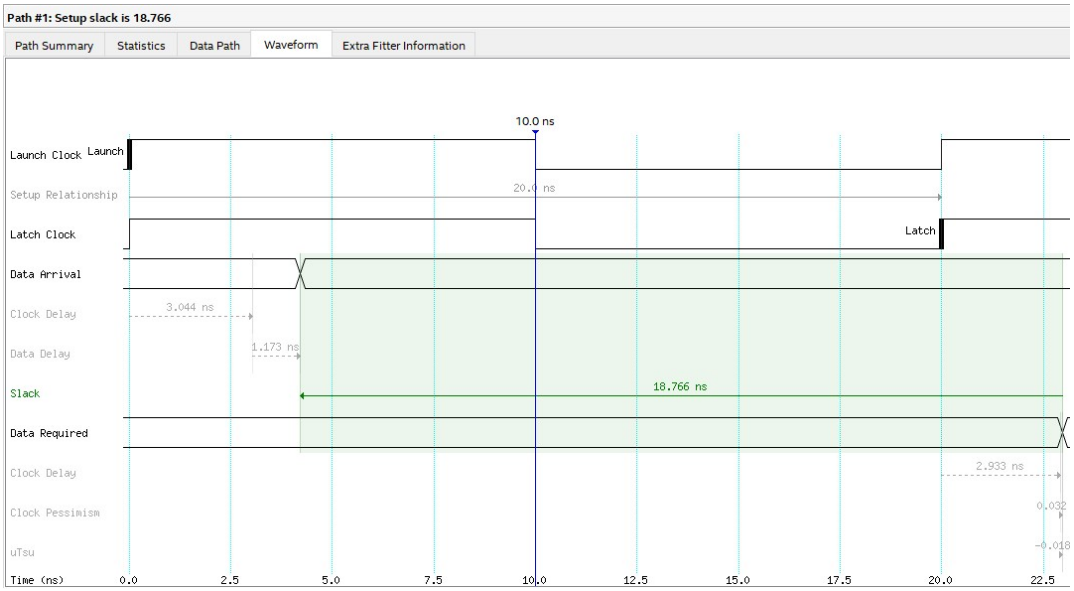

Figura 2. 4. 5 – Formas de Onda

La herramienta divide los retardos en retardos de reloj y de información, para un mejor entendimiento. Aunque los retardos estén presentes, estos no producen mayor problema a la frecuencia con la que se trabaja, denotando que se podría aumentar la misma. Sin embargo, esto es una limitación para la placa FPGA usada, ya que no puede proporcionar un reloj de mayor frecuencia, o por lo menos internamente.

Para finalizar, 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 2.4.6). En esta figura se observan las partes del sistema implementado que inciden en el retado máximo.

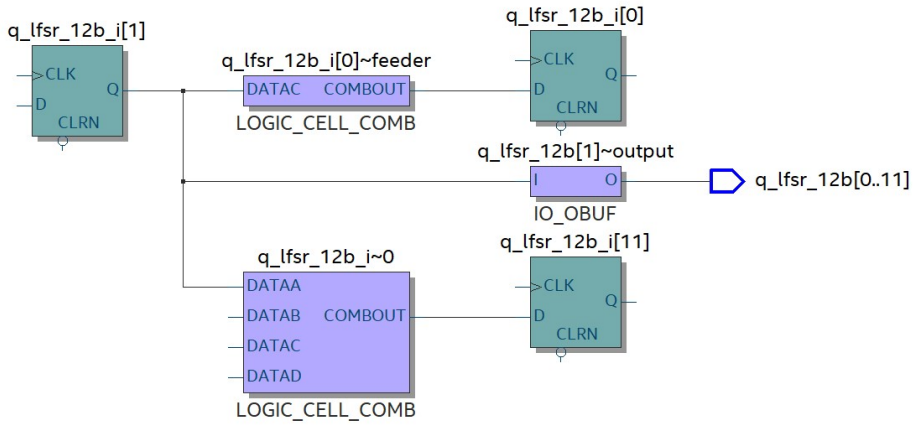

Figura 2. 4. 6 - Technology View of the Data Path del Contador LFSR

Como se puede apreciar, el camino critico está formado por la transmisión de información entre el registro interno del contador y la salida total del mismo. Tener en cuenta que las salidas fueron asignadas a los leds rojos de la placa FPGA utilizada. Como los retardos no son críticos para nuestra aplicación, no habría razón para reducirlos, sin embargo una de las formas seria modificar los retardos de ruteo, si esto fuera necesario.

2.5. Divisor de Frecuencia (Contador)

El cristal interno del FPGA otorga una frecuencia de aproximadamente $50MHz$, la que suele ser muy alta para el trabajo de algunos componentes, por ello es necesario disminuir de alguna manera este valor. Existen varias formas de realizar esto, pero se optó por la utilización de un contador simple, el cual tiene la función de contar los ciclos del reloj de alta frecuencia y entregar una señal de reloj de menor frecuencia. De esta manera, se obtuvieron varias señales con distintas frecuencias ($0.1Hz, 0.5Hz, 1Hz, 2Hz\ y\ 5Hz$) que pueden elegirse por medio de tres entradas de selección. Además, a través de dos display de 7 segmentos es posible mostrar la frecuencia seleccionada.

Una vez explicado el funcionamiento, la descripción del hardware en VHDL (Apéndice 4.1.7) presenta una *entidad* con tres entradas (1 bit de reloj, 1 bit de reset y un vector de 3 bits de selección) y dos salidas (dos vectores de 7 bits correspondientes a dos display). La *arquitectura* implemento el sistema utilizando un único proceso, en donde se describe el funcionamiento del contador simple junto con el procedimiento de selección de las frecuencias. Como los componentes anteriores, se simulo el funcionamiento describiendo las señales con instrucciones concurrentes, usando un TestBench (Apéndice 4.2.7) descripto al finalizar la arquitectura principal.

El esquemático del divisor obtenido (Figura 2.5.1) se visualizó mediante la herramienta de Quartus llamada RTL Viewer. Por otra parte, usando el software ModelSim, y siguiendo los pasos descriptos al inicio de esta sección, se observa la evolución de las entradas y salidas en el tiempo (Figura 2.5.2).

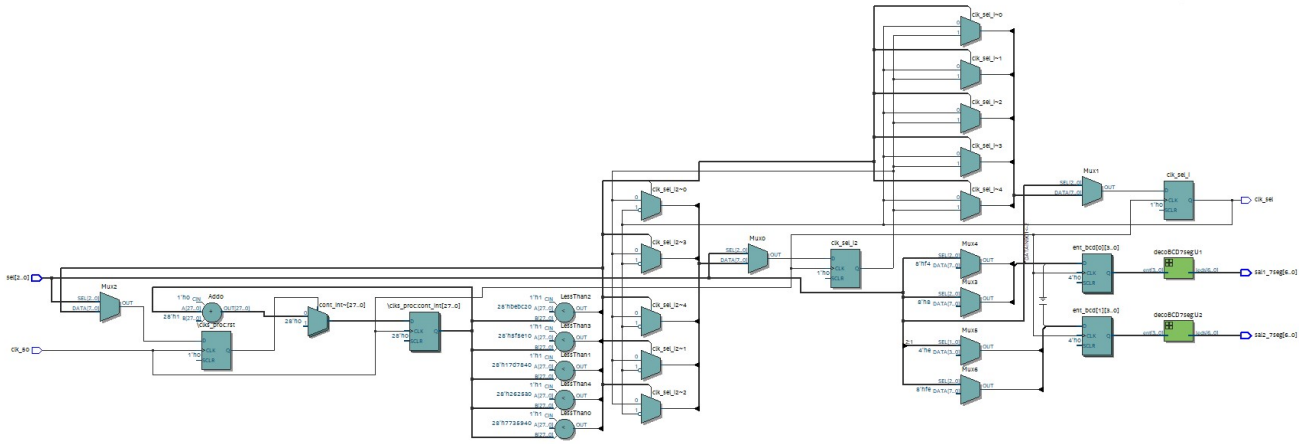

Figura 2. 5. 1 – Esquemático del Divisor de Frecuencia (Contadores)

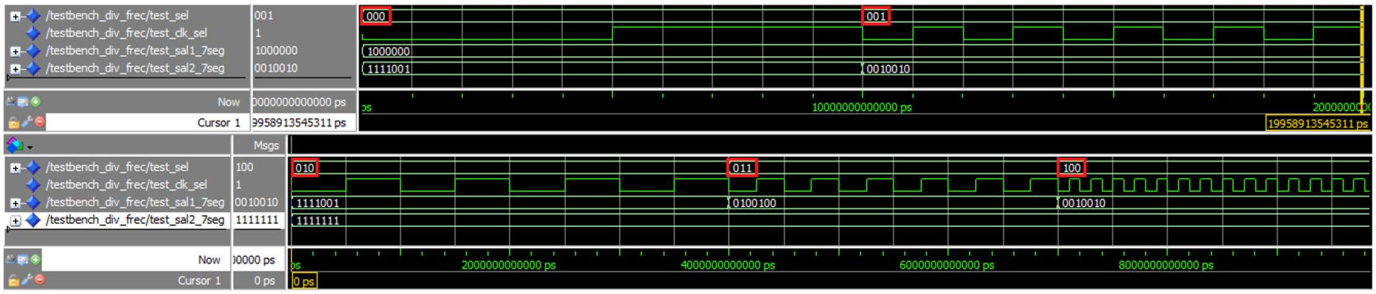

Figura 2. 5. 2 – Simulación del Divisor de Frecuencia (Contadores)

Se puede apreciar, recuadrados en color rojo, las diferentes selecciones de las frecuencias obtenidas con el divisor ($000 \rightarrow 0.1Hz, 001 \rightarrow 0.5Hz, 010 \rightarrow 1Hz, 011 \rightarrow 2Hz, 100 \rightarrow 5Hz$). A su vez, se puede observar los valores correspondientes a cada una de las frecuencias

codificadas para su visualización en los display 7 segmentos. Por estos resultados se puede concluir que el divisor descripto tiene un buen funcionamiento.

La asignación de pines para la placa FPGA utilizada, se puede ver en la siguiente tabla (Tabla 2.5.1), donde se usaron los switches como entradas de selección y el LED_0 rojo que es excitado por la señal de salida del divisor. De igual forma que con los componentes anteriores, se verifico el Reporte de Área (Tabla 2.5.2) para observar los recursos usados.

Tabla 2. 5. 1 - Asignación de Pines del Divisor de Frecuencia (Contador)

SEÑAL	PIN
clk_50	PIN_Y2
clk_sel	PIN_G19
sel[2]	PIN_AC27
sel[1]	PIN_AC28
sel[0]	PIN_AB28
sal1_7seg[6]	PIN_AH18
sal1_7seg[5]	PIN_AF18
sal1_7seg[4]	PIN_AG19
sal1_7seg[3]	PIN_AH19
sal1_7seg[2]	PIN_AB18
sal1_7seg[1]	PIN_AC18
sal1_7seg[0]	PIN_AD18
sal2_7seg[6]	PIN_AE18
sal2_7seg[5]	PIN_AF19
sal2_7seg[4]	PIN_AE19
sal2_7seg[3]	PIN_AH21
sal2_7seg[2]	PIN_AG21
sal2_7seg[1]	PIN_AA19
sal2_7seg[0]	PIN_AB19

Tabla 2. 5. 2 - Reporte de Área del Divisor de Frecuencia (Contador)

Recurso	Porcentaje
Total de elementos lógicos	84 / 114,480 (< 1 %)
Total registros	36
Total pins	19 / 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 %)

Se añadió, aunque antes no fue mencionado, el archivo de restricción correspondiente (*.sdc), con una restricción en el periodo del reloj de $20ns$. Esto se realizó mediante la herramienta “Timing Analyzer” que proporciona Quartus. Los pasos seguidos son los mismos que en el contador LFSR.

2.6. Divisor de Frecuencia (PLL)

Al igual que el módulo anterior, el objetivo es obtener diferentes frecuencias de menor valor a partir de la otorgada por el FPGA. Por lo tanto, el resultado obtenido por ambos módulos es el mismo, pero la diferencia en este caso es que la primera división se realizó usando uno de los PLL internos de la placa y además del contador simple, pero de menor longitud. La configuración del PLL permite la obtención de un reloj con una frecuencia mínima de $1.2KHz$. Para la configuración del mismo se seleccionó la opción *Tools>IP Catalog* en la interfaz de Quartus. Luego, en la parte derecha de la misma ventana se despliega una pequeña sección

donde se realizó la siguiente secuencia *Library>Basic Functions>Clocks; PLLs and Resets>PLL>ALTPLL.* En la ventana que aparece se tildo la opción *VHDL* y se seleccionó la ruta donde se deseaba guardar los archivos que se generen. Después, se abrió la ventana de configuración del PLL donde se configuro con una señal de entrada de $50MHz$ y de salida $1.2KHz$. Finalmente, se generaron los archivos necesarios para la instanciación del PLL dentro del proyecto principal.

Finalizada la configuración del PLL, y presentada una información breve del módulo que se implementó, se procedió a la descripción del mismo en VHDL (Apéndice 4.1.8), el cual contiene una *entidad* con tres entradas (1 bit de reloj, 1 bit de reset y un vector de 3 bits de selección) y cuatro salidas (dos vectores de 7 bits correspondientes a dos displays, 1 bit del reloj dividido y 1 bit de señal de lock del PLL). En la *arquitectura* inicialmente se instancio el PLL configurado, del cual se extrae la señal de reloj a dividir con el contador simple. Este último se implementó de igual manera que el divisor anterior, por medio de un proceso. Otra vez, se realizó la verificación funcional por medio del TestBench (Apéndice 4.2.8).

De igual manera que el divisor anterior, el esquemático (Figura 2.6.1) fue obtenido mediante la herramienta de Quartus llamada RTL Viewer. Por otra parte, debido a que el tiempo necesario para la simulación del módulo completo era muy largo, solo se realizó sobre el PLL configurado. Para ello, se usó el software ModelSim, donde se obtuvieron las evoluciones de sus salidas en el tiempo (Figura 2.6.2). El top level realizado (Apéndice 4.1.9) solo cuenta con la instanciación del PLL, mientras que el TestBench (Apéndice 4.2.9) correspondiente se realizó para verificar su funcionamiento.

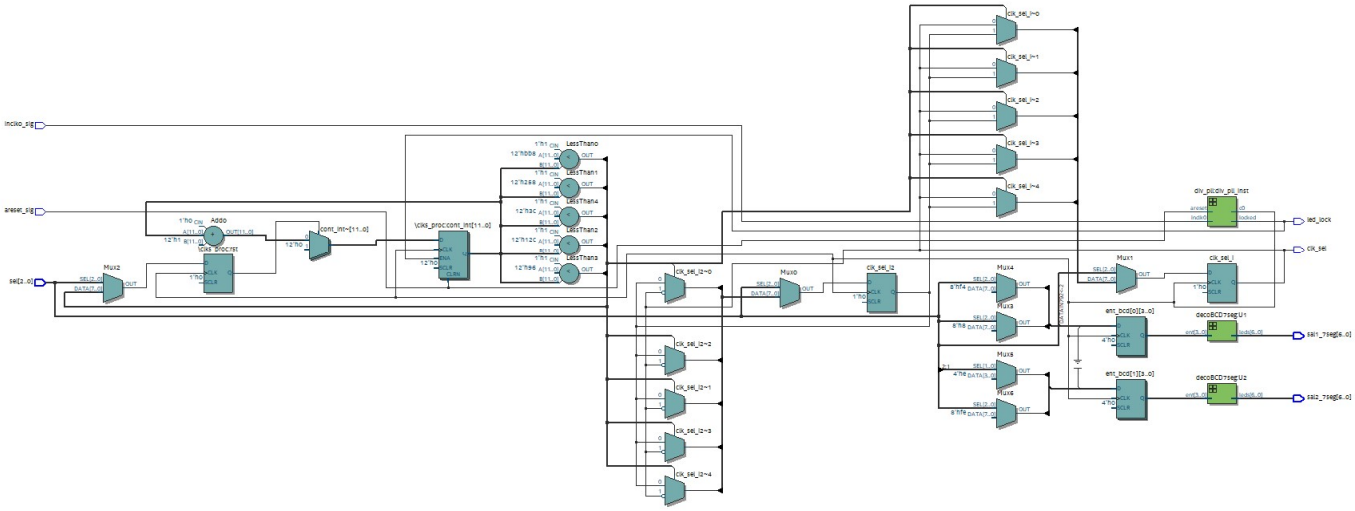

Figura 2. 6. 2 - Esquemático del Divisor de Frecuencia (PLL)

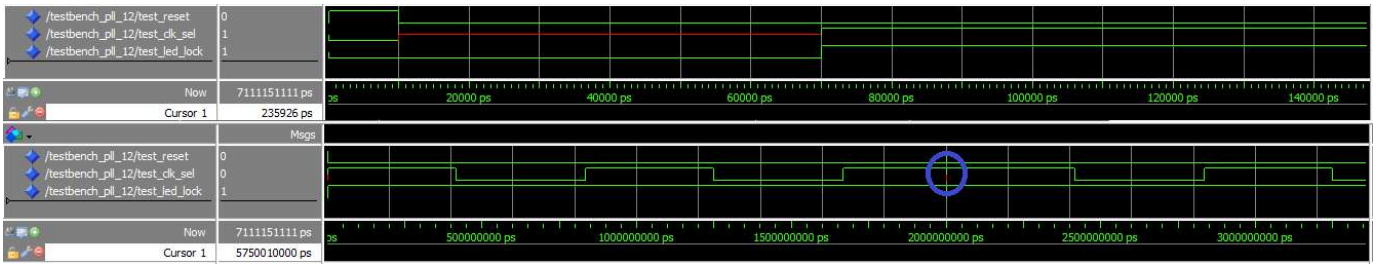

Figura 2. 6. 1 – Simulación del PLL

En la primera parte de la simulación se puede apreciar como a pesar de que se aplicó un reset al inicio de la misma, el valor de salida del PLL no tomo un valor hasta que se activó la señal de lock. A partir de la activación de esta señal, el PLL entrego un reloj de $1.2kHz$. Encerrado en un circulo azul se observa un reset, donde nuevamente la señal tomo un valor

indeterminado hasta la activación del lock y luego de esto comenzó a funcionar normalmente. Por lo anterior, se puede concluir que el PLL tiene un funcionamiento correcto, y combinado con el sistema de contadores del divisor anterior, se obtuvo un sistema funcional.

Como en todos los componentes hasta el momento, se asignaron los pines de E/S del FPGA (Tabla 2.6.1) usado para la prueba del sistema descripto. Para ello se utilizó como entradas de selección los switches, un pulsador como entrada de reset, un led verde para mostrar la señal de lock del PLL y dos de los display disponible para mostrar la frecuencia seleccionada.

Tabla 2. 6. 1 - Asignación de Pines del Divisor de Frecuencia (PLL)

SEÑAL	PIN
Inclk0_sig	PIN_Y2
clk_sel	PIN_G19
areset_sig	PIN_M23
sel[2]	PIN_AC27
sel[1]	PIN_AC28
sel[0]	PIN_AB28
sal1_7seg[6]	PIN_AH18
sal1_7seg[5]	PIN_AF18
sal1_7seg[4]	PIN_AG19
sal1_7seg[3]	PIN_AH19
sal1_7seg[2]	PIN_AB18
sal1_7seg[1]	PIN_AC18
sal1_7seg[0]	PIN_AD18
sal2_7seg[6]	PIN_AE18
sal2_7seg[5]	PIN_AF19
sal2_7seg[4]	PIN_AE19
sal2_7seg[3]	PIN_AH21
sal2_7seg[2]	PIN_AG21
sal2_7seg[1]	PIN_AA19
sal2_7seg[0]	PIN_AB19
led_lock	PIN_E21

De igual manera se obtuvo el Reporte de Área (Tabla 2.6.2), el cual muestra la utilización de uno de los cuatro PLLs disponibles en la placa, como así también los registros y otros recursos usados. La cantidad de registro usados se ve disminuida respecto al divisor de frecuencia implementado con únicamente el contador simple. Esto lógicamente se debe a que, al realizar una parte de la división de la frecuencia con el PLL, el contador necesito menos registros para llegar al valor final.

Tabla 2. 6. 2 - Reporte de Área del Divisor de Frecuencia (PLL)

Recurso	Porcentaje
Total de elementos lógicos	51 / 114,480 (< 1 %)
Total registros	21
Total pins	21 / 529 (4 %)
Total pins virtuales	0
Total bits de memoria	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	1 / 4 (25 %)

2.7. Sistema de Contadores

En continuación con el laboratorio anterior, donde se describió un sistema compuesto con multiplexers 4 a 1 y un decodificador BCD a 7segmentos, se describieron los contadores (Contador LFSR, Contador BCD, Contador Binario Down y Contador Binario Up) que fueron usados como entrada de los multiplexers. También, el sistema se complementó con un divisor de frecuencia, un módulo de reset con activación asincrónica y desactivación sincrónica y sincronizadores. Inicialmente se dibujó un diagrama de bloques (Figura 2.7.1) detallado con el fin de aclarar el diseño que se implementaría.

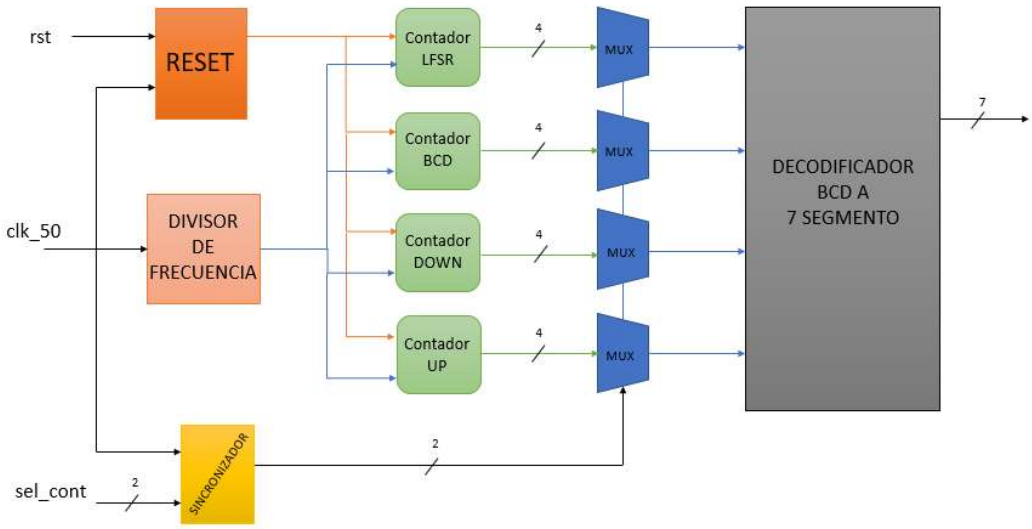

Figura 2. 7. 1 - Diagrama de Bloque Previo

Con el diagrama en bloque realizado en mente, se instanciaron todos los componentes (ya descriptos anteriormente) y se relacionaron sus entradas/salidas. En resumen, su funcionamiento se basa en cuatro contadores que funcionan a una frecuencia de $1Hz$ y cuentan con entrada de reset, mientras que por medio de una entrada de selección y cuatro multiplexers se elige la señal del contador que ingresa al decodificador BCD a 7segmento.

Teniendo en cuenta lo anterior, la descripción del hardware en VHDL (Apéndice 4.1.10) presenta una *entidad* con tres entradas (1 bit de reloj, 1 bit de reset y un vector de 2 bits de selección) y cuatro salidas (tres vectores de 7 bits correspondientes a tres display y un bit del reloj obtenido). La *arquitectura* se basó únicamente en instanciaciones, debido a que todos los componentes ya estaban descriptos. Únicamente se describieron los contadores faltantes, Contador BCD (Apéndice 4.1.11), Contador DOWN (Apéndice 4.1.12), Contador UP (Apéndice 4.1.13). Para verificar su correcto funcionamiento se realizó un TestBench (Apéndice 4.2.10) descripto al finalizar la arquitectura principal.

Como primer punto se obtuvo el esquemático del sistema (Figura 2.7.2), mediante la herramienta RTL Viewer de Quartus, para verificar si coincide con el diagrama en bloque antes realizado, y como se puede apreciar se sintetizo el mismo diagrama, a excepción de un detalle, la señal de reset asignada a los sincronizadores. También, se extrajeron las evoluciones de las salidas en el tiempo (Figura 2.7.3), con la ayuda del software ModelSim, frente a las entradas descriptas en el TestBench. Los pasos para esto último se describen al inicio de este documento.

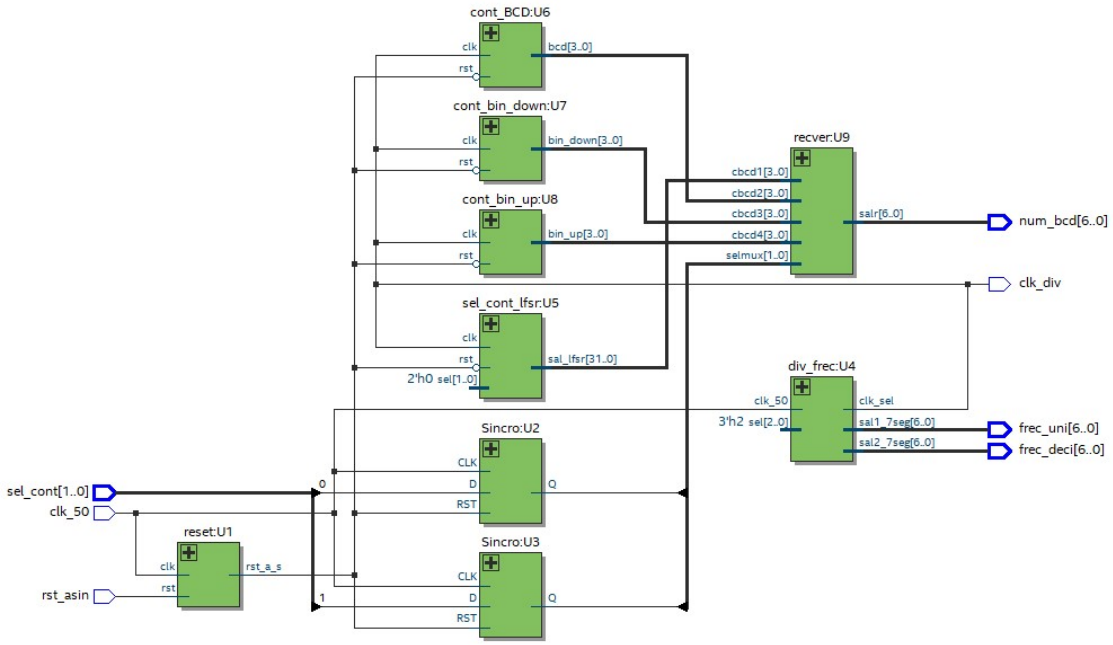

Figura 2. 7. 2 - Esquemático del Sistema de Contadores

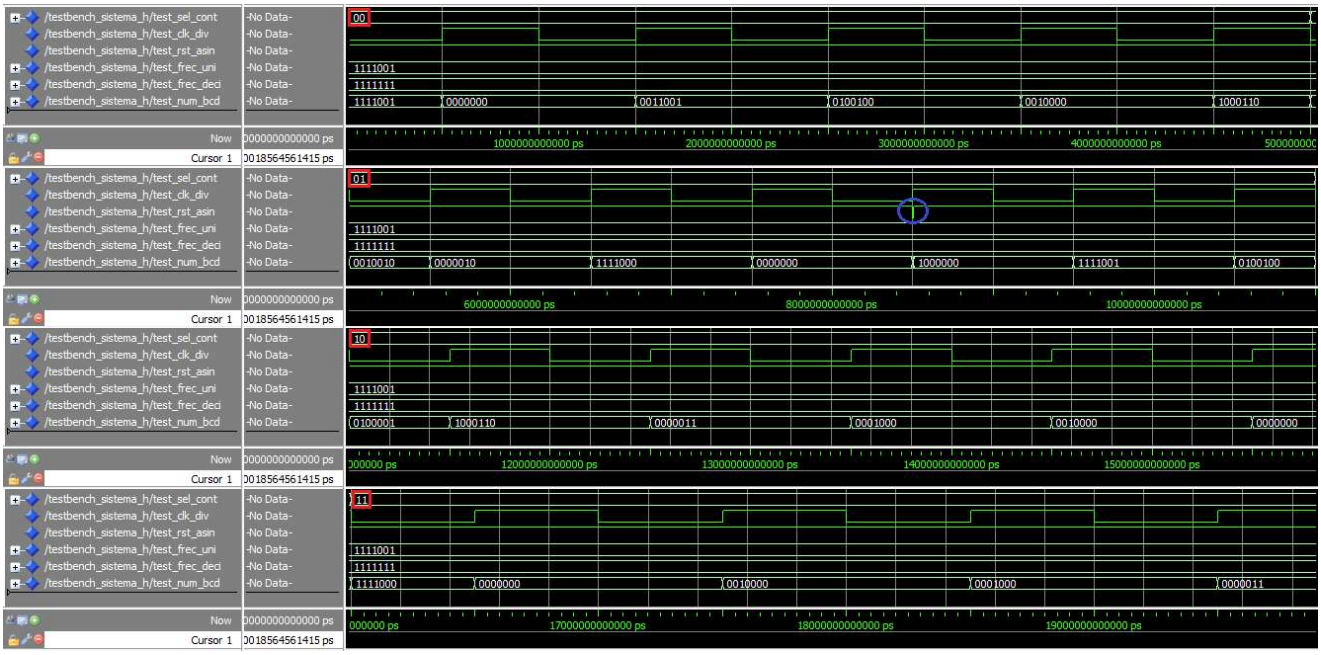

Figura 2. 7. 3 - Simulación del Sistema de Contadores

En esta última figura se pueden apreciar, recuadrado en rojo, las diferentes selecciones de los contadores ($00 \rightarrow Contador\ LFSR, 01 \rightarrow Contador\ BCD, 10 \rightarrow Contador\ DOWN, 11 \rightarrow Contador\ UP$), y se observa la evolución de la salida del decodificador BCD a 7segmento de cada contador. También se puede ver, encerrado en un círculo azul, la aplicación de un reset, el cual pone a todos los contadores en su valor inicial. Por lo obtenido en la simulación y el esquemático, se puede concluir que el sistema descripto tiene un buen funcionamiento.

La asignación de pines de E/S a la placa FPGA usada se muestra en la siguiente tabla (Tabla 2.7.1), donde se utilizaron los switches como entrada de selección y uno de los

pulsadores como entrada de reset. Respecto a la salida, se usaron tres de los displays disponibles en la placa y un led para visualizar la frecuencia usada.

Tabla 2. 7. 1 - Asignación de Pines del Sistema de Contadores

SEÑAL	PIN
clk_50	PIN_Y2
clk_sel	PIN_G19
rst_asin	PIN_M23
sel[1]	PIN_AC28
sel[0]	PIN_AB28
frec_deci[6]	PIN_AE18
frec_deci[5]	PIN_AF19
frec_deci[4]	PIN_AE19
frec_deci[3]	PIN_AH21
frec_deci[2]	PIN_AG21
frec_deci[1]	PIN_AA19
frec_deci[0]	PIN_AB19
frec_uni[6]	PIN_AH18
frec_uni[5]	PIN_AF18
frec_uni[4]	PIN_AG19
frec_uni[3]	PIN_AH19
frec_uni[2]	PIN_AB18
frec_uni[1]	PIN_AC18
frec_uni[0]	PIN_AD18
num_bcd[6]	PIN_AC17
num_bcd[5]	PIN_AA15
num_bcd[4]	PIN_AB15
num_bcd[3]	PIN_AB17
num_bcd[2]	PIN_AA16
num_bcd[1]	PIN_AB16
num_bcd[0]	PIN_AA17

El Reporte de Área (Tabla 2.7.2) muestra que los recursos usados son superiores comparados con los componentes anteriores, lo que es lógico debido a que dentro de este sistema se instanciaron todos estos compoenentes.

Tabla 2. 7. 2 - Reporte de Área del Sistema de Contadores

Recurso	Porcentaje
Total de elementos lógicos	72 / 114,480 (< 1 %)
Total registros	51
Total pins	26 / 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 %)

Antes de finalizar, se investigó el Camino Critico del Sistema Secuencial mediante la herramienta “Timing Analyzer” que proporciona Quartus. Se pudieron obtener los diez principales caminos críticos dentro del sistema (Figura 2.7.4), sin embargo, solo se consideró el primero, debido a que es el que presenta mayor retardo.

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	15.060	div_fre...k_sel_i	div_fre..._sel_i2	clk_50	clk_50	20.000	-0.080	4.878
2	16.406	div_fre...int[10]	div_fre..._sel_i2	clk_50	clk_50	20.000	-0.082	3.530
3	16.406	div_fre...int[10]	div_fre...k_sel_i	clk_50	clk_50	20.000	-0.082	3.530
4	16.408	div_fre...int[11]	div_fre..._sel_i2	clk_50	clk_50	20.000	-0.082	3.528
5	16.408	div_fre...int[11]	div_fre...k_sel_i	clk_50	clk_50	20.000	-0.082	3.528
6	16.567	div_fre...int[13]	div_fre..._sel_i2	clk_50	clk_50	20.000	-0.082	3.369
7	16.567	div_fre...int[13]	div_fre...k_sel_i	clk_50	clk_50	20.000	-0.082	3.369
8	16.609	div_fre..._int[0]	div_fre...int[27]	clk_50	clk_50	20.000	-0.083	3.326
9	16.610	div_fre...int[18]	div_fre..._sel_i2	clk_50	clk_50	20.000	-0.079	3.329
10	16.610	div_fre...int[18]	div_fre...k_sel_i	clk_50	clk_50	20.000	-0.079	3.329

Figura 2. 7. 4 - Caminos Críticos del Sistema de Contadores

Podemos observar en la columna Slack el tiempo que “sobra” en cada camino respecto al configurado como máximo ($20ns$). El primer camino es quien presenta menor cantidad de tiempo ($15.06ns$), por lo tanto, es el camino más crítico. De este último camino, 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 2.7.5).

Path #1: Setup slack is 15.060

Path Summary | Statistics | Data Path | Waveform | Extra Fitter Information

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	0.000	0.000					launch edge time
2	⌄ 3.068	3.068					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_Y2	clk_50
3	0.000	0.000	RR	IC	1	IOIBUF_X0_Y36_N15	clk_50~input|i
4	0.720	0.720	RR	CELL	1	IOIBUF_X0_Y36_N15	clk_50~input|o
5	0.907	0.187	RR	IC	1	CLKCTRL_G4	clk_50~inputclkctrl|inclk[0]
6	0.907	0.000	RR	CELL	37	CLKCTRL_G4	clk_50~inputclkctrl|outclk
7	2.470	1.563	RR	IC	1	FF_X56_Y66_N3	U4|clk_sel_i|clk
8	3.068	0.598	RR	CELL	1	FF_X56_Y66_N3	div_frec:U4|clk_sel_i
3	⌄ 7.946	4.878					data path
1	3.300	0.232		uTco	1	FF_X56_Y66_N3	div_frec:U4|clk_sel_i
2	3.300	0.000	FF	CELL	3	FF_X56_Y66_N3	U4|clk_sel_i|q
3	5.248	1.948	FF	IC	1	LCCOMB_X56_Y66_N14	U4|clk_sel_i2~0|dataa
4	5.677	0.429	FR	CELL	1	LCCOMB_X56_Y66_N14	U4|clk_sel_i2~0|combout
5	7.572	1.895	RR	IC	1	LCCOMB_X56_Y66_N18	U4|clk_sel_i2~feeder|datac
6	7.859	0.287	RR	CELL	1	LCCOMB_X56_Y66_N18	U4|clk_sel_i2~feeder|combout
7	7.859	0.000	RR	IC	1	FF_X56_Y66_N19	U4|clk_sel_i2|d
8	7.946	0.087	RR	CELL	1	FF_X56_Y66_N19	div_frec:U4|clk_sel_i2

Figura 2. 7. 5 - Información del Camino Critico con mayor retardo

Podemos ver que la herramienta divide los retardos producidos por el reloj y aparte los producidos por la información. A pesar de que están presentes estos delays, no son gran problema para la frecuencia a la que trabaja el componente. Esto denota que se podría trabajar a mayor frecuencia, pero esto no es posible debido a que la placa FPGA entrega internamente esa frecuencia como máximo.

La herramienta también proporciona la opción Locate Path, la cual permite obtener el Technology View of the Data Path (Figura 2.7.6). En esta figura se observan las partes del sistema implementado que inciden en el retado máximo.

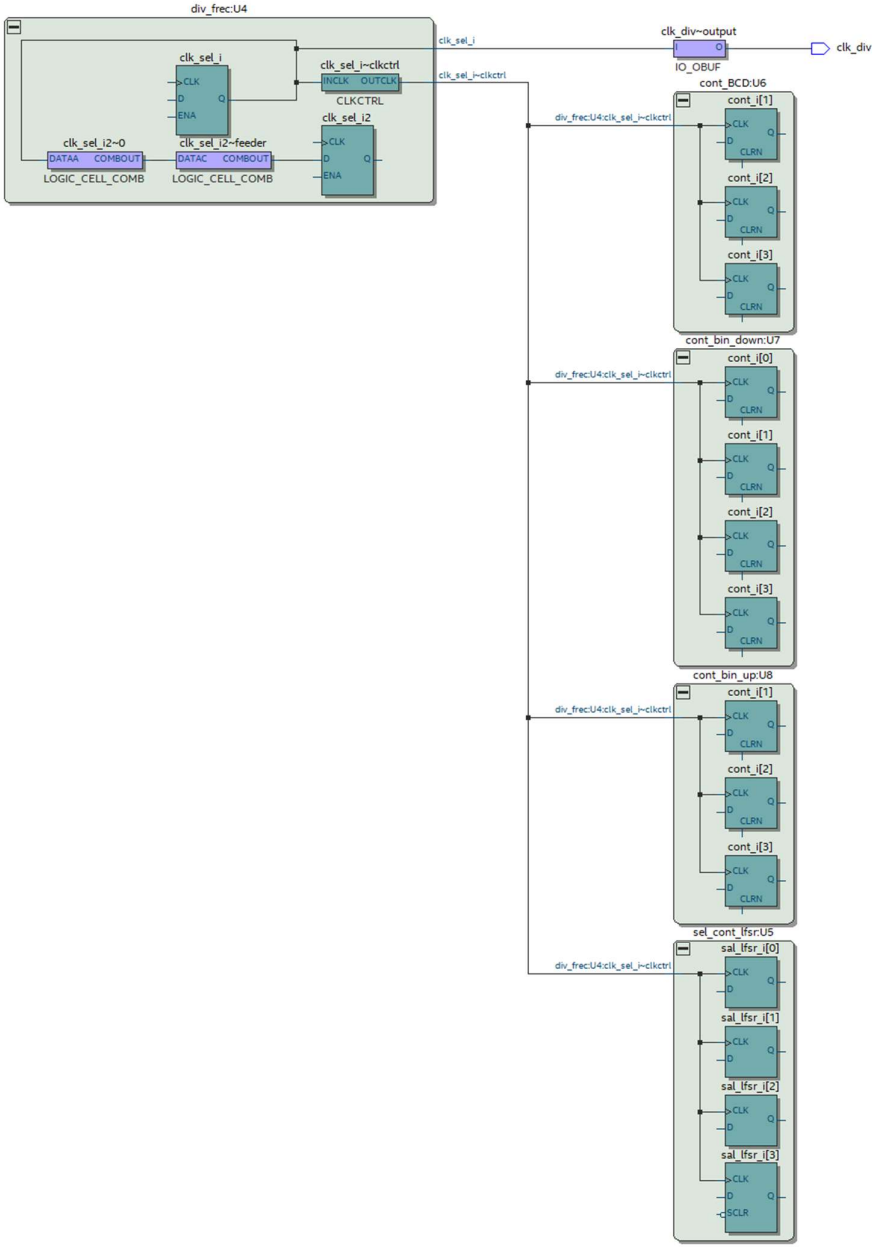

Figura 2. 7. 6 - Technology View of the Path del Sistema de Contadores

Se puede apreciar que el camino critico está compuesto por la circulación de información desde el divisor de frecuencia con cada uno de los cuatro contadores instanciados, junto con el led que muestra la frecuencia seleccionada. Esto solo se podría reducir modificando los retardos de ruteo, ya que los retardos lógicos son prácticamente fijos, pero para nuestra aplicación los retardos son más que aceptables.

Finalmente, para poder implementar el módulo en la placa y en relación al camino crítico, se realizó la comprobación correspondiente a nivel de compuertas usando el mismo TestBench que se utilizó para la simulación funcional. En este caso, se tiene en cuenta los retardos de

ruteo y retardos lógicos dentro de la simulación. Las formas de ondas de las entradas y salidas se obtuvieron de igual manera, por medio del software ModelSim.

Para esta simulación se siguió el siguiente camino de configuración: en primer lugar, selecciono la pestaña de *Assignaments>Settings* lo que desplego una nueva ventana, en la cual se eligió *Simulation>Test Benches*, donde nuevamente se abre una ventana y seleccionando el archivo correspondiente al TestBench del componente, aparece la opción de editar. Dentro de esta opción es posible permitir que el TestBench sea usado para la simulación a nivel compuertas y escribir el nombre del componente bajo prueba (DUT). Una vez que se realizo esta configuración, se dirigió a la pestaña de *Tools>Run Simulation Tool>Gate Level Simulation*, con lo cual se abrió la ventana de ModelSim y comenzó la simulación correspondiente.

Dicha simulación (Figura 2.7.7) entrego resultados similares a los obtenidos en la simulación funcional. Esto se debe principalmente a que el sistema bajo prueba trabaja a bajas frecuencias, por lo que es muy difícil que se produzcan retardos que se puedan considerar críticos para el desempeño final del componente.

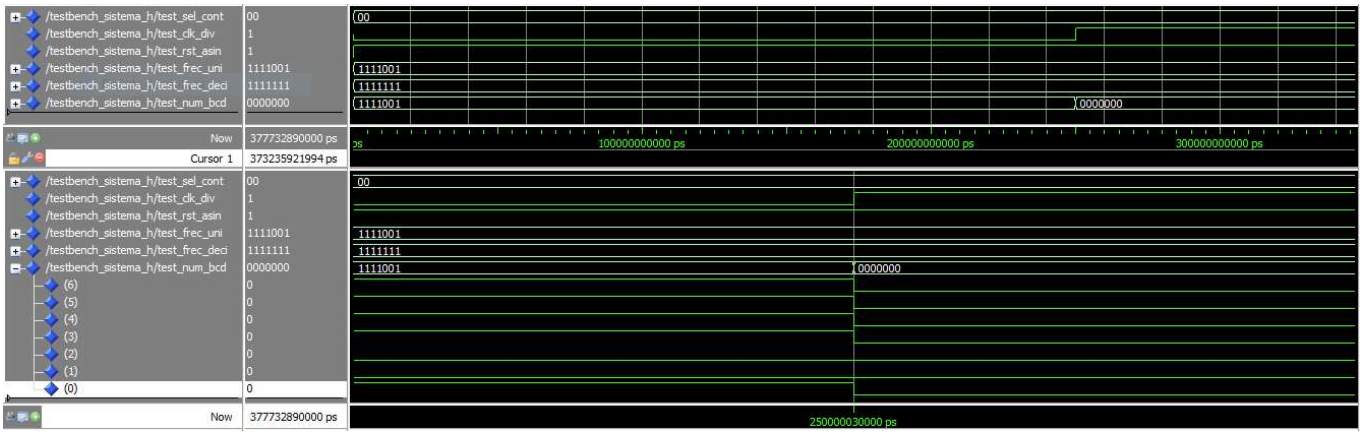

Figura 2. 7. 7 - Simulación a Nivel Compuertas

2.8. Generador de PWM

La modulación por ancho de pulsos consiste en generar una señal digital periódica con período (T) constante, pero con distintos tiempos para el estado alto (T_A) y estado en bajo (T_B). La relación entre el tiempo en alto y el periodo (T_A/T) se denomina ciclo de trabajo (duty cicle). Al variar el ciclo de trabajo de esta señal, lo que se hace es variar su tensión media, la cual es muy útil en algunas aplicaciones.

Con el funcionamiento descripto, la descripción del hardware en VHDL (Apéndice 4.1.14) cuenta con una *entidad* con cuatro entradas (1 bit de reloj, 1 bit de reset, 1 bit de carga del ciclo de trabajo y un vector de 8 bits de ciclo de trabajo) y dos salidas (1 bit del ciclo de trabajo y un vector de 4 bits que muestra los bits menos significativos del ciclo de trabajo ingresado). La *arquitectura* describe el funcionamiento del generador de PWM usando dos procesos principales, uno que genera la modulación por ancho de pulso dependiendo de la entrada del ciclo de trabajo y el otro que habilita el registro o no de la entrada del ciclo de trabajo recibida. A su vez, se encuentra la instanciación del decodificador a 7segmento para la visualización de los bits menos significativos en uno de los displays de la placa. También, se escribió el TestBench (Apéndice 4.2.11) correspondiente.

Como verificación del módulo descripto se obtuvieron tanto el esquemático (Figura 2.8.1), mediante la herramienta RTL Viewer de Quartus, y la evolución de las salidas respecto a entradas preestablecidas en el TestBench (Figura 2.8.2), esto a través del software ModelSim.

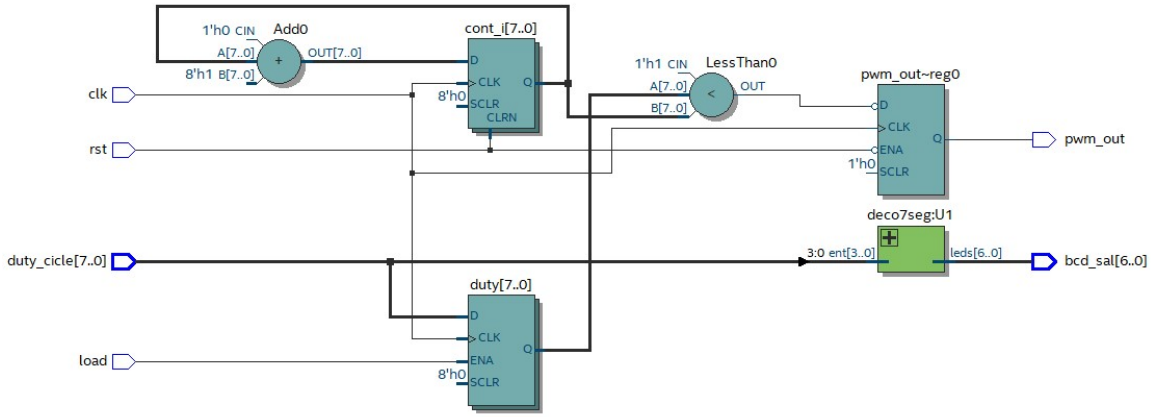

Figura 2. 8. 2 - Esquemático del Generador PWM

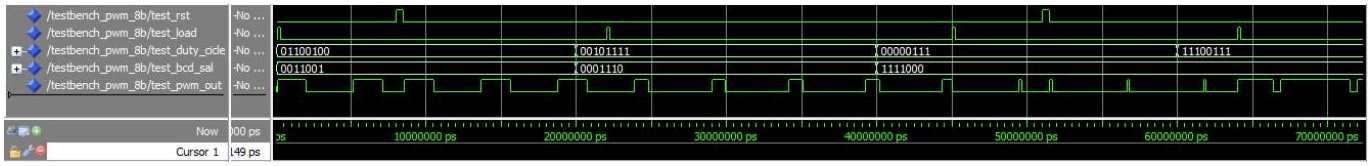

Figura 2. 8. 1 - Simulación del Generador PWM

Se aprecia que los ciclos de trabajaos no cambian hasta que se activa la señal de load, como así también que luego de activarse el reset, el ciclo de trabajo vuelve a comenzar. De igual manera, se ve el equivalente en 7segmento de los bits menos significativos de la entrada de ciclo de trabajo. Por las simulaciones y el esquemático, se concluye que tiene un buen funcionamiento el hardware descripto.

Se realizo la asignación de los pines de E/S en el FPGA (Tabla 2.8.1) como los módulos anteriores, donde se utilizaron los switches como entrada de ciclo de trabajo, y dos pulsadores para las entradas de reset y load. Para las salidas se usaron un display y un led que muestra el tiempo en alto y bajo de la señal PWM. De manera informativa, también se obtuvo el

Reporte de Área (Tabla 2.8.2), tratando de dar una idea de la cantidad de recursos usado por el componente.

Tabla 2. 8. 1 - Asignación de Pines del Generador PWM

SEÑAL	PIN
clk	PIN_Y2
rst	PIN_M23
load	PIN_Y23
duty_cicle[7]	PIN_AB26
duty_cicle[6]	PIN_AD26
duty_cicle[5]	PIN_AC26
duty_cicle[4]	PIN_AB27
duty_cicle[3]	PIN_AD27
duty_cicle[2]	PIN_AC27
duty_cicle[1]	PIN_AC28
duty_cicle[0]	PIN_AB28
bcd_sal[6]	PIN_AE18
bcd_sal[5]	PIN_AF19
bcd_sal[4]	PIN_AE19
bcd_sal[3]	PIN_AH21
bcd_sal[2]	PIN_AG21
bcd_sal[1]	PIN_AA19
bcd_sal[0]	PIN_AB19
pwm_out	PIN_G19

Tabla 2. 8. 2 - Reporte de Área del Generador PWM

Recurso	Porcentaje
Total de elementos lógicos	24 / 114,480 (< 1 %)
Total registros	17
Total pins	19 / 529 (4 %)
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 %)

2.9. Bloque de Generadores PWM

Utilizando el módulo descripto en el inciso anterior, se formó un bloque de ocho generadores de PWM. Para ello se describió el hardware en VHDL (Apéndice 4.1.15), el cual conto con una *entidad* de tres entradas (1 bit de reloj, 1 bit de reset y un bit de load) y una salida (Un vector de 8 bits), y también se especificó un *generic* con los valores fijos de ciclo de trabajo de cada generador PWM. En la *arquitectura* se realizaron únicamente instanciaciones por medio de un for-generate, el cual facilito esta descripción. Como lo hecho con todos los módulos, se realizó la verificación funcional por medio de un TestBench (Apéndice 4.2.12).

La verificación se complementó con la obtención del esquemático (Figura 2.9.1) que sintetiza el simulador, por medio de la herramienta RTL Viewer. Además, la evolución de las salidas en el tiempo (Figura 2.9.2) respecto a las entradas descriptas en el TestBench.

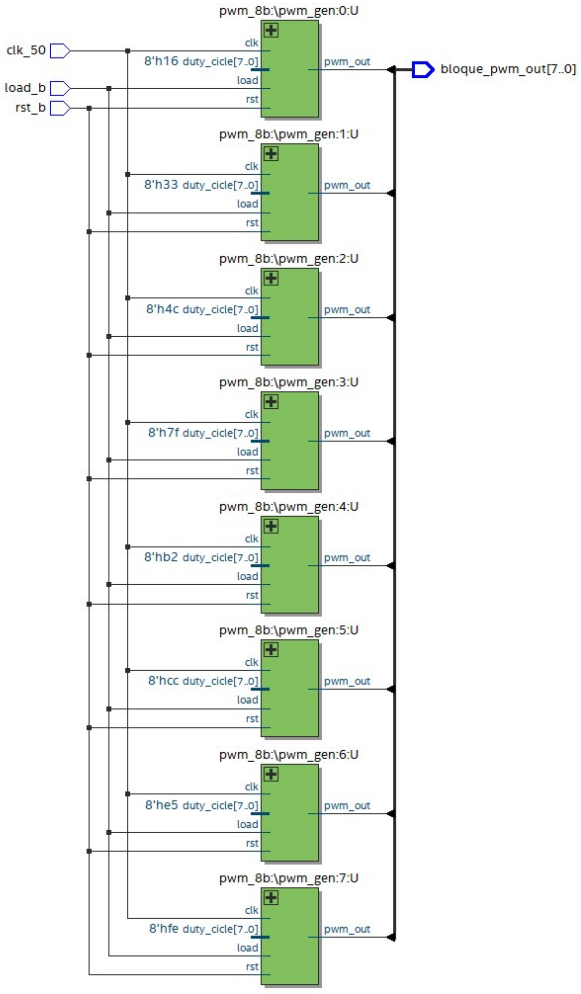

Figura 2. 9. 1 - Esquemático del Bloque Generador PWM

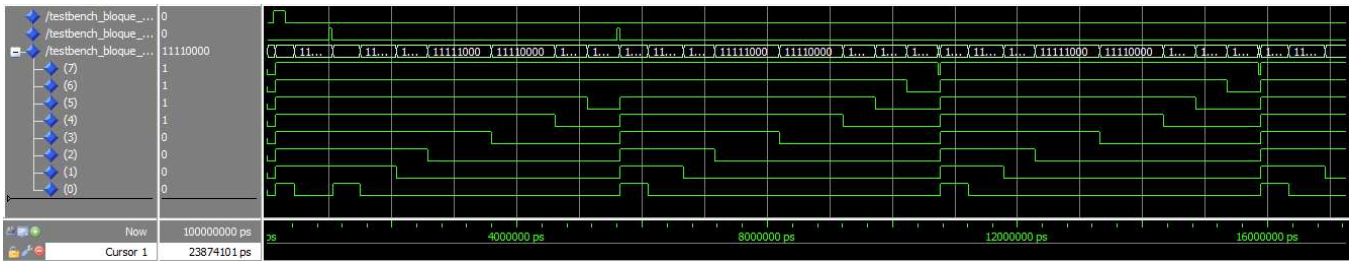

Figura 2. 9. 2 - Simulación del Bloque Generador PWM

Se puede observar cómo al inicio de la simulación se activa el load para poder cargar los ciclos de trabajo a todos los generadores PWM, donde cada salida cuenta con sus tiempos en alto y bajo. También, se aplicó en diferentes tiempos varios reset, donde se puede apreciar que todos los generadores comienzan nuevamente sus ciclos. Por la simulación y el esquemático obtenido, se determina el correcto funcionamiento del módulo.

Finalmente, se realizó la asignación de pines al FPGA (Tabla 2.9.1) para la verificación practica en la placa utilizada. Para las entradas se usaron dos pulsadores, mientras que para la salida se utilizaron los leds verdes. Por último, y no menos importante que lo anterior, también se obtuvo el Reporte de Área (Tabla 2.9.2) para ver la cantidad de recursos usados

por el módulo, el cual muestra que a pesar de que se usan ocho módulos unitarios del generador PWM, se sintetiza prácticamente con la misma cantidad recursos.

Tabla 2. 9. 1 - Asignación de Pines del Bloque Generador PWM

SEÑAL	PIN
clk_50	PIN_Y2
rst_b	PIN_M23
load_b	PIN_Y23
bloque_pwm_out[7]	PIN_G21
bloque_pwm_out[6]	PIN_G22
bloque_pwm_out[5]	PIN_G20
bloque_pwm_out[4]	PIN_H21
bloque_pwm_out[3]	PIN_E24
bloque_pwm_out[2]	PIN_E25
bloque_pwm_out[1]	PIN_E22
bloque_pwm_out[0]	PIN_E21

Tabla 2. 9. 2 - Reporte de Área del Bloque Generador PWM

Recurso	Porcentaje
Total de elementos lógicos	27 / 114,480 (< 1 %)
Total registros	17
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 %)

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 descriptos es posible concluir que los mismos, al ser circuitos lógicos secuenciales y a su vez combinacionales, debieron ser descriptos en VHDL usando instrucciones concurrentes y secuenciales. A pesar de que esto presento algunos inconvenientes, los mismos fueron resueltos de forma satisfactoria.

4. Apéndice

4.1. Descripciones VHDL

4.1.1. Sincronizador

```
-- Descripcion: Sincronizador controlado por flanco positivo del reloj y
--          reset activo en bajo. Se usan 2 ff-D en cascada para
--         evitar una señal metaestable a la salida.
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
-------------------------------------------------------------
-- Declaracion de Entidad
-------------------------------------------------------------
entity Sincro is
    generic( sincro_width : natural := 2);
    port(D, CLK, RST: in std_logic;
                Q: out std_logic);
end Sincro;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture beh of Sincro is
------- Declaracion de señales internas --------------------
    signal reg_i: std_logic_vector(sincro_width-1 downto 0) := (others =>
'0');
begin
-- Los registros cuentan con un reset sincronico y
-- se controlan con el flanco positivo del clock
------- Descripcion del sincronizador -----------------------
    reg_d:process (CLK, RST)
     begin
          if(RST = '0') then
                reg_i <= (others => '0');
          elsif(rising_edge (CLK)) then
                for i in 1 to reg_i'high loop
                      reg_i(i) <= reg_i(i-1);
                end loop;
                reg_i(0) <= D;
          end if;
    end process reg_d;
----- Asignacion de la salida del sincronizador-------------
    Q <= reg_i(reg_i'high);
end beh;
```

4.1.2. Circuito Antirebote

```
-- Descripcion: Circuito antirebote conformado por compuertas NAND,
--               donde sus salidas se encuentran interconectadas entre si
--               permitiendo obtener una señal estable para llaves
--               mecanicas. El circuito es controlado por flanco positivo
--               del reloj y reset activo en bajo.
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
------------------------------------------------------------
-- Declaracion de Entidad
------------------------------------------------------------
entity Antirebote is
    port(ent, clk, rst: in std_logic;
                sal: out std_logic);
end entity;
------------------------------------------------------------
-- Arquitectura
------------------------------------------------------------
architecture beh of Antirebote is
------- Declaracion de señales internas --------------------
    signal sal1_i, sal2_i : std_logic;
    signal vcc1 : std_logic := '1';
begin
------- Compuertas OR con entradas negadas (NAND) ----------
    sal1_i <= not(ent) or not(sal2_i);
    sal2_i <= ent or not(sal1_i);
------- Proceso sincronizador ---------------------
    anti_r: process(clk, rst)
    begin
          if(rst = '0') then
                sal <= '0';
          elsif(rising_edge(clk)) then
                sal <= sal2_i;
          end if;
    end process anti_r;
end beh;
```

4.1.3. Circuito Reset

```
--Descripcion: circuito de reset activo asincronicamente y desactivado
--              sincronicamente. El circuito es controlado por flanco
--              positivo del clock.
--              Se usa un reset activado asincronicamente para evitar
--              problemas con registros que funcionan con distintos clocks.
--              Mientras que la desactivacion sincronica se utiliza para
--              evitar la violacion de los tiempos de recuperacion y
--              remocion.
------------------------------------------------------------
--Declaracion de Librerias
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
------------------------------------------------------------
-- Declaracion de Entidad
------------------------------------------------------------
entity reset is
------ Generic que indica la cantidad de flip flop ---------
------ internos y su valor de activacion -------------------
    generic(rst_width:integer := 2;
                  rst_active_value:std_logic := '0');
    port( clk : in std_logic;
                rst : in std_logic;
                rst_a_s: out std_logic);
end entity reset;
```

```
------------------------------------------------------------
-- Arquitectura
------------------------------------------------------------
architecture rst_beh of reset is
------- Declaracion de señales internas --------------------
    signal rst_i: std_logic_vector(rst_width downto 0);
begin
------- Proceso que describe el reset activado de ----------
------- de forma asincronica y desactivado de forma --------
------- sincronica -----------------------------------------
    process(rst, clk)
    begin
    ---- Recepcion del reset de forma asincronica -----------
    if (rst = rst_active_value) then
          rst_i <= (others => rst_active_value);
          rst_a_s <= '0';
    ---- Al recibir un flanco positivo del reloj se ---------
    ---- desactiva el reset de forma sincronica, ------------
    ---- manteniendo el valor de salida de los f-f ----------
    ---- en el valor del rst desactivado --------------------
    elsif (rising_edge(clk)) then
          rst_i(0) <= not rst_active_value;
          for i in 0 to rst_width-1 loop
                rst_i(i+1) <= rst_i(i);
          end loop;
    end if;
    ---- Asignacion del valor de salida ---------------------
          rst_a_s <= rst_i(rst_width-1);
    end process;
end rst_beh;
```

4.1.4. Contadores LFSR

```
-- Descripcion: Contador tipo LFSR con entrada de seleccion para elegir
--        entre 4, 8, 16, 32 bits. Este tipo de contador genera secuencias
--        pseudoaleatorias.
--        La implentacion se realiza a traves de un registro de
--        desplazamiento realimentado a la entrada con una compuerta XOR.
--        Las salidas no usadas (caso 4, 8 y 16 bits) se colocan en alta -
--         impedancia.
------------------------------------------------------------
--Declaracion de Librerias
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
------------------------------------------------------------
-- Declaracion de Entidad
------------------------------------------------------------
entity sel_cont_lfsr is
------ Generic que parametriza el largo de los -------------
------ contadores LFSR -------------------------------------
    generic(     ancho_lfsr_a : integer := 4;
                       ancho_lfsr_b : integer := 8;
                       ancho_lfsr_c : integer := 16;
                       ancho_lfsr_d : integer := 32);
    port(  sel: in std_logic_vector(1 downto 0);
                rst: in std_logic;
                clk: in std_logic;
                sal_lfsr: out std_logic_vector(ancho_lfsr_d-1 downto 0));
end sel_cont_lfsr;
```

```
------------------------------------------------------------
-- Arquitectura
------------------------------------------------------------
architecture rtl of sel_cont_lfsr is
------- Declaracion de señales internas --------------------
    --signal initial_value: std_logic_vector(31 downto 0):= (0 => '1',
others => '0');
    signal sal_lfsr_i : std_logic_vector(31 downto 0) := (0 => '1', others
=> '0');
    signal serial_in : std_logic;
    signal realim : std_logic;
begin
------ Entrada Serie al Contador LFSR ----------------------
    serial_in <= realim;
------ Proceso de Realimentacion XOR de la -----------------
------ ecuacion de realimentacion --------------------------
    realim_proc: process (sal_lfsr_i, sel)
          begin
          case sel is
                when "00" =>
                      realim <= sal_lfsr_i(1) xor sal_lfsr_i(0);
                when "01" =>
                      realim <= sal_lfsr_i(4) xor sal_lfsr_i(3) xor
sal_lfsr_i(2) xor sal_lfsr_i(0);
                when "10" =>
                      realim <= sal_lfsr_i(5) xor sal_lfsr_i(4) xor
sal_lfsr_i(3) xor sal_lfsr_i(0);
                when others =>
                      realim <= sal_lfsr_i(22) xor sal_lfsr_i(2) xor
sal_lfsr_i(1) xor sal_lfsr_i(0);
          end case;
    end process realim_proc;
------ Proceso de seleccion de largo del ------------------
------ contador LFSR --------------------------------------
    sel_proc: Process (rst, clk, sel)
          variable ancho_lfsr: integer;
    begin
    --- El cambio entre un contador y otro se debe ---------
    --- realizar un reset para inicializar todas las -------
    --- salidas (Alta impedancia) --------------------------
          if(rst= '0') then
                sal_lfsr_i <= (0 => '1', others => '0');
          elsif(rising_edge(clk)) then
          case sel is
                when "00" =>
                      sal_lfsr_i(31 downto ancho_lfsr_a) <= (others =>
'Z');
                      sal_lfsr_i(ancho_lfsr_a-1 downto 0) <= serial_in &
sal_lfsr_i(ancho_lfsr_a-1 downto 1);
                when "01" =>
                      sal_lfsr_i(31 downto ancho_lfsr_b) <= (others =>
'Z');
                      sal_lfsr_i(ancho_lfsr_b-1 downto 0) <= serial_in &
sal_lfsr_i(ancho_lfsr_b-1 downto 1);
                when "10" =>
                      sal_lfsr_i(31 downto ancho_lfsr_c) <= (others =>
'Z');
                      sal_lfsr_i(ancho_lfsr_c-1 downto 0) <= serial_in &
sal_lfsr_i(ancho_lfsr_c-1 downto 1);
```

```
                when others =>
                      sal_lfsr_i(ancho_lfsr_d-1 downto 0) <= serial_in &
sal_lfsr_i(ancho_lfsr_d-1 downto 1);
          end case;
          end if;
    end process sel_proc;
---- Asignacion del valor de salida --------------------
    sal_lfsr <= sal_lfsr_i;
end rtl;
```

4.1.5. Contadores LFSR [Modificado]

```
-- Descripcion: Contador tipo LFSR con entrada de seleccion para elegir
--        entre 4, 8, 16, 32 bits. Este tipo de contador genera secuencias
--        pseudoaleatorias.
--        La implentacion se realiza a traves de un registro de
--        desplazamiento realimentado a la entrada con una compuerta XOR.
--        Las salidas no usadas (caso 4, 8 y 16 bits) se colocan en alta -
--        impedancia.
----------------------------------------------------------
--Declaracion de Librerias
----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
----------------------------------------------------------
-- Declaracion de Entidad
----------------------------------------------------------
entity sel_cont_lfsr is
------ Generic que parametriza el largo de los -------------
------ contadores LFSR -------------------------------------
    generic(     ancho_lfsr_a : integer := 4;
                       ancho_lfsr_b : integer := 8;
                       ancho_lfsr_c : integer := 16;
                       ancho_lfsr_d : integer := 32);
    port(  sel: in std_logic_vector(1 downto 0);
                rst: in std_logic;
                clk: in std_logic;
                bcd1, bcd2, bcd3, bcd4,bcd5, bcd6, bcd7, bcd8 : out
std_logic_vector(6 downto 0));
end sel_cont_lfsr;
----------------------------------------------------------
-- Arquitectura
----------------------------------------------------------
architecture rtl of sel_cont_lfsr is
------- Declaracion de señales internas --------------------
    signal sal_lfsr_i : std_logic_vector(31 downto 0) := (0 => '1', others
=> '0');
    signal serial_in : std_logic;
    signal realim : std_logic;
begin
------ Instanciacion de los decodificadores ----------------
          U1: entity work.deco7seg port map(ent => sal_lfsr_i(3 downto 0),
                                            leds => bcd1);
          U2: entity work.deco7seg port map(ent => sal_lfsr_i(7 downto 4),
                                            leds => bcd2);
          U3: entity work.deco7seg port map(ent => sal_lfsr_i(11 downto
                                            8), leds => bcd3);
          U4: entity work.deco7seg port map(ent => sal_lfsr_i(15 downto
                                            12), leds => bcd4);
```

```
          U5: entity work.deco7seg port map(ent => sal_lfsr_i(19 downto
                                                16), leds => bcd5);
          U6: entity work.deco7seg port map(ent => sal_lfsr_i(23 downto
                                                20), leds => bcd6);
          U7: entity work.deco7seg port map(ent => sal_lfsr_i(27 downto
                                                24), leds => bcd7);
          U8: entity work.deco7seg port map(ent => sal_lfsr_i(31 downto
                                                28), leds => bcd8);
------ Instanciacion del Divisor de Frecuencia ----------------
          U9: entity work.div_frec port map(sel => "001", clk_50 => clk,
clk_sel => clk_div, sal1_7seg => open, sal2_7seg => open);
------ Entrada Serie al Contador LFSR ----------------------
    serial_in <= realim;
------ Proceso de Realimentacion XOR de la -----------------
------ ecuacion de realimentacion --------------------------
    realim_proc: process (sal_lfsr_i, sel)
          begin
          case sel is
                when "00" =>
                      realim <= sal_lfsr_i(1) xor sal_lfsr_i(0);
                when "01" =>
                      realim <= sal_lfsr_i(4) xor sal_lfsr_i(3) xor
                                  sal_lfsr_i(2) xor sal_lfsr_i(0);
                when "10" =>
                      realim <= sal_lfsr_i(5) xor sal_lfsr_i(4) xor
                                  sal_lfsr_i(3) xor sal_lfsr_i(0);
                when others =>
                      realim <= sal_lfsr_i(22) xor sal_lfsr_i(2) xor
                                  sal_lfsr_i(1) xor sal_lfsr_i(0);
          end case;
    end process realim_proc;
------ Proceso de seleccion de largo del ------------------
------ contador LFSR --------------------------------------
    sel_proc: Process (rst, clk_div, sel)
          variable ancho_lfsr: integer;
    begin
    --- El cambio entre un contador y otro se debe ---------
    --- realizar un reset para inicializar todas las -------
    --- salidas (Alta impedancia) --------------------------
          if(rst= '0') then
                sal_lfsr_i <= (0 => '1', others => '0');
          elsif(rising_edge(clk_div)) then
          case sel is
                when "00" =>
                      sal_lfsr_i(31 downto ancho_lfsr_a) <= (others =>
'Z');
                      sal_lfsr_i(ancho_lfsr_a-1 downto 0) <= serial_in &
sal_lfsr_i(ancho_lfsr_a-1 downto 1);
                when "01" =>
                      sal_lfsr_i(31 downto ancho_lfsr_b) <= (others =>
'Z');
                      sal_lfsr_i(ancho_lfsr_b-1 downto 0) <= serial_in &
sal_lfsr_i(ancho_lfsr_b-1 downto 1);
                when "10" =>
                      sal_lfsr_i(31 downto ancho_lfsr_c) <= (others =>
'Z');
                      sal_lfsr_i(ancho_lfsr_c-1 downto 0) <= serial_in &
sal_lfsr_i(ancho_lfsr_c-1 downto 1);
                when others =>
```

```
                        sal_lfsr_i(ancho_lfsr_d-1 downto 0) <= serial_in &
sal_lfsr_i(ancho_lfsr_d-1 downto 1);
          end case;
          end if;
end rtl;
```

4.1.6. Contador LFSR 12 bits

```
------------------------------------------------------------
--Declaracion de Librerias
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
------------------------------------------------------------------
-- Entidad --
------------------------------------------------------------------
entity cont_lfsr_12b is
    generic(     initial_value: std_logic_vector(11 downto 0):=
"100000000000";
                        lfsr_width : integer := 12);
    port( clk : in std_logic;
                rst : in std_logic;
                q_lfsr_12b: out std_logic_vector(lfsr_width-1 downto 0) );
end cont_lfsr_12b;
------------------------------------------------------------------
-- Architectura --
------------------------------------------------------------------
architecture beh of cont_lfsr_12b is
    signal q_lfsr_12b_i: std_logic_vector(lfsr_width-1 downto 0);
begin
    -- Shifter Process
    lfsr_cnt_proc: process(rst, clk)
    begin
          if(rst= '1' ) then
                q_lfsr_12b_i <= initial_value;
          elsif (rising_edge(clk)) then
          shifter_loop: for i in 11 downto 1 loop
                q_lfsr_12b_i(i-1) <= q_lfsr_12b_i(i);
          end loop shifter_loop;
                q_lfsr_12b_i(11) <= q_lfsr_12b_i(6) xor q_lfsr_12b_i(4)
xor q_lfsr_12b_i(1) xor q_lfsr_12b_i(0);
          end if;
    end process lfsr_cnt_proc;
    q_lfsr_12b <= q_lfsr_12b_i;
end architecture beh;
```

4.1.7. Divisor de Frecuencia (Contador)

```
--Descripcion    : Divisor de frecuencia con entrada de seleccion para
--        elegir entre las frecuencias 5Hz, 2Hz, 1Hz, 0.5Hz y 0.1Hz.
--        El mismo se implemento utilizando un contador simple. El
--        circuito tambien cuenta con decodificadores BCD a 7
--        segmentos con el fin de mostrar por los display el valor ---
    de frecuencia seleccionado.
```

```
----------------------------------------------------------
--Declaracion de Librerias
----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
----------------------------------------------------------
-- Declaracion de Entidad
----------------------------------------------------------
entity div_frec is
------ Generic que permite seleccionar el largo ------------
------ del contador para obtener la frecuencia -------------
------ deseada ---------------------------------------------
    generic( ancho_cont: natural := 28);
    port( sel: in std_logic_vector(2 downto 0);
                clk_50 : in std_logic;
                clk_sel: out std_logic;
                sal1_7seg, sal2_7seg: out std_logic_vector(6 downto 0));
end entity;
----------------------------------------------------------
-- Arquitectura
----------------------------------------------------------
architecture beh of div_frec is
------- Declaracion de señales internas --------------------
    signal clk_sel_i : std_logic := '0';
    signal clk_sel_i2 : std_logic := '0';
--  signal cont_int : unsigned (ancho_cont-1 downto 0) := (others => '0');
--  signal sal_int : std_logic_vector (ancho_cont-1 downto 0);
    ---- Declaracion de un tipo especial para el deco -------
    type bcd is array(0 to 1) of std_logic_vector(3 downto 0);
    signal ent_bcd: bcd;
begin
------ Instanciacion de los decodificadores que ------------
------ permiten mostrar por display las frecuencias --------
    U1: entity work.decoBCD7seg port map(ent => ent_bcd(0) ,leds =>
sal1_7seg); --Unidad de la frecuencia
    U2: entity work.decoBCD7seg port map(ent => ent_bcd(1) ,leds =>
sal2_7seg); --Decimas de la frecuencia
------ Proceso que cuenta con el contador simple -----------
------ y que entrega los clocks de 5Hz 1Hz 2Hz 0.5Hz 0.1Hz -
    clks_proc : process(clk_50,sel)
          variable rst: std_logic := '0';
          variable cont_int : unsigned (ancho_cont-1 downto 0) := (others
=> '0');
          variable sal_int : std_logic_vector (ancho_cont-1 downto 0);
    begin
    --- Contador simple con reset generado internamente -----
          if(rising_edge(clk_50)) then
                if(rst = '1') then
                      cont_int := (others => '0');
                else
                      cont_int := cont_int + 1;
                end if;
          end if;
```

```
            sal_int := std_logic_vector(cont_int);
            if(rising_edge(clk_50)) then
                  rst := '0';
      --- Seleccion de la frecuencia del clock de salida ------
            case sel is
            when "000" =>
                  if(sal_int >= "01110111001101011001010000000") then
                        clk_sel_i2 <= not(clk_sel_i);
                        clk_sel_i <= clk_sel_i2;
                        rst := '1';
                  end if;
                  ent_bcd(0) <= "0000";           --Muestra un 0 en la
unidad del 7segmento
                  ent_bcd(1) <= "0001";                 --Muestra un 1 en
las decimas del 7segmento
            when "001" =>
                  if(sal_int >= "00010111110101111000010000000") then
                        clk_sel_i2 <= not(clk_sel_i);
                        clk_sel_i <= clk_sel_i2;
                        rst := '1';
                  end if;
                  ent_bcd(0) <= "0000";           --Muestra un 0 en la
unidad del 7segmento
                  ent_bcd(1) <= "0101";                 --Muestra un 5 en
las decimas del 7segmento
            when "010" =>
                  if(sal_int >= "000010111111010111110000100000") then
                        clk_sel_i2 <= not(clk_sel_i);
                        clk_sel_i <= clk_sel_i2;
                        rst := '1';
                  end if;
                  ent_bcd(0) <= "0001";           --Muestra un 1 en la
unidad del 7segmento
                  ent_bcd(1) <= "1111";                 --No muestra nada
en las decimas del 7segmento
            when "011" =>
                  if(sal_int >= "000001011111101011111000010000") then
                        clk_sel_i2 <= not(clk_sel_i);
                        clk_sel_i <= clk_sel_i2;
                        rst := '1';
                  end if;
                  ent_bcd(0) <= "0010";           --Muestra un 2 en la
unidad del 7segmento
                  ent_bcd(1) <= "1111";                 --No muestra nada
en las decimas del 7segmento
            when others =>
                  if(sal_int >= "00000010011000100010110100000") then
                        clk_sel_i2 <= not(clk_sel_i);
                        clk_sel_i <= clk_sel_i2;
                        rst := '1';
                  end if;
                  ent_bcd(0) <= "0101";           --Muestra un 5 en la
unidad del 7segmento
                  ent_bcd(1) <= "1111";                 --No muestra nada
en las decimas del 7segmento
```

```
          end case;
          end if;
    end process clks_proc;
------ Asignacion del valor de salida -------------------
    clk_sel <= clk_sel_i;
end beh;
```

4.1.8. Divisor de Frecuencia (PLL)

```
--Descripcion    : Divisor de frecuencia implementado con un PLL. El mismo
--        cuenta con entrada de seleccion para elegir entre las
--        frecuencias 5Hz, 2Hz, 1Hz,  0.5Hz y 0.1Hz. El circuito
--        cuenta con decodificadores BCD a 7 segmentos con el fin de --
    mostrar la freceuncia seleccionada en display.
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
-------------------------------------------------------------
-- Declaracion de Entidad
-------------------------------------------------------------
entity div_frec_pll is
------ Generic que permite seleccionar el largo ------------
------ del contador para obtener la frecuencia -------------
------ deseada ---------------------------------------------
          generic(ancho_cont: natural := 12);
          port( sel: in std_logic_vector(2 downto 0);
                      inclk0_sig, areset_sig: in std_logic;
                      clk_sel, led_lock: out std_logic;
                      sal1_7seg, sal2_7seg: out std_logic_vector(6 downto
0));
end entity;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture rtl of div_frec_pll is
------ Declaracion del PLL ----------------------------------
component div_pll
    PORT(  areset       : IN STD_LOGIC  := '0';
                inclk0       : IN STD_LOGIC  := '0';
                c0           : OUT STD_LOGIC ;
                locked       : OUT STD_LOGIC);
end component;
------- Declaracion de señales internas --------------------
    signal clk_sel_i : std_logic := '0';
    signal clk_sel_i2 : std_logic := '0';
    signal c0_sig, locked_sig: std_logic;
    signal led_lock_i: std_logic := '0';
    type bcd is array(0 to 1) of std_logic_vector(3 downto 0);
    signal ent_bcd: bcd;
begin
```

```
------ Instanciacion del PLL -------------------------------
div_pll_inst : div_pll PORT MAP (
          areset  => areset_sig,
          inclk0  => inclk0_sig,
          c0      => c0_sig,
          locked  => locked_sig);

------ Instanciacion de los decodificadores ----------------
    U1: entity work.decoBCD7seg port map(ent => ent_bcd(0) ,leds =>
sal1_7seg); --Unidad de la frecuencia
    U2: entity work.decoBCD7seg port map(ent => ent_bcd(1) ,leds =>
sal2_7seg); --Decimas de la frecuencia
------ Proceso que cuenta con el contador simple -----------
------ y que entrega los clocks de 5Hz 1Hz 2Hz 0.5Hz 0.1Hz -
    clks_proc : process(c0_sig,sel,areset_sig,locked_sig)
          variable rst: std_logic := '0';
          variable cont_int : unsigned (ancho_cont-1 downto 0) := (others
=> '0');
          variable sal_int : std_logic_vector (ancho_cont-1 downto 0);
    begin
    --- Contador simple con reset generado internamente -----
    if(areset_sig = '1') then
          cont_int := (others => '0');
          --clk_sel_i <= '0';
    elsif(rising_edge(c0_sig))then
          if(locked_sig = '1') then
                if(rst = '1') then
                      cont_int := (others => '0');
                else
                      cont_int := cont_int + 1;
                end if;
          end if;
    end if;
          sal_int := std_logic_vector(cont_int);

    if(rising_edge(c0_sig)) then
          rst := '0';
    --- Seleccion de la frecuencia del clock de salida ------
          case sel is
          when "000" =>
                if(sal_int >= "101110111000") then
                      clk_sel_i2 <= not(clk_sel_i);
                      clk_sel_i <= clk_sel_i2;
                      rst := '1';
                end if;
                ent_bcd(0) <= "0000";           --Muestra un 0 en la
unidad del 7segmento
                ent_bcd(1) <= "0001";                 --Muestra un 1 en
las decimas del 7segmento
          when "001" =>
                if(sal_int >= "001001011000") then
                      clk_sel_i2 <= not(clk_sel_i);
                      clk_sel_i <= clk_sel_i2;
                      rst := '1';
                end if;
```

```
                ent_bcd(0) <= "0000";           --Muestra un 0 en la
unidad del 7segmento
                ent_bcd(1) <= "0101";                   --Muestra un 5 en
las decimas del 7segmento
          when "010" =>
                if(sal_int >= "000100101100") then
                      clk_sel_i2 <= not(clk_sel_i);
                      clk_sel_i <= clk_sel_i2;
                      rst := '1';
                end if;
                ent_bcd(0) <= "0001";           --Muestra un 1 en la
unidad del 7segmento
                ent_bcd(1) <= "1111";                   --No muestra nada
en las decimas del 7segmento
          when "011" =>
                if(sal_int >= "000010010110") then
                      clk_sel_i2 <= not(clk_sel_i);
                      clk_sel_i <= clk_sel_i2;
                      rst := '1';
                end if;
                ent_bcd(0) <= "0010";           --Muestra un 2 en la
unidad del 7segmento
                ent_bcd(1) <= "1111";                   --No muestra nada
en las decimas del 7segmento
          when others =>
                if(sal_int >= "000000111100") then
                      clk_sel_i2 <= not(clk_sel_i);
                      clk_sel_i <= clk_sel_i2;
                      rst := '1';
                end if;
                ent_bcd(0) <= "0101";           --Muestra un 5 en la
unidad del 7segmento
                ent_bcd(1) <= "1111";                   --No muestra nada
en las decimas del 7segmento
          end case;
    end if;
    end process clks_proc;
---- Asignacion del valor de salida ----------------------
    clk_sel <= clk_sel_i;
    led_lock <= locked_sig;
end rtl;
```

4.1.9. PLL

```
--Descripcion: Instanciacion del PLL con el cual se divide la frecuencia
--             de 50Mhz del clock interno del FPGA Cyclone IV, a un valor
--             de 1.2Khz
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

-------------------------------------------------------------
```

```
-- Declaracion de Entidad
-------------------------------------------------------------
entity pll_12 is
    port(  clk_50, reset : in std_logic;
                clk_sel: out std_logic;
                led_lock: out std_logic);
end entity;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture rtl of pll_12 is
------ Declaracion del PLL ---------------------------------
    component div_pll
          PORT(  areset       : IN STD_LOGIC  := '0';
                      inclk0       : IN STD_LOGIC  := '0';
                      c0           : OUT STD_LOGIC ;
                      locked       : OUT STD_LOGIC);
    end component;
------- Declaracion de señales internas --------------------
    signal locked_sig : std_logic;
begin
------ Instanciacion del PLL -------------------------------
div_pll_inst : div_pll PORT MAP (
          areset  => reset,
          inclk0  => clk_50,
          c0      => clk_sel,
          locked  => locked_sig);
          led_lock <= locked_sig;
end rtl;
```

4.1.10. Sistema de Contadores

```
--Descripcion    : Sistema conformado por 4 diferentes contadores (contador
--        LFSR de 4 bits, contador BCD, contador ascendente binario,
--        contador descendente binario). Utilizando un circuito de 4
--        multiplexer y un decodificador de BCD a 7 segmentos, se
--        muestra por display el valor del contador seleccionado.
--        Las señales asincronicas recibidas se sincronizan con el ---
    clock del sistema. Cuenta con un circuito de reset
--        activado asincronicamente y desactivado sincronicamente.
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

-------------------------------------------------------------
-- Declaracion de Entidad
-------------------------------------------------------------
entity sistema_h is
    port(  clk_50, rst_asin: in std_logic;
                sel_cont: in std_logic_vector(1 downto 0);
```

```
                clk_div: out std_logic;
                num_bcd, frec_uni, frec_deci: out std_logic_vector(6
downto 0));
end sistema_h;
------------------------------------------------------------
-- Arquitectura
------------------------------------------------------------
architecture rtl of sistema_h is
------- Declaracion de señales internas --------------------
    signal rst, clk_1: std_logic;
    signal sel_cont_sincro: std_logic_vector(1 downto 0);
    signal cont_lfsr: std_logic_vector(31 downto 0) := (others => '0');
    signal cont_bcd, cont_down, cont_up, cont_lfsr_i: std_logic_vector(3
downto 0) := (others => '0');
begin
------ Señales auxiliares para la correcta instanciacion ---
    cont_lfsr_i <= cont_lfsr(3 downto 0);
    clk_div <= clk_1;
------ Instanciacion de todos los componentes --------------
------ necesarios para el sistema --------------------------
    U1: entity work.reset port map( clk => clk_50, rst => rst_asin,
rst_a_s => rst);
    U2: entity work.Sincro port map(D => sel_cont(0), CLK => clk_50, RST
=> rst, Q => sel_cont_sincro(0));
    U3: entity work.Sincro port map(D => sel_cont(1), CLK => clk_50, RST
=> rst, Q => sel_cont_sincro(1));
    U4: entity work.div_frec port map(clk_50 => clk_50, sel => "010",
clk_sel => clk_1, sal1_7seg => frec_uni, sal2_7seg => frec_deci);
    U5: entity work.sel_cont_lfsr port map(clk => clk_1, rst => rst,
sal_lfsr => cont_lfsr, sel => "00");
    U6: entity work.cont_BCD port map(clk => clk_1, rst => rst, bcd =>
cont_bcd);
    U7: entity work.cont_bin_down port map(clk => clk_1, rst => rst,
bin_down => cont_down);
    U8: entity work.cont_bin_up port map(clk => clk_1, rst => rst, bin_up
=> cont_up);
    U9: entity work.recver port map(selmux => sel_cont_sincro, cbcd1 =>
cont_lfsr_i, cbcd2 => cont_bcd, cbcd3 => cont_down, cbcd4 => cont_up, salr
=> num_bcd);
end rtl;
```

4.1.11. Contador BCD

```
--Descripcion    : Contador BCD ascendente simple
------------------------------------------------------------
--Declaracion de Librerias
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
------------------------------------------------------------
-- Declaracion de Entidad
```

```
------------------------------------------------------------
entity cont_BCD is
    generic(cont_ancho: natural := 4);
    port(  clk, rst: in std_logic;
                bcd: out std_logic_vector(cont_ancho-1 downto 0));
end cont_BCD;
------------------------------------------------------------
-- Arquitectura
------------------------------------------------------------
architecture rtl of cont_BCD is
------- Declaracion de señales internas --------------------
    signal cont_i: unsigned(cont_ancho-1 downto 0) := (others =>'0');
begin
------- Proceso que describe el contador -------------------
    cont_bcd_proc: process(clk, rst)
    begin
          if(rst = '0') then
                cont_i <= (others => '0');
          elsif(rising_edge(clk)) then
                if(cont_i = "1001") then
                      cont_i <= (others => '0');
                else
                      cont_i <= cont_i + 1;
                end if;
          end if;
    end process cont_bcd_proc;
------- Asignacion de la salida ----------------------------
    bcd <= std_logic_vector(cont_i);
end rtl;
```

4.1.12. Contador DOWN

```
--Descripcion    : Contador descendente simple
------------------------------------------------------------
--Declaracion de Librerias
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
------------------------------------------------------------
-- Declaracion de Entidad
------------------------------------------------------------
entity cont_bin_down is
    generic(cont_ancho: natural := 4);
    port(  clk, rst: in std_logic;
                bin_down: out std_logic_vector(cont_ancho-1 downto 0));
end cont_bin_down;
------------------------------------------------------------
-- Arquitectura
------------------------------------------------------------
architecture rtl of cont_bin_down is
------- Declaracion de señales internas --------------------
    signal cont_i: unsigned(cont_ancho-1 downto 0) := (others =>'1');
```

```
begin
------- Proceso que describe el contador -------------------
    cont_bcd_proc: process(clk, rst)
    begin
          if(rst = '0') then
                cont_i <= (others => '1');
          elsif(rising_edge(clk)) then
                cont_i <= cont_i - 1;
          end if;
    end process cont_bcd_proc;
------- Asignacion de la salida ----------------------------
    bin_down <= std_logic_vector(cont_i);
end rtl;
```

4.1.13. Contador UP

```
--Descripcion    : Contador ascendente simple
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
-------------------------------------------------------------
-- Declaracion de Entidad
-------------------------------------------------------------
entity cont_bin_up is
    generic(cont_ancho: natural := 4);
    port(  clk, rst: in std_logic;
                bin_up: out std_logic_vector(cont_ancho-1 downto 0));
end cont_bin_up;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture rtl of cont_bin_up is
------- Declaracion de señales internas --------------------
    signal cont_i: unsigned(cont_ancho-1 downto 0) := (others =>'0');
begin
------- Proceso que describe el contador -------------------
    cont_bcd_proc: process(clk, rst)
    begin
          if(rst = '0') then
                cont_i <= (others => '0');
          elsif(rising_edge(clk)) then
                cont_i <= cont_i + 1;
          end if;
    end process cont_bcd_proc;
------- Asignacion de la salida ----------------------------
    bin_up <= std_logic_vector(cont_i);
end rtl;
```

4.1.14. Generador de PWM

```
--Descripcion    : Modulo generador de señal PWM con entrada de
--        establecimiento del ciclo de trabajo. El nuevo ciclo de
--        trabajo ingresado se refleja en la salida a partir de la
--        activacion de la señal LOAD. El circuito esta controlado
--        por flanco positivo del clock y un reset asincronico.
--        El modulo tambien cuenta con decodificador de HEX a 7
--        segmentos que muestra por display el valor de los 4 bits
--        menos significativos de la entrada del ciclo de trabajo.
------------------------------------------------------------
--Declaracion de Librerias
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
------------------------------------------------------------
-- Declaracion de Entidad
------------------------------------------------------------
entity pwm_8b is
    generic(cont_ancho: natural := 8);
    port(  clk, rst, load: in std_logic;
                duty_cicle: in std_logic_vector(cont_ancho-1 downto 0);
                pwm_out: out std_logic;
                bcd_sal: out std_logic_vector(6 downto 0));
end pwm_8b;
------------------------------------------------------------
-- Arquitectura
------------------------------------------------------------
architecture rtl of pwm_8b is
------- Declaracion de señales internas --------------------
    signal cont_i: unsigned(cont_ancho-1 downto 0) := (others =>'0');
    signal duty: std_logic_vector(cont_ancho-1 downto 0) := (others =>
'0');
begin
------ Instanciacion del decodificador ---------------------
    U1: entity work.deco7seg port map(ent => duty_cicle(3 downto 0) ,leds
=> bcd_sal);
------ Proceso que genera el Tiempo en alto y --------------
------ Tiempo en bajo de la señal de salida (PWM) ----------
    cont_proc: process(clk, rst)
    begin
          if(rst = '1') then
                cont_i <= (others => '0');
          elsif(rising_edge(clk)) then
                cont_i <= cont_i + 1;
                if(std_logic_vector(cont_i) >= duty) then
                      pwm_out <= '0';
                else
                      pwm_out <= '1';
                end if;
          end if;
    end process cont_proc;
------ Proceso que verifica el cambio del -----------------
```

```
------ ciclo de trabajo con la señal LOAD -----------------
    duty_proc: process(clk, duty_cicle, load)
    begin
          if(rising_edge(clk)) then
                if(load = '1') then
                      duty <= duty_cicle;
                end if;
          end if;
    end process duty_proc;
end rtl;
```

4.1.15. Bloque de Generadores PWM

```
--Descripcion    : Bloque de 8 generadores de señales PWM con ciclos
--         de trabajo fijos.
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
-------------------------------------------------------------
-- Declaracion de Entidad
-------------------------------------------------------------
entity bloque_pwm is
------ Generic que establece los valores fijos -------------
------ de los ciclos de trabajo (en porcentaje)-------------
    generic(bloque_ancho: natural := 8;
          duty_cicle8: std_logic_vector(7 downto 0) := "11111110";--99%
          duty_cicle7: std_logic_vector(7 downto 0) := "11100101";--90%
          duty_cicle6: std_logic_vector(7 downto 0) := "11001100";--80%
          duty_cicle5: std_logic_vector(7 downto 0) := "10110010";--70%
          duty_cicle4: std_logic_vector(7 downto 0) := "01111111";--50%
          duty_cicle3: std_logic_vector(7 downto 0) := "01001100";--30%
          duty_cicle2: std_logic_vector(7 downto 0) := "00110011";--20%
          duty_cicle1: std_logic_vector(7 downto 0) := "00010110");--10%
    port(  clk_50, rst_b, load_b: in std_logic;
                bloque_pwm_out: out std_logic_vector(bloque_ancho-1 downto
0));
end bloque_pwm;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture rtl of bloque_pwm is
---- Declaracion de un tipo especial para la asignacion ----
---- de los ciclos de trabajos ------------------------------
    type duty is array(0 to 7) of std_logic_vector(7 downto 0);
    signal duty_opc: duty := (0 => duty_cicle1,
                              1 => duty_cicle2,
                              2 => duty_cicle3,
                              3 => duty_cicle4,
                              4 => duty_cicle5,
                              5 => duty_cicle6,
```

```
                              6 => duty_cicle7,
                              7 => duty_cicle8);
begin
---- Instanciacion de los 8 generadores de PWM con sus -----
---- respectivos ciclos de trabajo -------------------------
    pwm_gen: for i in bloque_pwm_out'range generate
          U: entity work.pwm_8b port map(clk => clk_50 ,rst => rst_b, load
=> load_b, duty_cicle => duty_opc(i), pwm_out => bloque_pwm_out(i));
    end generate;
end rtl;
```

4.2. Descripciones Test Bench

4.2.1. Sincronizador

```
-- Descripcion: Simulacion funcional del sincronizador por medio de
--                       un Test Bench
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
-------------------------------------------------------------
-- Declaracion de Entidad (Vacia)
-------------------------------------------------------------
entity testbench_sincro is
end;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture tb_behave of testbench_sincro is
------- Declaracion de señales internas --------------------
    signal test_D, test_RST, test_Q: std_logic := '0';
    signal test_CLK: std_logic := '0';
begin
------ Instanciacion del Sincronizador ---------------------
    U: entity work.Sincro port map(D => test_D, CLK => test_CLK, RST =>
test_RST, Q => test_Q);
------ Generacion de Señales de simulacion -----------------
    test_CLK <= not test_CLK after 10ns;
    test_D <= '0', '1' after 23ns, '0' after 42ns, '1' after 47ns, '0'
after 77ns, '1' after 82ns, '0' after 95ns, '1' after 100ns, '0' after
117ns, '1' after 129ns, '0' after 137ns, '1' after 146ns, '0' after 179ns;
    test_RST <= '1', '0' after 53ns, '1' after 103ns;
end tb_behave;
```

4.2.2. Circuito Antirebote

```
-- Descripcion : Simulacion funcional del circuito antirebote por medio
--          de un Test Bench
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.numeric_std.all;
------------------------------------------------------------
-- Declaracion de Entidad (Vacia)
------------------------------------------------------------
entity testbench_antirebote is
end;
------------------------------------------------------------
-- Arquitectura
------------------------------------------------------------
architecture tb_behave of testbench_antirebote is
------- Declaracion de señales internas --------------------
    signal test_ent, test_rst, test_sal: std_logic := '0';
    signal test_clk: std_logic := '0';
begin
------ Instanciacion del Circuito Antirebote ---------------
    U: entity work.Antirebote port map(ent => test_ent, clk => test_clk,
rst => test_rst, sal => test_sal);
------ Generacion de Señales de simulacion -----------------
    test_clk <= not test_CLK after 10ns;
    test_ent <= '0', '1' after 50ns, '0' after 51ns,'1' after 52ns, '0'
after 53ns, '1' after 54ns, '0' after 55ns,'1' after 56ns,'0' after
57ns,'0' after 58ns, '1' after 59ns, '0' after 100ns, '1' after 101ns, '0'
after 102ns, '1' after 103ns, '0' after 104ns,'1' after 200ns, '0' after
201ns,'1' after 202ns, '0' after 203ns, '1' after 204ns, '0' after
205ns,'1' after 206ns,'0' after 207ns,'0' after 208ns, '1' after 209ns,
'0' after 250ns, '1' after 251ns, '0' after 252ns, '1' after 253ns, '0'
after 254ns ;
    test_rst <= '1', '0' after 240ns, '1' after 260ns;
------ Mensajes de error dentro de la Simulacion -----------
    test_signals: process
    begin
          wait for 35ns;
                assert test_sal = '0'
                      Report "salida no es correcta"
                            severity ERROR;
          wait for 95ns;
                assert test_sal = '1'
                      Report "salida no es correcta"
                            severity ERROR;
    end process test_signals;
end tb_behave;
```

4.2.3. Circuito Reset

```
-- Descripcion: Simulacion funcional del circuito de reset por medio de
--                      un Test Bench
------------------------------------------------------------
--Declaracion de Librerias
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
------------------------------------------------------------
-- Declaracion de Entidad (Vacia)
------------------------------------------------------------
entity testbench_reset is
```

```
end;
------------------------------------------------------------
-- Arquitectura
------------------------------------------------------------
architecture tb_behave of testbench_reset is
------- Declaracion de señales internas --------------------
    signal test_rst, test_rst_a_s: std_logic := '0';
    signal test_clk: std_logic := '0';
begin
------ Instanciacion del Circuito de Reset -----------------
    U: entity work.reset port map(clk => test_clk, rst => test_rst,
rst_a_s => test_rst_a_s);
------ Generacion de Señales de simulacion -----------------
    test_clk <= not test_clk after 10ns;
    test_rst <= '1', '0' after 53ns, '1' after 68ns, '0' after 96ns, '1'
after 108ns, '0' after 176ns, '1' after 188ns;
------ Mensajes de error dentro de la Simulacion -----------
    test_signals: process
    begin
          wait for 35ns;
                assert test_rst_a_s = '1'
                      Report "salida no es correcta"
                            severity ERROR;
          wait for 100ns;
                assert test_rst_a_s = '0'
                      Report "salida no es correcta"
                            severity ERROR;
    end process test_signals;
end tb_behave;
```

4.2.4. Contador LFSR

```
-- Descripcion: Simulacion funcional del Contador LFSR por medio de
--                      un Test Bench
------------------------------------------------------------
--Declaracion de Librerias
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
------------------------------------------------------------
-- Declaracion de Entidad
------------------------------------------------------------
entity testbench_sel_cont_lfsr is
end testbench_sel_cont_lfsr;
------------------------------------------------------------
-- Arquitectura
------------------------------------------------------------
architecture tb_behave of testbench_sel_cont_lfsr is
------- Declaracion de señales internas --------------------
    signal test_sel: std_logic_vector(1 downto 0);
    signal test_rst: std_logic := '0';
    signal test_clk: std_logic := '0';
    signal test_sal_lfsr: std_logic_vector(31 downto 0);
begin
```

```
------ Instanciacion del Contador LFSR ---------------------
    U: entity work.sel_cont_lfsr port map(clk => test_clk, rst =>
test_rst, sal_lfsr => test_sal_lfsr, sel => test_sel);
------ Generacion de Señales de simulacion -----------------
    test_sel <= "00", "10" after 500ns, "01" after 1000ns, "11" after
1500ns;
    test_clk <= not test_CLK after 10ns;
    test_rst <= '0', '1' after 20ns, '0' after 510ns, '1' after 530ns, '0'
after 1010ns, '1' after 1030ns,  '0' after 1510ns, '1' after 1530ns;
end tb_behave;
```

4.2.5. Contadores LFSR [Modificado]

```
-- Descripcion: Simulacion funcional del Contador LFSR por medio de
--                      un Test Bench
----------------------------------------------------------
--Declaracion de Librerias
----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
----------------------------------------------------------
-- Declaracion de Entidad
----------------------------------------------------------
entity testbench_sel_cont_lfsr is
end testbench_sel_cont_lfsr;
----------------------------------------------------------
-- Arquitectura
----------------------------------------------------------
architecture tb_behave of testbench_sel_cont_lfsr is
------- Declaracion de señales internas --------------------
    signal test_sel: std_logic_vector(1 downto 0);
    signal test_rst: std_logic := '0';
    signal test_clk: std_logic := '0';
    --signal test_sal_lfsr: std_logic_vector(31 downto 0);
    signal test_bcd1, test_bcd2, test_bcd3, test_bcd4, test_bcd5,
test_bcd6, test_bcd7, test_bcd8: std_logic_vector(6 downto 0);
begin
------ Instanciacion del Contador LFSR ---------------------
    U: entity work.sel_cont_lfsr port map(clk => test_clk, rst =>
test_rst, sel => test_sel, bcd1 => test_bcd1, bcd2 => test_bcd2, bcd3 =>
test_bcd3, bcd4 => test_bcd4, bcd5 => test_bcd5, bcd6 => test_bcd6, bcd7
=> test_bcd7, bcd8 => test_bcd8);
------ Generacion de Señales de simulacion -----------------
    test_sel <= "11", "00" after 5000ms, "01" after 10000ms, "10" after
17000ms;
    test_clk <= not test_CLK after 10ns;
    test_rst <= '0', '1' after 1ms, '0' after 4500ms, '1' after 4510ms,
'0' after 11000ms, '1' after 11010ms,  '0' after 17000ms, '1' after
17010ms;
end tb_behave;
```

4.2.6. Contador LFSR 12 bits

```
-- Descripcion: Simulacion funcional del Contador LFSR 12 bits por medio
--                      de un Test Bench
----------------------------------------------------------
--Declaracion de Librerias
----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
----------------------------------------------------------
-- Declaracion de Entidad
----------------------------------------------------------
entity testbench_cont_lfsr_12b is
end testbench_cont_lfsr_12b;
----------------------------------------------------------
-- Arquitectura
----------------------------------------------------------
architecture tb_behave of testbench_cont_lfsr_12b is
------- Declaracion de señales internas --------------------
    signal test_rst: std_logic := '0';
    signal test_clk: std_logic := '0';
    signal test_q_lfsr_12b: std_logic_vector(11 downto 0);
begin
------ Instanciacion del Contador LFSR ---------------------
    U: entity work.cont_lfsr_12b port map(clk => test_clk, rst =>
test_rst, q_lfsr_12b => test_q_lfsr_12b);
------ Generacion de Señales de simulacion -----------------
    test_clk <= not test_CLK after 10ns;
    test_rst <= '0';
end tb_behave;
```

4.2.7. Divisor de Frecuencia (Contador)

```
-- Descripcion: Simulacion funcional del Divisor de Frecuencia por medio
--         de un Test Bench
----------------------------------------------------------
--Declaracion de Librerias
----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
----------------------------------------------------------
-- Declaracion de Entidad (Vacia)
----------------------------------------------------------
entity testbench_div_frec is
end entity;
----------------------------------------------------------
-- Arquitectura
----------------------------------------------------------
architecture beh of testbench_div_frec is
------- Declaracion de señales internas --------------------
    signal test_sel: std_logic_vector(2 downto 0);
    signal test_clk_50: std_logic := '0';
```

```
    signal test_clk_sel: std_logic := '0';
    signal test_sal1_7seg, test_sal2_7seg: std_logic_vector(6 downto 0) :=
"0000000";
begin
------ Instanciacion del Divisor de Frecuencia -------------
    U: entity work.div_frec port map(clk_50 => test_clk_50, sel =>
test_sel, clk_sel => test_clk_sel, sal1_7seg => test_sal1_7seg, sal2_7seg
=> test_sal2_7seg);
------ Generacion de Señales de simulacion -----------------
    test_clk_50 <= not test_clk_50 after 10ns;
    test_sel <= "010","011" after 4000ms, "100" after 7000ms; --"ZZ" after
400ns, "11" after 600ns;
end beh;
```

4.2.8. Divisor de Frecuencia (PLL)

```
-- Descripcion: Simulacion funcional del Divisor de Frecuencia con PLL
--        por medio de un Test Bench
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
-------------------------------------------------------------
-- Declaracion de Entidad (Vacia)
-------------------------------------------------------------
entity testbench_div_frec_pll is
end entity;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture beh of testbench_div_frec_pll is
------- Declaracion de señales internas --------------------
    signal test_sel: std_logic_vector(2 downto 0);
    signal test_inclk0_sig, test_areset_sig: std_logic := '0';
    signal test_clk_sel, test_led_lock: std_logic := '0';
    signal test_sal1_7seg, test_sal2_7seg: std_logic_vector(6 downto 0) :=
"0000000";
begin
------ Instanciacion del Divisor de Frecuencia -------------
    U: entity work.div_frec_pll port map(inclk0_sig => test_inclk0_sig,
areset_sig => test_areset_sig, sel => test_sel, clk_sel => test_clk_sel,
sal1_7seg => test_sal1_7seg, sal2_7seg => test_sal2_7seg, led_lock =>
test_led_lock);
------ Generacion de Señales de simulacion -----------------
    test_inclk0_sig <= not test_inclk0_sig after 10ns;
    test_sel <= "000","001" after 10000ms; --"ZZ" after 400ns, "11" after
600ns;
    test_areset_sig <= '0';
end beh;
```

4.2.9. PLL

```
-- Descripcion: Simulacion funcional del PLL por medio de
--               un Test Bench
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
-------------------------------------------------------------
-- Declaracion de Entidad (Vacia)
-------------------------------------------------------------
entity testbench_pll_12 is
end entity;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture beh of testbench_pll_12 is
------- Declaracion de señales internas --------------------
    signal test_clk_50, test_reset: std_logic := '0';
    signal test_clk_sel, test_led_lock: std_logic := '0';
begin
------ Instanciacion del Divisor de Frecuencia -------------
    U: entity work.pll_12 port map(clk_50 => test_clk_50, reset =>
test_reset, clk_sel => test_clk_sel, led_lock => test_led_lock);
------ Generacion de Señales de simulacion -----------------
    test_clk_50 <= not test_clk_50 after 10ns;
    test_reset <= '1', '0' after 10000ps,'1' after 2000000000ps, '0' after
2000010000ps;
end beh;
```

4.2.10. Sistema de Contadores

```
-- Descripcion: Simulacion funcional del Bloque de Contadores por medio
--         de un Test Bench
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
-------------------------------------------------------------
-- Declaracion de Entidad (Vacia)
-------------------------------------------------------------
entity testbench_sistema_h is
end testbench_sistema_h;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture beh of testbench_sistema_h is
------- Declaracion de señales internas --------------------
    signal test_sel_cont: std_logic_vector(1 downto 0);
    signal test_clk_50: std_logic := '0';
```

```
    signal test_clk_div: std_logic;
    signal test_rst_asin: std_logic := '0';
    signal test_frec_uni, test_frec_deci, test_num_bcd: std_logic_vector(6
downto 0);
begin
------ Instanciacion del Bloque de Contadores -------------
    U: entity work.sistema_h port map(clk_50 => test_clk_50, sel_cont =>
test_sel_cont, rst_asin => test_rst_asin, frec_uni => test_frec_uni,
frec_deci => test_frec_deci, num_bcd => test_num_bcd, clk_div =>
test_clk_div);
------ Generacion de Señales de simulacion -----------------
    test_clk_50 <= not test_clk_50 after 10ns;
    test_sel_cont <= "00","01" after 5000ms, "10" after 11000ms, "11"
after 16000ms; --"ZZ" after 400ns, "11" after 600ns;
    test_rst_asin <= '0','1' after 500ns, '0' after 8500ms, '1' after
8501ms;
end beh;
```

4.2.11. Generador de PWM

```
-- Descripcion: Simulacion funcional del Modulo generador de PWM por
--         medio de un Test Bench
----------------------------------------------------------
--Declaracion de Librerias
----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
----------------------------------------------------------
-- Declaracion de Entidad (Vacia)
----------------------------------------------------------
entity testbench_pwm_8b is
end;
----------------------------------------------------------
-- Arquitectura
----------------------------------------------------------
architecture tb_behave of testbench_pwm_8b is
------- Declaracion de señales internas --------------------
    signal test_rst, test_load: std_logic := '0';
    signal test_clk: std_logic := '0';
    signal test_pwm_out: std_logic := '1';
    signal test_duty_cicle: std_logic_vector(7 downto 0);
    signal test_bcd_sal: std_logic_vector(6 downto 0);
begin
------ Instanciacion del Modulo PWM ------------------------
    U: entity work.pwm_8b port map(clk => test_clk, rst => test_rst, load
=> test_load, duty_cicle => test_duty_cicle, pwm_out => test_pwm_out,
bcd_sal => test_bcd_sal);
------ Generacion de Señales de simulacion -----------------
    test_clk <= not test_clk after 10ns;
    test_duty_cicle <= "01100100", "00101111" after 20000ns, "00000111"
after 40000ns, "11100111" after 60000ns;
```

```
    test_load <= '0', '1' after 100ns, '0' after 300ns, '1' after 22000ns,
'0' after 22200ns, '1' after 45000ns, '0' after 45200ns, '1' after
64000ns, '0' after 64200ns;
    test_rst <= '0', '1' after 8000ns, '0' after 8500ns, '1' after
51000ns, '0' after 51500ns;
end tb_behave;
```

4.2.12. Bloque de Generadores PWM

```
-- Descripcion: Simulacion funcional del Bloque de PWMs por medio de
--                      un Test Bench
----------------------------------------------------------
--Declaracion de Librerias
----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
----------------------------------------------------------
-- Declaracion de Entidad (Vacia)
----------------------------------------------------------
entity testbench_bloque_pwm is
end entity;
----------------------------------------------------------
-- Arquitectura
----------------------------------------------------------
architecture tb_behave of testbench_bloque_pwm is
------- Declaracion de señales internas --------------------
    signal test_rst_b, test_load_b: std_logic := '0';
    signal test_clk_50: std_logic := '0';
    signal test_bloque_pwm_out: std_logic_vector(7 downto 0);
begin
------ Instanciacion del Bloque de PWMs --------------------
    U: entity work.bloque_pwm port map(clk_50 => test_clk_50, rst_b =>
test_rst_b, load_b => test_load_b, bloque_pwm_out => test_bloque_pwm_out);
------ Generacion de Señales de simulacion -----------------
    test_clk_50 <= not test_clk_50 after 10ns;
    test_load_b <= '0', '1' after 100ns, '0' after 300ns;
    test_rst_b <= '0', '1' after 1000ns, '0' after 1040ns, '1' after
5600ns, '0' after 5640ns;
    --El reset pone todo en alto para empezar el ciclo de trabajo
end tb_behave;
```