DEPARTAMENTO DE ELECTRÓNICA Y AUTOMÁTICA

FACULTAD DE INGENIERÍA – UNIVERSIDAD NACIONAL DE SAN JUAN

Informe de Laboratorio Nº 4

Máquina de Estados Finitos

[Fecha de Presentación: 7 de julio 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 realizan las descripciones de componentes, en lenguaje VHDL, como así también la utilización de Maquinas de Estados Finitos (FSM) que permitan establecer una comunicación serie usando el protocolo RS-232, y un Circuito Antirebote simple. Las descripciones pertinentes se realizarán 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:

- Utilización de un Máquina de Estados Finitas como solución de un problema específico.
- Utilización de las instrucciones secuenciales, concurrentes y paquetes.
- Uso de un constraint file para asignación de pines de I/O, pulsadores, switches y LEDs.
- Uso de código VHDL genérico para inferir una memoria de solo lectura tipo ROM.
- Uso de la herramienta de Quartus para la implementación de una memoria de solo lectura tipo ROM.
- Generar archivo de configuración de FPGA (Cyclone IV EP4CE115F29C7).
- Comunicación entre el módulo descripto e implementado en el Cyclone IV y una PC a través del puerto RS-232.

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 (10492): VHDL Process Statement warning at div_frec.vhd(44): signal "xxx" is read inside the Process Statement but isn't in the Process Statement's sensitivity list]: la advertencia surge básicamente porque no se han declarado en la lista sensitiva todas las señales que son ocupadas como entradas dentro del proceso. La solución es agregar las señales que se usan como asignaciones en el proceso.

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.

Para el caso de las máquinas de estados finitos, se obtuvieron los diagramas de estados también por una de las herramientas que proporciona Quartus, en este caso la herramienta *State Machine Viewer*.

2.1. 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, a diferencia del laboratorio anterior, se basó en una máquina de estados finitos junto con un contador temporizador.

Lo primero que se realizó, fue el diagrama de estados (Figura 2.1.1) con las transiciones correspondientes entre los mismos. Esto se complemento con un diagrama de bloques (Figura 2.1.2) para su mejor entendimiento y facilitar su futura descripción.

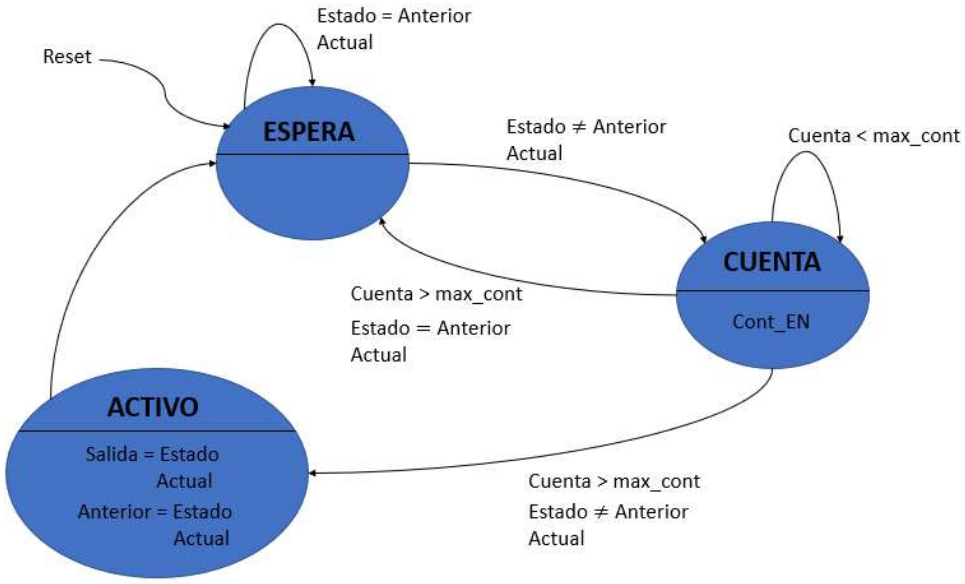

Figura 2. 1. 1 – Diagrama de Estados Circuit Antirebote

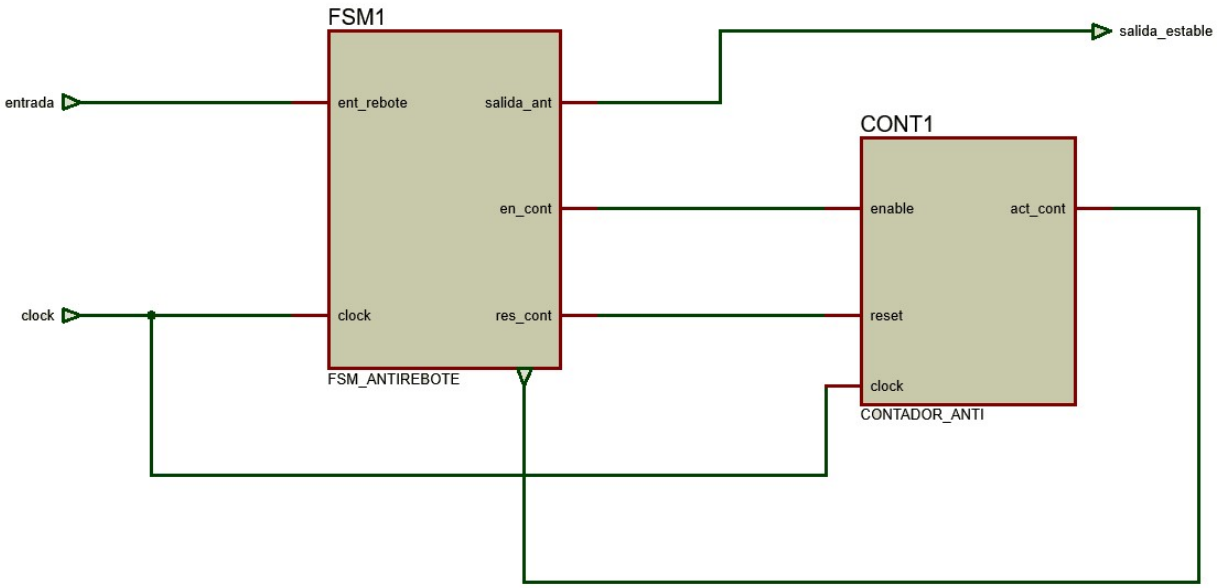

Figura 2. 1. 2 – Diagrama de Bloques Circuito Antirebote

Respecto a los componentes, se mostrará la implementación del contador temporizador, donde su descripción del hardware en VHDL (Apéndice 4.1.1) tiene una *entidad* con cuatro entradas (1 bit de habilitación, 1 bit de reloj, 1 bit de reset y 2 bits de selección) y una salida (1 bit de aviso de fin de temporización). En la *arquitectura* se describió el funcionamiento utilizando dos procesos principales, uno describe la forma de trabajo del contador simple y el otro establece el máximo valor de cuenta dependiendo de la entrada de selección, donde se definen los valores para temporizar durante 20ms de acuerdo a las diferentes frecuencias de trabajo.

El esquemático del diseño (Figura 2.1.3) 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 4.2.1) se utilizaron instrucciones concurrentes para la generación de las ondas de entrada. Se siguieron los mismos pasos que se explicaron anteriormente para poder visualizar las señales de entrada y salida con la herramienta ModelSim (Figura 2.1.4).

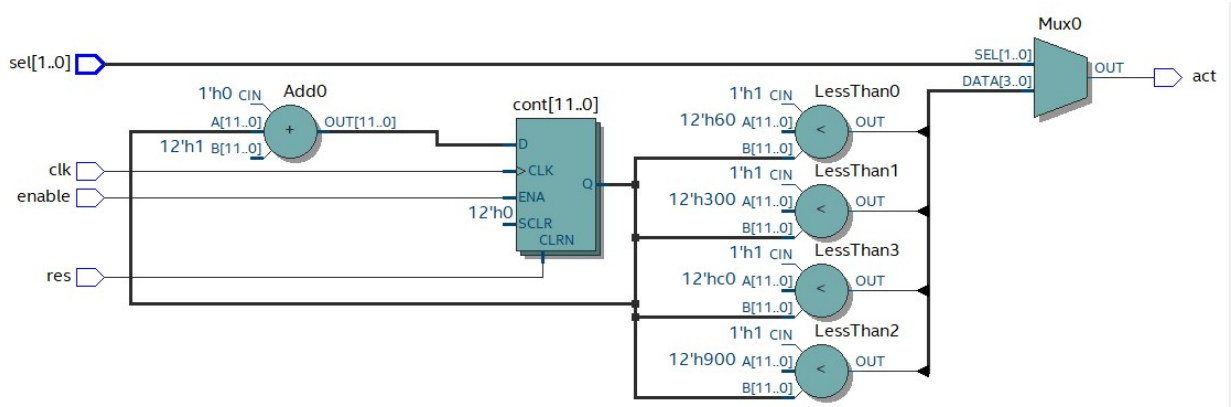

Figura 2. 1. 3 - Esquemático del Contador Temporizador

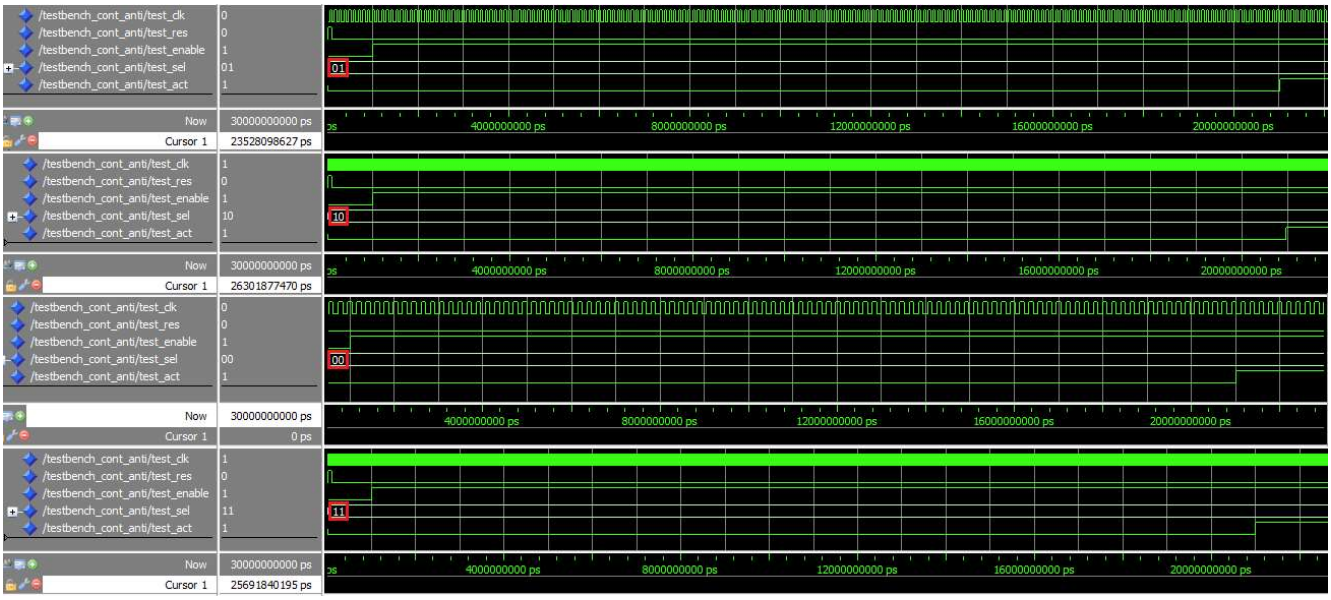

Figura 2. 1. 4 – Simulación del Contador Temporizador

Como se observa, para diferentes selecciones de frecuencia, recuadrado en rojo, se obtiene el mismo tiempo de temporización, que es igual a 20ms. Estas graficas denotan el correcto funcionamiento del diseño realizado.

Finalmente, se obtuvo el Reporte de Área (Tabla 2.1.1), en el cual se especifica los porcentajes utilizados de los recursos disponibles del FPGA.

Tabla 2. 1. 1 - Reporte de Área del Contador Temporizador

Recurso	Porcentaje
Total de elementos lógicos	17 / 114,480 (< 1 %)
Total registros	12
Total pins	6 / 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 %)

Una vez obtenido el contador temporizador, se procedió a describir la máquina de estados finitos (Apéndice 4.1.2) que controla y elimina los rebotes generados. La misma se describió usando el modelo Moore, el cual cuenta con tres procesos principales (Estado Presente, Próximo Estado y Lógica de Salida), siendo el primer y ultimo controlados por reloj.
La descripción del hardware cuenta con una *entidad* de cuatro entradas (1 bit de reloj, 1 bit de reset, 1 bit de señal de entrada y un vector de 2 bits para la selección de frecuencia) y una salida (1 bit de señal de salida sin rebotes). La *arquitectura*, como se mencionó antes, se compone básicamente de tres procesos principales intercomunicados entre ellos.

La sinterización realizada por el software Quartus entrego un diagrama de estados (Figura 2.1.5) similar al establecido al inicio del inciso, permitiendo concluir que las transiciones y comunicaciones establecidas dentro de la descripción son las correctas.
Por otra parte, al igual que el componente anterior, se obtuvo el esquemático del diseño (Figura 2.1.6) a través de la opción RTL Viewer de Quartus, donde se comprueba que no se generen latch erróneos. Para comprobar su funcionamiento, se generaron ondas de prueba para simular las entradas y verificar las salidas. En este TestBench (Apéndice 4.2.2) se siguieron los mismos pasos que se explicaron anteriormente, para poder visualizar las señales de entrada y salida con la herramienta ModelSim (Figura 2.1.7).

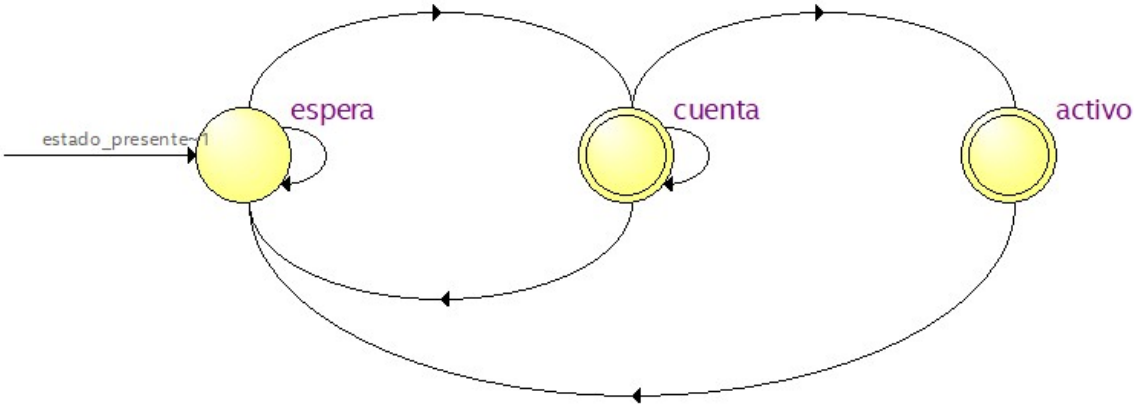

Figura 2. 1. 5 – Diagrama de Estados Circuito Antirebote (Quartus)

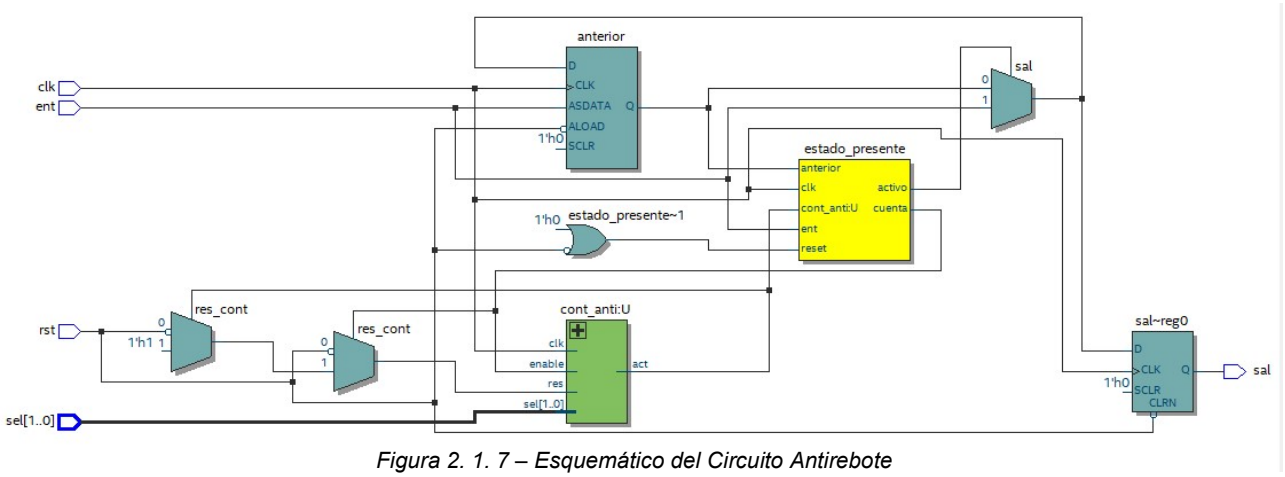

Figura 2. 1. 7 – Esquemático del Circuito Antirebote

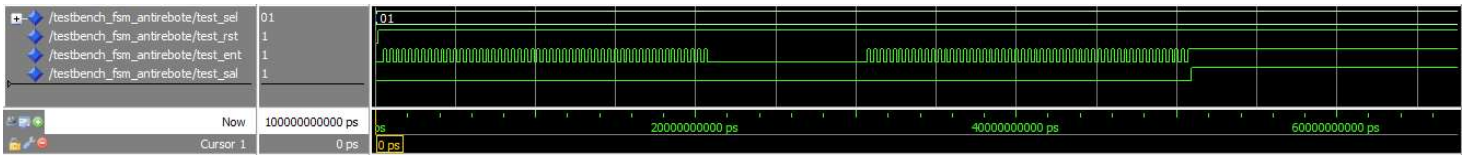

Figura 2. 1. 6 – Simulación del Circuito Antirebote

Se puede apreciar en la simulación como ante una entrada con rebotes constantes, los cuales en la practica duran poco menos que 20ms, y que al estabilizarse no cambia de estado, la salida tampoco lo hace. Lo que si sucede en el segundo tramo, donde luego del tiempo de temporización, la entrada se estabiliza en alto, permitiendo que la salida también lo haga. Por lo descripto, se puede concluir que le circuito cuenta con un funcionamiento correcto.

Para este módulo, a diferencia del anterior, se asignaron las E/S a pines específicos del FPGA (Tabla 2.1.2), se utilizaron los switches como entradas y un led verde como salida.

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

SEÑAL	PIN
clk	PIN_Y2
ent	PIN_AB28
rst	PIN_AC28
sel[0]	PIN_AC27
sel[1]	PIN_AD27
sal	PIN_E21

Como información adicional, se extrajo el Reporte de Área (Tabla 2.1.3) con el objetivo de visualizar la cantidad de recursos usado en esta implementación.

Tabla 2. 1. 3 - Reporte de Área del Circuito Antirebote

Recurso	Porcentaje
Total de elementos lógicos	26 / 114,480 (< 1 %)
Total registros	17
Total pins	6 / 529 (<1 %)
Total pins virtuales	0
Total bits de memoria	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)

Como punto final, también se describió un circuito antirebote específicamente para una frecuencia de 50MHz, el mismo también se compone de un contador temporizador (Apéndice 4.1.3) y la maquina de estado correspondiente (Apéndice 4.1.4) junto con su respectivo TestBench (Apéndice 4.2.3).

2.2. Sistema Digital de Transmisión de Datos (RS-232)

La transmisión serie de datos es una de las formas de comunicación más utilizadas, debido a su sencillez a la hora de implementarlo y de su fiabilidad. En este caso, se utilizo el protocolo de comunicación RS-232, el cual define las normas con las cuales se deben enviar los datos. El mismo establece que el formato del dato a transmitir debe contener un bit de start, ocho bits de datos, un bit de paridad y un bit de stop. De igual manera, se definen las frecuencias estándar de comunicación ($4.8KHz, 9.6KHz, 38.4KHz\ y\ 115.2KHz$).

Con lo antes descripto, lo primero que se realizó fueron los diagramas de estados, con sus respectivas transiciones, de las dos máquinas de estado de tiempo finito que controlan toda la transmisión. Estas son la FSM de Control (Figura 2.2.1), que se encarga de controlar la mayoría de los componentes complementarios, mientras que la FSM de Transmisión (Figura 2.2.2) acondiciona los datos a transmitir, para cumplir con el protocolo RS-232. De forma complementaria, para entender el funcionamiento del sistema, también se dibujo el diagrama en bloques (Figura 2.2.3) con todos los componentes necesarios para llevar a cabo una comunicación exitosa.

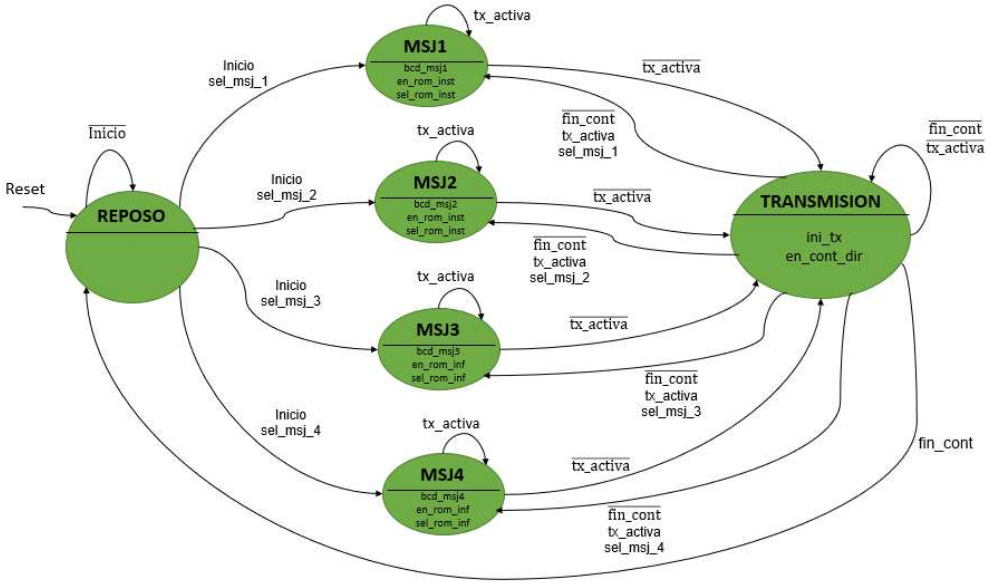

Figura 2. 2. 1 - Diagrama de Estados del Control RS232

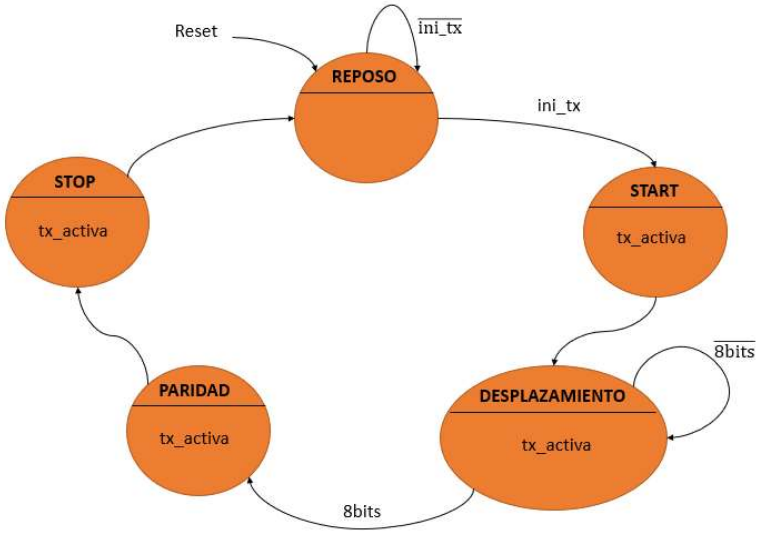

Figura 2. 2. 2 - Diagrama de Estados del Transmisor RS232

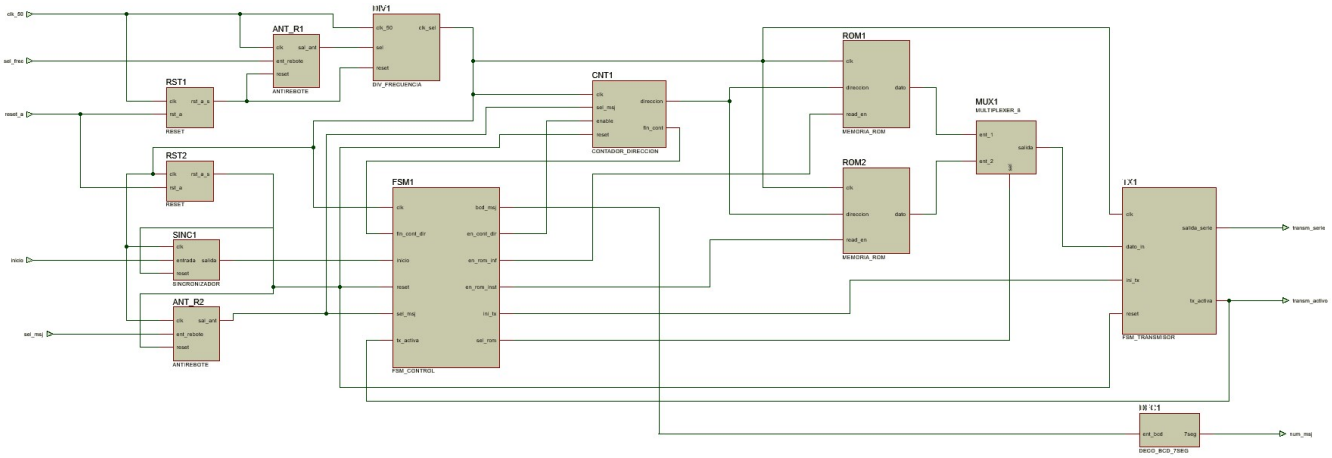

Figura 2. 2. 3 - Diagrama de Bloques del Sistema Digital de Transmisión de Datos

El diagrama anterior muestra la comunicación que tienen los diferentes componentes que componen todo el sistema. Algunos de estos fueron descriptos y probados en los laboratorios anteriores, sin embargo, la mayoría se describieron por primera vez, por lo que a continuación se muestra las descripciones y pruebas cada uno de ellos.

2.2.1. Divisor de Frecuencia

El cristal interno del FPGA otorga una frecuencia de aproximadamente $50MHz$, la que es muy elevada para las transmisiones serie con el protocolo RS-232, 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 las frecuencias necesarias $(4.8KHz, 9.6KHz, 38.4KHz\ y\ 115.2KHz)$ que pueden elegirse por medio de dos entradas de selección.

Una vez explicado el funcionamiento, la descripción del hardware en VHDL (Apéndice 4.1.5) 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 (1 bit de señal de salida). La *arquitectura* implemento el sistema utilizando dos procesos, en donde uno describe el funcionamiento del contador simple y otro 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 4.2.4) descripto al finalizar la arquitectura principal.

El esquemático del divisor obtenido (Figura 2.2.4) 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 2.2.5).

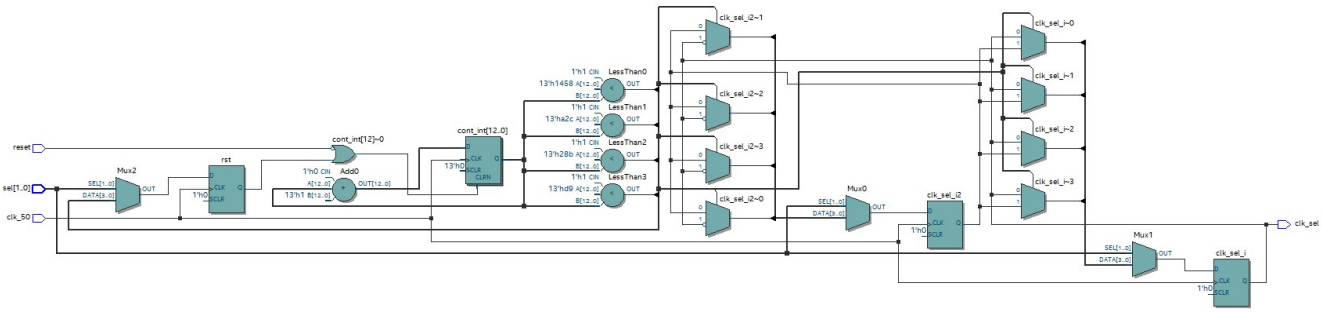

Figura 2. 2. 4 - Esquemático del Divisor de Frecuencia

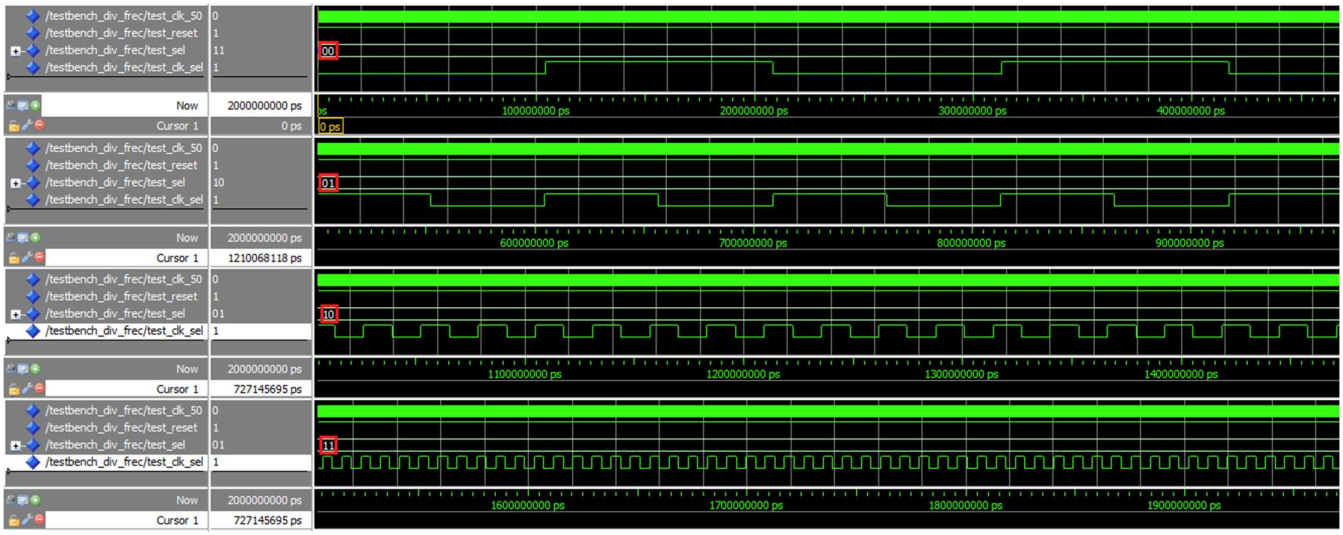

Figura 2. 2. 5 - Simulación del Divisor de Frecuencia

Se puede apreciar, recuadrados en color rojo, las diferentes selecciones de las frecuencias obtenidas con el divisor ($00 \rightarrow 4.8KHz, 01 \rightarrow 9.6KHz, 10 \rightarrow 38.4KHz, 11 \rightarrow 115.2Hz$). Por estos resultados se puede concluir que el divisor descripto tiene un buen funcionamiento.

En este caso no se realizó la asignación de pines, debido a que el componente será instanciado en un proyecto de mayor nivel. De igual forma que con los componentes anteriores, se verifico el Reporte de Área (Tabla 2.2.1) para observar los recursos usados.

Tabla 2. 2. 1 - Reporte de Área del Divisor de Frecuencia

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 %)

2.2.2. Contador de Direcciones

Los mensajes a transmitir se envían de a una letra, las cuales se almacenan en una dirección cada una. Es por ello, que es necesario algún componente que permita seleccionar las direcciones donde estas se ubican para la correcta transmisión. Por lo tanto, se optó por la utilización de un contador simple, el cual contiene las direcciones de fin e inicio de los mensajes a transmitir. El mismo es controlado por la FSM de Control quien envía las señales de habilitación y reset, y el contador realimenta una señal de fin de direcciones.

Teniendo en cuenta el funcionamiento expuesto, se realizó la descripción en VHDL (Apéndice 4.1.6). La *entidad* se compone por cuatro entradas (1 bit de reloj, 1 bit de reset, 1 bit de habilitación y un vector de 2 bits de selección de mensaje) y dos salidas (1 bit de fin de dirección y un vector de 7 bits de direccion). La *arquitectura* se compone de un único proceso, el cual describe el funcionamiento del contador junto con la selección de las direcciones de mensaje. El mismo se simulo por medio de un TestBench (Apéndice 4.2.5) descrito al final de la entidad principal.

El esquemático del contador de direcciones (Figura 2.2.6) se obtuvo mediante la opción de Quartus llamada RTL Viewer, Por otra parte, usando la herramienta ModelSim, y siguiendo los pasos antes descriptos, se observa la evolución de las entradas y salidas en el tiempo (Figura 2.2.7). Como en el esquemático, también se debió hacer algunas modificaciones en las gráficas extraídas para poder visualizar el caso de cada selección de mensaje.

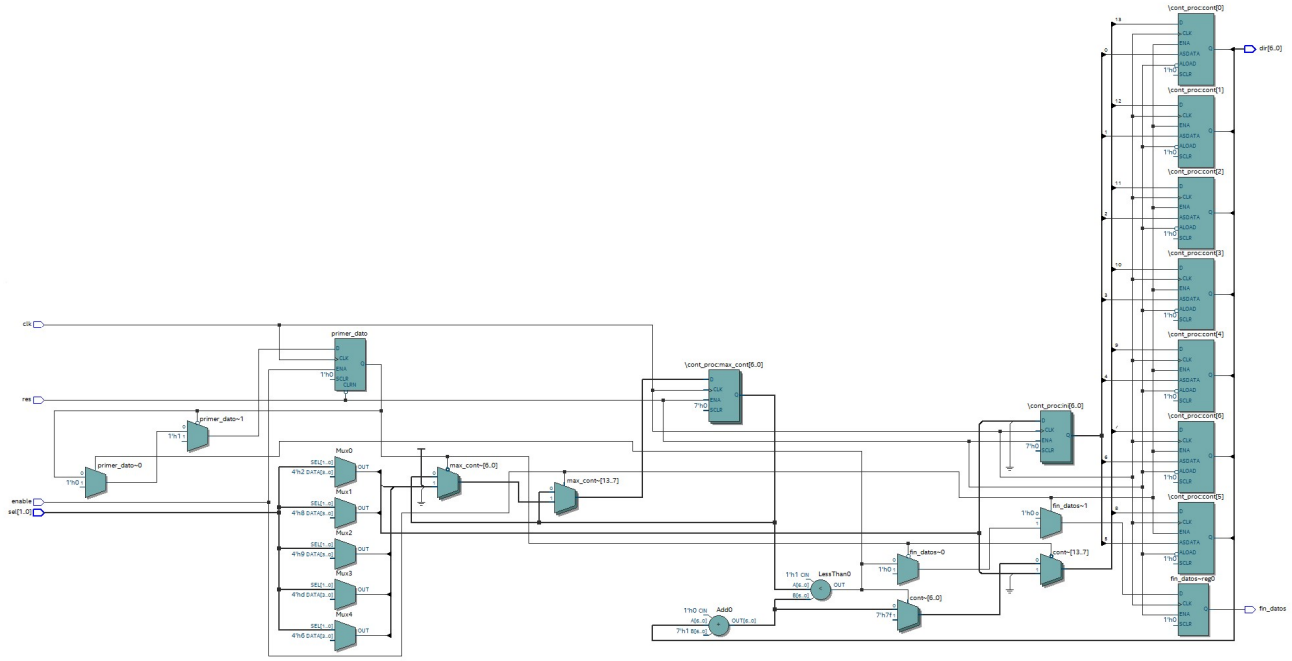

Figura 2. 2. 6 - Esquemático del Contador de Direcciones

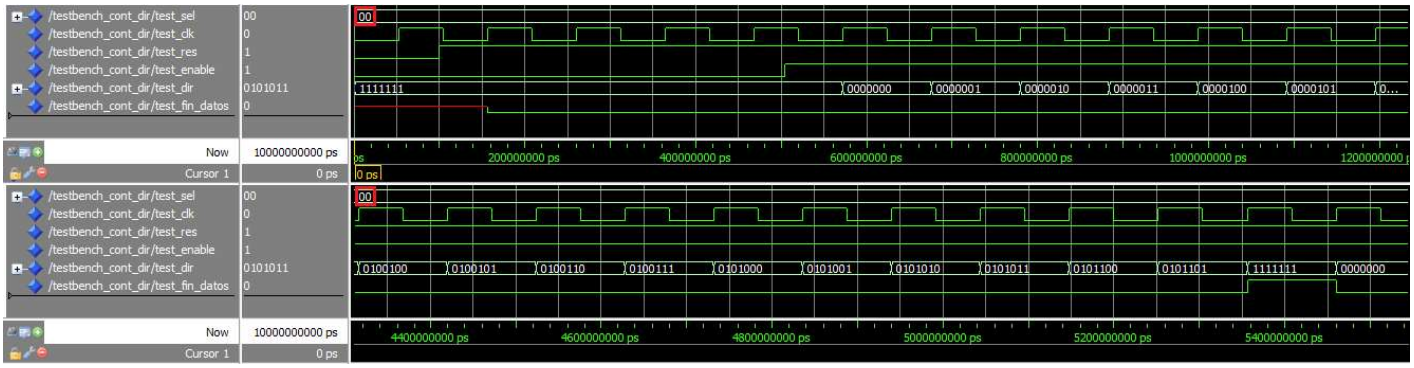

Figura 2. 2. 7 - Simulación del Contador de Direcciones

La simulación muestra recuadrado en rojo la selección del primer mensaje, donde las direcciones van aumentando a partir de que la habilitación se pone en alto. En el segundo tramo observamos como al llegar a la dirección final del mensaje, se activa la señal de fin de dirección que va dirigida a la maquina de estados correspondiente. Con lo obtenido se puede concluir que el componente funciona correctamente.

Finalmente, como los componentes anteriores, se obtuvo el Reporte de Área (Tabla 2.2.2) para comprobar los recursos necesarios para su implementación.

Tabla 2. 2. 2 - Reporte de Área del Contador de Direcciones

Recurso	Porcentaje
Total de elementos lógicos	50 / 114,480 (< 1 %)
Total registros	17
Total pins	13 / 529 (2 %)
Total pins virtuales	0
Total bits de memoria	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

2.2.3. Memoria ROM Inferida

Los mensajes deben almacenarse en algún lugar del FPGA, y como este cuenta con bloques dedicados de memoria, es posible describir una memoria mediante VHDL. Esto se debe a que la herramienta infiere que es una memoria, debido a la forma en que esta descrito el componente. La ventaja de esta forma de implementación es que el código es reusable y transferible a diferentes tipos de FPGA. El mismo cuenta con entradas de habilitación y dirección junto con una salida de datos, que se seleccionan por medio de las direcciones ingresadas.

Con el funcionamiento expuesto, se realizó la descripción del componente en VHDL (Apéndice 4.1.7). La *entidad* esta compuesta por tres entradas (1 bit de reloj, 1 bit de habilitación y un vector de 7 bits de dirección) y una salida (un vector de 8 bits del dato a transmitir). En la *arquitectura* se inicializa la memoria con los datos que serán transmitidos junto con los atributos que permiten implementar la memoria en los bloques dedicados del FPGA. A su vez, se utilizó un único proceso para describir el funcionamiento de la memoria. El TestBench (Apéndice 4.2.6) correspondiente se describió al final de la entidad principal.

Como primer punto se obtuvo el esquemático de la memoria (Figura 2.2.8), mediante la herramienta RTL Viewer de Quartus, verificando que no se generaron latches erróneos. También, se extrajeron las evoluciones de las salidas en el tiempo (Figura 2.2.9), 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

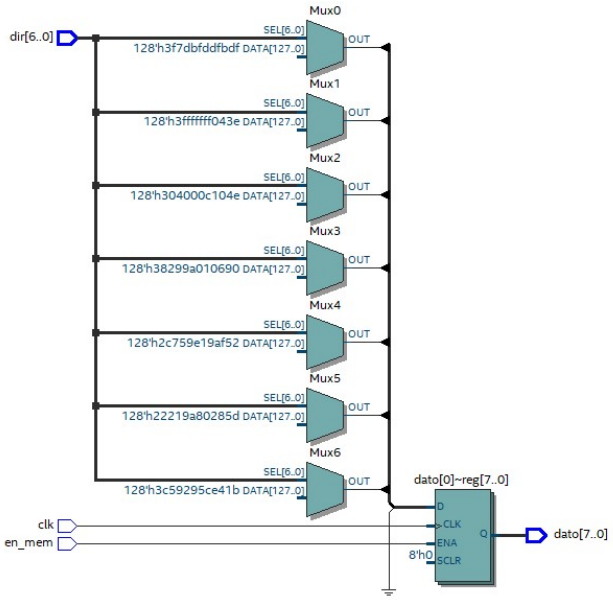

Figura 2. 2. 8 - Esquemático de la Memoria ROM Inferida

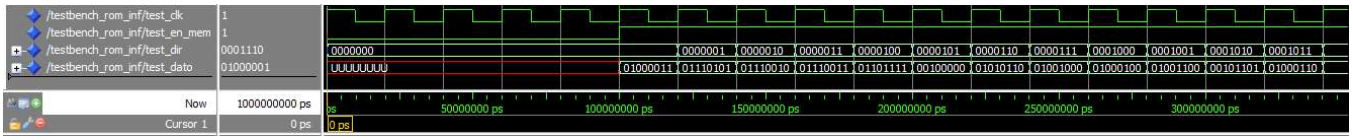

Figura 2. 2. 9 - Simulación de la Memoria ROM Inferida

En la simulación se puede apreciar como hasta que no se activa la habilitación, en la salida de la memoria no hay dato a enviar. Una vez activada, al cambiar las direcciones de entrada se colocando los respectivos datos a la salida. Es por ello que se puede deducir que el componente tiene un buen funcionamiento.

Como información final, se obtuvo el Reporte de Área (Tabla 2.2.3) donde se observa que el ítem de bits de memoria deja de tener valor nulo, a diferencia de los módulos anteriores.

Tabla 2. 2. 3 - Reporte de Área de la Memoria ROM Inferida

Recurso	Porcentaje
Total de elementos lógicos	35 / 114,480 (< 1 %)
Total registros	7
Total pins	17 / 529 (3 %)
Total pins virtuales	0
Total bits de memoria	1,024 / 3,981,312 (<1 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

2.2.4. Memoria ROM Instanciada

La principal diferencia con este tipo de implementación respecto a la anterior, es que una memoria ROM instanciada se obtiene a través de la utilización de una herramienta proporcionada por el fabricante del FPGA. Esto ultimo permite tener un uso mas eficiente de los recursos, pero a su vez lo convierte en un código no reusable en otro tipo de FPGA. Sin embargo, el funcionamiento es muy similar, tanto para una ROM inferida y una ROM instanciada. Cuenta con entradas de habilitación y de dirección, con los cuales se decide el valor de salida.

Teniendo en cuenta lo anterior, la obtención de la memoria se realizó a través del software Quartus, siguiendo los pasos explicitados a continuación: Lo primero que se procedió a hacer es la creación del archivo .mif, en el cual se especifican las dimensiones de la memoria (tamaño) y los datos que contendrá la misma. Para ello se seleccionó en el menú de la ventana de Quartus la opcion *File> New..* donde se despliegan nuevas opciones, y se eligió crear un *Memory Intalzation File*. Luego de esto, se abrió una pequeña ventana donde se estableció el largo de los datos y la cantidad de los mismos dentro de la memoria. Finalmente, se procedió a ingresar el valor de los datos a almacenar en la dirección correspondiente, para terminar con guardar el archivo donde se desee.
Una vez obtenido el archivo .mif, se pudo pasar a generar el componente de la memoria ROM, donde se accedió a la opción *Tools> IP Catalog*. Dentro de ella se siguió el siguiente camino *Library> Basic Functions > On Chip Memory> ROM-1 PORT.* Luego, se abre una pequeña ventana donde se debe especificar la dirección y nombre del componente, junto con lenguaje que se usara para describirlo (VHDL). Después, se despliega un utilitario que permitió empezar a configurar el componente, como fueron los parámetros de su tamaño, las entradas, como activar una de habilitación. Al final de la configuración, se da la opción de generar los archivos que tienen el código necesario para instancia la memoria ROM. Sin mucha diferencia a la memoria anterior, se realizó el TestBench (Apéndice 4.2.7) correspondiente. Para ello, se instancio el componente en un proyecto de mayor jerarquía (Apéndice 4.1.8).

Como verificación del módulo entregado por la herramienta, se obtuvieron tanto el esquemático (Figura 2.2.10), mediante la herramienta RTL Viewer de Quartus, y la evolución de las salidas respecto a entradas preestablecidas en el TestBench (Figura 2.2.11), esto a través del software ModelSim.

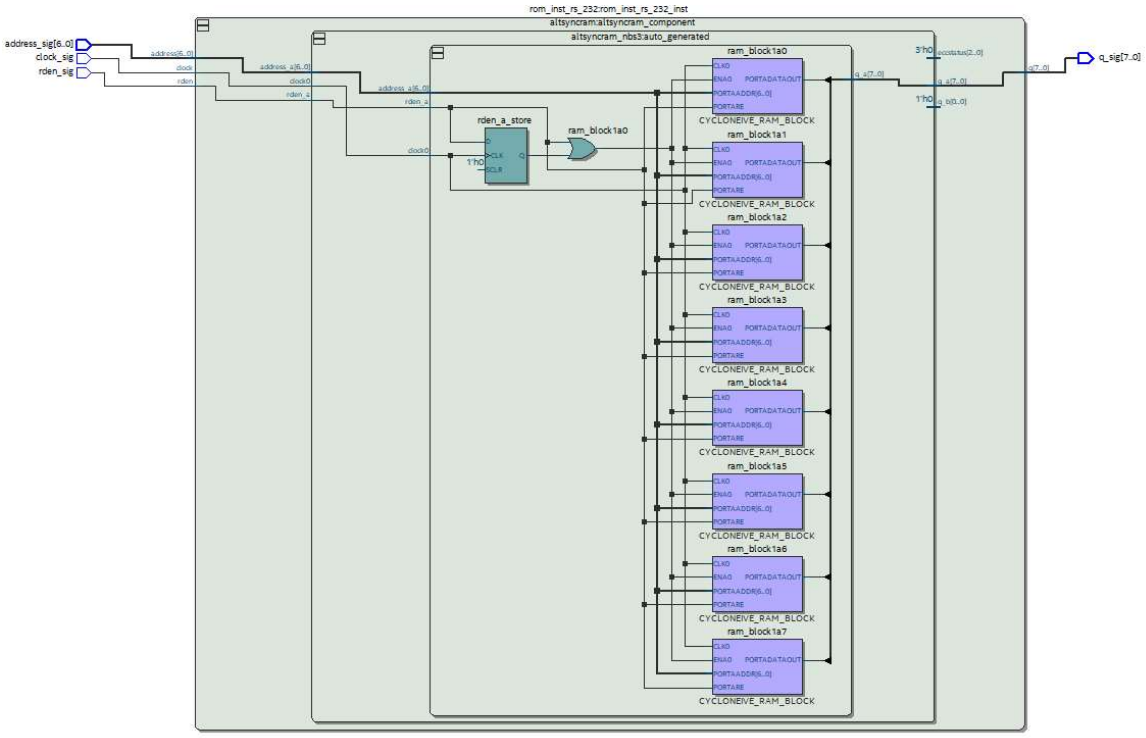

Figura 2. 2. 10 - Esquemático de la Memoria ROM Instanciada

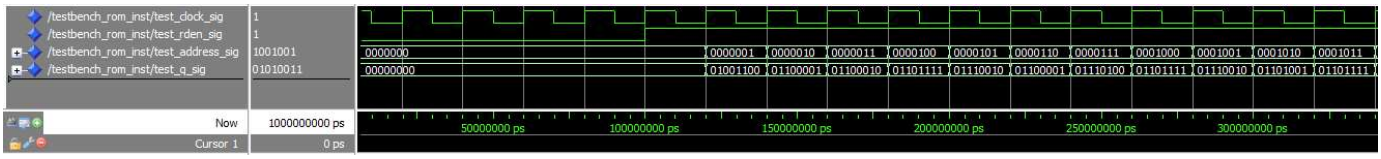

Figura 2. 2. 11 - Simulación de la Memoria ROM Instanciada

La simulación denota cómo hasta que no se activa la habilitación, en la salida de la memoria hay un dato nulo. Una vez activada la habilitación, al cambiar las direcciones de entrada se colocan los respectivos datos a la salida. Es por ello, que se puede concluir que el componente fue creado y configurado de manera correcta.

No menos importante, también se obtuvo el Reporte de Área (Tabla 2.2.4) con el que se comprobó como la memoria se implemento dentro de los bloques dedicados de memoria del FPGA.

Tabla 2. 2. 4 - Reporte de Área de la Memoria ROM Instanciada

Recurso	Porcentaje
Total de elementos lógicos	1 / 114,480 (< 1 %)
Total registros	1
Total pins	17 / 529 (3 %)
Total pins virtuales	0
Total bits de memoria	1,024 / 3,981,312 (<1 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

2.2.5. FSM de Control

Como se explico al inicio del inciso, la maquina de estado de control se encarga de la gestión de todos los componentes (excepto el divisor de frecuencia), utilizando las entradas al sistema para generar las salidas correspondientes hacia los demás módulos. También, se mostro en una de las figaras anteriores (Figura 2.2.1) el diagrama de estado con sus respectivas transiciones.

Entonces, la máquina de estados finitos se describió, por medio de lenguaje VHDL (Apéndice 4.1.9), usando el modelo Moore, el cual cuenta con tres procesos principales (Estado Presente, Próximo Estado y Lógica de Salida), siendo el primer y último controlados por reloj. La descripción de la misma cuenta con una *entidad* de seis entradas (1 bit de reloj, 1 bit de reset, 1 bit de inicio, 1 bit de fin de cuenta del contador de direcciones, 1 bit de transmisión activa y un vector de 2 bits para la selección del mensaje) y seis salidas (1 bit de habilitación de la memoria ROM inferida, 1 bit de habilitación de la memoria ROM instanciada, 1 bit de habilitación del contador de direcciones, 1 bit de selección de ROM, 1 bit de inicio de transmisión y un vector de 4 bits que indica el número de mensaje seleccionado). La *arquitectura*, como se hizo referencia, se compone de tres procesos principales intercomunicados entre ellos, con los cuales se describen las transiciones entre estados y las correspondientes salidas. Su funcionamiento se comprobó mediante un TestBench (Apéndice 4.2.8) descrito al final del proyecto.

El software Quartus sintetizo un diagrama de estados (Figura 2.2.12) similar al dibujado anteriormente, permitiendo concluir que las transiciones y comunicaciones establecidas dentro de la descripción son las correctas.
Por otra parte, al igual que los componentes anteriores, se obtuvo el esquemático del diseño (Figura 2.2.13) a través de la opción RTL Viewer de Quartus, donde se comprueba que no se generen latch erróneos. También, se extrajeron las evoluciones de las salidas en el tiempo (Figura 2.2.14), 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.

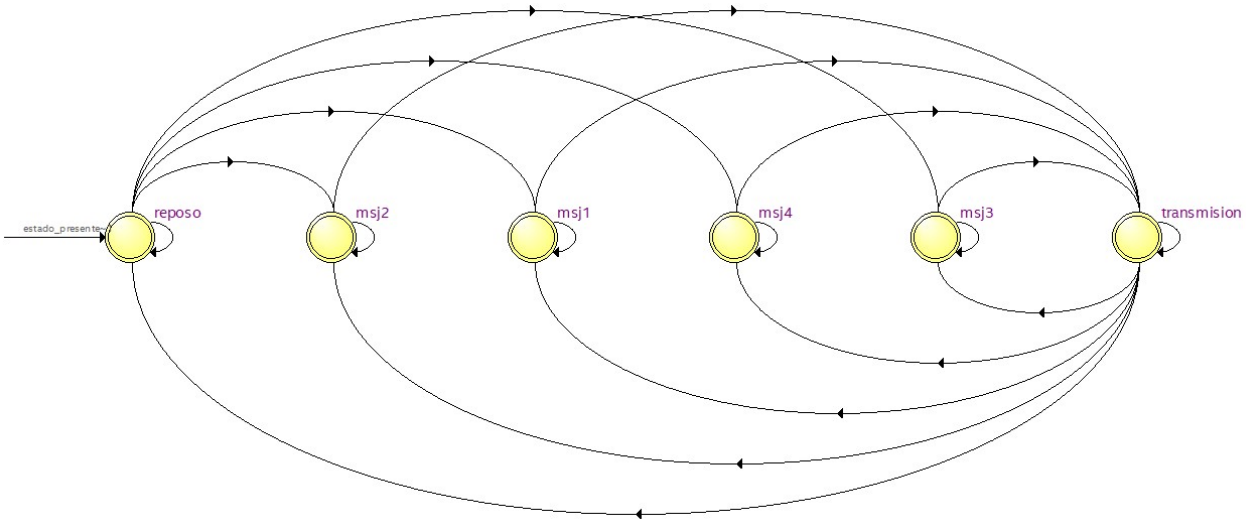

Figura 2. 2. 12 - Diagrama de Estados de la FSM de Control

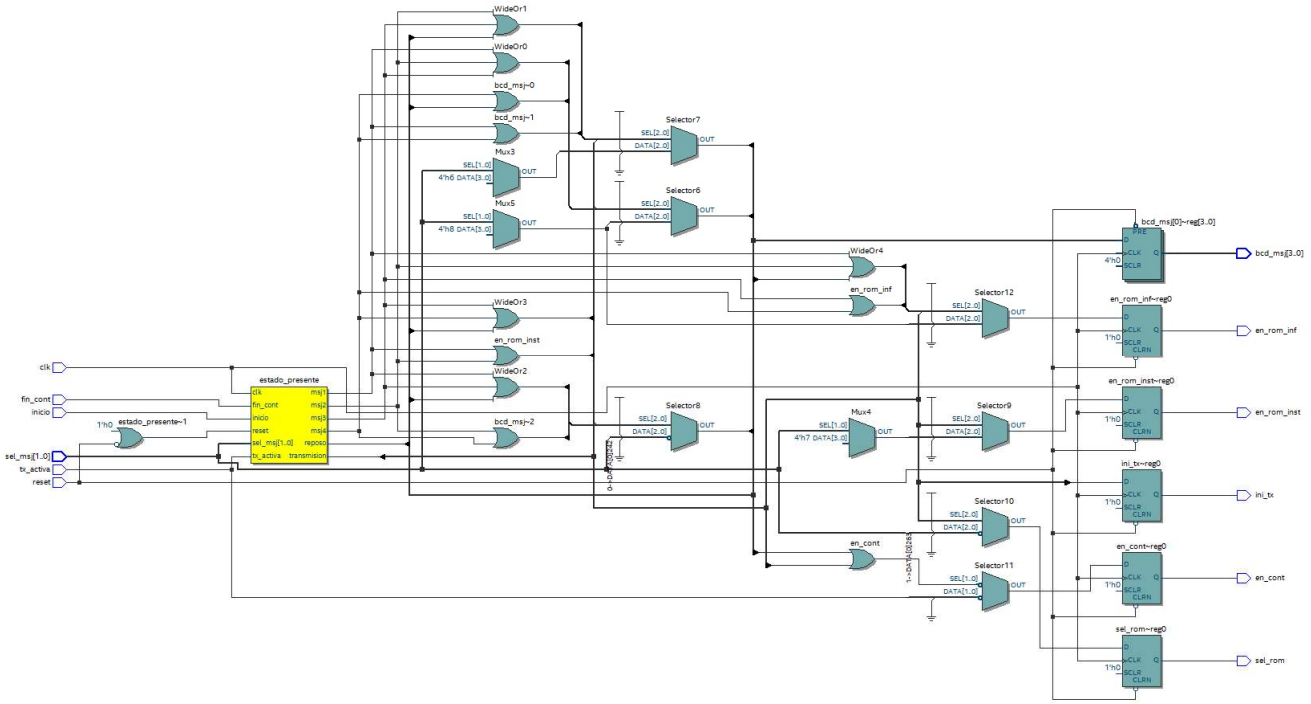

Figura 2. 2. 13 - Esquemático de la FSM de Control

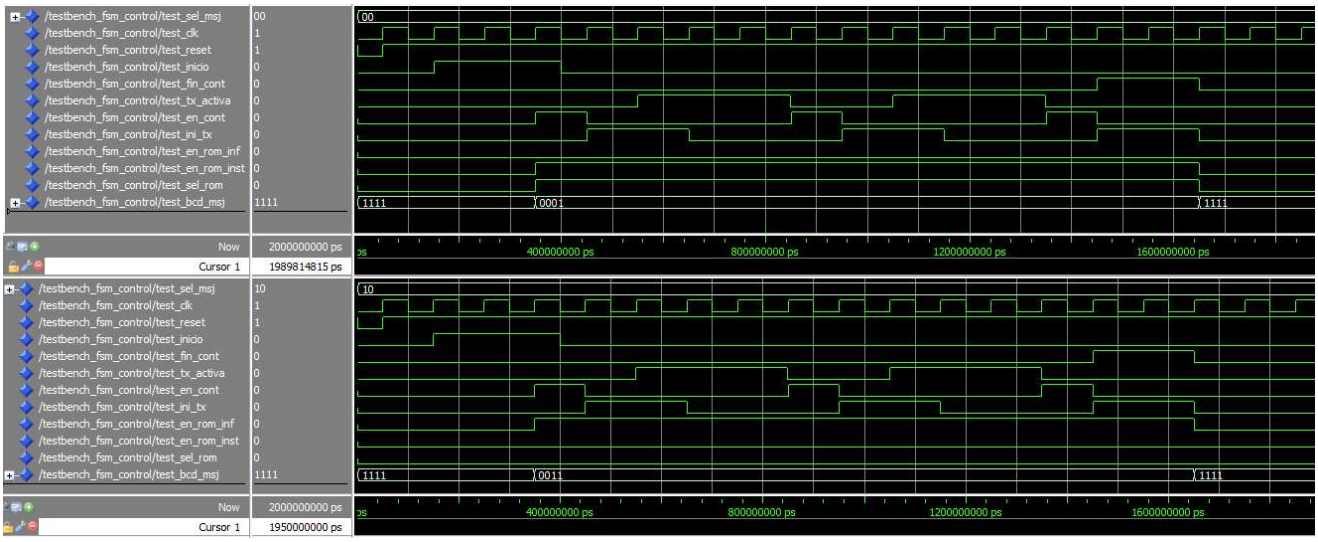

Figura 2. 2. 14 - Simulación de la FSM de Control

Se debe tener en cuenta que para comprender la simulación es necesario correlacionar todas las salidas con las entradas que pueden modificarlas. Teniendo en cuenta esto, se puede apreciar que las señales de salida no comienzan a generarse hasta que no se activa la señal de inicio. Luego, dependiendo de la entrada de selección de mensaje, se activa la habilitación de la memoria ROM inferida o de la instanciada, como asi también la selección de los multiplexores y la salida en BCD. La transmisión siempre comienza con alto en la señal de habilitación del contador de direcciones, seguido de un alto en la señal de inicio del transmisor RS-232, descrito en el inciso siguiente. Al activarse la señal de fin del contador, se genera un último inicio de transmisión y luego esta se finaliza hasta una nueva activación del inicio. Es por ello que al observar lo obtenido, se puede concluir que su funcionamiento es el adecuado.

Como con todos los componentes anteriores, el Reporte de Área (Tabla 2.2.5) se obtiene con el fin de visualizar la cantidad de recursos necesarios para la actual implementación.

Tabla 2. 2. 5 - Reporte de Área de la FSM de Control

Recurso	Porcentaje
Total de elementos lógicos	22 / 114,480 (< 1 %)
Total registros	15
Total pins	16 / 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 %)

2.2.6. FSM de Transmisión RS-232

Al igual que la maquina de estados finitos anterior, esta máquina también fue explicada al inicio del inciso. El objetivo principal de esta FSM es acondicionar el dato a transmitir para cumplir con el protocolo de comunicación serie RS-232, el cual establece el agregado de algunos bits especiales. Todos los bits son enviados consecutivamente, uno por cada periodo del reloj. De igual manera, al inicio se mostró en una de las figuras (Figura 2.2.2) el diagrama de estado con sus respectivas transiciones.

Entonces, la máquina de estados finitos se describió, por medio de lenguaje VHDL (Apéndice 4.1.10), usando el modelo Moore, el cual cuenta con tres procesos principales (Estado Presente, Próximo Estado y Lógica de Salida), siendo el primer y último controlados por reloj. La *entidad* de la FSM se compone de cuatro entradas (1 bit de reloj, 1 bit de reset, 1 bit de inicio de transmisión y un vector de 8 bits del dato a transmitir) y dos salidas (1 bit de para la transmisión serie y 1 bit de aviso de transmisión activa). En la *arquitectura*, como se aclaró en el párrafo anterior, se compone de tres procesos principales intercomunicados entre ellos, con los cuales se describen las transiciones entre estados y se establecen las correspondientes salidas.

Para comprobar el código realizado, se obtuvo del software Quartus la sinterización del diagrama de estados (Figura 2.2.15), el cual resulto ser similar al dibujado anteriormente, permitiendo de esta manera concluir que las transiciones y comunicaciones establecidas dentro de la descripción son las correctas.

El esquemático de la FSM (Figura 2.2.16) 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 4.2.9) correspondiente, utilizando nuevamente las instrucciones concurrentes. La visualización de las señales (Figura 2.2.17) se realizó siguiendo los pasos descriptos al inicio de esta sección.

Otro punto importante a aclarar, es que la obtención del bit de paridad par se realizo por medio de una función descrita dentro de un paquete (Apéndice 4.1.11), el cual fue importado dentro del proyecto para su utilización.

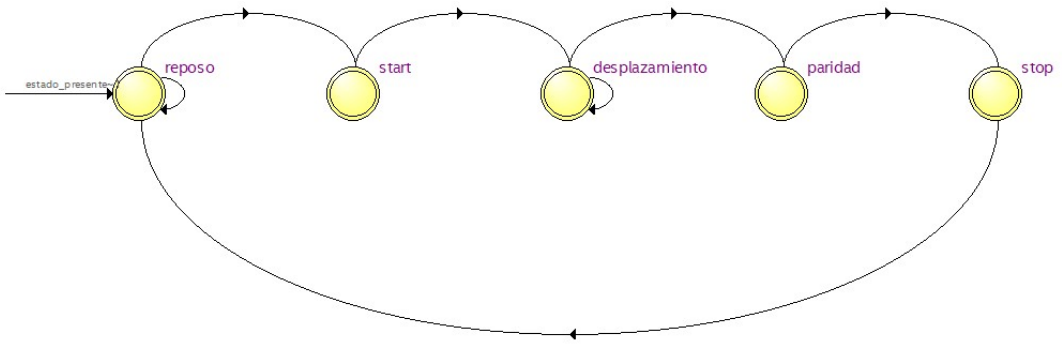

Figura 2. 2. 15 - Diagrama de Estados de la FSM de Transmisión

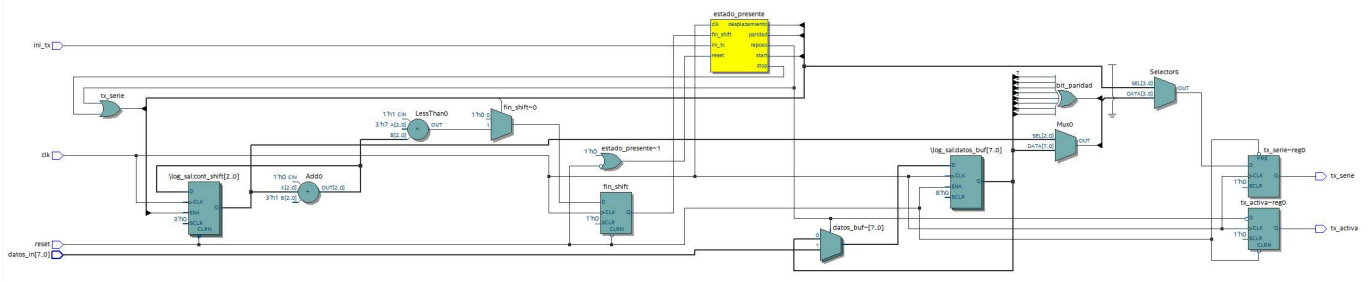

Figura 2. 2. 16 - Diagrama de Bloque de la FSM de Transmisión

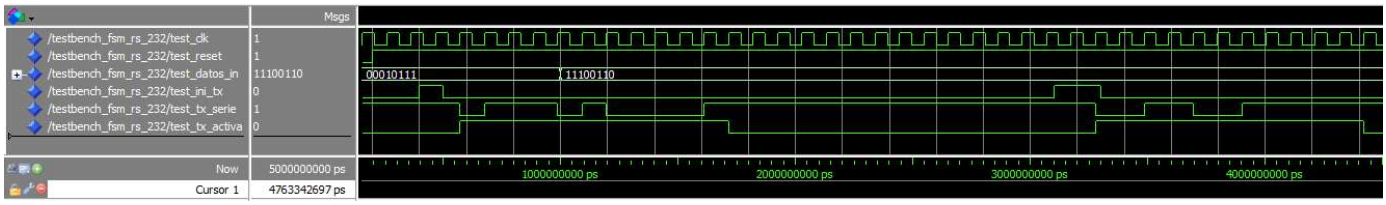

Figura 2. 2. 17 – Simulación de la FSM de Transmisión

La simulación demuestra de forma simple el funcionamiento del sistema, en donde luego de un alto en la entrada de inicio de transmisión, la misma comienza a enviar el dato de entrada cumpliendo con el protocolo RS-232. También, se puede apreciar que al iniciar una transmisión y al haber el cambio en el dato de entrada, no se modifica lo que se esta enviando, si no hasta un próximo alto en la entrada de inicio. Es correcto afirmar el buen funcionamiento de la máquina de estado implementada.

Como complemento de toda la información expuesta, se obtuvo el Registro de Área (Tabla 2.2.6).

Tabla 2. 2. 6 - Reporte de Área de la FSM de Transmisión

Recurso	Porcentaje
Total de elementos lógicos	20 / 114,480 (< 1 %)
Total registros	19
Total pins	13 / 529 (2 %)
Total pins virtuales	0
Total bits de memoria	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

2.2.7. Multiplexores de 2 a 1

En este caso, se describió un conjunto de ocho multiplexores de 2 a 1, con el objetivo de seleccionar entre las salidas de las memorias ROM implementadas. De esta manera, se facilita en gran medida el trabajar con los datos enviados de dos lugares diferentes. El mismo se controla, valga la redundancia, por la FSM de Control simplemente.

Se procedió a realizar la descripción de los multiplexores en lenguaje VHDL (Apéndice 4.1.12) junto con su correspondiente TestBench (Apéndice 4.2.10), para comprobar su correcto funcionamiento. La *entidad* cuenta con tres entradas (dos vectores de 8 bits de los datos y 1 bit de selección de dato) y una salida (vector de 8 bits del dato de salida). Mediante un *generic* se parametrizo el código, permitiendo en un futuro poder cambiar la cantidad de multiplexores a implementar. La *arquitectura* se compone básicamente de una instrucción concurrente with-select, con la cual se realiza la elección del dato de salida dependiendo de la entrada de selección.

La verificación se complementó con la obtención del esquemático (Figura 2.2.18) que sintetiza el simulador, por medio de la herramienta RTL Viewer. Además, la evolución de las salidas en el tiempo (Figura 2.2.19) respecto a las entradas descriptas en el TestBench.

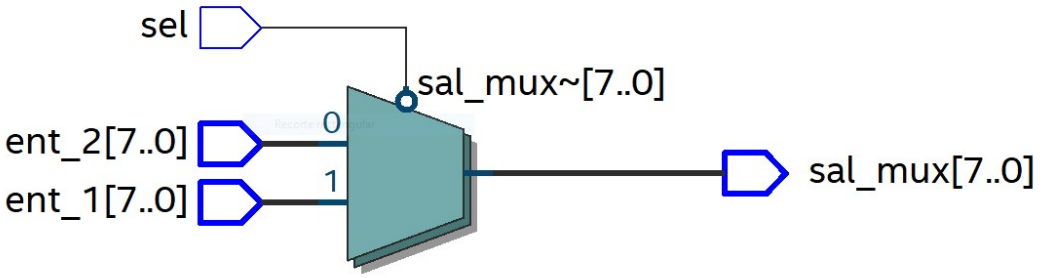

Figura 2. 2. 18 - Esquemático de los Multiplexores

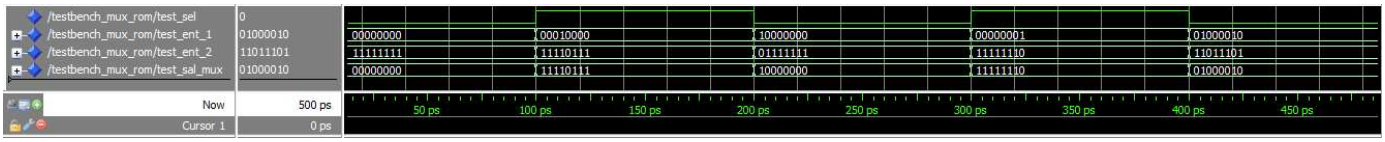

Figura 2. 2. 19 - Simulación de los Multiplexores

Como se puede apreciar en la simulación, las salidas se eligen correctamente dependiendo de la entrada de selección, comprobando su correcto funcionamiento. El Reporte de Área (Tabla 2.2.7) muestra cómo se utilizan una poca cantidad de registros de las LUP.

Tabla 2. 2. 7 - Reporte de Área de los Multiplexores

Recurso	Porcentaje
Total de elementos lógicos	8 / 114,480 (< 1 %)
Total registros	19
Total pins	25 / 529 (5 %)
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 %)

Finalizada la descripción de todos los componentes necesarios para la implementación del sistema de transmisión de datos, se creo un nuevo archivo .vhd. El mismo fue usado como top level, en el cual fueron instanciados todos los módulos descritos hasta el momento (Apéndice 4.1.13). Se agregaron algunos módulos descritos en laboratorios anteriores, como el circuito de reset, sincronizador y decodificador de BCD a 7segmento. De esta manera, se obtiene un código muy simple y segmentado, facilitando su comprensión y utilización.
Su *entidad* se compone básicamente de cinco entradas (1 bit de reloj, 1 bit de reset, 1 bit de inicio, un vector de 2 bits de selección de frecuencia y un vector de 2 bits de selección del mensaje a transmitir) y tres salidas (1 bit de transmisión serie, 1 bit de transmisión actica y un vector de 7 bits hacia el display de 7 segmento). La *arquitectura,* como se especificó anteriormente, solo se compone de instrucciones de instanciación. Sin embargo, de igual manera que los componentes anteriores, se simulo el funcionamiento describiendo las señales con instrucciones concurrentes. Este TestBench (Apéndice 4.1.11) se describió al finalizar la descripción de la arquitectura principal.

Como no sería de otra manera, el esquemático del sistema completo (Figura 2.2.20) 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 2.2.21). Como en el esquemático, también se debió hacer algunas modificaciones en las gráficas extraídas para poder visualizarlas de mejor manera.

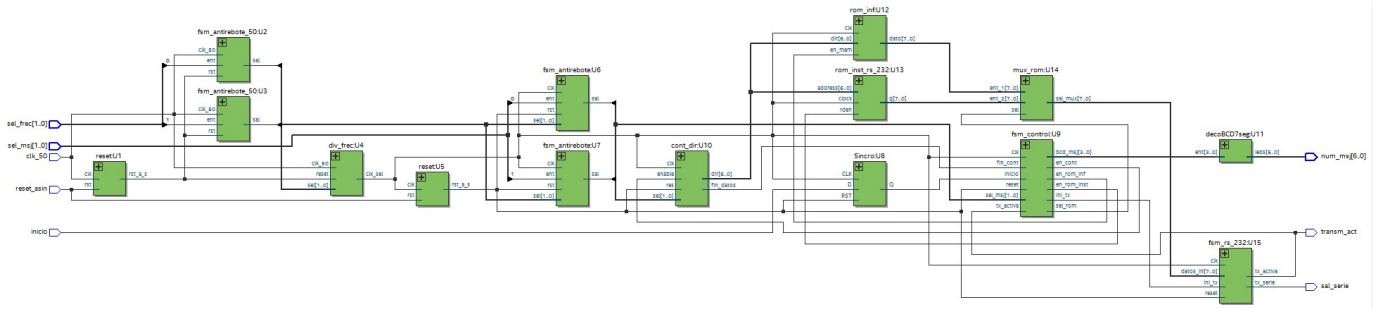

Figura 2. 2. 20 - Esquemático del Sistema Digital de Transmisión de Datos

Figura 2. 2. 21 - Simulación del Sistema Digital de Transmisión de Datos

La figura anterior muestra la evolución de las salidas respecto a las entradas generadas. Como se puede observar se realizaron transmisiones de distintos mensajes (recuadrado en azul) en diferentes frecuencias (recuadrado en rojo). También, apreciar todas las transmisiones series comienzan con un alto en la señal de inicio e inmediatamente el valor enviado al display 7 segmento cambia al valor del mensaje correspondiente. Por todo lo explicado, es posible concluir que el sistema tiene un buen funcionamiento.

Para completar la información expuesta, y no menos importante, los recursos que utiliza la implementación son un factor a tener en cuenta, debido a la gran cantidad de componentes instanciados en el top level. Es por eso, que se obtuvo el Reporte de Área (Tabla 2.2.8) con el cual se podrá verificar estos parámetros.

Tabla 2. 2. 8 - Reporte de Área del Sistema Digital de Transmisión de Datos

Recurso	Porcentaje
Total de elementos lógicos	298 / 114,480 (< 1 %)
Total registros	166
Total pins	16 / 529 (3 %)
Total pins virtuales	0
Total bits de memoria	1,024 / 3,981,312 (<1 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Los componentes descritos en los incisos anteriores no necesitaron asignación de pines debido a que estos serian instanciados, por lo que únicamente se verificaron a través de los TestBench. Entonces, para el archivo de más alto nivel (el sistema que se describe en esta sección) si se realizaron las asignaciones correspondientes (Tabla 2.2.9), con el objetivo de implementarlo y probarlo en la placa del FPGA. Para ello se utilizaron switches, pulsadores, un display 7 segmento y el chip transceptor ZT3232.

Tabla 2. 2. 9 - Asignación de Pines del Sistema Digital de Transmisión de Datos

SEÑAL	PIN
clk_50	PIN_Y2
inicio	PIN_M23
reset_asin	PIN_M21
sel_frec[1]	PIN_AC28
sel_frec[0]	PIN_AB28
sel_msj[1]	PIN_AD27
sel_msj[0]	PIN_AC27
num_msj[6]	PIN_H22
num_msj[5]	PIN_J22
num_msj[4]	PIN_L25
num_msj[3]	PIN_L26
num_msj[2]	PIN_E17
num_msj[1]	PIN_F22
num_msj[0]	PIN_G18
sal_serie	PIN_G9
transm_act	PIN_E21

Para finalizar, se realizó un análisis del Camino Crítico del sistema completo con la ayuda de la herramienta “Timing Analyzer” de Quartus. Se pudieron obtener los diez caminos críticos dentro del sistema (Figura 2.2.22), sin embargo, solo se consideró el primero que aparece en la lista, debido a que es el que presenta mayor retardo.

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	14.972	reset:U1|rst_i[1]	fsm_antirebote_50:U2|anterior~_emulated	clk_50	clk_50	20.000	-0.094	4.952
2	14.981	reset:U1|rst_i[1]	fsm_antirebote_50:U3|sal	clk_50	clk_50	20.000	-0.101	4.936
3	14.981	reset:U1|rst_i[1]	fsm_antirebote_50:U2|sal	clk_50	clk_50	20.000	-0.094	4.943
4	14.998	reset:U1|rst_i[1]	fsm_antirebote_50:U3|anterior~_emulated	clk_50	clk_50	20.000	-0.101	4.919
5	15.007	reset:U1|rst_i[1]	fsm_antirebote_50:U3|estado_presente.espera	clk_50	clk_50	20.000	-0.101	4.910
6	15.011	reset:U1|rst_i[1]	fsm_antirebote_50:U3|estado_presente.cuenta	clk_50	clk_50	20.000	-0.101	4.906
7	15.014	reset:U1|rst_i[1]	fsm_antirebote_50:U3|estado_presente.activo	clk_50	clk_50	20.000	-0.101	4.903
8	15.125	div_frec:U4|clk_sel_i	div_frec:U4|clk_sel_i2	clk_50	clk_50	20.000	-0.080	4.813
9	15.190	reset:U1|rst_i[1]	fsm_antirebote_50:U2|estado_presente.espera	clk_50	clk_50	20.000	-0.094	4.734
10	15.194	reset:U1|rst_i[1]	fsm_antirebote_50:U2|estado_presente.cuenta	clk_50	clk_50	20.000	-0.094	4.730

Figura 2. 2. 22 - Caminos Críticos del Sistema Digital de Transmisión de Datos

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 ($14.972ns$), por lo tanto, es el camino más crítico respecto al tiempo sobrante. 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 2.2.23). 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	⌄ 3.076	3.076					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	70	CLKCTRL_G4	clk_50~inputclkctrl|outclk
7	2.478	1.571	RR	IC	1	FF_X114_Y37_N25	U1|rst_i[1]|clk
8	3.076	0.598	RR	CELL	1	FF_X114_Y37_N25	reset:U1|rst_i[1]
3	⌄ 8.028	4.952					data path
1	3.308	0.232		uTco	1	FF_X114_Y37_N25	reset:U1|rst_i[1]
2	3.308	0.000	FF	CELL	6	FF_X114_Y37_N25	U1|rst_i[1]|q
3	6.161	2.853	FF	IC	1	LCCOMB_X110_Y36_N0	U2|anterior~2|datac
4	6.442	0.281	FF	CELL	5	LCCOMB_X110_Y36_N0	U2|anterior~2|combout
5	7.512	1.070	FF	IC	1	LCCOMB_X110_Y36_N18	U2|anterior~3|dataa
6	7.941	0.429	FR	CELL	1	LCCOMB_X110_Y36_N18	U2|anterior~3|combout
7	7.941	0.000	RR	IC	1	FF_X110_Y36_N19	U2|anterior~_emulated|d
8	8.028	0.087	RR	CELL	1	FF_X110_Y36_N19	fsm_antirebote_50:U2|anterior~_emulated

Figura 2. 2. 23 - 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 2.2.24), donde se ve claramente los retardos y sus valores.

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.

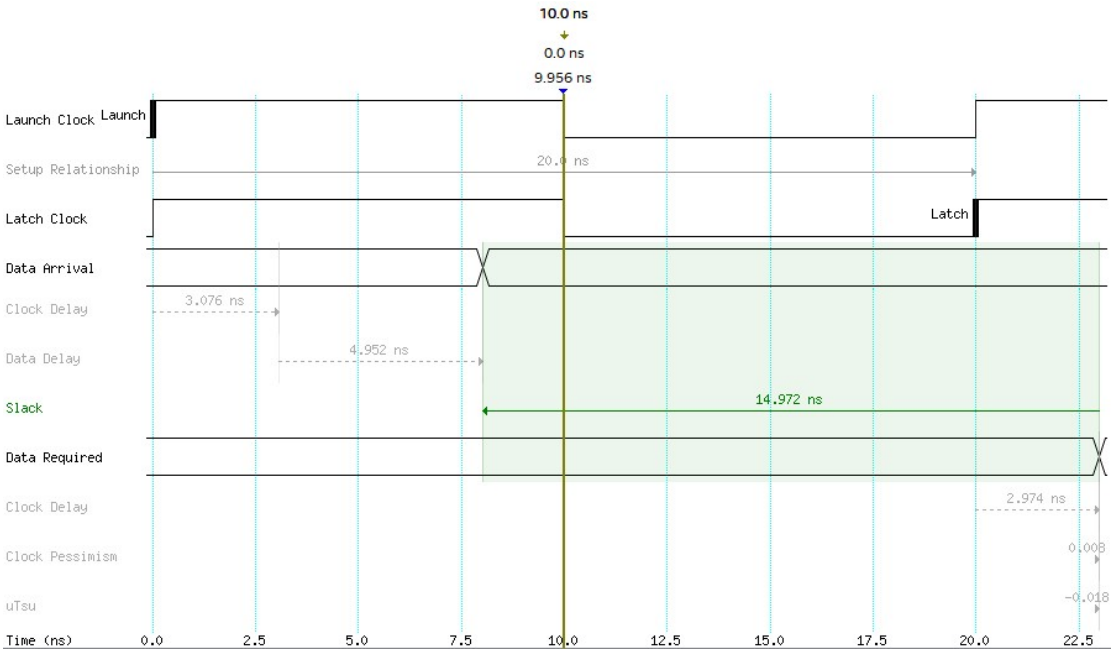

Figura 2. 2. 24 - Formas de Onda de los Retardos

Como punto final, 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 2.2.25). En esta figura se observan las partes del sistema implementado que inciden en el retado máximo.

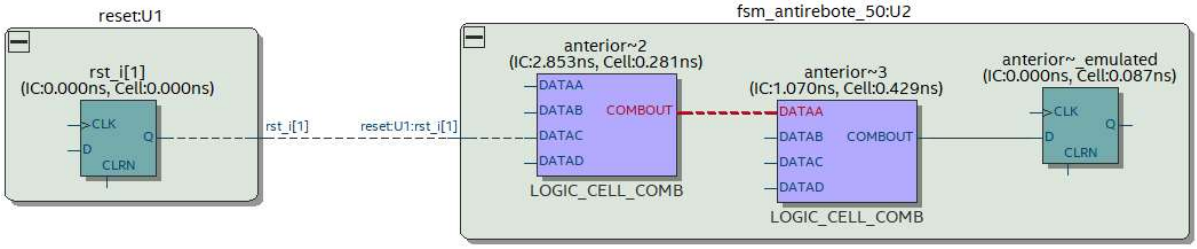

Figura 2. 2. 25 - Technology View of the Data Path del Sistema Digital de Transmisión de Datos

Como se puede apreciar, el camino critico está formado por la transmisión de información entre el componente sincronizador del reset y el módulo de antirebote de las entradas de selección de la frecuencia. Tener en cuenta que las entradas fueron asignadas a los switches 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.

3. Conclusiones

El presente informe permitió poner en práctica nuevamente los conocimientos adquiridos en materias anteriores, como así también los obtenidos durante este curso. Siendo más específicos, el tema principal fue el diseño e implementación de Maquinas de Estados Finitos. 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 tanto instrucciones concurrentes como 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. Contador Temporizador (Selección de Frecuencias)

```
--Descripcion    : Contador simple utilizado para contar durante 20ms
--               dependiendo de las frecuencias y entrada de seleccion
--               que ingresan. Salida de un bit que indica que termino
--               la temporizacion.
----------------------------------------------------------
--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_anti is
    generic( cont_ancho: natural := 12);
    port(  enable, res, clk: in std_logic;
                sel: in std_logic_vector(1 downto 0);
                act: out std_logic);
end cont_anti;
----------------------------------------------------------
-- Arquitectura
----------------------------------------------------------
architecture beh of cont_anti is
------- Declaracion de señales internas --------------------
    signal cont: unsigned(cont_ancho-1 downto 0);
    signal sal_int: std_logic_vector (cont_ancho-1 downto 0);
    signal act_i: std_logic;
begin
    -------- Proceso del Contador Simple -------------
    cont_proc: process(clk, res, enable)
    begin
          if(res = '1') then
                cont <= (others => '0');
          elsif(rising_edge(clk)) then
                if(enable = '1') then
                      cont <= cont + 1;
                end if;
          end if;
          sal_int <= std_logic_vector(cont);
    end process cont_proc;
    -------- Proceso que elige el maximo valor -------
    -------- del contador para llegar a 20ms ---------
    -------- segun la entrada de seleccion -----------
    sel_proc: process(sel, sal_int)
          constant max_cont1: std_logic_vector(cont_ancho-1 downto 0) :=
"000001100000"; --4800 baudios
          constant max_cont2: std_logic_vector(cont_ancho-1 downto 0) :=
"000011000000"; --9600 baudios
          constant max_cont3: std_logic_vector(cont_ancho-1 downto 0) :=
"001100000000"; --38400 baudios
          constant max_cont4: std_logic_vector(cont_ancho-1 downto 0) :=
"100100000000"; --115200 baudios
```

```
    begin
          act_i <= '0';
          case sel is
          ------------ 4800 baudios --------------
                when "00" =>
                      if(sal_int >= max_cont1) then
                            act_i <= '1';
                      end if;
          ------------ 38400 baudios --------------
                when "10" =>
                      if(sal_int >= max_cont3) then
                            act_i <= '1';
                      end if;
          ------------ 115200 baudios --------------
                when "11" =>
                if(sal_int >= max_cont4) then
                            act_i <= '1';
                      end if;
          ------------ 9600 baudios --------------
                when others =>
                      if(sal_int >= max_cont2) then
                            act_i <= '1';
                      end if;
          end case;
    end process sel_proc;
---------- Asignacion del valor de salida ------------
act <= act_i;
end beh;
```

4.1.2. FSM Antirebote (Selección de Frecuencias)

```
--Descripcion    : Circuito antirebote descripto mediante una maquina de
--               estados finito. La misma elimina los rebotes indeseados
--               ocacionados por el accionamiento de las llaves mecanicas.
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
-------------------------------------------------------------
-- Declaracion de Entidad
-------------------------------------------------------------
entity fsm_antirebote is
    port(  rst, clk, ent: in std_logic;
                sel: in std_logic_vector(1 downto 0);
                sal: out std_logic);
end fsm_antirebote;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture beh of fsm_antirebote is
------- Declaracion del tipo nuevo para --------------------
------- los diferentes estados -----------------------------
    type estados is (espera, cuenta, activo);
    signal siguiente_estado, estado_presente: estados;
------- Declaracion de señales internas --------------------
    signal anterior: std_logic;
    signal enable_cont, res_cont: std_logic;
```

```
    signal act_cont: std_logic;
begin
------ Instanciacion del contador simple externo ------------
------ que cuenta durante 20ms segun la frecuencia ----------
    U: entity work.cont_anti port map(clk => clk, res => res_cont, enable
=> enable_cont, sel => sel, act => act_cont);
-------------------------------------------------------------
-- Proceso de Estado Presente (Logica Secuencial)
-------------------------------------------------------------
    est_pr: process (clk, rst)
    begin
          if(rst = '0') then
                estado_presente <= espera;
          elsif(rising_edge(clk)) then
                estado_presente <= siguiente_estado;
          end if;
    end process est_pr;
-------------------------------------------------------------
-- Proceso de Proximo Estado (Logica Combinacional)
-------------------------------------------------------------
    prox_est: process (ent, estado_presente, sel, rst, anterior, act_cont)
    begin
          enable_cont <= '0';
          res_cont <= '0';
          siguiente_estado <= estado_presente;
          ------- Actualizacion del reset interno hacia -----------
          ------- el contador externo ----------------------------
          if(rst = '0') then
                res_cont <= '1';
          end if;
          ------- Eleccion del proximo estado dependiente ---------
          ------- del valor de las entradas -----------------------
          case estado_presente is
                when espera =>
                      if(ent = not(anterior)) then
                            siguiente_estado <= cuenta;
                      end if;
                when cuenta =>
                      enable_cont <= '1';
                      if(act_cont = '1') then
                            res_cont <= '1';
                            if(ent = anterior) then
                                  siguiente_estado <= espera;
                            else
                                  siguiente_estado <= activo;
                            end if;
                      end if;
                when others =>
                      siguiente_estado <= espera;
          end case;
    end process prox_est;
-------------------------------------------------------------
-- Proceso de Salida (Logica Secuencial)
-------------------------------------------------------------
    log_sal: process (clk, rst, ent)
    begin
          if(rst = '0') then
                sal <= '0';
                anterior <= ent;
```

```
        elsif(rising_edge(clk)) then
              case estado_presente is
                    when espera =>
                          sal <= anterior;
                    when cuenta =>
                          sal <= anterior;
                    when others =>
                          anterior <= ent;
                          sal <= ent;
              end case;
        end if;
    end process log_sal;
end beh;
```

4.1.3. Contador Temporizador (50MHz)

```
--Descripcion    : Contador simple utilizado para contar durante 20ms
--               para una frecuencia de 50MHz. Salida de un bit
--               que indica que termino la temporizacion.
-------------------------------------------------------------
--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_anti_50 is
    generic( cont_ancho: natural := 20);
    port(  enable, res, clk: in std_logic;
                act: out std_logic);
end cont_anti_50;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture beh of cont_anti_50 is
------- Declaracion de señales internas --------------------
    signal cont: unsigned(cont_ancho-1 downto 0);
    signal sal_int: std_logic_vector (cont_ancho-1 downto 0);
    signal act_i: std_logic;
begin
    -------- Proceso del Contador Simple -------------
    cont_proc: process(clk, res, enable)
          constant max_cont: std_logic_vector(cont_ancho-1 downto 0) :=
"11110100001001000000"; --20ms para un reloj de 50MHz
    begin
          if(res = '1') then
                cont <= (others => '0');
          elsif(rising_edge(clk)) then
                act_i <= '0';
                if(enable = '1') then
                      cont <= cont + 1;
                      if(std_logic_vector(cont) >= max_cont) then
                            act_i <= '1';
                      end if;
                end if;
          end if;
```

```
    end process cont_proc;
act <= act_i;
end beh;
```

4.1.4. FSM Antirebote (50MHz)

```
--Descripcion    : Circuito antirebote descripto mediante una maquina de
--               estados finito. La misma elimina los rebotes indeseados
--               ocacionados por el accionamiento de las llaves mecanicas.
------------------------------------------------------------
--Declaracion de Librerias
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
------------------------------------------------------------
-- Declaracion de Entidad
------------------------------------------------------------
entity fsm_antirebote_50 is
    port(  rst, clk_50, ent: in std_logic;
                sal: out std_logic);
end fsm_antirebote_50;
------------------------------------------------------------
-- Arquitectura
------------------------------------------------------------
architecture beh of fsm_antirebote_50 is
------- Declaracion del tipo nuevo para --------------------
------- los diferentes estados -----------------------------
    type estados is (espera, cuenta, activo);
    signal siguiente_estado, estado_presente: estados;
------- Declaracion de señales internas --------------------
    signal anterior: std_logic;
    signal enable_cont, res_cont: std_logic;
    signal act_cont: std_logic;
begin
------ Instanciacion del contador simple externo ------------
------ que cuenta durante 20ms segun la frecuencia ----------
    U: entity work.cont_anti_50 port map(clk => clk_50, res => res_cont,
enable => enable_cont, act => act_cont);
------------------------------------------------------------
-- Proceso de Estado Presente (Logica Secuencial)
------------------------------------------------------------
    est_pr: process (clk_50, rst)
    begin
          if(rst = '0') then
                estado_presente <= espera;
          elsif(rising_edge(clk_50)) then
                estado_presente <= siguiente_estado;
          end if;
    end process est_pr;
------------------------------------------------------------
-- Proceso de Proximo Estado (Logica Combinacional)
------------------------------------------------------------
    prox_est: process (ent, estado_presente, rst, anterior, act_cont)
    begin
          enable_cont <= '0';
          res_cont <= '0';
          siguiente_estado <= estado_presente;
          ------- Actualizacion del reset interno hacia -----------
          ------- el contador externo ----------------------------
```

```
            if(rst = '0') then
                  res_cont <= '1';
            end if;
            ------- Eleccion del proximo estado dependiente ---------
            ------- del valor de las entradas -----------------------
            case estado_presente is
                  when espera =>
                        if(ent = not(anterior)) then
                              siguiente_estado <= cuenta;
                        end if;
                  when cuenta =>
                        enable_cont <= '1';
                        if(act_cont = '1') then
                              res_cont <= '1';
                              if(ent = anterior) then
                                    siguiente_estado <= espera;
                              else
                                    siguiente_estado <= activo;
                              end if;
                        end if;
                  when others =>
                        siguiente_estado <= espera;
            end case;
      end process prox_est;
-------------------------------------------------------------
-- Proceso de Salida (Logica Secuencial)
-------------------------------------------------------------
      log_sal: process (clk_50, rst, ent)
      begin
            if(rst = '0') then
                  sal <= '0';
                  anterior <= ent;
            elsif(rising_edge(clk_50)) then
                  case estado_presente is
                        when espera =>
                              sal <= anterior;
                        when cuenta =>
                              sal <= anterior;
                        when others =>
                              anterior <= ent;
                              sal <= ent;
                  end case;
            end if;
      end process log_sal;
end beh;
```

4.1.5. Divisor de Frecuencia

```
--Descripcion    : Divisor de frecuencia con entrada de seleccion para
--               elegir entre las frecuencias 4.8KHz, 9.6KHz, 38.4KHz y
--        115.2KHz. El mismo se implemento utilizando un contador 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 div_frec is
------ Generic que permite seleccionar el largo ------------
------ del contador para obtener la frecuencia -------------
------ deseada ---------------------------------------------
    generic( ancho_cont: natural := 13);
    port( sel: in std_logic_vector(1 downto 0);
                clk_50: in std_logic;
                reset: in std_logic;
                clk_sel: out std_logic);
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);
    signal rst: std_logic := '0';
begin
------ Proceso del contador simple -----------
    cont_proc: process(clk_50, reset)
    begin
          if(reset = '0') then
                cont_int <= (others => '0');
          elsif(rst = '1') then
                cont_int <= (others => '0');
          elsif(rising_edge(clk_50)) then
                cont_int <= cont_int + 1;
          end if;
          sal_int <= std_logic_vector(cont_int);
    end process cont_proc;
------ Proceso que entrega los clocks de 4800Hz ------------
------ 9600Hz 38400Hz 115200Hz -----------------------------
    clks_proc : process(sel, sal_int, clk_sel_i, clk_sel_i2, clk_50)
          constant frec_4800: std_logic_vector(ancho_cont-1 downto 0) :=
"1010001011000";
          constant frec_9600: std_logic_vector(ancho_cont-1 downto 0) :=
"0101000101100";
          constant frec_38400: std_logic_vector(ancho_cont-1 downto 0) :=
"0001010001011";
          constant frec_115200: std_logic_vector(ancho_cont-1 downto 0) :=
"0000011011001";
    begin
    if(rising_edge(clk_50)) then
          rst <= '0';
    --- Seleccion de la frecuencia del clock de salida ------
          case sel is
          ------------------ 4800 baudios ----------------------
          when "00" =>
                if(sal_int >= frec_4800) then
                      clk_sel_i2 <= not(clk_sel_i);
                      clk_sel_i <= clk_sel_i2;
                      rst <= '1';
                end if;
          ------------------ 9600 baudios ----------------------
```

```
            when "01" =>
                  if(sal_int >= frec_9600) then
                        clk_sel_i2 <= not(clk_sel_i);
                        clk_sel_i <= clk_sel_i2;
                        rst <= '1';
                  end if;
            ------------------ 38400 baudios ---------------------
            when "10" =>
                  if(sal_int >= frec_38400) then
                        clk_sel_i2 <= not(clk_sel_i);
                        clk_sel_i <= clk_sel_i2;
                        rst <= '1';
                  end if;
            ----------------- 115200 baudios ----------------------
            when others =>
                  if(sal_int >= frec_115200) then
                        clk_sel_i2 <= not(clk_sel_i);
                        clk_sel_i <= clk_sel_i2;
                        rst <= '1';
                  end if;
            end case;
      end if;
      end process clks_proc;
------ Asignacion del valor de salida -------------------
      clk_sel <= clk_sel_i;
end beh;
```

4.1.6. Contador de Direcciones

```
--Descripcion    : Contador simple que entrega las direcciones de los datos
--               almacenados en memorias ROM e indica cuando se han enviado
--               todos los datos (direcciones).
-------------------------------------------------------------
--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_dir is
    -------- Generic que define las direcciones -------------
    -------- iniciales y finales de cada uno de -------------
    -------- los mensajes -----------------------------------
    generic( ancho_cont: natural := 7;
                        ini_msj1: std_logic_vector := "0000000";
                        fin_msj1: std_logic_vector := "0101110";
                        ini_msj2: std_logic_vector := "0110000";
                        fin_msj2: std_logic_vector := "1001011";
                        ini_msj3: std_logic_vector := "0000000";
                        fin_msj3: std_logic_vector := "0001111";
                        ini_msj4: std_logic_vector := "0001111";
                        fin_msj4: std_logic_vector := "0101110");
    port( enable, res, clk: in std_logic;
                sel: in std_logic_vector(1 downto 0);
```

```
                dir: out std_logic_vector(ancho_cont-1 downto 0);
                fin_datos: out std_logic);
end cont_dir;
------------------------------------------------------------
-- Arquitectura
------------------------------------------------------------
architecture beh of cont_dir is
------- Declaracion de las señales internas ----------------
    signal primer_dato: std_logic;
begin
    -------- Proceso del Contador Simple --------------------
    cont_proc: process(clk, res, enable)
          variable ini, fin, max_cont: std_logic_vector(ancho_cont-1
downto 0) := "1111111";
          variable cont: unsigned(ancho_cont-1 downto 0);
          constant ref: std_logic_vector(ancho_cont-1 downto 0) :=
"1111111";
    begin
          if(res = '0') then
                cont := unsigned(ini);
                primer_dato <= '0';
          elsif(rising_edge(clk)) then
                fin_datos <= '0';
                --------- Seleccion del inicio y final del --------
                --------- mensaje a transmitir --------------------
                case sel is
                      when "00" =>
                            ini := ini_msj1;
                            fin := fin_msj1;
                      when "01" =>
                            ini := ini_msj2;
                            fin := fin_msj2;
                      when "10" =>
                            ini := ini_msj3;
                            fin := fin_msj3;
                      when others =>
                            ini := ini_msj4;
                            fin := fin_msj4;
                end case;
                if(enable = '1') then
                ---------- Primer enable establece el inicio ---------
                ---------- del mensaje de la salida ------------------
                      if(primer_dato = '0') then
                            cont:= unsigned(ini);
                            max_cont := fin;
                            primer_dato <= '1';
                      else
                            cont := cont + 1;
                            if(std_logic_vector(cont) >= max_cont) then
                                  fin_datos <= '1';
                                  primer_dato <= '0';
                                  cont := unsigned(ref);
                            end if;
                      end if;
                end if;
```

```
          end if;
-------- Asignacion del valor de salida -----------------------
          dir <= std_logic_vector(cont);
    end process cont_proc;
end beh;
```

4.1.7. Memoria ROM Inferida

```
--Descripcion    : Memoria ROM inferida desde el codigo VHDL. Su
--               distribucion es  128 x 8
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
-------------------------------------------------------------
-- Declaracion de Entidad
-------------------------------------------------------------
entity rom_inf is
    ------ Generic que define la cantidad de datos ----------
    ------ y el largo de los mismos dentro de la ------------
    ------ memoria ------------------------------------------
    generic(     largo_dato: natural := 8;
                        largo_dir: natural := 7);
    port(  clk, en_mem: in std_logic;
                dir: in std_logic_vector(largo_dir-1 downto 0);
                dato: out std_logic_vector(largo_dato-1 downto 0));
end rom_inf;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture beh of rom_inf is
    --------- Tamaño de la memoria -------------------------
    constant mem_size: natural := 2**largo_dir;
    --------- Tipo que indica el largo de los datos  --------
    type mem_tipo is array(mem_size-1 downto 0) of
std_logic_vector(largo_dato-1 downto 0);
    --------- Inicializacion de los valores dentro de -------
    --------- la memoria ROM inferida ----------------------
    constant mem_rom: mem_tipo :=
          (     0 => x"43", 1 => x"75", 2 => x"72", 3 => x"73", 4 =>
x"6f", 5 => x"20", 6 => x"56",
                7 => x"48", 8 => x"44", 9 => x"4c", 10 => x"2d", 11 =>
x"46", 12 => x"50",
                13 => x"47", 14 => x"41", 15 => x"45", 16 => x"6c", 17 =>
x"20", 18 => x"71", -- 14 => "41" termina el primer msj
                19 => x"75", 20 => x"65", 21 => x"20", 22 => x"61", 23 =>
x"62", 24 => x"61",
                25 => x"6e", 26 => x"64", 27 => x"6f", 28 => x"6e",29 =>
x"61", 30 => x"20",
                31 => x"6e", 32 => x"6f", 33 => x"20", 34 => x"74", 35 =>
x"69", 36 => x"65",
```

```
                37 => x"6e", 38 => x"65", 39 => x"20", 40 => x"70", 41 =>
x"72", 42 => x"65",
                43 => x"6d", 44 => x"69", 45 => x"6f", others => x"00");
    -------- Atributo que fuerzan a que la memoria ----------
    -------- se implemente dentro de los bloques ------------
    -------- dedicados de memoria del FPGA ------------------
    attribute romstyle: string;
    attribute romstyle of mem_rom: constant is "M9K";
begin
    -------- Proceso que describe el funcionamiento ---------
    -------- de la memoria ----------------------------------
    rom_proc: process(clk, en_mem)
    begin
          if(rising_edge(clk)) then
                if(en_mem = '1') then
                      dato <= mem_rom(to_integer(unsigned(dir)));
                end if;
          end if;
    end process rom_proc;
end beh;
```

4.1.8. Memoria ROM Instanciada

```
--Descripcion    : Memoria ROM instanciada, la cual fue generada por la
--                herramienta de QUARTUS. Su distribucion es de 128 x 8
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
-------------------------------------------------------------
-- Declaracion de Entidad
-------------------------------------------------------------
entity rom_inst_rs232 is
    ------ Generic que define la cantidad de datos ----------
    ------ y el largo de los mismos dentro de la ------------
    ------ memoria ------------------------------------------
    generic(     largo_dato: natural := 8;
                      largo_dir: natural := 7);
    port(  clock_sig, rden_sig: in std_logic;
                address_sig: in std_logic_vector(largo_dir-1 downto 0);
                q_sig: out std_logic_vector(largo_dato-1 downto 0));
end rom_inst_rs232;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture beh of rom_inst_rs232 is
    -------- Declaracion de la memoria ROM ------------------
    component rom_inst_rs_232
          PORT(  address            : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
                      clock         : IN STD_LOGIC  := '1';
                      rden          : IN STD_LOGIC  := '1';
                      q             : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
    end component;
```

```
begin
    -------- Instanciacion de la memoria ROM ----------------
    rom_inst_rs_232_inst : rom_inst_rs_232 PORT MAP (
          address       => address_sig,
          clock   => clock_sig,
          rden    => rden_sig,
          q       => q_sig);
end beh;
```

4.1.9. FSM de Control

```
--Descripcion    : Maquina de Estados Finitos encargada del control del
--  sistema de transmision de mensajes almacenados dentro de memorias ROM.
-------------------------------------------------------------
--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 fsm_control is
    generic(     largo_dir: natural := 7);
    port(  reset, clk, inicio, fin_cont: in std_logic;
                 tx_activa: in std_logic;
                 sel_msj: in std_logic_vector(1 downto 0);
                 en_cont, en_rom_inf, en_rom_inst: out std_logic;
                 sel_rom, ini_tx: out std_logic;
                 bcd_msj: out std_logic_vector(3 downto 0));
end fsm_control;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture beh of fsm_control is
------- Declaracion del tipo nuevo para --------------------
------- los diferentes estados -----------------------------
    type estados is (reposo, msj1, msj2, msj3, msj4, transmision);
    signal siguiente_estado, estado_presente: estados;
begin
-------------------------------------------------------------
-- Proceso de Estado Presente (Logica Secuencial)
-------------------------------------------------------------
    est_pr: process (clk, reset, siguiente_estado)
    begin
          if(reset = '0') then
                estado_presente <= reposo;
          elsif(rising_edge(clk)) then
                estado_presente <= siguiente_estado;
          end if;
    end process est_pr;
-------------------------------------------------------------
-- Proceso de Proximo Estado (Logica Combinacional)
```

```
-------------------------------------------------------------
    prox_est: process(inicio, estado_presente, sel_msj, fin_cont,
tx_activa)
    begin
          siguiente_estado <= estado_presente;
          ------- Eleccion del proximo estado dependiente ---------
          ------- del valor de las entradas -----------------------
          case estado_presente is
                when reposo =>
                      if(inicio = '0') then
                            siguiente_estado <= reposo;
                      else
                      ------ Seleccion de los estados mensajes ----------
                            case sel_msj is
                                  when "00" =>
                                        siguiente_estado <= msj1;
                                  when "01" =>
                                        siguiente_estado <= msj2;
                                  when "10" =>
                                        siguiente_estado <= msj3;
                                  when others =>
                                        siguiente_estado <= msj4;
                            end case;
                      end if;
                ------ Logica de los estados mensajes -----------------
                when msj1 =>
                            if(tx_activa = '0') then
                                  siguiente_estado <= transmision;
                            end if;
                when msj2 =>
                            if(tx_activa = '0') then
                                  siguiente_estado <= transmision;
                            end if;
                when msj3 =>
                            if(tx_activa = '0') then
                                  siguiente_estado <= transmision;
                            end if;
                when msj4 =>
                            if(tx_activa = '0') then
                                  siguiente_estado <= transmision;
                            end if;
                ------ Transmision de los mensajes --------------------
                when others =>
                      if(fin_cont = '0') then
                      if(tx_activa = '1') then
                      case sel_msj is
                            when "00" =>
                                  siguiente_estado <= msj1;
                            when "01" =>
                                  siguiente_estado <= msj2;
                            when "10" =>
                                  siguiente_estado <= msj3;
                            when others =>
                                  siguiente_estado <= msj4;
                      end case;
```

```
                        end if;
                        else
                              siguiente_estado <= reposo;
                        end if;
            end case;
    end process prox_est;
-------------------------------------------------------------
-- Proceso de Salida (Logica Secuencial)
-------------------------------------------------------------
    log_sal: process (clk, reset, tx_activa, sel_msj, estado_presente)
    begin
            if(reset = '0') then
                  en_cont <= '0';
                  en_rom_inf <= '0';
                  en_rom_inst <= '0';
                  ini_tx <= '0';
                  sel_rom <= '0';
                  bcd_msj <= "1111";
            elsif(rising_edge(clk)) then
                  en_cont <= '0';
                  en_rom_inf <= '0';
                  en_rom_inst <= '0';
                  ini_tx <= '0';
                  sel_rom <= '0';
                  bcd_msj <= "1111";
                  case estado_presente is
                        when reposo =>
                              null;
                        ------ salidas de los estados mensajes ------------
                        when msj1 =>
                              bcd_msj <= "0001";
                              en_rom_inst <= '1';
                              sel_rom <= '1';
                              if(tx_activa = '0') then
                                    en_cont <= '1';
                              end if;
                        when msj2 =>
                              bcd_msj <= "0010";
                              en_rom_inst <= '1';
                              sel_rom <= '1';
                              if(tx_activa = '0') then
                                    en_cont <= '1';
                              end if;
                        when msj3 =>
                              bcd_msj <= "0011";
                              en_rom_inf <= '1';
                              if(tx_activa = '0') then
                                    en_cont <= '1';
                              end if;
                        when msj4 =>
                              bcd_msj <= "0100";
                              en_rom_inf <= '1';
                              if(tx_activa = '0') then
                                    en_cont <= '1';
                              end if;
```

```
                        ------ Salidas de la transmision ------------------
                        when others =>
                              ini_tx <= '1';
                              case sel_msj is
                                    when "00" =>
                                          bcd_msj <= "0001";
                                          en_rom_inst <= '1';
                                          sel_rom <= '1';
                                    when "01" =>
                                          bcd_msj <= "0010";
                                          en_rom_inst <= '1';
                                          sel_rom <= '1';
                                    when "10" =>
                                          bcd_msj <= "0011";
                                          en_rom_inf <= '1';
                                    when others =>
                                          bcd_msj <= "0100";
                                          en_rom_inf <= '1';
                              end case;
                  end case;
            end if;
      end process log_sal;
end beh;
```

4.1.10. FSM de Transmisión RS-232

```
--Descripcion    : Maquina de Estados Finitos encargada del control de la
--  transmision de datos usando el protocolo de comunicacion serie RS-232.
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
use work.pack_232.all;
-------------------------------------------------------------
-- Declaracion de Entidad
-------------------------------------------------------------
entity fsm_rs_232 is
    generic( largo_dato: natural := 8);
    port(  reset, clk, ini_tx: in std_logic;
                datos_in: in std_logic_vector(largo_dato-1 downto 0);
                tx_serie, tx_activa: out std_logic);
end fsm_rs_232;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture beh of fsm_rs_232 is
------- Declaracion del tipo nuevo para --------------------
------- los diferentes estados -----------------------------
    type estados is (reposo, start, desplazamiento, paridad, stop);
    signal siguiente_estado, estado_presente: estados;
------- Declaracion de las señales internas ----------------
```

```
    signal fin_shift: std_logic;
begin
-------------------------------------------------------------
-- Proceso de Estado Presente (Logica Secuencial)
-------------------------------------------------------------
    est_pr: process (clk, reset)
    begin
          if(reset = '0') then
                estado_presente <= reposo;
          elsif(rising_edge(clk)) then
                estado_presente <= siguiente_estado;
          end if;
    end process est_pr;
-------------------------------------------------------------
-- Proceso de Proximo Estado (Logica Combinacional)
-------------------------------------------------------------
    prox_est: process (ini_tx, estado_presente, fin_shift)
    begin
          siguiente_estado <= estado_presente;
          ------- Eleccion del proximo estado dependiente ---------
          ------- del valor de las entradas -----------------------
          case estado_presente is
                when reposo =>
                      if(ini_tx = '1') then
                            siguiente_estado <= start;
                      end if;
                when start =>
                      siguiente_estado <= desplazamiento;
                when desplazamiento =>
                      if(fin_shift = '1') then
                            siguiente_estado <= paridad;
                      end if;
                when paridad =>
                      siguiente_estado <= stop;
                when others =>
                      siguiente_estado <= reposo;
          end case;
    end process prox_est;
-------------------------------------------------------------
-- Proceso de Salida (Logica Secuencial)
-------------------------------------------------------------
    log_sal: process (clk, reset, datos_in)
          variable cont_shift: unsigned(2 downto 0) := (others => '0');
          constant max_datos: std_logic_vector(2 downto 0) := "111";
          variable datos_buf: std_logic_vector(largo_dato-1 downto 0);
    begin
          if(reset = '0') then
                tx_serie <= '1';
                tx_activa <= '0';
                fin_shift <= '0';
                cont_shift := (others => '0');
          elsif(rising_edge(clk)) then
                fin_shift <= '0';
                tx_serie <= '1';
                tx_activa <= '0';
```

```
                case estado_presente is
                        when reposo =>
                                datos_buf := datos_in;
                        ------ Transmision del bit de start ---------------
                        when start =>
                                tx_serie <= '0';
                                tx_activa <= '1';
                        ------ Transmision serie del dato ingresado -------
                        when desplazamiento =>
                                tx_activa <= '1';
                                tx_serie <=
datos_buf(to_integer(cont_shift));
                                cont_shift := cont_shift + 1;
                                if(std_logic_vector(cont_shift) >= max_datos)
then
                                        fin_shift <= '1';
                                end if;
                        ------ Transmision serie del bit de paridad -------
                        when paridad =>
                                tx_activa <= '1';
                                tx_serie <= paridad(datos_buf);
                        ------ Transmision serie del bit de stop ----------
                        when others =>
                                tx_activa <= '1';
                end case;
        end if;
    end process log_sal;
end beh;
```

4.1.11. Paquete Bit de Paridad

```
--Descripcion    : Paquete que contiene la funcion que permite calcular
--               el bit de paridad par de un cierto vector
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
-------------------------------------------------------------
-- Declaracion del Encabezado del Paquete
-------------------------------------------------------------
package pack_232 is
    function paridad(d_in: std_logic_vector) return std_logic;
end pack_232;
-------------------------------------------------------------
-- Declaracion del Cuerpo del Paquete
-------------------------------------------------------------
package body pack_232 is
    ------- Funcion que calcula el bit de paridad par -------
    function paridad(d_in: std_logic_vector) return std_logic is
        variable bit_paridad: std_logic;
    begin
        bit_paridad := '0';
        for i in 0 to d_in'length-1 loop
```

```
                bit_paridad := bit_paridad xor d_in(i);
            end loop;
            return bit_paridad;
    end paridad;
end;
```

4.1.12. Multiplexores de 2 a 1

```
--Descripcion    : Conjunto variable de Multiplexores 2 a 1
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
-------------------------------------------------------------
-- Declaracion de Entidad
-------------------------------------------------------------
entity mux_rom is
    ------ Generic que define la cantidad de ----------------
    ------ multiplexers a generar -------------------------
    generic(     largo_dato: natural := 8);
    port(ent_1, ent_2: in std_logic_vector(largo_dato-1 downto 0);
             sel: in std_logic;
             sal_mux: out std_logic_vector(largo_dato-1 downto 0));
end mux_rom;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture behav of mux_rom is
begin
    ------ Instruccion Concurrente que describe el ----------
    ------ comprtamiento de los multiplexers ----------------
    with sel select
           sal_mux <= ent_1 when '0',             --Primera ROM
                              ent_2 when others;      --Segunda ROM
end behav;
```

4.1.13. Sistema Digital de Transmisión de Datos (RS-232)

```
--Descripcion: Sistema de alto nivel que se encarga de interconectar los
--               componentes y hacer posible la transmision de los mensajes
--               almacenados dentro de las memorias ROM, a traves de un
--               transmisor serie RS-232
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.pack_232.all;
-------------------------------------------------------------
-- Declaracion de Entidad
-------------------------------------------------------------
entity sist_serie_rs232 is
    port(  clk_50, reset_asin, inicio: in std_logic;
```

```
                sel_msj, sel_frec: in std_logic_vector(1 downto 0);
                sal_serie, transm_act: out std_logic;
                num_msj: out std_logic_vector(6 downto 0));
end sist_serie_rs232;
------------------------------------------------------------
-- Arquitectura
------------------------------------------------------------
architecture beh of sist_serie_rs232 is
------- Declaracion de las señales internas ----------------
    signal clk: std_logic;
    signal reset_sin_50, reset_sin: std_logic;
    signal sel_frec_ant, sel_msj_ant: std_logic_vector(1 downto 0);
    signal inicio_sinc: std_logic;
    signal en_cont_dir, en_rom_inf, en_rom_inst, sel_rom_dir, ini_transm:
std_logic;
    signal cont_fin, transm_activa: std_logic;
    signal direccion: std_logic_vector(6 downto 0);
    signal dato_inf, dato_inst, dato: std_logic_vector(7 downto 0);
    signal bin_msj: std_logic_vector(3 downto 0);
begin
------- Instanciacion de los componentes necesarios --------
    ---- Componentes de control de la frecuencia de trabajo -
    U1: entity work.reset port map(clk => clk_50, rst => reset_asin,
rst_a_s => reset_sin_50);
    U2: entity work.fsm_antirebote_50 port map(clk_50 => clk_50, rst =>
reset_sin_50, ent => sel_frec(0), sal => sel_frec_ant(0));
    U3: entity work.fsm_antirebote_50 port map(clk_50 => clk_50, rst =>
reset_sin_50, ent => sel_frec(1), sal => sel_frec_ant(1));
    U4: entity work.div_frec port map(clk_50 => clk_50, sel =>
sel_frec_ant, clk_sel => clk, reset => reset_sin_50);
    ---- Componentes de entrada a la FSM principal ----------
    U5: entity work.reset port map(clk => clk, rst => reset_asin, rst_a_s
=> reset_sin);
    U6: entity work.fsm_antirebote port map(clk => clk, sel =>
sel_frec_ant, rst => reset_sin, ent => sel_msj(0), sal => sel_msj_ant(0));
    U7: entity work.fsm_antirebote port map(clk => clk, sel =>
sel_frec_ant, rst => reset_sin, ent => sel_msj(1), sal => sel_msj_ant(1));
    U8: entity work.Sincro port map(CLK => clk, RST => reset_sin, D =>
inicio, Q => inicio_sinc);
    ---- Componentes de Control de la eleccion de los -------
    ---- mensajes e inicio de la transmision ----------------
    U9: entity work.fsm_control port map(clk => clk, sel_msj =>
sel_msj_ant, reset => reset_sin,
                inicio => inicio_sinc, fin_cont => cont_fin, en_cont =>
en_cont_dir, en_rom_inf => en_rom_inf,
                en_rom_inst => en_rom_inst, sel_rom => sel_rom_dir, ini_tx
=> ini_transm,
                bcd_msj => bin_msj, tx_activa => transm_activa);
    U10: entity work.cont_dir port map(clk => clk, res => reset_sin,
enable => en_cont_dir, sel => sel_msj_ant, fin_datos => cont_fin, dir =>
direccion);
    U11: entity work.decoBCD7seg port map(ent => bin_msj,leds => num_msj);
    ---- Componentes encargados de entregar los mensajes ----
    ---- a transmitir ----------------------------------------
```

```
    U12: entity work.rom_inf port map(clk => clk, en_mem => en_rom_inf,
dir => direccion, dato => dato_inf);
    U13: entity work.rom_inst_rs_232 port map(clock => clk, rden =>
en_rom_inst, address => direccion, q => dato_inst);
    U14: entity work.mux_rom port map(sel => sel_rom_dir, ent_1 =>
dato_inf, ent_2 => dato_inst, sal_mux => dato);
    ---- Componente encargado de la transmision serie -------
    ---- usando el protocolo RS-232 -------------------------
    U15: entity work.fsm_rs_232 port map(clk => clk, datos_in => dato,
reset => reset_sin,
          ini_tx => ini_transm, tx_serie => sal_serie, tx_activa =>
transm_activa);
    transm_act <= transm_activa;
end beh;
```

4.2. Descripciones Test Bench

4.2.1. Contador Temporizador (Selección de Frecuencias)

```
-- Descripcion: Simulacion funcional del Contador temporizador 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
----------------------------------------------------------
entity testbench_cont_anti is
end testbench_cont_anti;
----------------------------------------------------------
-- Arquitectura
----------------------------------------------------------
architecture tb_behave of testbench_cont_anti is
------- Declaracion de señales internas -------------------
    signal test_res: std_logic := '0';
    signal test_enable: std_logic := '0';
    signal test_clk: std_logic := '0';
    signal test_sel: std_logic_vector(1 downto 0);
    signal test_act: std_logic := '0';
begin
------ Instanciacion del Contador temporizador ----------------------
    U: entity work.cont_anti port map(clk => test_clk, res => test_res,
enable => test_enable, sel =>  test_sel, act => test_act);
------ Generacion de Señales de simulacion -----------------
    test_clk <= not test_CLK after 4.34us;
    test_res <= '1', '0' after 100us; --'1' after 35000us, '0' after
35100us;
    test_enable <= '0', '1' after 1000us;
    test_sel <= "11";
end tb_behave;
```

4.2.2. FSM Antirebote (Selección de Frecuencias)

```
-- Descripcion: Simulacion funcional de la maquina de estado del
--                      circuito antirebote 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_fsm_antirebote is
end testbench_fsm_antirebote;
----------------------------------------------------------
-- Arquitectura
----------------------------------------------------------
architecture beh of testbench_fsm_antirebote is
------- Declaracion de señales internas --------------------
    signal test_sel: std_logic_vector(1 downto 0);
    signal test_rst: std_logic := '1';
    signal test_ent: std_logic := '0';
    signal test_clk: std_logic := '0';
    signal test_sal: std_logic;
begin
------ Instanciacion del circuito antirebote ---------------
    U: entity work.fsm_antirebote port map(clk => test_clk, sel =>
test_sel, rst => test_rst, ent => test_ent, sal => test_sal);
------ Generacion de Señales de simulacion -----------------
    test_clk <= not test_clk after 52us;
    test_sel <= "01";
    test_rst <= '0', '1' after 200us;
    ----------- Proceso que simula los rebotes --------------
    ----------- en la entrada -------------------------------
    process
    begin
          wait for 500us;
          for i in 0 to 100 loop
                test_ent <= not(test_ent);
                wait for 200us;
          end loop;
          wait for 1us;
          test_ent <= '0';
          wait for 10000us;
          test_ent <= '1';
          for i in 0 to 100 loop
                test_ent <= not(test_ent);
                wait for 200us;
          end loop;
          wait for 1us;
          test_ent <= '1';
          wait;
    end process;
```

```
end beh;
```

4.2.3. FSM Antirebote (50MHz)

```
-- Descripcion: Simulacion funcional de la maquina de estado del
--                      circuito antirebote 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_fsm_antirebote_50 is
end testbench_fsm_antirebote_50;
----------------------------------------------------------
-- Arquitectura
----------------------------------------------------------
architecture beh of testbench_fsm_antirebote_50 is
------- Declaracion de señales internas --------------------
    signal test_rst: std_logic := '1';
    signal test_ent: std_logic := '0';
    signal test_clk: std_logic := '0';
    signal test_sal: std_logic;
begin
------ Instanciacion del circuito antirebote ---------------
    U: entity work.fsm_antirebote_50 port map(clk_50 => test_clk, rst =>
test_rst, ent => test_ent, sal => test_sal);
------ Generacion de Señales de simulacion -----------------
    test_clk <= not test_clk after 10ns;
    test_rst <= '0', '1' after 200us;
    ----------- Proceso que simula los rebotes --------------
    ----------- en la entrada -------------------------------
    process
    begin
          wait for 500us;
          for i in 0 to 95 loop
                test_ent <= not(test_ent);
                wait for 200us;
          end loop;
          wait for 1us;
          test_ent <= '1';
          wait for 10000us;
          test_ent <= '0';
          for i in 0 to 95 loop
                test_ent <= not(test_ent);
                wait for 200us;
          end loop;
          wait for 1us;
          test_ent <= '0';
          wait;
    end process;
```

```
end beh;
```

4.2.4. Divisor de Frecuencia

```
-- 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 testbench_div_frec;
----------------------------------------------------------
-- Arquitectura
----------------------------------------------------------
architecture beh of testbench_div_frec is
------- Declaracion de señales internas --------------------
    signal test_sel: std_logic_vector(1 downto 0);
    signal test_reset: std_logic;
    signal test_clk_50: std_logic := '0';
    signal test_clk_sel: std_logic;
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, reset => test_reset);
------ Generacion de Señales de simulacion -----------------
    test_clk_50 <= not test_clk_50 after 10ns;
    test_sel <= "00","01" after 500us, "10" after 1000us, "11" after
100us;
    test_reset <= '0', '1' after 100ns;
end beh;
```

4.2.5. Contador de Direcciones

```
-- Descripcion: Simulacion funcional del Contador de direcciones 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
----------------------------------------------------------
entity testbench_cont_dir is
end testbench_cont_dir;
```

```
------------------------------------------------------------
-- Arquitectura
------------------------------------------------------------
architecture tb_behave of testbench_cont_dir is
------- Declaracion de señales internas --------------------
    signal test_res: std_logic := '0';
    signal test_enable: std_logic := '0';
    signal test_clk: std_logic := '0';
    signal test_dir: std_logic_vector (6 downto 0);
    signal test_sel: std_logic_vector (1 downto 0);
    signal test_fin_datos: std_logic;
begin
------ Instanciacion del Contador de direcciones -----------
    U: entity work.cont_dir port map(clk => test_clk, res => test_res,
enable => test_enable, sel => test_sel, fin_datos => test_fin_datos, dir
=> test_dir);
------ Generacion de Señales de simulacion -----------------
    test_clk <= not test_clk after 52us;
    test_res <= '0', '1' after 100us; --'0' after 2000us, '1' after
2100us;
    test_sel <= "00"; --"01" after 1000us, "10" after 2000us, "11" after
3000us;
    test_enable <= '0', '1' after 504us;
end tb_behave;
```

4.2.6. Memoria ROM Inferida

```
-- Descripcion: Simulacion funcional de la memoria ROM inferida 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_rom_inf is
end testbench_rom_inf;
------------------------------------------------------------
-- Arquitectura
------------------------------------------------------------
architecture beh of testbench_rom_inf is
------- Declaracion de señales internas --------------------
    signal test_en_mem: std_logic;
    signal test_clk: std_logic := '1';
    signal test_dir: std_logic_vector(6 downto 0);
    signal test_dato: std_logic_vector(7 downto 0);
begin
------ Instanciacion de la Memoria Inferida ----------------
    U: entity work.rom_inf port map(clk => test_clk, en_mem =>
test_en_mem, dir => test_dir, dato => test_dato);
------ Generacion de Señales de simulacion -----------------
```

```
    test_clk <= not test_clk after 10us;
    test_en_mem <= '0', '1' after 100us;
    test_dir <= "0000000", "0000001" after 120us, "0000010" after 140us,
"0000011" after 160us,
                              "0000100" after 180us,  "0000101" after
200us,  "0000110" after 220us,  "0000111" after 240us,
                              "0001000" after 260us, "0001001" after 280us,
"0001010" after 300us,  "0001011" after 320us,
                              "0001100" after 340us, "0001101" after 360us,
"0001110" after 380us;
end beh;
```

4.2.7. Memoria ROM Instanciada

```
-- Descripcion: Simulacion funcional de la memoria ROM instanciada por
--                      medio de un Test Bench
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
-------------------------------------------------------------
-- Declaracion de Entidad (Vacia)
-------------------------------------------------------------
entity testbench_rom_inst is
end testbench_rom_inst;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture beh of testbench_rom_inst is
------- Declaracion de señales internas --------------------
    signal test_rden_sig: std_logic;
    signal test_clock_sig: std_logic := '1';
    signal test_address_sig: std_logic_vector(6 downto 0);
    signal test_q_sig: std_logic_vector(7 downto 0);
begin
------ Instanciacion de la memoria -------------
    U: entity work.rom_inst_rs232 port map(clock_sig => test_clock_sig,
rden_sig => test_rden_sig,
          address_sig => test_address_sig, q_sig => test_q_sig);
------ Generacion de Señales de simulacion -----------------
    test_clock_sig <= not test_clock_sig after 10us;
    test_rden_sig <= '0', '1' after 100us;
    test_address_sig <= "0000000", "0000001" after 120us, "0000010" after
140us, "0000011" after 160us,
                              "0000100" after 180us,  "0000101" after
200us,  "0000110" after 220us,  "0000111" after 240us,
                              "0001000" after 260us, "0001001" after 280us,
"0001010" after 300us,  "0001011" after 320us,
                              "1000111" after 340us, "1001000" after 360us,
"1001001" after 380us;
end beh;
```

4.2.8. FSM de Control

```
-- Descripcion: Simulacion funcional de la maquina de estado de
--                      control del sistema total de transmision de
--                            mensajes por medio de un Test Bench
-------------------------------------------------------------
--Declaracion de Librerias
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
-------------------------------------------------------------
-- Declaracion de Entidad (Vacia)
-------------------------------------------------------------
entity testbench_fsm_control is
end testbench_fsm_control;
-------------------------------------------------------------
-- Arquitectura
-------------------------------------------------------------
architecture beh of testbench_fsm_control is
------- Declaracion de señales internas --------------------
    signal test_sel_msj: std_logic_vector(1 downto 0);
    signal test_reset: std_logic := '1';
    signal test_inicio: std_logic := '0';
    signal test_clk: std_logic := '0';
    signal test_fin_cont: std_logic := '0';
    signal test_tx_activa: std_logic := '0';
    signal test_en_cont, test_en_rom_inf, test_en_rom_inst: std_logic;
    signal test_sel_rom, test_ini_tx: std_logic;
    signal test_bcd_msj: std_logic_vector(3 downto 0);
begin
------ Instanciacion de la FSM Controladora de -------------
------ la transmision de mensajes --------------------------
    U: entity work.fsm_control port map(clk => test_clk, sel_msj =>
test_sel_msj, reset => test_reset,
                inicio => test_inicio, fin_cont => test_fin_cont, en_cont
=> test_en_cont, en_rom_inf => test_en_rom_inf,
                en_rom_inst => test_en_rom_inst, sel_rom => test_sel_rom,
ini_tx => test_ini_tx,
                bcd_msj => test_bcd_msj, tx_activa => test_tx_activa);
------ Generacion de Señales de simulacion -----------------
    test_clk <= not test_clk after 50us;
    test_sel_msj <= "10";
    test_inicio <= '0', '1' after 150us, '0' after 400us;
    test_reset <= '0', '1' after 50us;
    test_fin_cont <= '0', '1' after 1450us, '0' after 1650us;
    test_tx_activa <= '0', '1' after 549us, '0' after 849us, '1' after
1049us, '0' after 1349us;
end beh;
```

4.2.9. FSM de Transmisión RS-232

```
-- Descripcion: Simulacion funcional de la maquina de estado del
--               transmisor serie RS-232 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_fsm_rs_232 is
end testbench_fsm_rs_232;
----------------------------------------------------------
-- Arquitectura
----------------------------------------------------------
architecture beh of testbench_fsm_rs_232 is
------- Declaracion de señales internas --------------------
    signal test_datos_in: std_logic_vector(7 downto 0);
    signal test_reset: std_logic := '1';
    signal test_clk: std_logic := '0';
    signal test_ini_tx: std_logic := '0';
    signal test_tx_serie, test_tx_activa: std_logic;
begin
------ Instanciacion del Transmisor RS-232 -----------------
    U: entity work.fsm_rs_232 port map(clk => test_clk, datos_in =>
test_datos_in, reset => test_reset,
          ini_tx => test_ini_tx, tx_serie => test_tx_serie, tx_activa =>
test_tx_activa);
------ Generacion de Señales de simulacion -----------------
    test_clk <= not test_clk after 52us;
    test_datos_in <= "00010111", "11100110" after 1000us;
    test_reset <= '0', '1' after 200us;
    test_ini_tx <= '0', '1' after 400us, '0' after 500us, '1' after
3100us, '0' after 3300us;
end beh;
```

4.2.10. Multiplexores de 2 a 1

```
-- Descripcion: Simulacion funcional de 8 multiplexer de 2 a 1 por medio
--                      de un Test Bench
----------------------------------------------------------
--Declaracion de Librerias
----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
----------------------------------------------------------
-- Declaracion de Entidad
----------------------------------------------------------
entity testbench_mux_rom is
end testbench_mux_rom;
----------------------------------------------------------
-- Arquitectura
----------------------------------------------------------
architecture tb_behave of testbench_mux_rom is
------- Declaracion de señales internas --------------------
```

```
    signal test_sel: std_logic;
    signal test_ent_1, test_ent_2: std_logic_vector(7 downto 0);
    signal test_sal_mux: std_logic_vector(7 downto 0);
begin
------ Instanciacion del conjunto de multiplexers -------
    UUT: entity work.mux_rom port map(sel => test_sel, ent_1 =>
test_ent_1, ent_2 => test_ent_2, sal_mux => test_sal_mux);
------ Generacion de Señales de simulacion -----------------
    test_sel <=  '0', '1' after 100ps, '0' after 200ps, '1' after 300ps,
'0' after 400ps;
    test_ent_1 <= "00000000", "00010000" after 100ps, "10000000" after
200ps, "00000001" after 300ps, "01000010" after 400ps;
    test_ent_2 <= "11111111", "11110111" after 100ps, "01111111" after
200ps, "11111110" after 300ps, "11011101" after 400ps;
end tb_behave;
```

4.2.11. Sistema Digital de Transmisión de Datos (RS-232)

```
-- Descripcion: Simulacion funcional del sistema de comunicacion usando
--          el protocolo de comunicacion RS-232 por medio de
--          un Test Bench
----------------------------------------------------------
--Declaracion de Librerias
----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
----------------------------------------------------------
-- Declaracion de Entidad (Vacia)
----------------------------------------------------------
entity testbench_sist_serie is
end testbench_sist_serie;
----------------------------------------------------------
-- Arquitectura
----------------------------------------------------------
architecture beh of testbench_sist_serie is
------- Declaracion de señales internas --------------------
    signal test_sel_msj, test_sel_frec: std_logic_vector(1 downto 0);
    signal test_reset_asin: std_logic := '1';
    signal test_inicio: std_logic := '0';
    signal test_clk_50: std_logic := '0';
    signal test_sal_serie: std_logic;
    signal test_transm_act: std_logic;
    signal test_num_msj: std_logic_vector(6 downto 0);
begin
------ Instanciacion del Sistema de transmision ------------
------ serie completo -------------------------------------
    U: entity work.sist_serie_rs232 port map(clk_50 => test_clk_50,
sel_msj => test_sel_msj,
                  sel_frec => test_sel_frec, reset_asin => test_reset_asin,
inicio => test_inicio,
                  sal_serie => test_sal_serie, num_msj => test_num_msj,
transm_act => test_transm_act);
------ Generacion de Señales de simulacion -----------------
    test_clk_50 <= not test_clk_50 after 10ns;
```

```
    test_sel_frec <= "00", "11" after 100000us;
    test_sel_msj <= "01", "11" after 121000us;
    test_inicio <= '0', '1' after 400us, '0' after 800us, '1' after
142000us, '0' after 144000us;
    test_reset_asin <= '0', '1' after 50us;
end beh;
```