



DEPARTAMENTO DE ELECTRÓNICA Y AUTOMÁTICA
FACULTAD DE INGENIERÍA – UNIVERSIDAD NACIONAL DE SAN JUAN

Informe de Práctica Nº 5

Asignatura: Sistemas digitales avanzados
Ingeniería Electrónica

Autores:

Jalil, Simón – Registro 26514

Pelaez, Pablo – Registro 26506

2º Semestre

Año 2019

Indice de contenido

Tabla de ilustraciones.....	4
1. Introducción.....	5
2. Contenido.....	5
2.1 Descripción.....	5
2.2 Parte A	7
2.2.1 Registro.....	7
2.2.2 Contador del registro.....	7
2.2.3 Maquina de estado finito del registro.....	8
2.3 Parte B.....	9
2.3.1 Memoria ROM.....	9
2.3.2 Contador de memoria.....	10
2.3.3 Maquina de estado de finito de la memoria.....	10
2.4 Paquete (package).....	12
2.5 Bit de paridad.....	13
2.6 Receptor.....	13
2.7 Display LCD.....	15
2.8 Puesta a punto del proyecto.....	18
3. Conclusiones.....	22
4. Anexo.....	22
4.1Parte A.....	22
4.1.1 Registro de desplazamiento.	22
4.1.2 Contador binario.....	24
4.1.3 FSM registro de desplazamiento	25
4.1.4 Flip-flop D.....	28
4.1.5 Punto A.....	28
4.1.6 Test-bench Punto A	31
4.2 Parte B.....	33
4.2.1 Memoria ROM.....	33
4.2.2 Package memoria.....	34
4.2.3 Contador Memoria.....	38
4.2.4 FSM memoria.....	41

4.2.5 Punto B.....	44
4.2.6 Test-bench Punto BTest-bench Punto B.....	47
4.3 Receptor.....	49
4.3.1 UART RX.....	49
4.3.2 Mux Paralelos.....	52
4.3.3 Mux	53
4.3.4 RX.....	53
4.3.5 Test-bench UART RX.....	55
4.4 LCD.....	57
4.4.1 Contador binario LCD.....	57
4.4.2 FSM LCD.....	58
4.4.3 LCD main.....	68
4.4.4 Test-bench LCD.....	69
4.5 RS 232.....	71
4.5.1 RS 232 Main.....	71
4.5.2 Test-bench RS232	78

Tabla de ilustraciones

Figura N°1: RS-232 formato de transmisión.....	5
Figura N°2: Conector DB9.....	6
Figura N.º3: Interfaz DB9 a USB.....	6
Figura N°4: Esquema principal del proyecto.....	7
Figura N°5: Diagrama de flujo de maquina de estado del registro.....	8
Figura N.º6: Esquema RTL parte A.....	9
Figura N°7: Diagrama test-bench parte A.....	9
Figura N.º8: Estructura conceptual de memoria de solo lectura (ROM).....	10
Figura N.º9: Diagrama de flujo de maquina de estado de memoria.....	11
Figura N.º10: Esquema RTL parte B.....	12
Figura N.º11: Diagrama test-bench parte B.....	12
Figura N.º12: Diagrama de flujo de maquina de estado de la memoria.....	14
Figura N.º13: Diagrama temporal test-bench de receptor UART.....	14
Figura N.º14: LCD 16x02.....	16
Figura N.º15: Función de pines de display LCD.....	16
Figura N.º16: Set de instrucciones del controlador LCD (HD44780U or KS0066U).....	17
Figura 17: Diagrama de estados provisto por Quartus de la maquina de estados del controlador LCD.....	17
Figura N.º18: Diagrama test-bench del modulo LCD.....	18
Figura N.º19: Diagrama RTL RS232.....	18
Figura N.º20: Diagrama test-bench de protocolo RS232.....	19
Figura N°21: Esquema general del proyecto.....	19
Figura N.º22: Configuración de baudios. a) 4800baud, b) 9600baud, c) 38400baud d) 115200baud.....	20
Figura N°23: Frases enviadas al monitor serial.	20
Figura N°24: Envío de la primera frase y recepción del carácter t.....	21
Figura N.º25: Envío de la segunda frase y recepción del carácter g.....	21
Figura N.º26: Envío de la tercera frase y recepción del carácter j.....	21
Figura N.º27: Envío de la cuarta frase y recepción del carácter m.....	22

1 Introducción.

Objetivos

- Utilización de las instrucciones secuenciales, concurrentes y paquetes aprendidos en clase.
- Interpretación de información especificada en hojas de datos o especificaciones de diseño.
- Familiarizarse con el DE2-115 Cyclone IV development kit board. Uso de un constraint file para asignación de pines de I/O, pulsadores, switches y LEDs.
- Uso de la herramienta MegaWizard para la implementación de un DLL o PLL para generar un reloj de baja frecuencia (opcional).
- Uso de código VHDL genérico para inferir una memoria de solo lectura tipo ROM.
- Uso de la herramienta MegaWizard para la implementación de una memoria de solo lectura tipo ROM (opcional).
- Simulación a nivel de compuerta o post-place and route.
- Configuración del FPGA.
- Comunicación entre el módulo descripto e implementado en el Cyclone IV y una PC a través del puerto RS-232.

2 Contenido.

2.1 Descripción

Realizar la descripción en VHDL de un sistema digital que transmita datos en forma serie según el protocolo RS-232 y muestre los datos recibidos y transmitidos en un LCD (esto ultimo es opcional). Implementar el diseño en el DE2-115 development kit.

El dato a transmitir debe tener el formato detallado de la figura 1, con la siguiente configuración:

- **1 bit de start**
- **8 bits de datos**
- **Paridad par**
- **1 bit de stop**
- **Frecuencia de recepción/transmisión por defecto es de 9600 bauds, con la posibilidad de cambiar la velocidad de Rx/Tx a 4800, 38400, 115200 bauds, seleccionadas por las llaves disponibles en el board.**



Figura N°1: RS-232 formato de transmisión.

El protocolo RS232 fue diseñado para comunicación punto a punto, en donde se tiene una computadora que se encuentra transmitiendo hacia un equipo esclavo ubicado a distancias no mayores a 15 metros y a una velocidad máxima de 19200 bauds. Este tipo de transmisión es bastante

simple, pero también muy vulnerable al ruido aditivo en la linea y por esta razón es empleada para comunicación a distancias cortas.

En general, en la transmisión RS232, las cadenas de datos son caracteres ASCII, los cuales incluyen los códigos de letras, números y signos de puntuación, ademas de caracteres especiales. Se trata de un estándar orientado a la transmisión de texto.

El formato de transmisión de datos en las señales Tx y Rx del estándar RS232, se muestra en la figura 1. Se trata de una señal bipolar, normalmente entre +10 y -10 Voltios, con formato asíncrono.

El bit de inicio (start bit) tiene como función proporcionar, mediante el flanco ascendente, la señal de sincronía para que el circuito receptor puede muestrear el resto de los 8 bits de datos. La velocidad de transmisión define el periodo de cada uno de los bits. Por ejemplo, una velocidad de 1200 bps opera con una duración de 833 us por cada bit. A 9600 bps, se tendrían 104 us por bit.

Al final de la trama de 8 bits, se generan los llamados bits de paro(stop bits) cuya función es regresar la señal al estado bajo para preparar el siguiente flanco ascendente del bit de inicio. La transmisión se conoce como “asíncrona”, dado que no se requiere una señal separada para sincronía, sino que cada carácter incluye tanto los 8 bits de datos como los bits de inicio y de paro para establecerla.

Para la conexión se utiliza un cable con conectores DB9, con nueve señales, como el mostrado en la figura N°2. Adicionalmente a las señales de datos transmitidos y recibidos TX, RX, la norma original RS232 incluye definiciones para señales de control (handshake signals) que se usan para varias funciones auxiliares en el protocolo de envío y recepción de datos, así como para el diagnóstico de fallas.

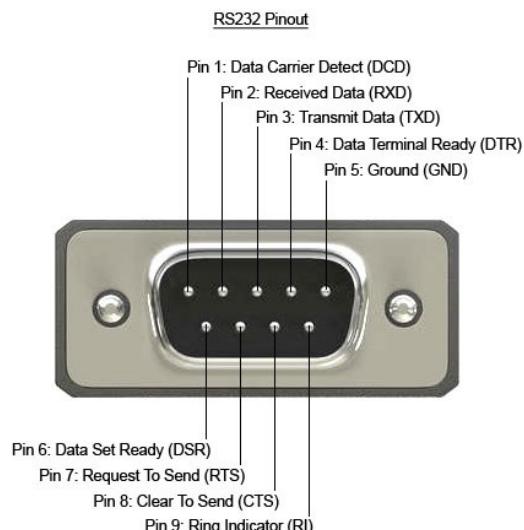


Figura N°2: Conector DB9.

Para la esta practica se remitió únicamente a las señales de transmisión TX, recepción RX y tierra GND, dejando sin utilizarse el resto de las señales. Ademas cabe destacar que utilizamos un interfaz DB9 a USB para la comunicación de la placa con la PC. Como se aprecia en la figura N.º3.



Figura N.º3: Interfaz DB9 a USB.

Una vez sentada las bases de lo que representa el protocolo de comunicación RS232 se continua con el diseño del mismo.

El proyecto en si se puede resumir en el siguiente esquema presentado en la figura N°4:

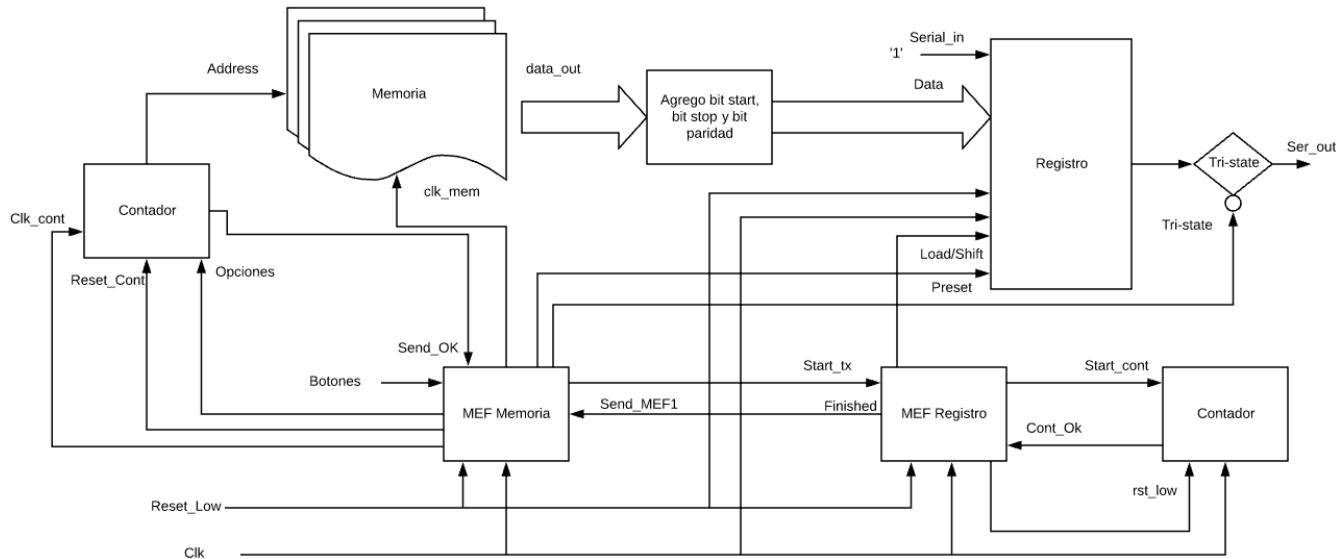


Figura N°4: Esquema principal del proyecto.

Para la realización de una manera mas ordenada del proyecto, se estipulo al inicio del mismo la división de dos partes: Parte A y parte B donde las mismas comprenden ciertas áreas del esquema:

- Parte A: Registro, Contador de registro y maquina de estado que controla al registro.
- Parte B: Memoria ROM, contador de memoria, maquina de estados que controla la memoria.

2.2 Parte A

2.2.1 Registro

Una colección de dos o mas flip-flops D con una entrada de reloj en común se denomina un *registro*. Los registros se utilizan con frecuencia para almacenar una colección de bits relacionados, tal como un byte de datos. Sin embargo, un registro simple también puede ser empleado para almacenar bits de datos sin relación o información de control; la única restricción real es que todos los bits se almacenan utilizando la misma señal de reloj.

Un *registro de corrimiento* es un registro de n bits con una disposición para recorrer sis datos almacenados por una posición de bit en cada tic de reloj.

Para esta práctica se requerirá de un registro de corrimiento de entrada serie y paralela, y salida serie. La entrada paralela se utiliza para depositar en el registro la palabra a transmitir de manera serie a través de acertar la señal de registro LOAD/SHIFT en '1', el cual es una palabra de 11 bits (start, dato 8 bits, bit paridad, stop), ahora la entrada serie nos servirá para depositar un '1' lógico en caso de que no haya transferencia y transferirlo en forma serie colocando la señal LOAD/SHIFT en '0'.

2.2.2 Contador del registro

El nombre *contador* generalmente se utiliza para cualquier circuito secuencial temporizado cuyo diagrama de estado contenga un solo lazo. El modulo de un contador es el numero de estados en el lazo. Un contador con m estados se conoce como un contador de modulo m, o a veces, un contador de división entre m.

Este contador de modulo 11, tiene como objetivo llevar la cuenta de la cantidad de bits que se extraen por la salida serie del registro una vez que se comienza una transmisión. Cuando este llega a cumplir su modulo, activa un flag avisando que se transfirió con éxito la palabra de datos.

2.2.3 Maquina de estado finito del registro

Para este apartado se tiene una maquina de estado finito el cual controla el accionar del registro en base al contador y una señal de start. La carta correspondiente a la maquina de estado se aprecia en la figura N°5.

En el inicio de la misma se tiene el estado idle al cual se llega a el a través de una señal de reset o la condición de que no haya comenzado una señal de comienzo de transmisión ($start = 0$). Una vez que se acciona la señal de comienzo de transmisión, se activa, en el estado load_shift, la salida I_s la cual ordena al registro que cargue la palabra de 11 bits a transmitir en el registro, luego en el estado cont_down se inicia el contador a través de la salida $start_cont$ la cual permanecerá en este estado hasta que el contador active el flag $cont_ok$ el cual indica que se transmitieron los 11 bits. Una vez finalizado se activa un flag de $finished$, el cual su uso se explicara mas adelante, para luego volver al estado de inicio.

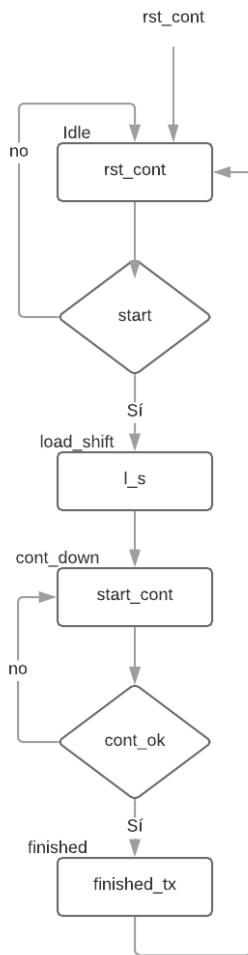


Figura N°5: Diagrama de flujo de maquina de estado del registro.

El esquema RTL ofrecido por Quartus al unir todos los elementos de la Parte A se puede apreciar en la figura N.º6, mientras que el diagrama test-bench que demuestra el funcionamiento del esquema se presenta en la figura N°7. Los códigos VHDL tanto del esquema como del test bench se encuentran en el anexo al final de este informe.

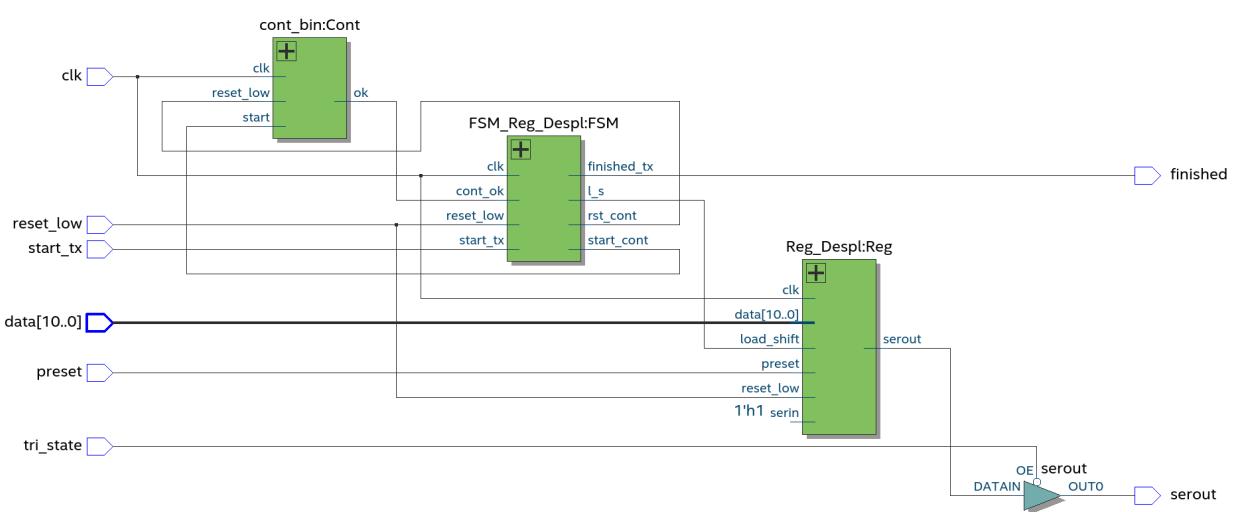


Figura N.º6: Esquema RTL parte A.

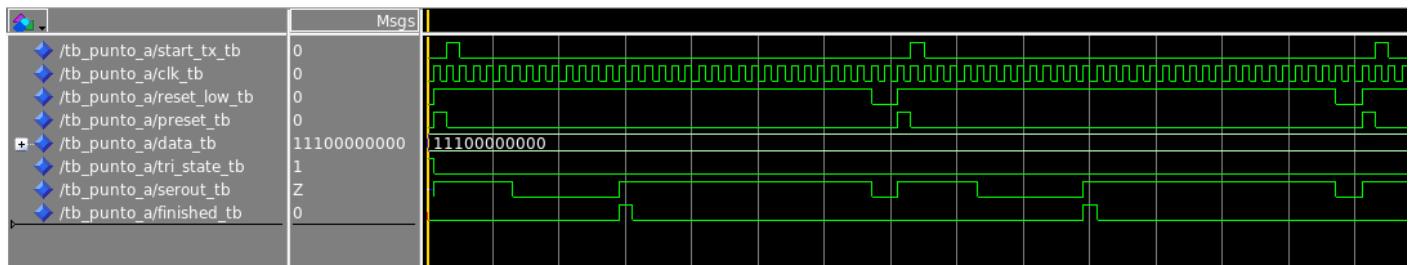


Figura N°7: Diagrama test-bench parte A.

De la figura N°7 podemos apreciar el correcto funcionamiento de la parte A enviando una palabra de 11 bits, donde al finalizar se activan el correspondiente flag de finalización de transmisión de palabra.

Con esto damos por finalizado el apartado A el cual se encarga de la transmisión de una palabra de datos de 11 bits por un puerto serie.

2.3 Parte B

2.3.1 Memoria ROM

Una memoria de solo lectura (ROM, read only memory) es un circuito combinacional con n entradas y b salidas. Las entradas se llaman *entradas de dirección* y se denominan de manera tradicional A₀, A₁, ..., A_{n-1}. Las salidas se conocen como *salidas de datos* y habitualmente se denominan D₀, d₁, ..., d_{b-1}. Un esquema conceptual de una memoria ROM se presenta en la figura N.º8.

Para este proyecto se uso una ROM de arreglo 64*8 para almacenar las distintas letras en ASCII de las distintas frases a enviar por el bus serial de salida. A medida que se selecciona una entrada del sistema, el mismo irá buscando, a través de las correspondientes direcciones, las letras en la memoria para ser extraídas en la salida hasta formar la frase completa.

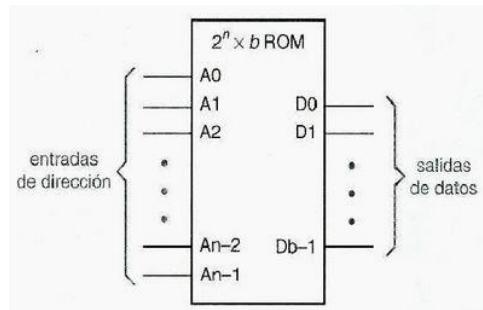


Figura N.º8: Estructura conceptual de memoria de solo lectura (ROM).

2.3.2 Contador de memoria

Este contador es el encargado de contar la cantidad de letras a extraer de la memoria pero ademas, es el encargado de brindar las distintas direcciones de cada una de las letras en la memoria. Estas direcciones las extraemos de una de las salidas del contador las cuales se conectan a las entradas de dirección de la memoria.

Básicamente el contador opera de la siguiente manera: En base a una opción de entrada el contador selecciona el primer carácter de la frase correspondiente a esa selección. Una vez seleccionado se obtiene a la salida la dirección de dicho carácter la cual esta es provista a la memoria para ser extraída. Una vez hecho esto, se procede el envío del carácter por parte del apartado A, esperando a que el mismo indique a través del flag *finished*, de la MEF del registro, que el carácter fue enviado exitosamente, para luego incrementar el contador y así apuntar a la dirección de memoria correspondiente al carácter posterior de la frase a enviar. Cuando se llega al ultimo carácter el contador activa una bandera, *send_ready*, la cual anuncia la finalización del envío de la frase poniendo luego el contador a cero, esperando a otra opción de entrada para transmitir.

2.3.3 Maquina de estado de finito de la memoria.

Para este apartado, se diseño una maquina de estados la cual en base al contador ya mencionado se regula el comportamiento de la memoria para que la misma en determinados momentos brinde a su salida los caracteres en ASCII correspondientes a las letras de las frases a transmitir. Dicha carta se puede apreciar en la figura N.º9.

Al principio de la carta, se puede ver un estado de inicio el cual se ejecuta una sola vez, este estado sirve para asegurar que a la salida no haya ningún valor lógico, ya que si existe la presencia de un cero lógico, el receptor del dispositivo externo entenderá que hay un bit de start y por lo tanto recibirá los siguientes 10 bits considerándolos como si fuera un carácter valido. Para evitar este inconveniente se conecta a la salida un bus tri-state, el cual asegura que la salida esté en alta impedancia al comienzo del encendido del sistema. Ademas de esto se presetea el registro de corrimiento, cargándolo enteramente a sus flip flops internos con un '1' lógico, de esta manera se transmitirá de aquí en mas solamente unos lógicos, traduciéndolo en que por el momento no se enviá nada por parte del sistema. Luego se ejecuta un estado de reset para setear el sistema con las configuraciones iniciales (contadores en cero y banderas en cero).

Una vez hecho lo anterior se procede a preguntar por las entradas de selección; como se puede apreciar se da una determinada jerarquía de entradas, siendo la frase 1, asociada a la señal *botones(0)*, la de mayor jerarquía y la frase 5, asociada a la señal *botones(4)*, la de menor jerarquía. Una vez se acierta una de estas entradas, se procede a acertar una señal de salida en correspondencia con la entrada activada. Esta señal es la que le indica al contador que frase debe ser extraída en la memoria, por lo tanto el contador sabe desde que inicio apuntar en memoria y hasta donde contar. Luego de esto se le da un clock al contador para que busque la dirección de memoria del carácter y despues, en el estado siguiente, se le brinda un clock a la memoria para que extraiga el carácter ASCII, lo cual en ese instante tendrá en su entrada la dirección de memoria otorgada por el contador.

Realizado lo anterior, se ejecuta mediante la señal *start_tx* el envío del carácter por parte de la maquina de estados del registro, y se espera hasta que se active el flag de envío correcto por parte de esta misma maquina de estados, una vez dada estas condiciones se pregunta si el contador termino de enviar la frase completa a través del flag *send_ok*, si es cierto se termina proceso

reiniciando el sistema y esperando un nuevo pedido del usuario, ahora si no lo es, vuelve a ejecutar los clocks correspondientes al contador y memoria y se vuelve a ejecutar una transmisión de carácter, repitiéndose el proceso hasta que termine de transmitirse la frase completa.

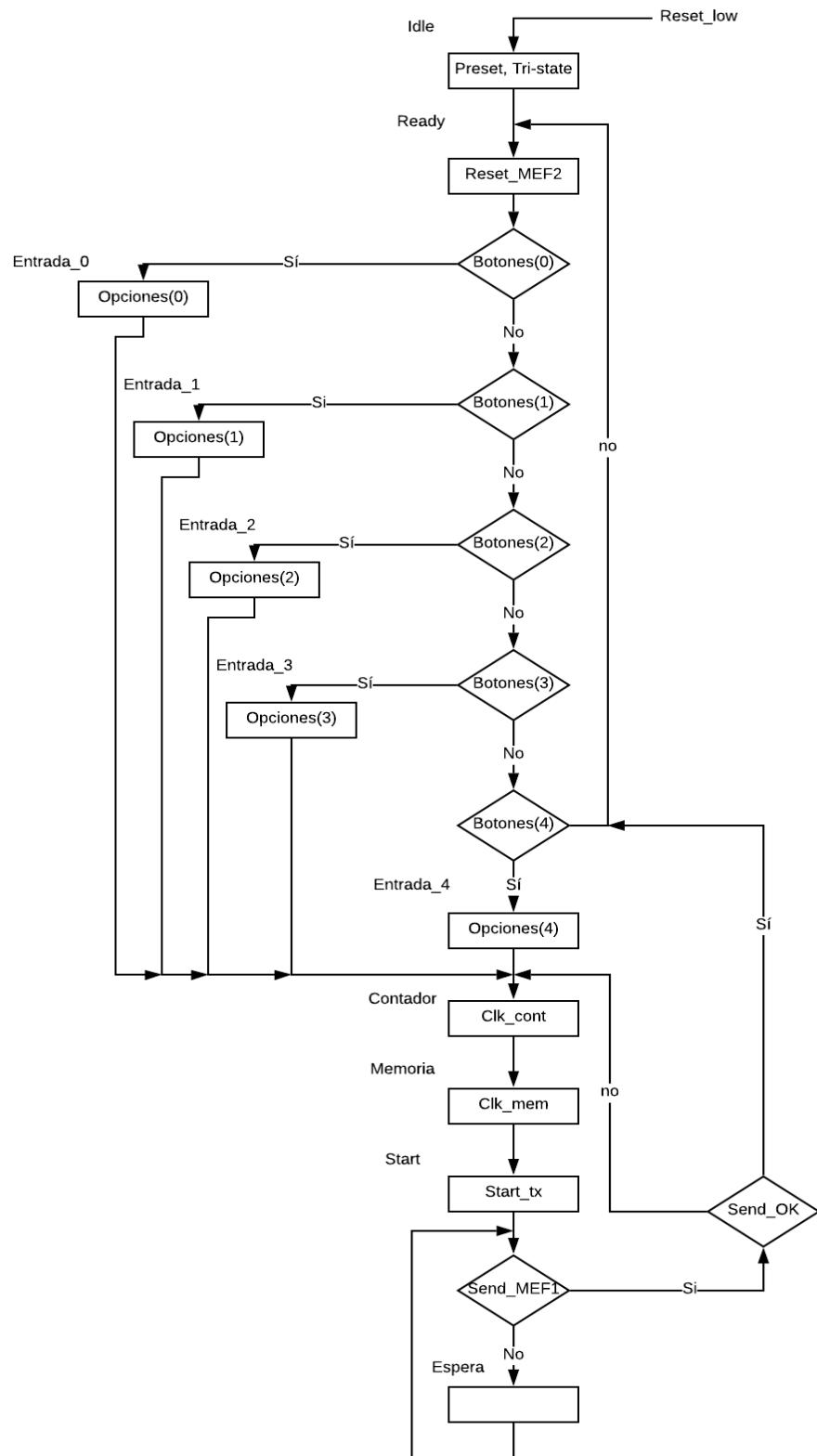


Figura N.^o9: Diagrama de flujo de maquina de estado de memoria.

El esquema RTL del apartado B es el presentado en la figura N.^o10, mientras que el diagrama de test-bench que prueba el funcionamiento del apartado es el de la figura N^o11.

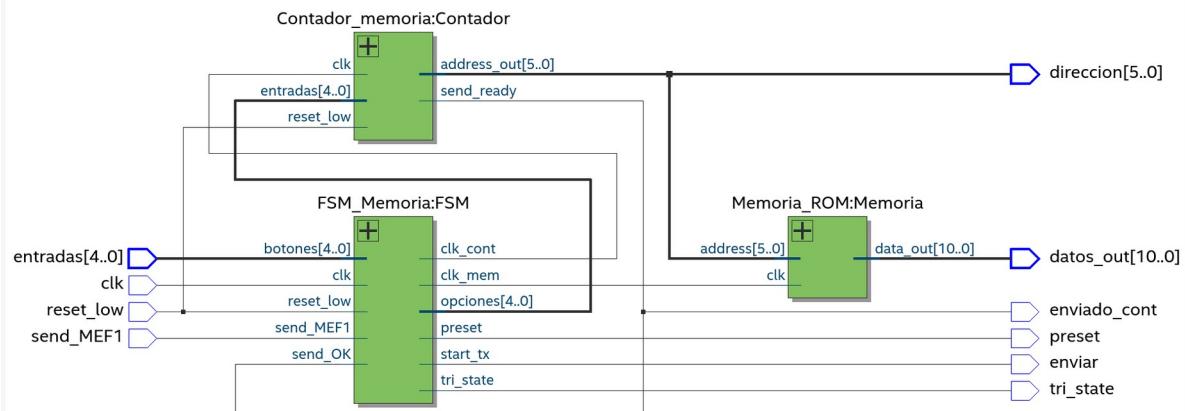


Figura N.º10: Esquema RTL parte B.

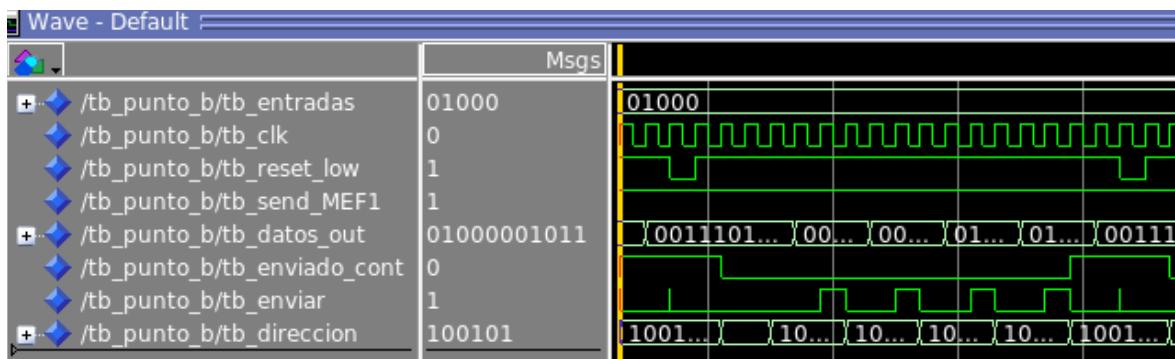


Figura N.º11: Diagrama test-bench parte B.

Como se puede apreciar en la imagen de la figura N.º11, el esquema muestra como se selecciona una entrada correspondiente a la cuarta frase, y tanto en la salida `tb_datos_out` como en `tb_direccion` se obtiene el datos de 11 bits y la dirección de memoria del dato respectivamente.

Una vez realizado todo esto procedimos a unir ambas partes A y B para verificar el funcionamiento de todo, pero antes de eso en las siguientes secciones se explicaran como se compone la memoria, la función de bit de paridad, el modulo LCD y el receptor los cuales son opcionales al proyecto.

2.4 Paquete (package)

Un paquete es una colección de declaraciones de tipo, constantes, programaras, etc., normalmente con la intención de implementar algún servicio en particular o aislar un grupo de elementos relacionados. De esta manera se pueden hacer visibles las interfaces de algunos elementos como funciones o procedimientos quedando ocultas las descripciones de estos elementos.

Los paquetes están separados en dos partes, una es la parte de declaraciones y la otra es la de cuerpo. La parte de cuerpo, donde se encuentran por ejemplo algunas definiciones de funciones y procedimientos, puede ser omitida si no hay ninguno de estos elementos. El nombre dado al paquete y al cuerpo del paquete debe coincidir para que se entienda que ambos forman un conjunto.

Para este proyecto se implemento un paquete en el cual contiene las constantes referidas a la estructura de la memoria ROM y ademas se tiene el contenido de la misma, el cual son los caracteres ASCII de las letras correspondientes a las frases a transmitir. Ademas de esto se implemento una función de bit de paridad la cual se explicara en la siguiente sección.

El código del paquete estará escrito en el anexo de este informe.

2.5 Bit de paridad

Un bit de paridad es un digito binario que indica si el numero de bits con un valor de 1 en un conjunto de bits es par o impar. Los bits de paridad conforman el método de detección de errores mas simple.

La paridad par es un caso especial del *control de redundancia cíclica* (CRC).

Este método detecta los errores, pero no los corrige (salvo en el caso de que la palabra transmitida sea de tamaño de 1 bit (lo cual no es habitual)). Existen dos variantes de este método, bit de paridad par e impar.

En el caso de la paridad par, se cuentan el numero de unos. Si el total es impar, el bit de paridad se establece en uno y por tanto la suma del total anterior con este bit de paridad, daría par. Si el conteo de bits uno es par, entonces el bit de paridad (par) se deja en 0, pues ya es par.

En el caso de la imparidad par, la situación es la contraria. Se suman los bits cuyo valor es uno, si da un numero impar de bits, entonces el bit de paridad (impar) es cero. Y si la suma de los bits cuyo valor es uno es par, entonces el bit de paridad (impar) se establece en uno, haciendo impar la cuenta total de bits uno.

Para este proyecto se considero el uso de un bit de paridad par el cual se implemento en base a una función que posee como entradas los bits a analizar y devuelve un bit el cual es el correspondiente al bit de paridad par, con esto se tiene en cuenta el bit de stop para la paridad. Se aprovecho el uso de paquetes en VHDL y se declaro y definió la función en dicho paquete.

El código, como siempre, estará en el anexo al final de este informe.

2.6 Receptor

Para el diseño del receptor se considero realizar una pequeña maquina de estados la cual su diseño se aprecia en el esquema de la figura N.^o 12.

Su funcionamiento se basa al comienzo en la detección del bit de start, una vez encontrado se espera una mitad de tiempo de bit y se pregunta de vuelta si el bit de start sigue activo, esto es mas que nada una medida de seguridad por si el receptor detecta de manera accidentada un glitch.

Una vez detectado el bit de start, se procede a recibir en los próximos estados los bits de datos, lo cual se van almacenando uno en uno en una señal de 8 bits.

Al finalizar se detecta el bit de stop, se espera otro ciclo de reloj mas y el proceso se vuelve a iniciar.

La figura N.^o13 muestra el diagrama realizado del test-bench correspondiente a este modulo, el cual comprueba la perfecta recepción de un byte enviado.

Una vez completado este modulo se produjo a través de la instrucción *for loop generate* un conjunto de receptores lo cual operan con cada uno de los baud-rate pedidos en este laboratorio.

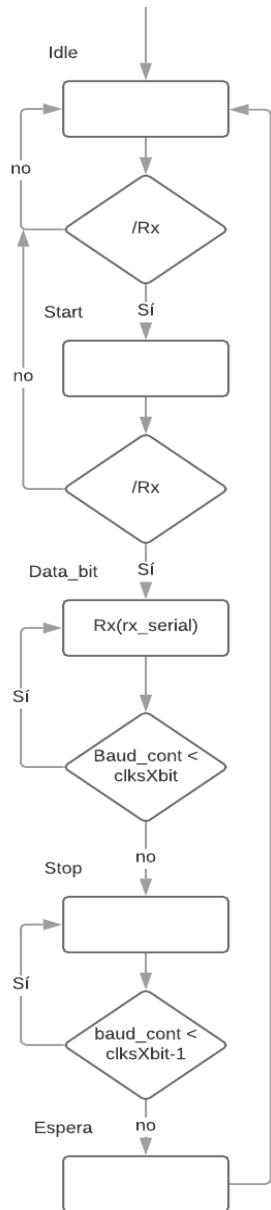


Figura N.º12: Diagrama de flujo de máquina de estado de la memoria

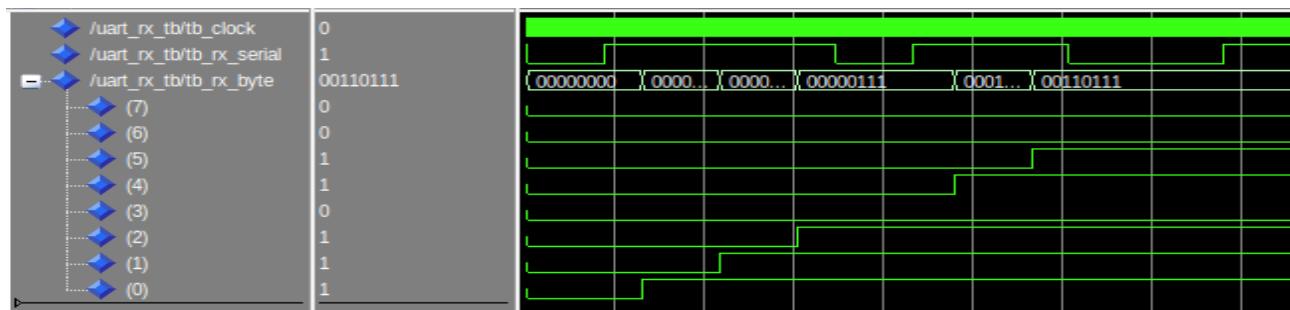


Figura N.º13: Diagrama temporal test-bench de receptor UART.

2.7 Display LCD

Para graficar cada uno de las acciones realizadas en este proyecto se decidió realizar un modulo controlador de display LCD, mostrado en la figura N°14. En el mismo se podrán graficar cada una de las frases enviadas y ademas un carácter recibido a partir de los receptores.

Para usar un LCD, el primer paso es entender el controlador. Observando la figura N°15 se puede entender que, ademas de los pines de alimentación, las siguientes cuatro señales deben ser tenidas en cuenta para el controlador:

- **RS(register select)**: '0' selecciona el registro de instrucción del controlador, mientras que '1' selecciona el registro de datos (este ultimo es para pasar caracteres que serán mostrados en el display).
- **R/W(Read/Write)**: Si es '0', el siguiente pulso de E (enable) causará que la presente instrucción o dato sera escrito en el registro seleccionado por RS, mientras sea '1' hace que los datos se lean desde el registro del controlador.
- **DB(Data bus)**: Es un bus de 8 bits el cual su contenido (datos o instrucciones) es escrito dentro del registro del controlador en el siguiente pulso de E si R/W='0', o a través del cual el dato es leído desde el registro del controlador si R/W='1'.
- **E(enable)**: Debe ser pulsada en alto para escribir en el registro del controlador.

Todas estas señales son enviadas al controlador. La señal principal recibida del controlador es descripta a continuación:

- **BusyFlag**: La señal es provista por el controlador a través del bit DB(7) del bus de datos, con un '1' indica que el controlador esta ocupado. En la practica, el uso de esta señal es normalmente evitada por la adaptación de las instrucciones con una separación de tiempo mas larga que el máximo requerido por las instrucciones a ser completadas. De esta manera, R/W puede ser mantenida en bajo permanentemente.

El set de instrucciones del controlador se muestra en la figura N.º16. Un resumen de sus características principales es el siguiente:

- Limpiar el display (Clear Display o Return Home).
- Incrementar o decrementar la posición del display (Entry Mode Set).
- Opción individual de modos del display, cursor, y parpadeo ON/OFF (Display ON/OFF Control)
- Funcionamiento con bus de 4 u 8 bits con una linea de caracteres de 5x8 o 5x10 puntos o con dos lineas de 5x8 puntos caracteres (conjunto de funciones).
- 7 bits para mostrar el direccionamiento de caracteres (Establecer dirección DD RAM), permitiendo acceso individual a las 128 posiciones del LCD, divididas en dos renglones de 64 caracteres cada uno. La dirección del primer carácter en el primer renglón es 0, mientras la dirección del primer carácter en el segundo renglón es 64, independientemente del numero actual de caracteres en el LCD.



Figura N.º14: LCD 16x02

Pin	Name	Direction	Function
1	Vss	Ground	0V
2	Vcc	Power in	+5V
3	Vo	Analog in	Contrast (0V to 5V)
4	RS	Input	Register Select (0: Instruction register, 1: Data register)
5	R/W-	Input	Read/Write (1: Read from display, 0: Write to display)
6	E	Input	Enable read/write
7	DB0	In-Out	Data (LSB)
8	DB1	In-Out	Data
9	DB2	In-Out	Data
10	DB3	In-Out	Data
11	DB4	In-Out	Data
12	DB5	In-Out	Data
13	DB6	In-Out	Data
14	DB7	In-Out	Data (MSB) and BF (busy flag)
15	A	Power in	Backlight anode (+) (optional)
16	K	Power in	Backlight cathode (-) (optional)

Figura N.º15: Función de pines de display LCD.

Instruction	RS	RW-	DB7 ... DB0	Description	Max exec. time (*)
1) Clear Display	0	0	0 0 0 0 0 0 1	Clears display and sets DD RAM address to zero.	1.52 ms
2) Return Home	0	0	0 0 0 0 0 0 1 X (X=don't care)	Returns display to origin and sets DD RAM address to zero.	1.52 ms
3) Entry Mode Set	0	0	0 0 0 0 1 I/D S	Sets cursor direction and display shift during read and write. I/D=1 increment DD RAM address, =0 decrement S=1 shift display, =0 do not shift	37 us
4) Display ON/OFF Control	0	0	0 0 0 0 1 D C B	D=1 display on, =0 off C=1 cursor on, =0 off B=1 blink char., =0 do not blink	37 us
5) Cursor or Display Shift	0	0	0 0 0 1 S/C R/L X X	Moves cursor or display without changing DD RAM contents. S/C=1 shift display, =0 shift cursor R/L=1 shift to right, =0 shift to left	37 us
6) Function Set	0	0	0 0 1 DL N F X X	Set bus size, number of lines, and digit size (font). DL=1 8-bit bus, =0 4-bit bus N=0 1-line operation, =1 2-line F=0 5x8 dots, =1 5x10 dots	37 us
7) Set CG RAM Address	0	0	0 1 A A A A A A	Sets CG RAM address to AAAAAAA	37 us
8) Set DD RAM Address	0	0	1 A A A A A A	Sets DD RAM address to AAAAAAA	37 us
9) Read Busy Flag and Address	0	1	B F A A A A A A	Reads busy flag and address counter	0 us
10) Write Data to CG or DD RAM	1	0	D D D D D D D D	Writes data into DD RAM or CG RAM (defined by last DD or CG RAM address set)	41 us
11) Read Data from CG or DD RAM	1	1	D D D D D D D D	Reads data from DD RAM or CG RAM (defined by last DD or CG RAM address set)	41 us

(*) For 270 kHz internal oscillator; for other frequencies (100 to 500 kHz), multiply time given by 270 kHz/foscillator.

Figura N.º16: Set de instrucciones del controlador LCD (HD44780U or KS0066U).

A partir de todas estas consideraciones se realizo una maquina de estados como controlador en la cual se inicializa el display un modo de funcionamiento de bus de 8 bits, se configuran los retornos y estados de limpieza en función de la frase a ser enviada por el protocolo RS232 y ademas se tiene en cuenta un renglón del mismo para graficar el carácter recibido por el receptor. Dicha maquina de estados cuenta con 58 estados los cuales una aproximación grafica de la misma nos la provee el programa Quartus con la herramienta visor de maquinas de estados la cual se aprecia en la figura N°17:

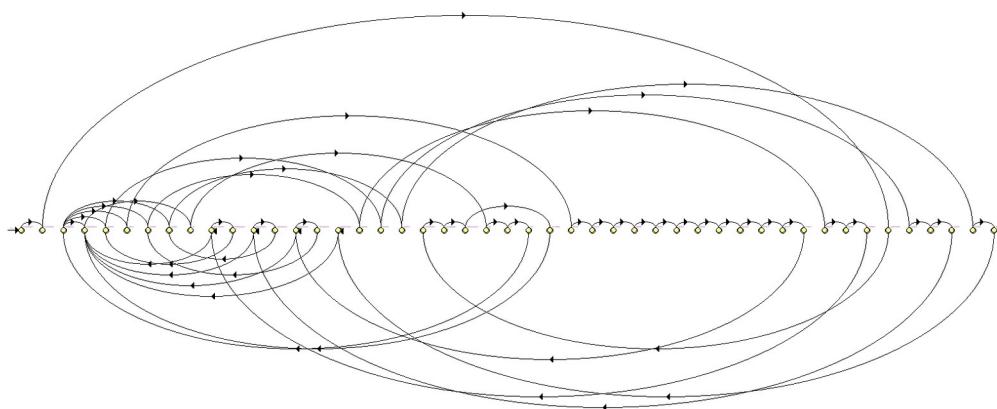


Figura 17: Diagrama de estados provisto por Quartus de la maquina de estados del controlador LCD.

Antes de subir el programa en placa se realizo el correspondiente test-bench para la prueba del modulo. Este mismo genero un diagrama que se puede apreciar en la figura N.º 18:

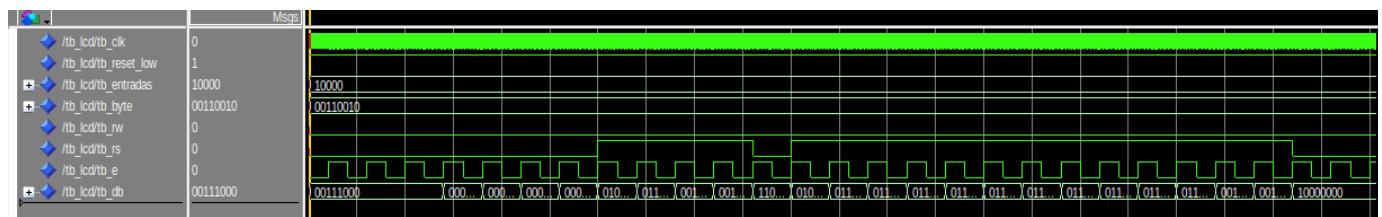


Figura N.º18: Diagrama test-bench del modulo LCD.

2.8 Puesta a punto del proyecto.

Una vez comprobado el correcto funcionamiento de todos los módulos y partes del proyecto se procedió a realizar un nuevo proyecto en Quartus que reuniera a todos los módulos ya hechos hasta el momento: Display LCD, receptor, Parte A (transmisor), Parte B (transmisor), memoria ROM, sincronizadores, anti-rebotes y sistema de reloj. Todo esto se puede apreciar en el diagrama RTL otorgado por Quartus de la figura N°19 y su correspondiente test-bench en la figura N°20:

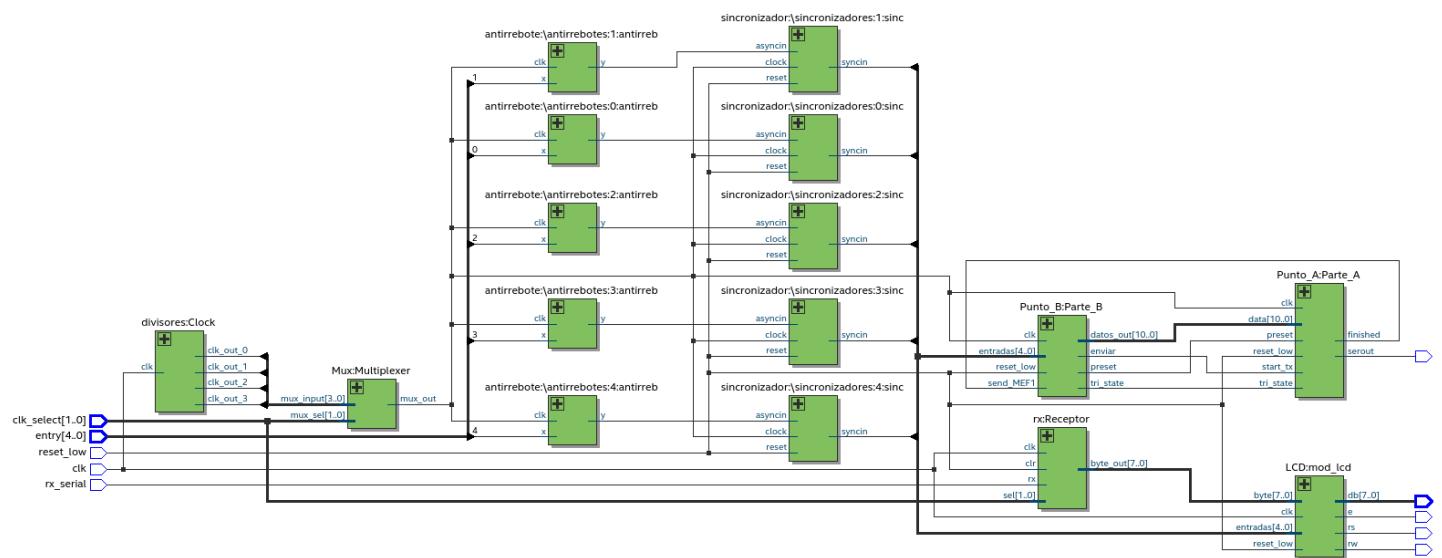


Figura N.º19: Diagrama RTL RS232.

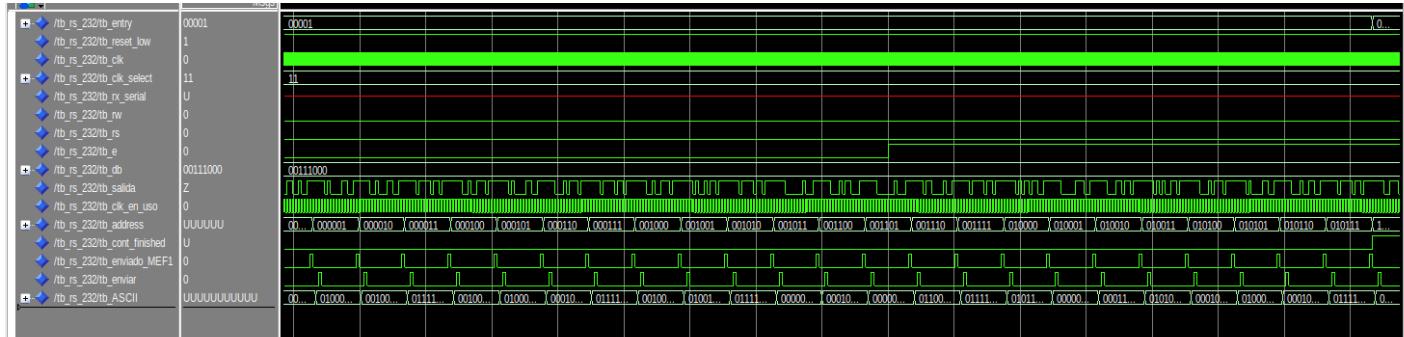


Figura N.º20: Diagrama test-bench de protocolo RS232

Terminado de analizar a fondo el diagrama test-bench del protocolo RS232, se procedió a realizar la puesta a punto del proyecto. Para ello se utilizó una placa FPGA Altera Cyclone IV modelo EP4CE6E22C8 donde se conectó el módulo display LCD, y el cable interfaz DB9 a USB. Además debido a limitaciones del hardware de la placa, se utilizó un par de entradas externas al FPGA las cuales consisten en llaves con resistencias pull-up y pull-down. Todas estas conexiones se pueden apreciar en la figura N.º21.

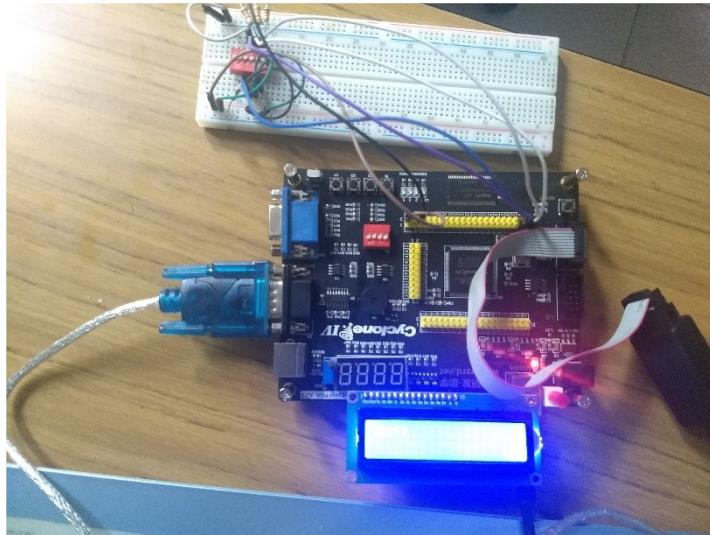


Figura N°21: Esquema general del proyecto.

Para poder realizar la conexión entre PC y placa se utilizó un programa llamado TeraTerm el cual administra el canal de comunicación serie adaptando la frecuencia de comunicación según la voluntad del usuario. Imágenes de la configuración del mismo se pueden apreciar en la figura 22 a), b), c) y d).

Una vez configurado la frecuencia de baudios correcta se procedió a transmitir desde placa a PC, donde se graficaron correctamente las frases enviadas a través del monitor y por supuesto el display LCD. En la figura N.º23 se muestra lo graficado por el programa y las figuras N°24, 25, 26 y 27 nos muestran lo graficado en el display LCD.

Práctica N° 5

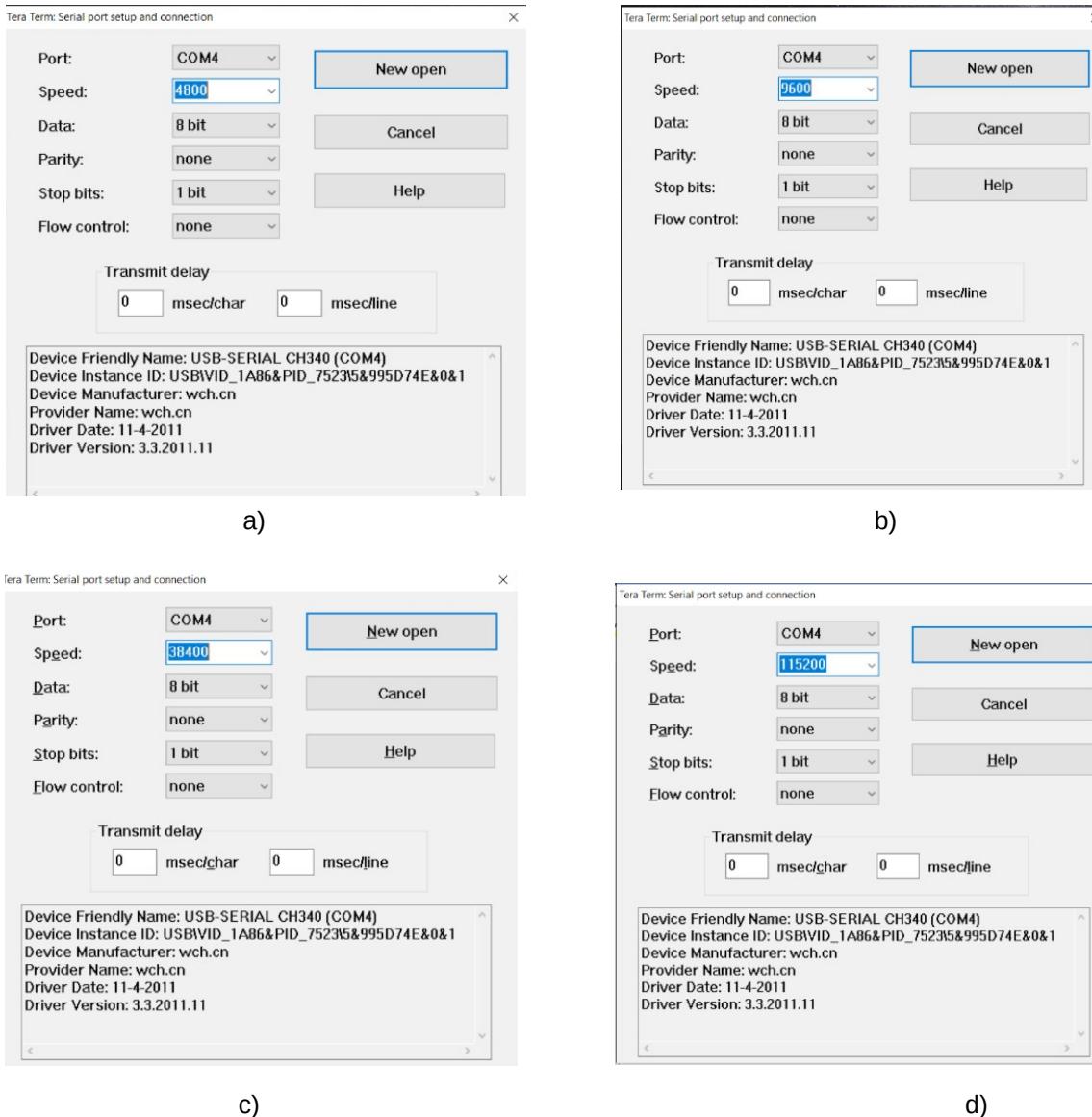


Figura N.º22: Configuración de baudios. a) 4800baud, b) 9600baud, c) 38400baud d) 115200baud.

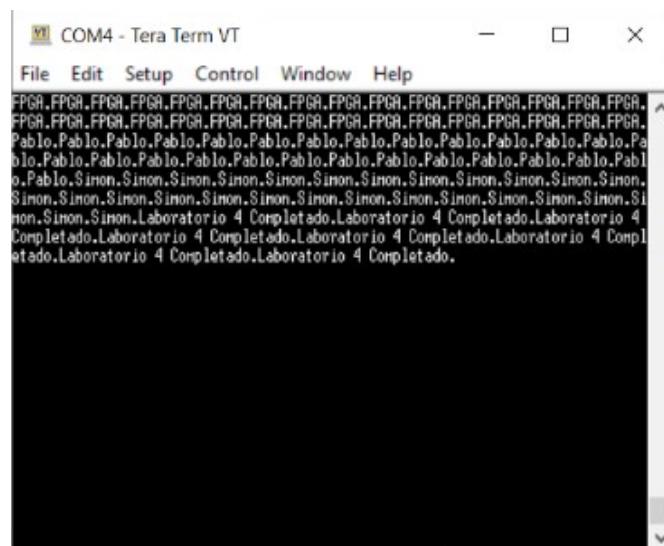


Figura N°23: Frases enviadas al monitor serial.



Figura N°24: Envío de la primera frase y recepción del carácter t.



Figura N.º25: Envío de la segunda frase y recepción del carácter g.



Figura N.º26: Envío de la tercera frase y recepción del carácter j.



Figura N.º27: Envío de la cuarta frase y recepción del carácter m.

3 Conclusiones.

Como conclusión general del proyecto se obtuvo un buen resultado de la implementación del protocolo, añadiendo ademas conceptos tales como uso de memorias, paquetes y cuerpos de paquetes en VHDL y la implementación del modulo controlador de display LCD el cual dio como resultado un sistema transmisor-receptor como así también una interfaz grafica de usuario que permita observar toda acción del proceso.

4 Anexo.

4.1 Parte A.

4.1.1 Registro de desplazamiento.

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

entity Reg_Despl is
    generic (ancho : integer :=11);
    port (
        serin           :in std_logic;
        --Entrada serial por donde se estableceria un 1 logico constante.
        clk             :in std_logic;
        --Entrada de reloj.
        rst_low         :in std_logic;
        --Entrada asincronica de reset en bajo.
        preset          :in std_logic;
        --Entrada asincronica de seteo de FFD's.
        data            :in std_logic_vector(ancho-1 downto 0);
        --Entrada paralela de datos.
```

```

        load_shift  :in std_logic;
                    --Entrada la cual permite cargar los datos en
paralelo a cada uno de los ffd.

        serout          :out std_logic
                    --Salida serie del registro.

    );

end Reg_Despl;

architecture behav of Reg_Despl is

--Declaro componente FFD con reset asincronico y carga sincronica.

component FFD is

port(
    d                  :in std_logic;
    data               :in std_logic;
    reset_low         :in std_logic;
    preset             :in std_logic;
    clk                :in std_logic;
    load_shift        :in std_logic;
    q                  :out std_logic
);

end component;

--Señal interna entre FFD de registr.

signal intercon :std_logic_vector(0 to ancho-1);

--Señal auxiliar de salida.

signal bus_out :std_logic;

begin

reg_despl: for i in 0 to ancho-1 generate
            --Para el primer ffd conecto a su entrada
la entrada serie del registro.

            primero: if(i = 0) generate
                        priff: FFD port
map (preset => preset, clk => clk, reset_low => rst_low, load_shift =>
load_shift, d => serin, data => data(i), q => intercon(i));
                        end generate;
            --En los ffd siguientes conecto a su
entrada, la salida del ffd anterior.

```

```

        medios:      if (i /= 0 and i /= ancho-1)
generate

        medff:  FFD  port
map  (preset => preset, clk => clk, reset_low => rst_low, load_shift =>
load_shift, d=> intercon(i-1), data => data(i), q => intercon(i));
                                         end generate;

--En el ultimo la salida se conecta con
la salida serie del registro.

        ultimo:      if(i = ancho-1) generate
        ultff:  FFD  port
map  (preset => preset, clk => clk, reset_low => rst_low, load_shift =>
load_shift, d => intercon (i-1),data => data(i), q=> bus_out);
                                         end generate;

end generate reg_despl;

serout <= bus_out;

end behav;

```

4.1.2 Contador binario.

```

library ieee;
use ieee.std_logic_1164.all;

entity cont_bin is
port (
    start_cont :in std_logic;      --Entrada de habilitacion de
conteo.

    clk          :in std_logic;    --Entrada de reloj.

    rst_low      :in std_logic;    --Entrada de reset en
bajo.

    cont_ok       :out std_logic   --Salida la cual se
usa como flag de termino de cuenta (en este caso 9).

);
end entity;

architecture behav of cont_bin is

--Señal de conteo.

signal cont :integer range 0 to 10 :=0;

--Señal de busd de salida.

signal bus_out    :std_logic := '0';

```

```
begin

    contador: process(clk,rst_low,start_cont)
    begin
        if (rst_low = '0') then
            cont <= 0;
            bus_out <= '0';
        elsif (clk'event and clk = '1' and start_cont = '1') then
            if cont = 9 then
                cont <= 0;
                bus_out <= '1';
            else
                cont <= cont + 1;
            end if;
        end if;
    end process;

    cont_ok <= bus_out;

end behav;
```

4.1.3 FSM registro de desplazamiento

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

entity FSM_Reg_Despl is
    port (
        start_tx          :in std_logic;           --Señal de comienzo de transmision de caracter ASCII.
        cont_ok           :in std_logic;           --Señal de flag de finalizacion del contador binario.
        clk               :in std_logic;           --Señal de reloj.
        rst_low           :in std_logic;           --Señal de reset en bajo.
        l_s               :out std_logic;          --Salida que indica al registro la carga paralela de datos.
        start_cont        :out std_logic;          --Salida de habilitacion del contador binario.
```

```
        rst_cont          :out std_logic;    --Salida de reset del
contador binario.

        finished_tx :out std_logic           --Salida para indicar
termino de la transmision de caracter ASCII.

);

end FSM_Reg_Despl;

architecture behav of FSM_Reg_Despl is

--Declaro los estados de la FSM.

type FSM_states is (idle,load_shift,cont_down,finished);

--Declaro las señales de la FSM.

signal current_state, next_state : FSM_states;

--Establezco el estilo de codificacion.

attribute syn_encoding: string;
attribute syn_encoding of FSM_States : type is "one hot";

begin

--Proceso de estado presente.

cs_pr: process (clk,rst_low)
begin
    if rst_low = '0' then
        current_state <= idle;
    elsif (rising_edge(clk)) then
        current_state <= next_state;
    end if;
end process;

--Proceso de proximo estado.

nxp: process(current_state,start_tx,cont_ok)
begin
    case current_state is

        when idle =>
            if (start_tx = '1') then
                next_state <= load_shift;
```

```
        else
            next_state <= idle;
        end if;

        when load_shift =>
            next_state <= cont_down;

        when cont_down =>
            if cont_ok = '1' then
                next_state <= finished;
            else
                next_state <= cont_down;
            end if;

        when finished =>
            next_state <= idle;

    end case;
end process;

--Proceso de salida Moore.
moore_pr: process (current_state)
begin
    --Valores por default.
    l_s             <= '0';
    start_cont     <= '0';
    rst_cont       <= '1';
    finished_tx   <= '0';
    case current_state is
        when idle =>
            rst_cont <= '0';
        when load_shift =>
            l_s <= '1';
        when cont_down =>
            start_cont <= '1';
        when finished =>
            finished_tx <= '1';
        when others => null;
    end case;
```

```
    end process;

end behav;

4.1.4 Flip-flop D
library ieee;
use ieee.std_logic_1164.all;

entity FFD is
port (
    d           :in std_logic;
    data        :in std_logic;
    reset_low   :in std_logic;
    preset      :in std_logic;
    clk         :in std_logic;
    load_shift  :in std_logic;
    q           :out std_logic
);
end FFD;
```

```
architecture behav of FFD is
```

```
begin

ffd: process (clk,reset_low,preset)
begin
    if (reset_low = '0') then q <= '0';
    elsif (preset = '1') then q <= '1';
    elsif (rising_edge(clk)) then
        if load_shift = '1' then q <= data;
        else q <= d;
        end if;
    end if;
end process;
end behav;
```

4.1.5 Punto A

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

entity Punto_A is
```

```
generic (ancho : integer :=11);
port (
    start_tx      :in std_logic;
    --Señal de comienzo de Tx.
    clk           :in std_logic;
    --Señal de clock.
    reset_low     :in std_logic;
    --Señal de reset asincronico.
    preset        :in std_logic;
    --Entrada asincronica de seteo de FFD's.
    data          :in std_logic_vector (ancho-1 downto 0);
--Dato completo, 11 bits: paridad,start,stop,ASCII.
    tri_state     :in std_logic;
    --Buffer tri-state a la salida.
    serout        :out std_logic;
    --Señal de salida serie.
    finished      :out std_logic
);
end Punto_A;

architecture behav of Punto_A is

--Declaro Contador binario.
component cont_bin is
port(
    start_cont   :in std_logic;
    clk          :in std_logic;
    rst_low      :in std_logic;
    cont_ok      :out std_logic
);
end component;

--Declaro Registro de Desplazamiento.
component Reg_Despl is
generic (ancho : integer :=11);
port(
    serin        :in std_logic;
    clk          :in std_logic;
    rst_low      :in std_logic;
    preset       :in std_logic;
```

```
        data           :in      std_logic_vector(ancho-1
downto 0);

        load_shift  :in std_logic;
        serout       :out std_logic
    );

end component;

--Declaro FSM controladora del registro.

component FSM_Reg_Despl is

port(
    start_tx      :in std_logic;
    cont_ok       :in std_logic;
    clk           :in std_logic;
    rst_low       :in std_logic;
    l_s           :out std_logic;
    start_cont   :out std_logic;
    rst_cont      :out std_logic;
    finished_tx  :out std_logic
);

end component;

--Declaracion de señales.

signal l_s_bus           :std_logic;
signal serout_bus         :std_logic;
signal start_cont_bus    :std_logic;
signal cont_ok_bus        :std_logic;
signal reset_cont         :std_logic;

begin

    Cont: cont_bin port map (start_cont => start_cont_bus, clk => clk,
rst_low => reset_cont, cont_ok => cont_ok_bus);

    Reg: Reg_Despl port map (preset => preset, serin => '1' , clk => clk,
rst_low => reset_low, data => data, load_shift => l_s_bus, serout =>
serout_bus );

    FSM: FSM_Reg_Despl port map (start_tx  => start_tx, cont_ok  =>
cont_ok_bus, clk => clk, rst_low => reset_low, l_s => l_s_bus, start_cont
=> start_cont_bus,
```

```
        rst_cont  
=> reset_cont,finished_tx => finished);
```

```
    serout <= serout_bus when tri_state = '0' else 'Z';
```

```
end behav;
```

4.1.6 Test-bench Punto A

```
library ieee;
```

```
use ieee.std_logic_1164.all;
```

```
entity tb_Punto_A is  
end tb_Punto_A;
```

```
architecture behav of tb_Punto_A is
```

```
--Declaro componente a testear.
```

```
component Punto_A is
```

```
    generic (ancho : integer :=11);
```

```
    port (
```

```
        start_tx          :in std_logic;
```

```
        clk              :in std_logic;
```

```
        reset_low       :in std_logic;
```

```
        preset          :in std_logic;
```

```
        data            :in std_logic_vector (ancho-1  
downto 0);
```

```
        tri_state       :in std_logic;
```

```
        serout         :out std_logic;
```

```
        finished       :out std_logic
```

```
    );
```

```
end component;
```

```
--Definicion de constantes.
```

```
constant clk_semiperiod :time := 50us;
```

```
constant ancho           :integer :=11;
```

```
--Señal de test bench.
```

```
signal start_tx_tb      :std_logic := '0';
```

```
signal clk_tb           :std_logic := '0';
```

```
signal reset_low_tb      :std_logic := '0';
signal preset_tb         :std_logic := '0';
signal data_tb            :std_logic_vector(ancho-1 downto 0);
signal tri_state_tb       :std_logic :='1';
signal serout_tb          :std_logic;
signal finished_tb        :std_logic;

begin

clk_tb <= not clk_tb after clk_semiperiod;

    uut: Punto_A port map (start_tx => start_tx_tb, clk => clk_tb,
reset_low => reset_low_tb, preset => preset_tb, data => data_tb, tri_state
=> tri_state_tb,
                           serout           =>
serout_tb,finished => finished_tb);

data_gen: process
begin

--Ingresamos una palabra de 11 bits simulando al dato a
transmitir, con el canal de salida bloqueado por el bus tri-state.

    data_tb <= "11100000000";
    wait until clk_tb = '1';

--Habilitamos su funcionamiento incluyendo su salida tri-
state, y pre-seteando el registro.

    reset_low_tb <= '1';
    tri_state_tb <= '0';
    preset_tb <= '1';
    wait until clk_tb = '1';

--Comenzamos transmision de caracter ASCII.

    start_tx_tb <= '1';
    preset_tb <= '0';
    wait until clk_tb = '1';

    start_tx_tb <= '0';


```

```
        wait until serout_tb = '1';

        --Esperamos aproximadamente 11 clocks.
        wait until clk_tb = '1';
        wait until clk_tb = '1';

        --Reseteamos y comenzamos de cero.
        reset_low_tb <= '0';
        wait until clk_tb = '1';

    end process;

end behav;
```

4.2 Parte B

4.2.1 Memoria ROM

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.pkg_rom.all;
```

```
entity Memoria_ROM is
    port (
        clk           :in std_logic;
                                --Señal de reloj de la memoria
ROM.
        address       :in std_logic_vector(addr_length-1 downto 0);
        --Señal de entrada correspondiente a la dirección del carácter a
buscar.
        data_out      :out std_logic_vector(data_length-1 downto
0) --Dato ASCII junto con el bit start, stop y paridad par.
    );
end Memoria_ROM;

architecture behav of Memoria_ROM is

    --Señales auxiliares.
    signal dato_ok          :std_logic_vector(data_length-1
downto 0);

begin
    --La memoria ROM opera de la siguiente manera:
    --Cuando hay presencia de flanco positivo de reloj, la memoria
arroja a la salida el dato completo (bit start, dato ASCII, bit paridad par,
bit stop),
    --Pero antes se convierte el dato a bit_vector de manera de
poder acoplarle los bits correspondientes de paridad par, start y stop, una
vez hecho esto,
    --se vuelve a convertir a std_logic_vector y se brinda a la
salida.
    rom_proc : process (clk)
    begin
        if (rising_edge(clk) and clk = '1') then
            dato_ok          <= to_stdlogicvector('0' &
to_bitvector(mem(to_integer(unsigned(address)))) &
bit_paridad_par(to_bitvector(mem(to_integer(unsigned(address)))))) & '1');
        end if;
    end process rom_proc;
    data_out <= dato_ok;
end architecture behav;
```

4.2.2 Package memoria

```
library ieee;
```

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

package pkg_rom is

    constant data_length      : natural := 8;      --Constante de longitud de
dato.

    constant data_length_ok   : natural := 11;     --Constante de longitud de
datos, incluido el bit paridad, start y stop.

    constant addr_length      : natural := 6;      --Constante de longitud de
dirección.

    constant mem_size          : natural := 2**addr_length;  --
Constante de tamaño de memoria.

--DATOS MEMORIA ROM
subtype rom_word is std_logic_vector(data_length-1 downto 0);

type mem_type is array (mem_size-1 downto 0) of rom_word;

constant mem : mem_type :=
(
    --0 => x"4c",--L
    --1 => x"61",--a
    --2 => x"62",--b
    --3 => x"6f",--o
    --4 => x"72",--r
    --5 => x"61",--a
    --6 => x"74",--t
    --7 => x"6f",--o
    --8 => x"72",--r
    --9 => x"69",--i
    --10 => x"6f",--o
    --11 => x"20",--_
    --12 => x"34",--4
    --13 => x"20",--_
    --14 => x"43",--C
    --15 => x"6f",--o
    --16 => x"6d",--m
    --17 => x"70",--p
    --18 => x"6c",--l
    --19 => x"65",--e
```

```
--          20 => x"74", --t
--          21 => x"61", --a
--          22 => x"64", --d
--          23 => x"6f", --o

--          24 => x"53", --S
--          25 => x"69", --i
--          26 => x"6d", --m
--          27 => x"6f", --o
--          28 => x"6e", --n

--
--          29 => x"50", --P
--          30 => x"61", --a
--          31 => x"62", --b
--          32 => x"6c", --l
--          33 => x"6f", --o

--
--          34 => x"46", --F
--          35 => x"50", --P
--          36 => x"47", --G
--          37 => x"41", --A
--          38 => x"2E", --.

          0 => x"32", --L
          1 => x"86", --a
          2 => x"46", --b
          3 => x"f6", --o
          4 => x"4e", --r
          5 => x"86", --a
          6 => x"2e", --t
          7 => x"f6", --o
          8 => x"4e", --r
          9 => x"96", --i
         10 => x"f6", --o
         11 => x"04", --
         12 => x"2c", --4
         13 => x"04", --
         14 => x"c2", --C
         15 => x"f6", --o
```

```
16 => x"b6", --m
17 => x"0e", --p
18 => x"36", --l
19 => x"a6", --e
20 => x"2e", --t
21 => x"86", --a
22 => x"26", --d
23 => x"f6", --o

24 => x"ca", --s
25 => x"96", --i
26 => x"b6", --m
27 => x"f6", --o
28 => x"76", --n

29 => x"0a", --p
30 => x"86", --a
31 => x"46", --b
32 => x"36", --l
33 => x"f6", --o

34 => x"62", --f
35 => x"0a", --p
36 => x"e2", --g
37 => x"82", --a
38 => x"74", --.

others => x"00"

);

function bit_paridad_par (dato: in bit_vector) return bit;

end package;

package body pkg_rom is
```

--Funcion paridad par, la misma opera de la siguiente manera:

--Dicha funcion tiene como entrada un dato de tipo bit_vector mientras que la misma arroja como dato un bit.

--La funcion realiza un loop en todo el rango del bit_vector y realiza la funcion XOR entre cada uno de ellos con un bit de testeo el cual,

--para obtener un bit de paridad par, el bit de testeo debe ser '1' al inicio del programa, si se desea obtener un bit de paridad impar el bit de testeo

--debe ser '0'.

```
function bit_paridad_par (dato: in bit_vector) return bit is
    variable bit_test : bit;
begin
    bit_test := '1';
    for i in dato'range loop
        bit_test := bit_test xor dato(i);
    end loop;
    return bit_test;
end function;
```

```
end package body pkg_rom;
```

4.2.3 Contador Memoria

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

entity Contador_memoria is
    port (
        entradas          :in std_logic_vector(4 downto 0);    --
        Señales que seleccionan la frase a buscar en memoria.
        clk               :in std_logic;
        --Señal de reloj.
        reset_low         :in std_logic;
        --Señal de reset en bajo.
        send_ready        :out std_logic;
        --Señal de finalizacion del contador.
        address_out       :out std_logic_vector (5 downto 0) --Salida
        que representa la direccion del caracter en memoria ROM.
    );
end Contador_memoria;
```

```
architecture behav of Contador_memoria is

    --Señales de conteo.
    signal cont :integer range 0 to 38;

    --Señal de seleccion de opcion de palabras.
    signal opcion      :std_logic_vector(4 downto 0);
    signal proceso     :std_logic_vector(4 downto 0);

    --Señales auxiliar de salida
    signal contToSend_ready      :std_logic;
    signal contToaddress_out      :std_logic_vector (5 downto 0);

begin

    --En este apartado a traves de latchs se selecciona el tipo de opcion
    --del contador en base a las entradas.

    opciones: for i in 0 to 4 generate
        opcion(i)  <=  '1'  when  entradas(i)  =  '1'  else  '0'  when
        (reset_low = '0');
    end generate opciones;

    --En esta seccion se asegura en base a un with select que no se hayan
    --seleccionado al mismo instante dos opciones.

    --Es una manera de asegurarse que se seleccione una opcion del
    --contador.

    with opcion select
        proceso <=  "00001"  when  "00001",
                    "00010"  when  "00010",
                    "00100"  when  "00100",
                    "01000"  when  "01000",
                    "10000"  when  "10000",
                    "00000"  when  others;

    --El contador opera de la siguiente manera:
    --En base a una opcion el contador debe arrojar a la salida la
    --direccion de memoria correspondiente al caracter.
```

--Como previamente se conoce la ubicacion de cada caracter en memoria y ademas el inicio y final de cada frase,

--el contador proporciona el lugar de memoria del primer caracter inicio de cada frase hasta su caracter final, lo cual en

--ese instante se activa una bandera de finalizacion por parte del contador.

```
contador: process (clk,reset_low)
begin
    if (reset_low = '0') then
        cont <= 0;
        contTosend_ready <= '0';
        contToaddress_out <= "111111";
    elsif (clk'event and clk = '1') then
        if((cont = 24 and proceso = "00001") or (cont = 5
and proceso = "00010") or (cont = 5 and proceso = "00100") or (cont = 4 and
proceso = "01000") or (cont = 38 and proceso = "10000")) then
            contTosend_ready <= '1';
            contToaddress_out <= "100110"; --Direccion de memoria correspondiente al caracter punto '.';
        else
            contTosend_ready <= '0';
            case proceso is
                when "00001" => contToaddress_out <=
conv_std_logic_vector(cont,6);
                cont
                <= cont + 1;
                when "00010" => contToaddress_out <=
conv_std_logic_vector(cont+24,6);
                cont
                <= cont + 1;
                when "00100" => contToaddress_out <=
conv_std_logic_vector(cont+29,6);
                cont
                <= cont + 1;
                when "01000" => contToaddress_out <=
conv_std_logic_vector(cont+34,6);
                cont
                <= cont + 1;
                when "10000" => contToaddress_out <=
conv_std_logic_vector(cont,6);
                cont
                <= cont + 1;
                when others => contToaddress_out <=
"ZZZZZZ";
            end case;
        end if;
    end if;
end process;
```

```
        end case;
    end if;
end if;
end process;

send_ready <= contToSend_ready;
address_out <= contToaddress_out;

end behav;
```

4.2.4 FSM memoria

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

entity FSM_Memoria is
port (
    botones           :in std_logic_vector(4 downto 0);
    --Interruptores fisicos que seleccionan la frase a transmitir
    clk               :in std_logic;
    --Señal de reloj.
    send_MEF1         :in std_logic;
    --Señal que avisa que la transmision de un caracter
    ha finalizado.
    send_OK           :in std_logic;
    --Señal que indica que el envio de la frase se
    ha completado.
    reset_low         :in std_logic;
    --Señal de reset en bajo.
    opciones          :out std_logic_vector(4 downto 0);
    --Señal que indica que frase se selecciono.
    clk_cont          :out std_logic;
    --Señal de reloj del contador.
    clk_mem           :out std_logic;
    --Señal de reloj de la memoria ROM.
    start_tx          :out std_logic;
    --Señal de comienzo de transmision de un caracter.
    reset_cont         :out std_logic;
    --Señal de reset de contador.
    preset             :out std_logic;
    --Señal de seteo asincronico para el registro de
    desplazamiento.
    tri_state          :out std_logic
    --Señal tri_state de salida del proyecto RS-232.
);
end FSM_Memoria;
```

```
architecture behav of FSM_Memoria is
    --Declaro estados de la FSM.
    type FSM_states is (idle, ready, entrada_0, entrada_1, entrada_2,
    entrada_3, entrada_4, contador, memoria, start, espera);

    --Declaro las señales de la FSM.
    signal current_state, next_state : FSM_states;

    --Establezco el estilo de codificacion.
    attribute syn_encoding :string;
    attribute syn_encoding of FSM_states : type is "one_hot";

begin
    --Proceso de estado presente.
    cs_pr: process (clk,reset_low)
        begin
            if reset_low = '0' then
                current_state <= idle;
            elsif (rising_edge(clk)) then
                current_state <= next_state;
            end if;
        end process;

    --Proceso de proximo estado.
    nxs_pr: process (current_state, send_MEF1, send_OK, botones)
        begin
            case current_state is
                when idle =>
                    next_state <= ready;
                when ready =>
                    case botones is
                        when "00001" => next_state <= entrada_0;
                        when "00010" => next_state <= entrada_1;
                        when "00100" => next_state <= entrada_2;
                        when "01000" => next_state <= entrada_3;
                        when "10000" => next_state <= entrada_4;
                        when others => next_state <= ready;
                    end case;
            end case;
        end process;
end;
```

```
        end case;
when entrada_0 =>
    next_state <= contador;
when entrada_1 =>
    next_state <= contador;
when entrada_2 =>
    next_state <= contador;
when entrada_3 =>
    next_state <= contador;
when entrada_4 =>
    next_state <= contador;
when contador =>
    next_state <= memoria;
when memoria =>
    next_state <= start;
when start =>
    if (send_MEF1 = '0') then
        next_state <= espera;
    elsif(send_OK = '0') then
        next_state <= contador;
    else
        next_state <= ready;
    end if;
when espera =>
    if (send_MEF1 = '0') then
        next_state <= espera;
    elsif(send_OK = '0') then
        next_state <= contador;
    else
        next_state <= ready;
    end if;
when others => next_state <= idle;
end case;
end process;

--Proceso de salida Moore.
moore_pr: process (current_state)
begin
    --Valores por default de salidas.
```

```
opciones <= "00000";
clk_cont      <= '0';
clk_mem <= '0';
start_tx <= '0';
reset_cont <= '1';
preset <= '0';
tri_state <= '0';
case current_state is
    when idle =>
        preset <= '1';
        tri_state <= '1';
    when ready =>
        reset_cont <= '0';
    when entrada_0 =>
        opciones(0) <= '1';
    when entrada_1 =>
        opciones(1) <= '1';
    when entrada_2 =>
        opciones(2) <= '1';
    when entrada_3 =>
        opciones(3) <= '1';
    when entrada_4 =>
        opciones(4) <= '1';
    when contador =>
        clk_cont <= '1';
    when memoria =>
        clk_mem <= '1';
    when start =>
        start_tx <= '1';
    when others => null;
end case;
end process;

end behav;
```

4.2.5 Punto B

```
library ieee;
use ieee.std_logic_1164.all;
use work.pkg_rom.all;
```

```
entity Punto_B is
    port(
        entradas      :in std_logic_vector(4 downto 0);
        --Entradas fisicas que seleccionan la frase a enviar.
        clk           :in std_logic;
        --Señal de reloj.
        reset_low     :in std_logic;
        --Señal de reset en bajo.
        send_MEF1    :in std_logic;
        --Señal de enviado de caracter
ASCII por parte de la MEF 1.
        datos_out    :out std_logic_vector(10 downto 0);
        --Señal de dato completo a enviar.
        preset        :out std_logic;
        --Señal asincronica de seteo del
registro.
        tri_state     :out std_logic;
        --Señal de tri_state de salida en el
punto A.

        --Estas señales se utilizaron para hacer un analisis mas
exhaustivo en el test bench.
        --Si se desea se pueden comentar, haciendolo ademas con
las ultimas tres lineas de este codigo.
        enviado_cont:out std_logic;
        enviar         :out std_logic;
        direccion     :out std_logic_vector(addr_length-1 downto 0)

    );
end Punto_B;

architecture behav of Punto_B is

    --Declaro componentes.
    component Memoria_ROM is
        port(
            clk           :in std_logic;
            address       :in      std_logic_vector(addr_length-1
downto 0);
            data_out      :out      std_logic_vector(data_legh_ok-1
downto 0)
        );
    end component;
```

```
component FSM_Memoria is
    port (
        botones           :in      std_logic_vector(4
downto 0);
        clk               :in      std_logic;
        send_MEF1         :in      std_logic;
        send_OK           :in      std_logic;
        reset_low         :in      std_logic;
        opciones          :out     std_logic_vector(4
downto 0);
        clk_cont          :out    std_logic;
        clk_mem           :out    std_logic;
        start_tx          :out    std_logic;
        reset_cont         :out    std_logic;
        preset             :out    std_logic;

        tri_state          :out    std_logic
    );
end component;

component Contador_Memoria is
    port (
        entradas          :in      std_logic_vector(4 downto 0);
        clk               :in      std_logic;
        reset_low         :in      std_logic;
        send_ready        :out    std_logic;
        address_out       :out    std_logic_vector (5 downto 0)
    );
end component;

--Declaro señales
signal contTofsm            : std_logic;
                                --Señal de termino del contador. O
sea, termino la frase.
signal fsmTocont             : std_logic_vector(4 downto 0);
                                --Señales que indican la oracion a enviar.
signal clk_contador          :std_logic;
                                --Señal de clock de la memoria y
el contador.
signal clk_memoria           :std_logic;
```

```

    signal start                      :std_logic;
    signal reset_cont                 :std_logic;
                                         --Señal de reset del contador.
    signal contTomem                  :std_logic_vector(addr_length-1      downto
0);   --Señal de dirección del contador a memoria.

begin

    Memoria: Memoria_ROM port map (clk  => clk_memoria, address  =>
contTomem, data_out => datos_out);

    FSM: FSM_Memoria port map(botones => entradas, clk => clk, send_MEF1
=> send_MEF1, send_OK => contTofsm, reset_low => reset_low, opciones =>
fsmTocont,
                                         clk_cont      =>
clk_contador, clk_mem => clk_memoria, start_tx => start, reset_cont =>
reset_cont, preset => preset, tri_state => tri_state);

    Contador: Contador_Memoria port map(entradas => fsmTocont, clk =>
clk_contador, reset_low => reset_cont, send_ready => contTofsm, address_out
=> contTomem);

    enviar <= start;
    dirección <= contTomem;
    enviado_cont <= contTofsm;

end behav;

```

4.2.6 Test-bench Punto B

```

library ieee;
use ieee.std_logic_1164.all;
use work.pkg_rom.all;

entity tb_Punto_B is
end tb_Punto_B;

architecture behav of tb_Punto_B is

component Punto_B is
    port (
        entradas      :in std_logic_vector(4 downto 0);
        clk           :in std_logic;
        reset_low     :in std_logic;

```

```
    send_MEF1      :in std_logic;
    datos_out     :out std_logic_vector(10 downto 0);

    enviado_cont:out std_logic;
    enviar         :out std_logic;
    direccion     :out      std_logic_vector(addr_length-1
downto 0)
);

end component;

--Declaro señales auxiliares.

signal tb_entradas      : std_logic_vector(4 downto 0) := "00000";
signal tb_clk            :std_logic := '0';
signal tb_reset_low      :std_logic := '0';
signal tb_send_MEF1      :std_logic := '0';

signal tb_datos_out      :std_logic_vector(10 downto 0);
signal tb_enviado_cont   :std_logic;
signal tb_enviar          :std_logic;
signal tb_direccion       :std_logic_vector(addr_length-1 downto 0);

constant semiperiod : time := 50us;

begin

clk_gen: tb_clk <= not tb_clk after semiperiod;

uut: Punto_B port map (entradas => tb_entradas, clk => tb_clk,
reset_low  => tb_reset_low, send_MEF1  => tb_send_MEF1, datos_out  =>
tb_datos_out,
                                         enviado_cont           =>
tb_enviado_cont, enviar => tb_enviar, direccion => tb_direccion);

data_gen: process
begin

--Damos una entrada correspondiente a la ultima frase.
tb_entradas <= "01000";
tb_reset_low <= '1';
for i in 0 to 15 loop
```

```
        wait until tb_clk = '1';
    end loop;

    --La MEF que controla al registro envia el caracter...
    tb_send_MEF1 <= '1';
    wait until tb_clk = '1';

--Y una vez finalizado, enviamos una entrada invalida.
tb_entradas <= "00011";
--wait until tb_clk = '1';
--wait until tb_clk = '1';
--wait until tb_clk = '1';

--Reseteamos y comenzamos de vuelta.
tb_reset_low <= '0';
wait until tb_clk = '1';

end process;
end behav;
```

4.3 Receptor

4.3.1 UART RX

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

entity UART_RX is
    generic (
        clksXbit : integer
    );
    port (
        clk      : in  std_logic;
        rx_serial : in  std_logic;
        rx_byte   : out std_logic_vector(7 downto 0)
    );
end UART_RX;
```

```
architecture behav of UART_RX is

type FSM_states is (idle, start, data_bit, stop, espera);
signal state      : FSM_states := idle;

attribute syn_encoding      :string;
attribute syn_encoding of FSM_states : type is "one hot";

signal baud_cont      : integer range 0 to clksXbit-1 := 0;
signal bit_cont       : integer range 0 to 7 := 0;
signal rx              : std_logic_vector(7 downto 0) := (others => '0');

begin

proc : process (clk)
begin
if rising_edge(clk) then

case state is

when idle =>
baud_cont <= 0;
bit_cont <= 0;

if rx_serial = '0' then
state <= start;
else
state <= idle;
end if;

when start =>
if baud_cont = (clksXbit-1)/2 then
if rx_serial = '0' then
baud_cont <= 0;
state  <= data_bit;
else
state <= idle;
end if;
```

```
        else
            baud_cont <= baud_cont + 1;
            state    <= start;
        end if;

when data_bit =>
    if baud_cont < clksXbit-1 then
        baud_cont <= baud_cont + 1;
        state    <= data_bit;
    else
        baud_cont           <= 0;
        rx(bit_cont) <= rx_serial;

        if bit_cont < 7 then
            bit_cont <= bit_cont + 1;
            state    <= data_bit;
        else
            bit_cont <= 0;
            state    <= stop;
        end if;
    end if;

when stop =>
    if baud_cont < clksXbit-1 then
        baud_cont <= baud_cont + 1;
        state    <= stop;
    else
        baud_cont <= 0;
        state    <= espera;
    end if;

when espera =>
    state <= idle;
when others =>
    state <= idle;

end case;
end if;
end process;
```

```
rx_byte <= rx;

end behav;

4.3.2 Mux Paralelos
library ieee;
use      ieee.std_logic_1164.all;
use      ieee.numeric_std.all;

entity mux_paralelos is
port(
    --Seleccion de datos.
    sel: in std_logic_vector (2 downto 0);
    --Entrada de datos.
    in1, in2, in3, in4, in5, in6, in7, in8: in std_logic_vector (7 downto
0);
    --Salida.
    out_8mx: out std_logic_vector (7 downto 0)
);
end mux_paralelos;

architecture behav of mux_paralelos is

component mux is
generic (ent_sel: integer:= 3);
port(
    mux_input    : in  std_logic_vector ((2**ent_sel)-1 downto
0);
    mux_sel       : in  std_logic_vector (ent_sel-1 downto
0);
    mux_out       : out std_logic
);
end component;

end component;

type arreglo is array (7 downto 0) of std_logic_vector (7 downto 0);
signal mi_arreglo: arreglo;
```

```
begin
    t:for i in out_8mx'range generate
        mi_arreglo(i) <= in1(i) & in2(i) & in3(i) & in4(i) & in5(i) &
        in6(i) & in7(i) & in8(i);

        u1: mux port map(
            mux_input => mi_arreglo(i),
            mux_out => out_8mx(i),
            mux_sel => sel
        );
    end generate t;

end behav;
```

4.3.3 Mux

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

entity Mux is
    generic (ent_sel: integer:= 3);
    port (
        mux_input : in std_logic_vector ((2**ent_sel)-1 downto
0);
        mux_sel      : in std_logic_vector (ent_sel-1 downto
0);
        mux_out      : out std_logic
    );
end Mux;
```

```
architecture behav of Mux is
begin
    mux_out <= mux_input (to_integer(unsigned(mux_sel)));
end behav;
```

4.3.4 RX

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

entity rx is
```

```
port (
    clk          :in std_logic;
    --Entrada de clk.
    clr          :in std_logic;
    --Entrada de clr.
    rx           :in std_logic;
    --Entrada del receptor.
    sel          :in std_logic_vector(1 downto 0);  --
    Entrada de seleccion del mux paralelo.

    byte_out     :out std_logic_vector(7 downto 0)   --
    Extraccion del byte para LCD
);
end rx;

architecture behav of rx is

type arreglo is array (3 downto 0) of integer;
constant baudios :arreglo := (10417,5209,1303,435);

type arreglo2 is array (3 downto 0) of std_logic_vector(7 downto 0);
signal v      :arreglo2;

component UART_RX is
generic(
    clksXbit : integer
);
port (
    clk       : in  std_logic;
    rx_serial : in  std_logic;
    rx_byte   : out std_logic_vector(7 downto 0)
);
end component;

signal byte :std_logic_vector(7 downto 0);

begin
--uart: UART_RX port map(clk => clk, rx_serial => rx, rx_byte =>
byte);
t: for i in 0 to 3 generate
    uart: UART_RX      generic map(clksXbit => baudios(i))
```

```
port map(clk => clk, rx_serial => rx, rx_byte
=> v(i));
end generate t;

mux: with sel select
    byte <=      v(2) when "01",
                  v(1) when "10",
                  v(0) when "11",
                  v(3) when others;

byte_out <= byte;
end behav;
```

4.3.5 Test-bench UART RX

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

entity UART_RX_TB is
end UART_RX_TB;

architecture behav of UART_RX_TB is

constant semiperiod : time := 10 ns;

-- Queremos una interfaz de 115200
-- 25000000 / 115200 = 217 Clocks Por Bit.
constant clksXbit : integer := 435;

-- 1/115200:
constant perXbit : time := 8680 ns;

signal tb_clock      : std_logic := '0';
signal tb_rx_serial: std_logic := '1';
signal tb_rx_byte    : std_logic_vector(7 downto 0);

procedure send_byte (
    data_in          : in  std_logic_vector(7 downto 0);
    signal serial_out : out std_logic) is
```

```
begin

    -- Envio el bit de start
    serial_out <= '0';
    wait for perXbit;

    -- Envio el dato
    for ii in 0 to 7 loop
        serial_out <= data_in(ii);
        wait for perXbit;
    end loop; -- ii

    -- Envio el bit de stop
    serial_out <= '1';
    wait for perXbit;
end send_byte;

begin

    -- Instancio el receptro UART
    uut : entity work.UART_RX
    generic map (
        clksXbit => clksXbit
    )
    port map (
        clk      => tb_clock,
        rx_serial => tb_rx_serial,
        rx_byte   => tb_rx_byte
    );
    clock_generate: tb_clock <= not tb_clock after semiperiod;

    data_gen: process
    begin
        -- Envio un comando UART
        wait until rising_edge(tb_clock);
        send_byte(x"37", tb_rx_serial);
        wait until rising_edge(tb_clock);
```

```
-- Chequeo si el correspondiente byte fue enviado con éxito.  
if tb_rx_byte = X"37" then  
    report "Test aprobado - Correcta recepción de byte" severity note;  
else  
    report "Test desaprobado - Incorrecta recepción de byte" severity note;  
end if;  
  
assert false report "Tests Completado" severity failure;  
  
end process;  
  
end behav;
```

4.4 LCD

4.4.1 Contador binario LCD

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity cont_bin_lcd is  
    port (  
        clk           :in std_logic;      --Entrada de reloj.  
        reset_low    :in std_logic;      --Entrada de reset en bajo.  
        ok           :out std_logic     --Salida la cual  
se usa como flag de término de cuenta (en este caso 9).  
    );  
end cont_bin_lcd;  
  
architecture behav of cont_bin_lcd is  
  
--Señal de conteo.  
signal cont :integer range 0 to 100000000;  
  
--Señal de busd de salida.  
signal bus_out    :std_logic := '0';  
  
begin
```

```
contador: process(clk,reset_low)
begin
    if (reset_low = '0') then
        cont <= 0;
        bus_out <= '0';
    elsif (clk'event and clk = '1') then
        if (cont = 100000000) then
            cont <= 0;
            bus_out <= '1';
        else
            cont <= cont + 1;

        end if;
    end if;
end process;

ok <= bus_out;

end behav;
```

4.4.2 FSM LCD

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

entity FSM_LCD is
    generic(clk_divisor: integer := 100000); -- 50Mhz a 500Hz.
    port(
        clk           :in std_logic;
        --Reloj del sistema.
        reset_low     :in std_logic;
        --Reinicio activo en bajo.
        cont_ok       :in std_logic;
        --Entrada del contador binario.
        entradas      :in std_logic_vector(4 downto 0); --Entradas de seleccion de oraciones.
        byte          :in std_logic_vector(7 downto 0); --Byte recibido del receptor.

        rw            :out std_logic;
        --Escritura/lectura.
```

```
        rs          :out std_logic;
        --Datos/Instrucciones.

        reset_cont :out std_logic;
        --Reset en bajo del contador binario.

        e           :buffer std_logic := '0';
        --Señal de habilitacion del modulo LCD.

        db          :out std_logic_vector(7 downto 0)
        --Señales de datos para el LCD.
        --
    );
end FSM_LCD;

architecture behav of FSM_LCD is

    --Estados de la FSM y sus respectivas señales.

    type FSM_states is (    FunctionSet1,      FunctionSet2,      FunctionSet3,
FunctionSet4,DisplayControl,EntryMode,
                           ClearDisplay1,ClearDisplay2,ClearDisplay3,ClearDisplay4,
ClearDisplay5,
                           rx1,         rx2,         rx3,         rx4,
SetAddress,
                           ReturnHome1,     ReturnHome2,
ReturnHome3, ReturnHome4,
                           Write1,        Write2,        Write3,
Write4, Write5, Write6, Write7, Write8, Write9, Write10, Write11, Write12,
Write13,
                           Write24,        Write25,        Write26,
Write27, Write28,
                           Write29,        Write30,        Write31,
Write32, Write33,
                           Write34,        Write35,        Write36,
Write37);

    signal current_state, next_state : FSM_states;

    --Estilo de codificacion.

    attribute syn_encoding :string;
    attribute syn_encoding of FSM_states :type is "one hot";

begin

    --Generador de clock.
```

```
clock: process(clk)
    variable cont: integer range 0 to clk_divisor;
begin
    if(clk'event and clk = '1') then
        cont := cont + 1;
        if(cont = clk_divisor) then
            e <= not e;
            cont := 0;
        end if;
    end if;
end process;

--Proceso de estado presente.
cs_proc: process(e)
begin
    if (reset_low = '0') then
        current_state <= FunctionSet1;
    elsif (rising_edge(e)) then
        current_state <= next_state;
    end if;
end process;

--Proceso de proximo estado.
nx_proc: process(current_state, entradas, cont_ok)
variable flag          :bit := '0';
variable contador :integer range 0 to 1000 := 0;
begin
    case current_state is
        when FunctionSet1 =>
            next_state <= FunctionSet2;
        when FunctionSet2 =>
            next_state <= FunctionSet3;
        when FunctionSet3 =>
            next_state <= FunctionSet4;
        when FunctionSet4 =>
            next_state <= ClearDisplay1;
        when ClearDisplay1 =>
            next_State <= DisplayControl;
        when DisplayControl =>
```

```
        next_state <= EntryMode;
when EntryMode =>
        next_state <= ClearDisplay2;
when ClearDisplay2 =>
        next_state <= rx1;
when rx1 =>
        next_state <= rx2;
when rx2 =>
        next_state <= rx3;
when rx3 =>
        next_state <= rx4;
when rx4 =>
        next_state <= SetAddress;
when SetAddress =>
    case entradas is
        when "00001" =>
            next_state <= Write1;
        when "00010" =>
            next_state <= Write24;
        when "00100" =>
            next_state <= Write29;
        when "01000" =>
            next_state <= Write34;
        when "10000" =>
            flag := '1';
            next_state <= Write1;
        when others =>
            next_state <= ClearDisplay2;
    end case;
when Write1 =>
        next_state <= Write2;
when Write2 =>
        next_state <= Write3;
when Write3 =>
        next_state <= Write4;
when Write4 =>
        next_state <= Write5;
when Write5 =>
        next_state <= Write6;
```

```
when Write6 =>
    next_state <= Write7;
when Write7 =>
    next_state <= Write8;
when Write8 =>
    next_state <= Write9;
when Write9 =>
    next_state <= Write10;
when Write10 =>
    next_state <= Write11;
when Write11 =>
    next_state <= Write12;
when Write12 =>
    next_state <= Write13;
when Write13 =>
    next_state <= ReturnHome1;
when ReturnHome1 =>
    if (cont_ok = '1') then
        if (flag = '1') then
            next_state <= ClearDisplay3;
        else
            next_state <= ClearDisplay2;
        end if;
    else
        next_state <= ReturnHome1;
    end if;
when ClearDisplay3 =>
    next_state <= Write24;
when Write24 =>
    next_state <= Write25;
when Write25 =>
    next_state <= Write26;
when Write26 =>
    next_state <= Write27;
when Write27 =>
    next_state <= Write28;
when Write28 =>
    next_state <= ReturnHome2;
when ReturnHome2 =>
```

```
        if (cont_ok = '1') then
            if (flag = '1') then
                next_state <= ClearDisplay4;
            else
                next_state <= ClearDisplay2;
            end if;
        else
            next_state <= ReturnHome2;
        end if;

when ClearDisplay4 =>
    next_state <= Write29;
when Write29 =>
    next_state <= Write30;
when Write30 =>
    next_state <= Write31;
when Write31 =>
    next_state <= Write32;
when Write32 =>
    next_state <= Write33;
when Write33 =>
    next_state <= ReturnHome3;
when ReturnHome3 =>
    if (cont_ok = '1') then
        if (flag = '1') then
            next_state <= ClearDisplay5;
        else
            next_state <= ClearDisplay2;
        end if;
    else
        next_state <= ReturnHome3;
    end if;

when ClearDisplay5 =>
    next_state <= Write34;
when Write34 =>
    next_state <= Write35;
when Write35 =>
    next_state <= Write36;
when Write36 =>
    next_state <= Write37;
```

```
when Write37 =>
    next_state <= ReturnHome4;
when ReturnHome4 =>
    flag := '0';
    if (cont_ok = '1') then
        next_state <= ClearDisplay2;
    else
        next_state <= ReturnHome4;
    end if;
when others =>
    next_state <= FunctionSet1;
end case;
end process;

--Proceso de salida Moore.
moore_proc: process (current_state)
begin
    rw <= '0';
    rs <= '0';
    reset_cont <= '0';
    db <= x"00";
    case current_state is
        when FunctionSet1 =>
            db <= x"38";
        when FunctionSet2 =>
            db <= x"38";
        when FunctionSet3 =>
            db <= x"38";
        when FunctionSet4 =>
            db <= x"38";
        when ClearDisplay1 =>
            db <= x"01";
        when DisplayControl =>
            db <= x"0c";
        when EntryMode =>
            db <= x"06";
        when ClearDisplay2 =>
            db <= x"01";
        when rx1 =>
```

```
        rs <= '1';
        db <= x"52";           --R
when rx2 =>
        rs <= '1';
        db <= x"78";           --x
when rx3 =>
        rs <= '1';
        db <= x"3A";           --:
when rx4 =>
        rs <= '1';
        db <= byte;
when SetAddress =>
        db <= x"C0";           --Me posiciono en la segunda
fila
when Write1 =>
        rs <= '1';
        db <= x"4C";
when Write2 =>
        rs <= '1';
        db <= x"61";
when Write3 =>
        rs <= '1';
        db <= x"62";
when Write4 =>
        rs <= '1';
        db <= x"6F";
when Write5 =>
        rs <= '1';
        db <= x"72";
when Write6 =>
        rs <= '1';
        db <= x"61";
when Write7 =>
        rs <= '1';
        db <= x"74";
when Write8 =>
        rs <= '1';
        db <= x"6F";
when Write9 =>
```

```
        rs <= '1';
        db <= x"72";
when Write10 =>
        rs <= '1';
        db <= x"69";
when Write11 =>
        rs <= '1';
        db <= x"6f";
when Write12 =>
        rs <= '1';
        db <= x"20";
when Write13 =>
        rs <= '1';
        db <= x"34";
when ReturnHome1 =>
        reset_cont <= '1';
        db <= x"80";
when ClearDisplay3 =>
        db <= x"01";
when Write24 =>
        rs <= '1';
        db <= x"53";
when Write25 =>
        rs <= '1';
        db <= x"69";
when Write26 =>
        rs <= '1';
        db <= x"6d";
when Write27 =>
        rs <= '1';
        db <= x"6f";
when Write28 =>
        rs <= '1';
        db <= x"6e";
when ReturnHome2 =>
        reset_cont <= '1';
        db <= x"80";
when ClearDisplay4 =>
        db <= x"01";
```

```
when Write29 =>
    rs <= '1';
    db <= x"50";
when Write30 =>
    rs <= '1';
    db <= x"61";
when Write31 =>
    rs <= '1';
    db <= x"62";
when Write32 =>
    rs <= '1';
    db <= x"6c";
when Write33 =>
    rs <= '1';
    db <= x"6f";
when ReturnHome3 =>
    reset_cont <= '1';
    db <= x"80";
when ClearDisplay5 =>
    db <= x"01";
when Write34 =>
    rs <= '1';
    db <= x"46";
when Write35 =>
    rs <= '1';
    db <= x"50";
when Write36 =>
    rs <= '1';
    db <= x"47";
when Write37 =>
    rs <= '1';
    db <= x"41";
when ReturnHome4 =>
    reset_cont <= '1';
    db <= x"80";
when others =>
    null;
end case;
end process;
```

```
end behav;
```

4.4.3 LCD main

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

entity LCD is
    port(
        clk           :in std_logic;
        reset_low     :in std_logic;
        entradas      :in std_logic_vector(4 downto 0);
        byte          :in std_logic_vector(7 downto 0);

        rw            :out std_logic;
        rs            :out std_logic;
        e             :buffer std_logic := '0';
        db            :out std_logic_vector(7 downto 0)
    );
end LCD;
```

```
architecture behav of LCD is
```

```
component FSM_LCD is
    port(
        clk           :in std_logic;
        reset_low     :in std_logic;
        cont_ok       :in std_logic;
        entradas      :in std_logic_vector(4 downto 0);
        byte          :in std_logic_vector(7 downto 0);

        rw            :out std_logic;
        rs            :out std_logic;
```

```
        reset_cont  :out std_logic;
        e           :buffer  std_logic  :=  '0';
        db          :out     std_logic_vector(7
downto 0)
    );
end component;

component cont_bin_lcd is
port(
    clk           :in std_logic;
    reset_low    :in std_logic;
    ok           :out std_logic
);
end component;

signal contTofsm :std_logic;
signal fsmTocont :std_logic;

begin

fsm: FSM_LCD port map(clk => clk, reset_low => reset_low, cont_ok =>
contTofsm, entradas => entradas, byte => byte, rw => rw, rs => rs,
reset_cont => fsmTocont, e => e, db => db);

cont: cont_bin_lcd port map(clk => clk, reset_low => fsmTocont, ok =>
contTofsm);

end behav;
```

4.4.4 Test-bench LCD

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

entity tb_LCD is
end tb_LCD;

architecture behav of tb_LCD is
component LCD is
port(
```

```
clk           :in std_logic;
reset_low    :in std_logic;
entradas     :in std_logic_vector(4 downto 0);
byte         :in std_logic_vector(7 downto 0);

rw           :out std_logic;
rs           :out std_logic;
e            :buffer std_logic;
db           :out      std_logic_vector(7
downto 0)
);

end component;

signal tb_clk           :std_logic := '0';
signal tb_reset_low     :std_logic := '1';
signal tb_entradas      :std_logic_vector(4 downto 0);
signal tb_byte          :std_logic_vector(7 downto 0);
signal tb_rw             :std_logic;
signal tb_rs             :std_logic;
signal tb_e              :std_logic := '0';
signal tb_db             :std_logic_vector(7 downto 0);

constant semiperiod     :time :=10us;

begin
  uut: LCD port map(clk => tb_clk, reset_low => tb_reset_low, entradas
=> tb_entradas, byte => tb_byte,
                     rw  => tb_rw, rs  => tb_rs, e   =>
                     tb_e, db => tb_db);

  clk_gen: tb_clk <= not tb_clk after semiperiod;

  data_gen: process
  begin
    tb_byte <= x"32";
    tb_entradas <= "10000";
  end;

```

```
        wait until tb_clk = '1';
--      for i in 0 to 8 loop
--          wait until tb_clk = '1';
--      end loop;
--
--      tb_entradas <= "00001";
--      wait until tb_clk = '1';

--      tb_entradas <= "00010";
--      for i in 0 to 8 loop
--          wait until tb_clk = '1';
--      end loop;

    end process;

end behav;
```

4.5 RS 232

4.5.1 RS 232 Main

```
library ieee;
use ieee.std_logic_1164.all;
use work.pkg_rom.all;

entity RS_232 is
    port (
        entry           :in std_logic_vector(4 downto 0);    --
Entradas para seleccionar la frase a transmitir.

        reset_low      :in std_logic;
        --Señal de reset en bajo.

        clk            :in std_logic;
        --Señal de reloj.

        clk_select    :in std_logic_vector(1 downto 0);    --
Entrada
de seleccion de reloj de sistema (baudios).

        rx_serial     :in std_logic;
        --Entrada serial de datos.

        salida         :out std_logic;
        --Señal de salida de registro serie.

        --Modulo LCD.
```

```
rw          :out std_logic;
--Escritura/lectura.

rs          :out std_logic;
--Datos/Instrucciones.

e           :buffer std_logic;
--Señal de hanilitacion del modulo LCD.

db          :out std_logic_vector(7 downto 0); --
Señales de datos para el modulo LCD.

--Estas señales se añadieron para evaluar de manera mas
eficiente el test bench y entender que procesos
--actuaban de por medio, se pueden eliminar a traves de
comentarios, haciendolo ademas con las ultimas
--cinco lineas de este codigo.

clk_en_uso      :out std_logic;
address         :out std_logic_vector(addr_length-
1 downto 0);

cont_finished    :out std_logic;
enviado_MEF1    :out std_logic;
enviar          :out std_logic;
ASCII           :out std_logic_vector(10 downto 0)
);

end RS_232;

architecture behav of RS_232 is

--Declaro parte A.

component Punto_A is
generic (ancho : integer :=11);
port (
start_tx        :in std_logic;      --Señal      de
comienzo de Tx.

clk             :in std_logic; --Señal de clock.

reset_low       :in std_logic;     --Señal      de      reset
asincronico.

preset          :in std_logic;
data            :in std_logic_vector (ancho-1
downto 0); --Dato completo, 11 bits: paridad,start,stop,ASCII.

tri_state       :in std_logic;
serout          :out std_logic;     --Señal      de
salida serie.

finished        :out std_logic
```

```
        );
end component;

--Declaro parte B.

component Punto_B is
    port (
        entradas      :in std_logic_vector(4 downto 0);
        clk           :in std_logic;
        reset_low     :in std_logic;
        send_MEF1    :in std_logic;
        datos_out    :out std_logic_vector(10 downto 0);
        preset        :out std_logic;
        tri_state     :out std_logic;

        enviado_cont:out std_logic;
        enviar         :out std_logic;
        direccion     :out      std_logic_vector(addr_length-1
downto 0)
    );
end component;

--Declaro divisores de frecuencia.

component divisores is
    port (
        clk           :in std_logic;
        clk_out_0    :out std_logic;    --4800 baud
        clk_out_1    :out std_logic;    --9600 baud
        clk_out_2    :out std_logic;    --38400 baud
        clk_out_3    :out std_logic;    --115200 baud
    );
end component;

--Declaro multiplexer.

component Mux is
    generic (ent_sel: integer:= 2);
    port (
        mux_input     : in  std_logic_vector ((2**ent_sel)-1
downto 0);

```

```
        mux_sel      : in std_logic_vector (ent_sel-1
downto 0);

        mux_out      : out std_logic
    );

end component;

--Declaro LCD.

component LCD is
    port (
        clk           :in std_logic;

        reset_low    :in std_logic;

        entradas     :in std_logic_vector(4 downto 0);
        byte         :in std_logic_vector(7 downto 0);

        rw           :out std_logic;

        rs           :out std_logic;

        e            :buffer std_logic := '0';

        db           :out      std_logic_vector(7
downto 0)
    );
end component;

--Declaro antirrebote.

component antirrebote is
    port (
        x            :in std_logic;
        clk          :in std_logic;
        y            :buffer std_logic
    );
end component;

--Declaro sincronizador.

component sincronizador is
    port(
        asyncin      :in std_logic;
        clock        :in std_logic;
```

```
        reset      :in std_logic;
        syncin     :out std_logic
    );
end component;

--Declaro top receptor
component rx is
port (
    clk          :in std_logic;
    --Entrada de clk.
    clr          :in std_logic;
    --Entrada de clr.
    rx           :in std_logic;
    --Entrada del receptor.
    sel          :in std_logic_vector(1 downto 0);  --
    Entrada de seleccion del mux paralelo.

    byte_out     :out std_logic_vector(7 downto 0)   --
    Extraccion del byte para LCD
);
end component;

--Declaro Receptor UART
component UART_RX is
generic (
    clksXbit : integer
);
port (
    clk       : in  std_logic;
    rx_serial : in  std_logic;
    rx_byte   : out std_logic_vector(7 downto 0)
);
end component;

--Declaro controlador de display 7 segmentos
component mod_7seg is
port(
    x          :in std_logic_vector(15 downto 0);
    clk       :in std_logic;
    clr       :in std_logic;

```

```
        salida:out std_logic_vector(6 downto 0);
        dig   :out std_logic_vector(3 downto 0);
        dp    :out std_logic
      );
end component;

--Declaro señales auxiliares
signal enviado          :std_logic;
                           --Señal de caracter ASCII enviado
por parte de Punto_A.

signal dato              :std_logic_vector(10 downto 0);
                           --Señal RS232 a transmitir.

signal enviar_dato       :std_logic;
                           --Comienzo de transmision de dato ASCII.

signal reg_out           :std_logic;
                           --Salida serie del registro de
desplazamiento.

signal direccion         :std_logic_vector(addr_length-1 downto 0);
                           --Direccion de memoria ROM.

signal enviadoXcont      :std_logic;
                           --Termino de envio de la frase completa.

signal set               :std_logic;
                           --Señal para precargar el registro
de desplazamiento.

signal triState          :std_logic;
                           --Señal de triState de salida del RS232.

signal clocks            :std_logic_vector(3 downto 0);
                           --Señales de reloj proveniente de divisores de
frecuencia.

signal clk_RS_232         :std_logic;
                           --Señal de clock para RS232.

signal antirrebTosinc:std_logic_vector(4 downto 0);
                           --Señales de antirebotes a sincronizadores.

signal sincToRS232        :std_logic_vector(4 downto 0);
                           --Señales de sincronizadores a RS232 y LCD.

signal byte_for_lcd       :std_logic_vector(7 downto 0);
                           --Señal que conecta el rx con LCD para entrega del byte
recibido.

begin

antirrebotes: for i in 0 to 4 generate
begin
  antirreb: antirrebot port map(x => entry(i), clk =>
clk_RS_232, y => antirrebTosinc(i));

```

```

    end generate;

    sincronizadores: for i in 0 to 4 generate
        begin
            sinc: sincronizador port map(asyncin      =>
antirrebTosinc(i), clock => clk_RS_232, reset => reset_low, syncin =>
sincToRS232(i));
        end generate;

    Parte_A: Punto_A port map(start_tx => enviar_dato, clk => clk_RS_232,
reset_low => reset_low, data => dato, serout => reg_out, finished =>
enviado, preset => set, tri_state =>
triState);

    Parte_B: Punto_B port map(entradas => sincToRS232, clk => clk_RS_232,
reset_low => reset_low, send_MEF1 => enviado, datos_out => dato,
enviado_cont => enviadoXcont,
                           enviar      =>
enviar_dato, direccion => direccion, preset => set, tri_state => triState);

    Clock: divisores port map(clk => clk, clk_out_0 => clocks(0),
clk_out_1 => clocks(1), clk_out_2 => clocks(2), clk_out_3 => clocks(3));

    Multiplexer: Mux port map(mux_input => clocks, mux_sel => clk_select,
mux_out => clk_RS_232);

    mod_lcd: LCD port map(clk => clk, reset_low => reset_low, entradas =>
sincToRS232, byte => byte_for_lcd, rw => rw, rs => rs, e => e, db => db);

    Receptor: rx port map(clk => clk, clr => reset_low, rx => rx_serial,
sel => clk_select, byte_out => byte_for_lcd);

    Salida <= reg_out;

    --Señales extras para test bench.
    clk_en_uso <= clk_RS_232;
    address <= direccion;
    cont_finished <= enviadoXcont;
    enviado_MEF1 <= enviado;
    enviar <= enviar_dato;
    ASCII <= dato;
end behav;

```

4.5.2 Test-bench RS232

```
library ieee;
use ieee.std_logic_1164.all;
use work.pkg_rom.all;

entity tb_RS_232 is
end tb_RS_232;

architecture behav of tb_RS_232 is

--Declaro componente a testear.
component RS_232 is
    port (
        entry           :in      std_logic_vector(4
downto 0);
        reset_low       :in      std_logic;
        clk             :in      std_logic;
        clk_select     :in      std_logic_vector(1 downto 0);
        rx_serial      :in      std_logic;
        salida          :out     std_logic;

        rw              :out    std_logic;
        rs              :out    std_logic;
        e               :buffer  std_logic;
        db              :out    std_logic_vector(7
downto 0);

        clk_en_uso      :out    std_logic;
        address         :out    std_logic_vector(addr_length-1 downto 0);
        cont_finished   :out    std_logic;
        enviado_MEF1   :out    std_logic;
        enviar          :out    std_logic;
        ASCII           :out    std_logic_vector(10
downto 0)
    );
end component;
```

```
--Declaro señales auxiliares de test bench.

signal tb_entry           :std_logic_vector(4      downto      0)      := "00001";
signal tb_reset_low       :std_logic := '1';
signal tb_clk              :std_logic := '0';
signal tb_clk_select      :std_logic_vector(1 downto 0) := "11";
signal tb_rx_serial        :std_logic;

signal tb_rw                :std_logic;
signal tb_rs                :std_logic;
signal tb_e                 :std_logic;
signal tb_db                :std_logic_vector(7 downto 0);

signal tb_salida            :std_logic;
signal tb_clk_en_uso        :std_logic;
signal tb_address           :std_logic_vector(addr_length-1      downto
0);
signal tb_cont_finished     :std_logic;
signal tb_enviado_MEF1      :std_logic;
signal tb_enviar             :std_logic;
signal tb_ASCII              :std_logic_vector(10 downto 0);

constant semiperiod :time := 50us;

begin

clk_gen: tb_clk <= not tb_clk after semiperiod;

uut: RS_232 port map(entry => tb_entry, reset_low => tb_reset_low,
clk => tb_clk, clk_select => tb_clk_select, salida => tb_salida,
rx_serial => tb_rx_serial,
rw => tb_rw, rs => tb_rs, e
=> tb_e, db => tb_db,
clk_en_uso => tb_clk_en_uso,
address  => tb_address, cont_finished => tb_cont_finished, enviado_MEF1
=>tb_enviado_MEF1,enviar => tb_enviar, ASCII => tb_ASCII);

data_gen: process
begin
```

```
--Esperamos 11 clk para que el registro tome todos los
valores de los FFD en 1.

for i in 0 to 11 loop
    wait until tb_clk_en_uso <= '1';
end loop;

--Seleccionamos como entrada la primera frase.

tb_entry <= "00001";
wait until tb_clk_en_uso <= '1';

--Esperamos a que el sistema envie la primera frase.

wait until tb_cont_finished <= '1';

--Seleccionamos como entrada la segunda frase.

tb_entry <= "00010";
wait until tb_clk_en_uso <= '1';

--Esperamos a que el sistema envie la segunda frase.

wait until tb_cont_finished = '1';

--Seleccionamos como entrada la tercera frase.

tb_entry <= "00100";
wait until tb_clk_en_uso <= '1';

--Esperamos a que el sistema envie la tercera frase.

wait until tb_cont_finished = '1';

--Seleccionamos como entrada la cuarta frase.

tb_entry <= "01000";
wait until tb_clk_en_uso <= '1';

--Esperamos a que el sistema envie la cuarta frase.

wait until tb_cont_finished = '1';

--Seleccionamos como entrada todas las frases.

tb_entry <= "10000";
wait until tb_clk_en_uso <= '1';

--Esperamos a que el sistema envie hasta la ultima frase.
```

```
    wait until tb_cont_finished = '1';

    --Seleccionamos como entrada una condicion indeseada para
    ver su funcionamiento.

    tb_entry <= "11000";
    wait until tb_clk_en_uso <= '1';

    --Observamos que sucede con las salidas y las señales.
    for i in 0 to 60 loop
        wait until tb_clk_en_uso <= '1';
    end loop;

    --Reiniciamos y comenzamos de vuelta.
    tb_reset_low <= '0';
    wait until tb_clk_en_uso <= '1';
    tb_reset_low <= '1';
    wait until tb_clk_en_uso <= '1';
end process;

end behav;
```