Escuela Superior de Ingenieros

Departamento de Ingeniería Electrónica
Universidad de Sevilla

Tercera práctica de VHDL:

SISTEMAS ELECTRÓNICOS DE COMUNICACIONES

Javier Nápoles

Hipólito Guzmán

Fernando Muñoz

Miguel A. Aguirre

Práctica 3. Transmisión por línea serie

Tercera práctica de VHDL.
Transmisión por línea serie

Esta práctica es la tercera de un grupo de tres dedicada a entrenar al alumno con el entorno de trabajo Xilinx ISE y la tarjeta Digilent S3.

Durante la práctica se diseñará un transmisor serie asíncrono que utilizará la línea RS232 disponible en la tarjeta. Dicho transmisor permitirá enviar datos ASCII almacenados en una memoria ROM desde la FPGA hasta un PC a través de su puerto serie.

Los objetivos de la práctica son los siguientes:

1. *Perfeccionar el entrenamiento con el entorno Xilinx ISE..*
2. *Aprender a diseñar utilizando máquinas de estado.*
3. *Aprender a generar dispositivos utilizando la herramienta "Core Generator".*

1. Objetivo propuesto

Los objetivos didácticos propuestos en la práctica son los siguientes:

- Generación de una memoria ROM utilizando la herramienta Core Generator (o "coregen"). En la memoria se almacenará un mensaje en ASCII.
- Diseño de una máquina de estado que extraiga los datos de la ROM y los transmita a través del puerto serie
- Proveer de un mecanismo de transferencia de información entre la tarjeta y el usuario para posibles aplicaciones en los proyectos.

2. Enlace Serie

Las principales características del enlace serie a diseñar serán:

- Comunicación asíncrona: El RS232 (Recommended Standard 232) es un estándar de comunicación serie que define tanto comunicaciones síncronas como asíncronas, aunque en esta práctica nos centraremos exclusivamente en la comunicación asíncrona.
- Sólo diseñaremos el transmisor: Al definir líneas independientes para la transmisión y recepción, el estándar RS232 permite una comunicación *full dúplex*. En esta práctica sólo necesitaremos transmitir datos desde la FPGA al ordenador.
- Transmitiremos los datos en ASCII: El estándar RS232 no define nada sobre la codificación de los datos. Nosotros transmitiremos caracteres ASCII que serán interpretados por el *HyperTerminal* de Windows para su representación en pantalla.
- Características básicas de la transmisión: 8 bits de datos (ASCII), paridad par, 1 *start*, 1 de *stop* y velocidad de 9600 baudios. En la Figura 1 se presenta un diagrama temporal de la transmisión de un byte tal como habría que realizarla en la presente práctica. El objetivo de la práctica consistirá en generar una señal como la de la Figura 1 con la temporización adecuada.
- La transmisión se realizará por el pin R13 de la FPGA: La conexión del puerto serie con la FPGA se representa en la Figura 2. Se puede observar que el pin de transmisión del conector DB9 (pin 2) está conectado a pin R13 de la FPGA a través del circuito integrado MAX3232. Dicho circuito integrado realizará la translación entre los 0-1.2V de tensión a la salida de la FPGA a los ±15V requeridos por el estándar RS232.

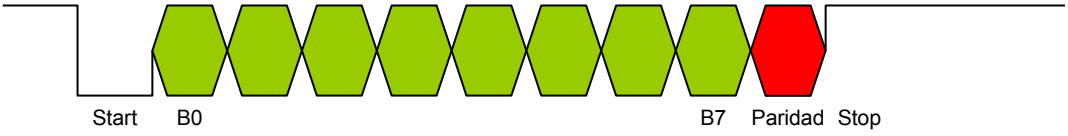

Figura 1. Ejemplo de transmisión de un byte.

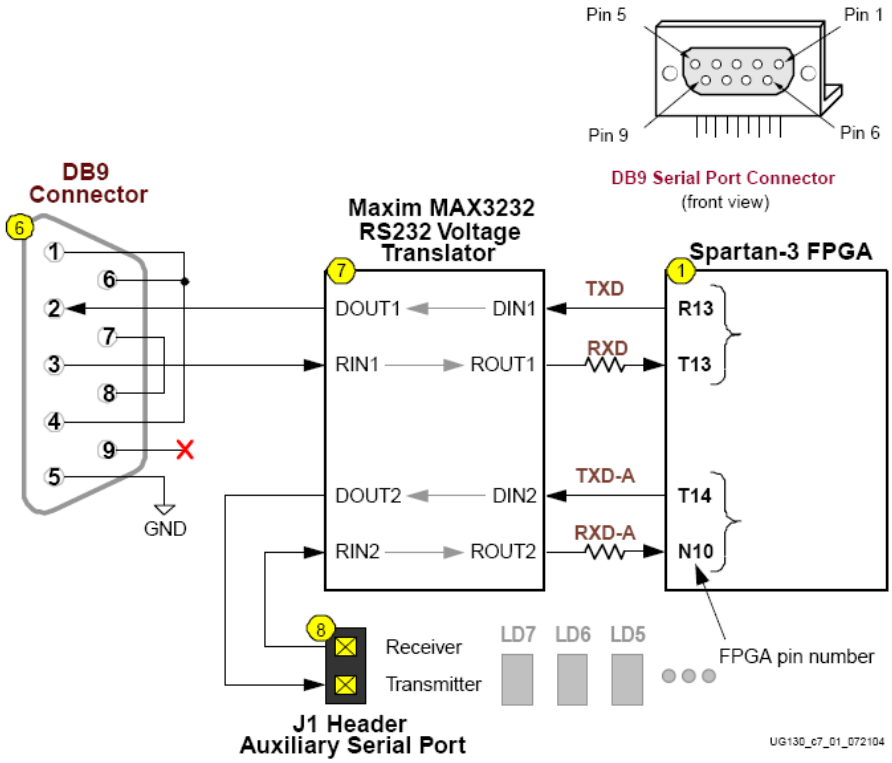

Figura 2. Conexión entre el puerto RS232 y la FGA

Para recibir los datos en el PC utilizaremos programa HyperTerminarl de Windows. Dicho programa nos permitirá configurar el puerto COM1 con las características básicas de la comunicación (9600 baudios, paridad pare, etc), interpretará los caracteres ASCII y los mostrará por pantalla. Se realizará una explicación del HyperTerminal en el apartado 3.3.

3. Realización de la práctica

La práctica consta de tres apartados:

- En primer lugar generaremos la memoria ROM con el mensaje en ASCII que pretendemos comunicar al PC.
- A continuación diseñaremos una máquina de estado que leerá los datos de la ROM y los trasmitirá de forma serie por el pin R13. Uniremos la máquina de estado con la memoria ROM mediante un bloque de jerarquía superior.
- Configuraremos el programa HyperTerminal para que utilice el mismo protocolo serie que nuestro diseño.

3.1. Diseño de la memoria ROM utilizando CORE Generator

Para implementar la memoria ROM empotrada en la FPGA utilizaremos la herramienta *CORE Generator*. Esta herramienta permite utilizar primitivas específicas de la FPGA que normalmente no son accesibles directamente desde código VHDL[1], como por ejemplo memorias empotradas, bloques DSP48, Digital Clock Managers (DCMs), etc. Al utilizarla, nos generará un 'core', una caja negra que podremos instanciar en nuestro diseño como si de un componente más se tratara.

Los *cores* generados por la herramienta están optimizados para FPGAs[2] de Xilinx, por lo que como diseñadores podremos confiar (hasta cierto punto[3]) en el correcto funcionamiento de estos bloques.

Generación de nuestro primer *core*: Memoria de Bloque Empotrada

Si bien podemos arrancar CORE Generator de forma independiente, para esta práctica nos servirá arrancarlo desde ISE. Seleccionamos *Project -> New Source*, y seleccionamos la opción *IP (Coregen & Architecture Wizard)*. Ver Figura 3.

[1] Realmente muchas de las primitivas se pueden acceder directamente desde código VHDL, pero su manejo es bastante más complicado (por ejemplo: describir unos registros de forma que el sintetizador infiera una memoria empotrada)

[2] Por esta razón es necesario especificar qué modelo de FPGA concreto estamos utilizando en nuestro proyecto. Si se realiza a través del Coregen & Architecture Wizard, como se explica en este apartado, se tomará por defecto la FPGA definida al crear el proyecto.

[3] Ante la duda, se pueden consultar los datasheet de los cores, utilizando el botón *View Data Sheet* que aparece en la esquina inferior izquierda de las ventanas del Wizard de CORE Generator

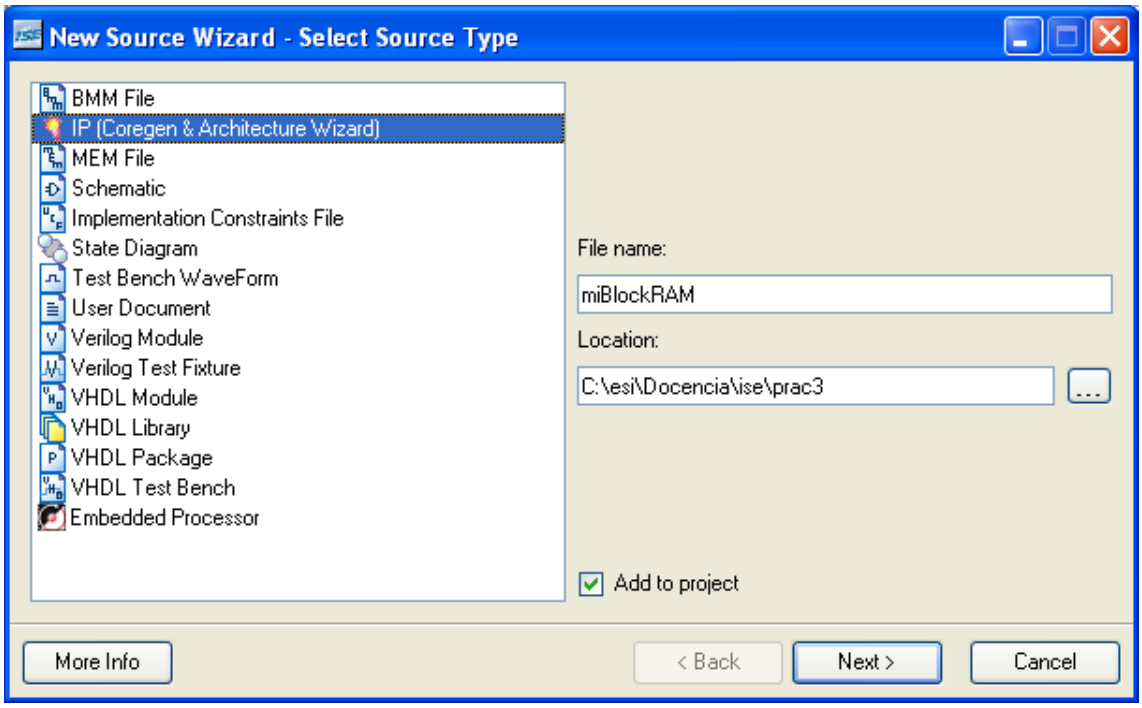

Figura 3. Cómo arrancar el Coregen directamente desde ISE

En la pantalla de selección de IP seleccionamos la primitiva que queremos instanciar, en nuestro caso una memoria de bloque o BlockRAM. El nombre "BlockRAM" significa que toda la memoria se encuentra, implementada en silicio, en una zona dedicada de la FPGA, a diferencia de las "Distributed RAM", que utilizan los bits de memoria las LUTs de distintos bloques lógicos repartidos por la FPGA (Figura 4).

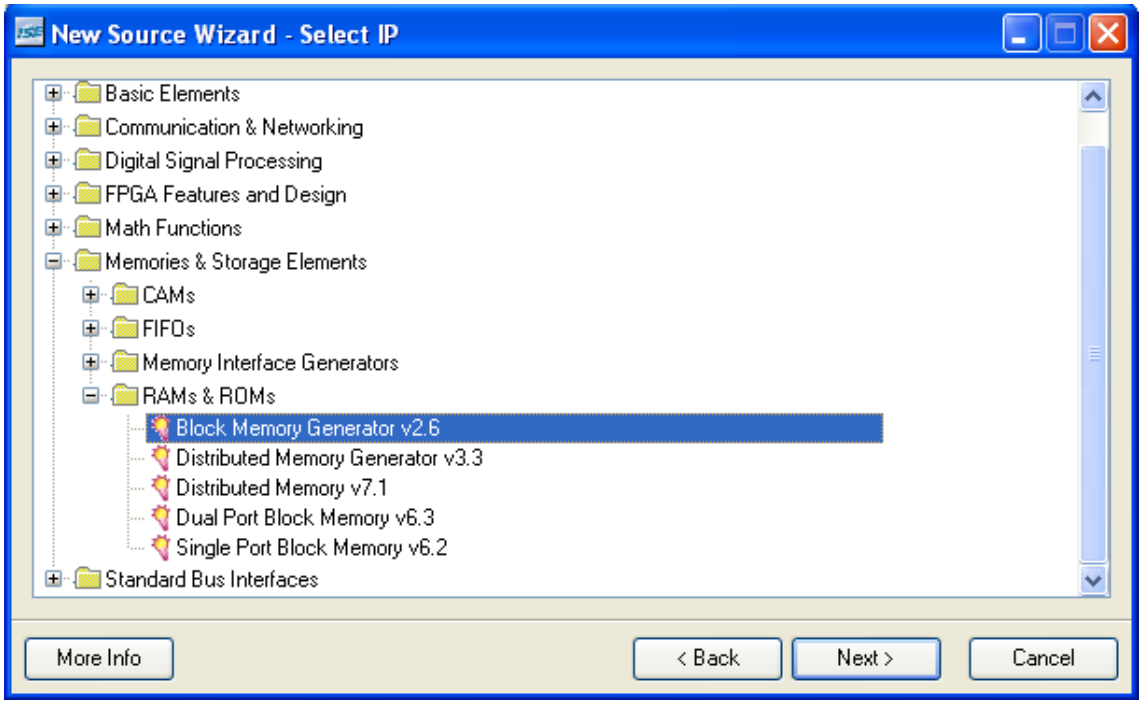

Figura 4. Queremos generar una memoria del tipo Block Memory

Tras lo que nos aparece una pantalla de resumen de las opciones seleccionadas (Figura 5). Fijémonos en el 'Source Name' del fichero que nos va a generar. Si queremos guardar únicamente los ficheros fuente de un proyecto que use cores, guardaremos, además de los ficheros .vhd y .ucf, los ficheros .xco (Xilinx Core) correspondientes a los cores generados.

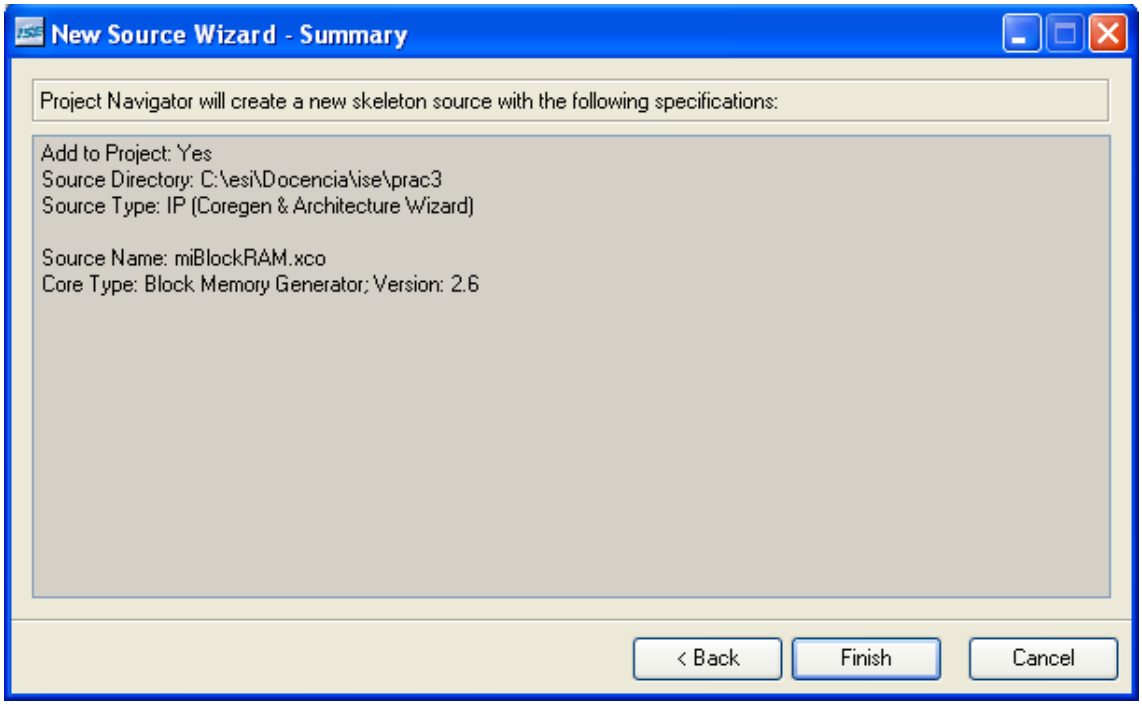

Figura 5. Resumen de las opciones seleccionadas en el asistente. Tras pulsar *Finish* se llamará a la aplicación Coregen con estas opciones.

Al pulsar 'Finish' finalizará el asistente y arrancará propiamente la herramienta CORE Generator. Como ya le hemos dicho qué primitiva queremos, directamente nos saldrá el Generador de Memorias de Bloque, en el que seleccionaremos el tipo de memoria que queremos (Memoria ROM de un solo puerto). Ver Figura 6.

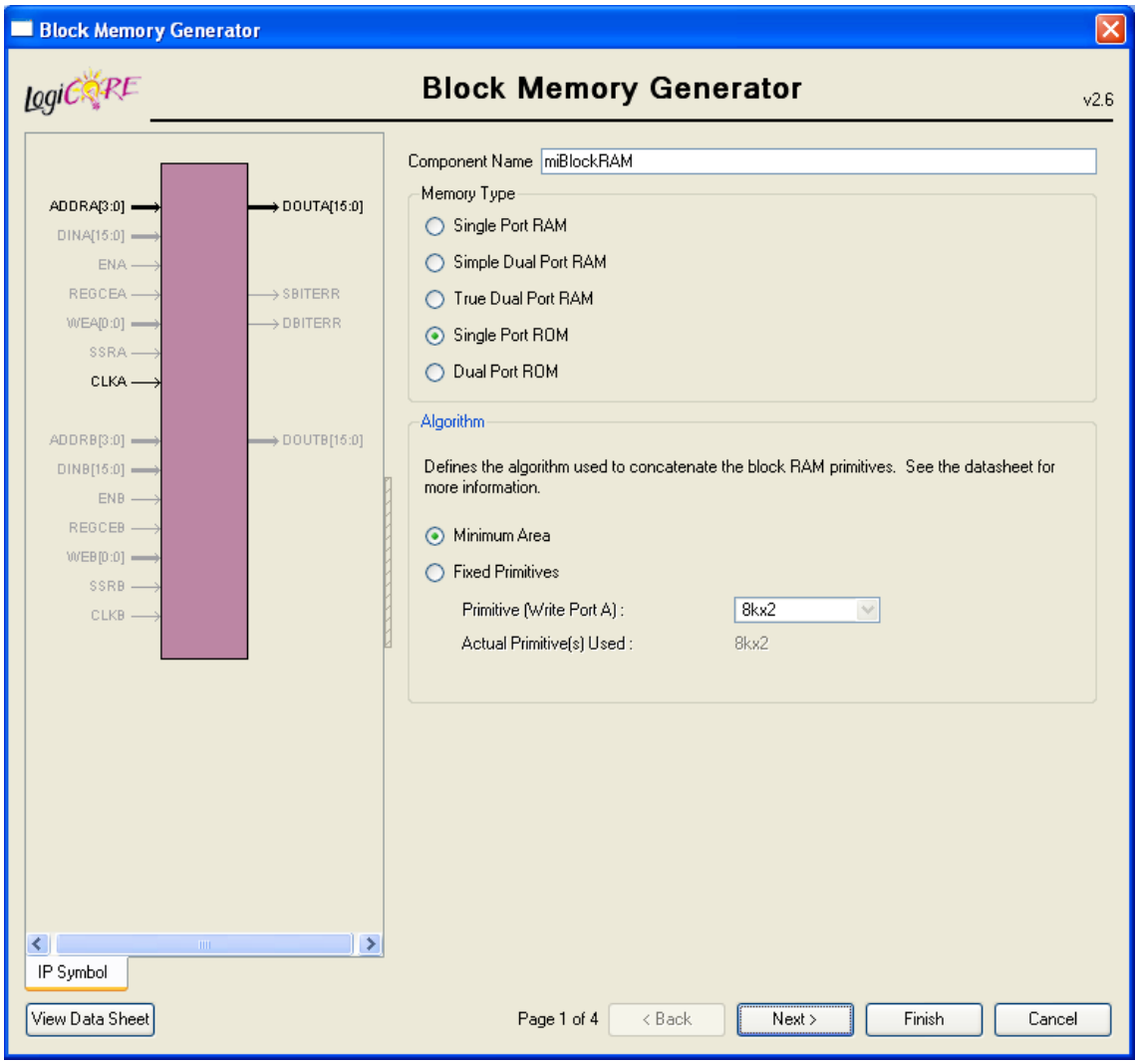

Figura 6. Primer menú del Core Generator para la generación de una memoria Block Memory. Se especifica el tipo de memoria.

Ya que queremos almacenar caracteres ASCII en la memoria, configuramos la misma con una anchura de 8 bits. Seleccionamos una profundidad de 16 palabras (cada palabra será del ancho que hayamos configurado). Ver Figura 7.

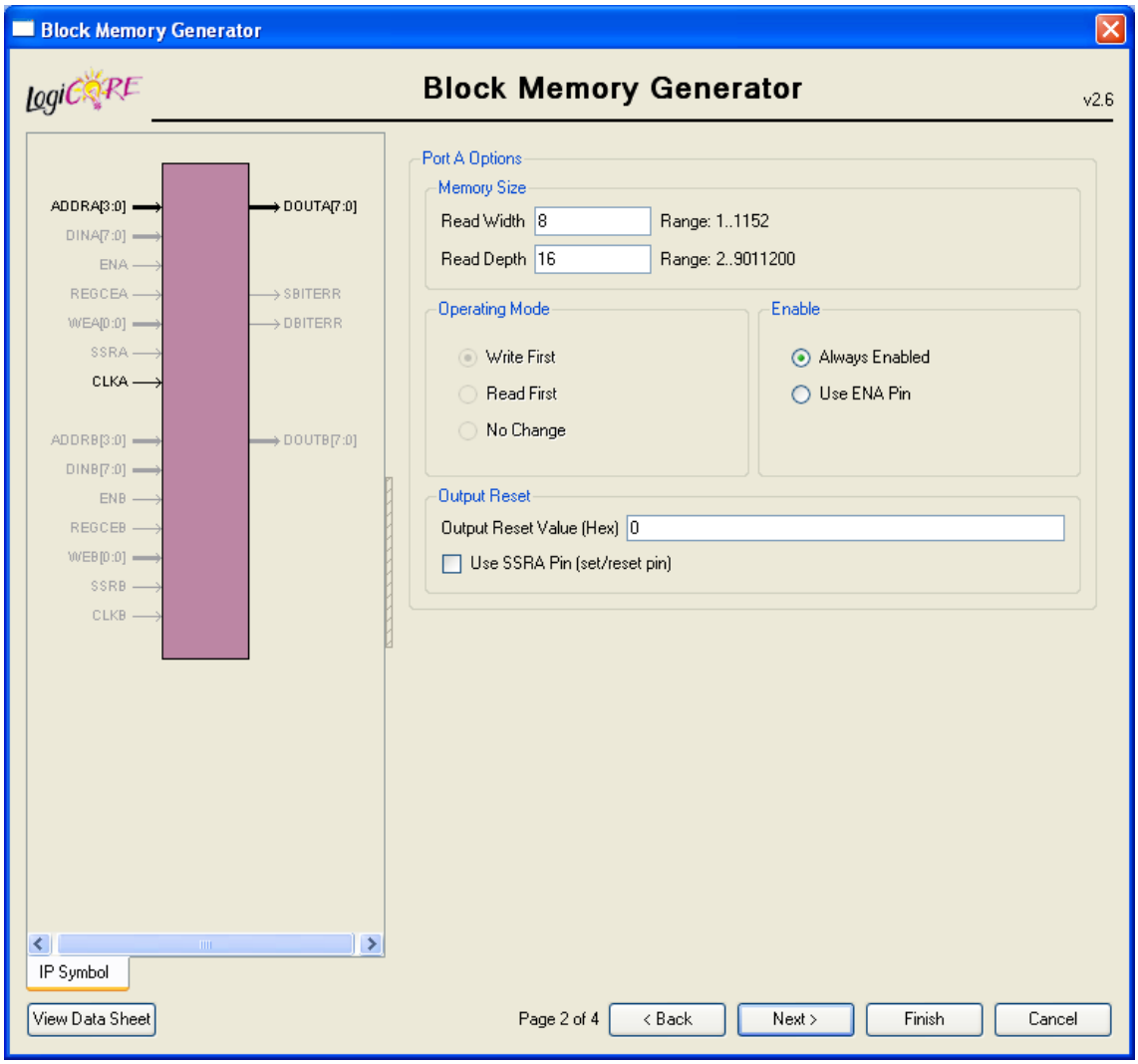

Figura 7. Segundo menú del Core Generator para la generación de Block Memory. Entre otras opciones se configura el tamaño de la memoria.

Marcamos la opción *Register Port A of Output Memory Primitives*, esta opción nos sintetizará un registro a la salida de la memoria, de forma que cuando se cambie el valor del bus de direcciones (ADDRA), el nuevo dato (DOUTA) estará disponible al siguiente ciclo de reloj (en este caso no tenemos señal Enable, la memoria siempre está habilitada para lectura, ya que seleccionaremos la opción *Always Enabled*).

El último paso es utilizar un fichero .coe (Coefficients) para dar los valores iniciales a la memoria (en este caso, tratándose de una ROM, es la única forma que tenemos de introducir dichos valores en la misma). En la siguiente sección se detalla la sintaxis que han de seguir los ficheros .coe. Marcamos también la opción *Fill Remaining Memory Locations*, con el valor 0, para asegurarnos de que las posiciones de memoria que no queden descritas por el fichero .coe se inicializen a cero.

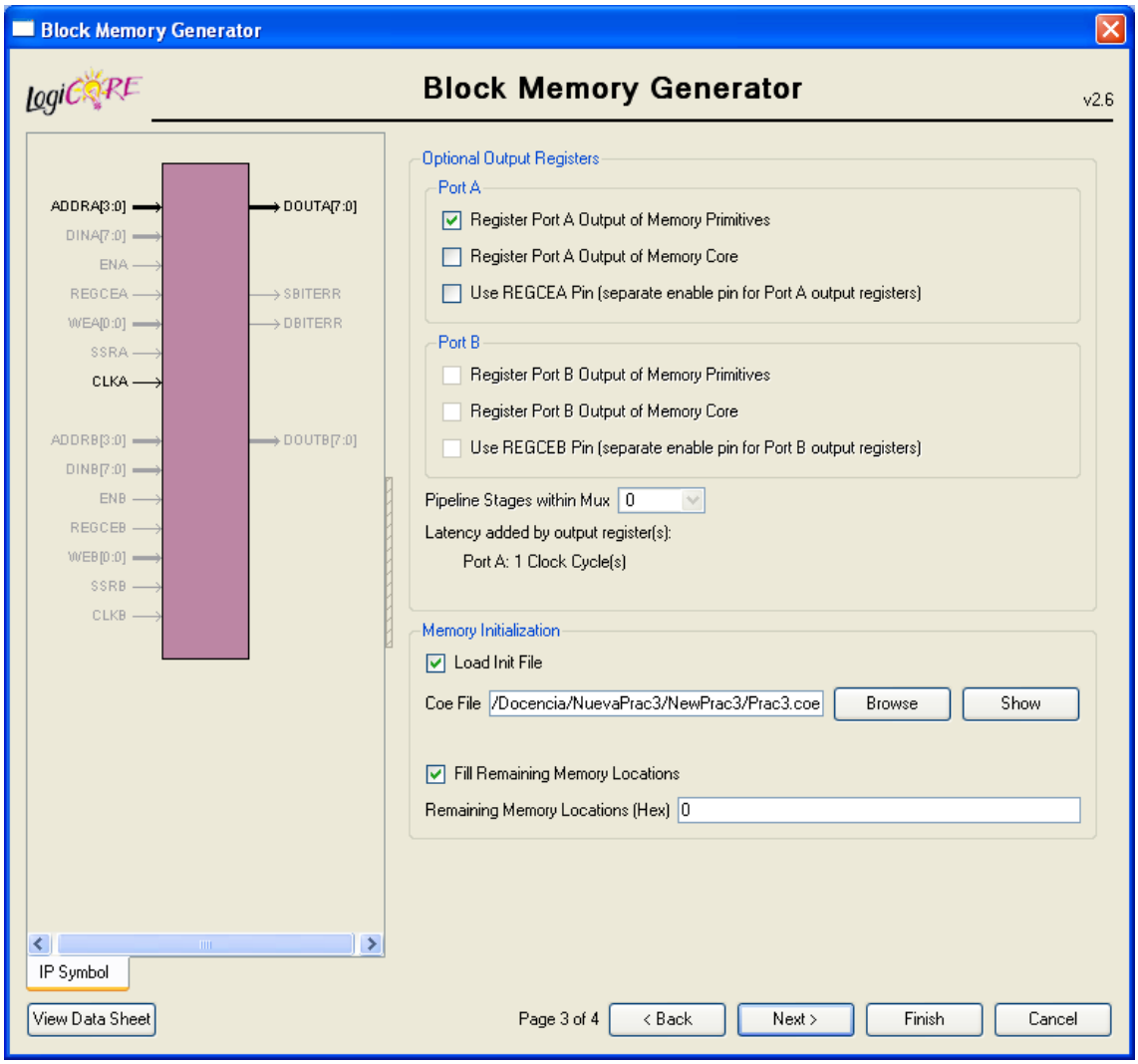

Figura 8. Tercer menú del Core Generator para la generación de Block Memory. Se direcciona un fichero .coe con los datos inciales de la memoria.

En la última pantalla (opciones de simulación) dejamos las opciones por defecto:

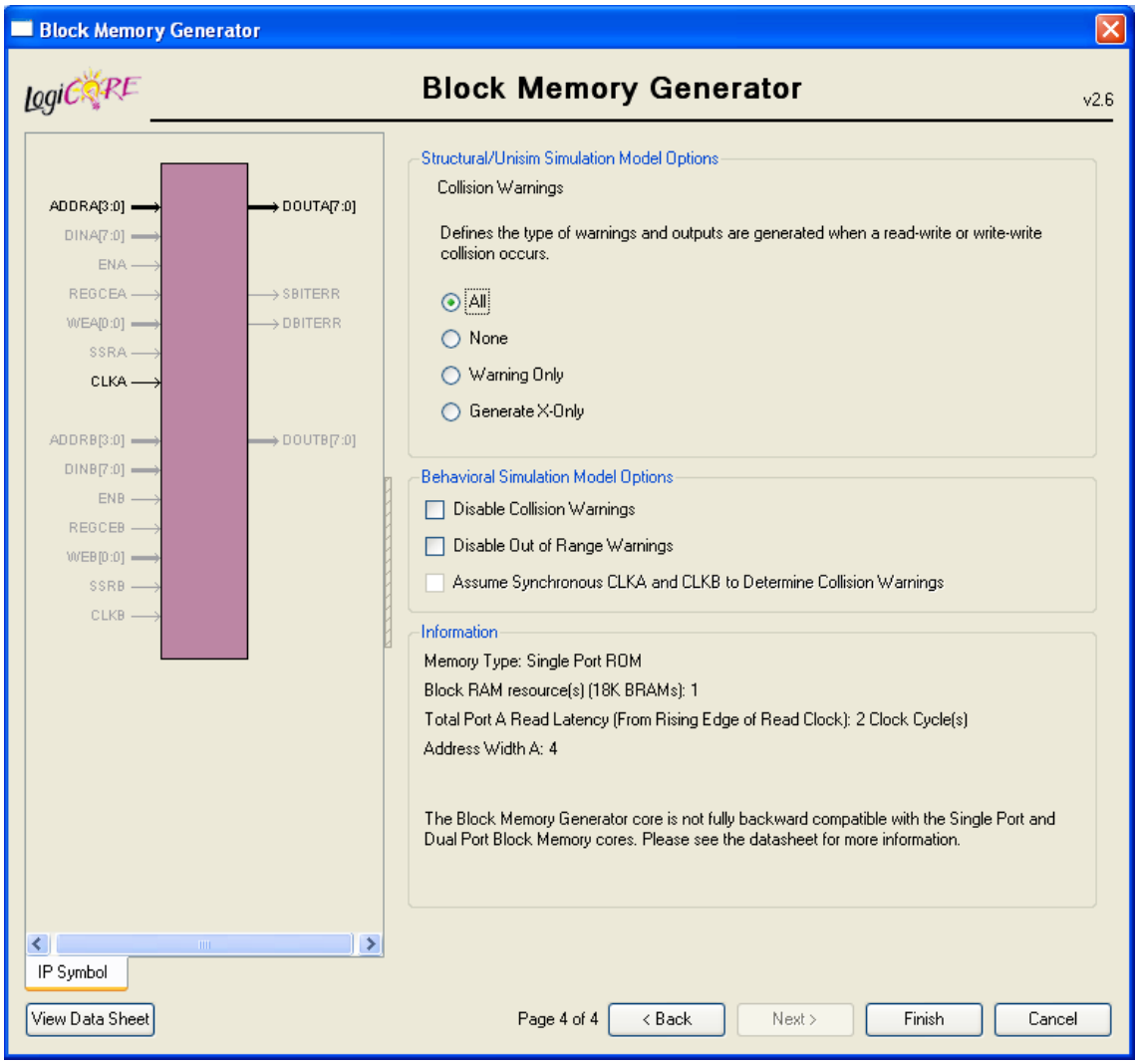

Figura 9. Cuarto menú del Core Generator para la generación de Block Memory.

Cuando pulsemos 'Finish' nos generará la IP. Es un proceso que tarda un poco, y por la consola de ISE nos avisará cuando haya terminado, o si detecta algún error.

Sintaxis del fichero .coe

El fichero .coe es un fichero de texto plano, que contiene un vector con los valores que se han de introducir en la memoria, y una variable 'radix' (que significa 'base numérica' en inglés), que indica en qué base numérica debemos interpretar los datos[4].

Un ejemplo de fichero .coe es el siguiente:

```
; Un punto y coma (semicolon) significa comentario
memory_initialization_radix = 16;
memory_initialization_vector =
<LISTA DE VALORES SEPARADOS POR ESPACIO> ;
```

[4] Recordemos que, por ejemplo, los dígitos "10", según se interpreten en binario, decimal, hexadecimal u octal, significan "dos", "diez", "dieciséis" u "ocho" respectivamente

Recordamos que radix 16 sería hexadecimal (base 16), radix 8 octal, radix 10 decimal…

Instanciación de la memoria y manejo de los puertos

Para poder utilizar la memoria que acabamos de generar, necesitaremos una definición de componente, igual que para los componentes que tengamos definidos en VHDL. ¿De dónde obtenemos esta definición?

Una vez que coregen ha terminado de generar la memoria, si seleccionamos el nuevo core en la ventana de procesos (lo reconoceremos porque tiene un icono diferente al de los módulos VHDL), nos aparecerán tres procesos para realizar sobre el mismo:

- *Manage Cores*: Nos permite volver a configurar nuestro core, en caso de que nos hayamos equivocado en algo o simplemente de que queramos cambiar su configuración.
- *Regenerate Core*: Genera todos los ficheros intermedios del software de Xilinx a partir del .xco original.
- *View HDL functional model*: Nos muestra el modelo HDL para simulación funcional de la primitiva generada. Este modelo tiene una sección ENTITY que podemos copiar para obtener la declaración de componente necesaria para instanciarlo.

El manejo de la memoria es muy sencillo. Tiene dos entradas (ADDRA y CLK), y una salida (DOUTA). Ya que no existe señal enable, y hemos puesto un registro a la salida de la memoria, cuando cambie ADDRA obtendremos un dato nuevo en DOUTA al siguiente flanco de reloj.

3.2. Realización de la práctica

El envío de los datos a través del puerto serie se va a controlar mediante una máquina de estados[5]. Dicha máquina de estados pedirá el dato que necesita en cada momento a través del puerto *direcc* y lo recibirá por el puerto *data*. Dichos puertos estarán conectados a los puertos de la memoria en un nivel jerárquico superior.

Por otro lado, para implementar la comunicación síncrona de la Figura 1, Irá contando el número de ciclos de reloj que hay que esperar para poder transmitir cada bit de forma correcta respetando el protocolo. Para ello utilizará un contador interno que deberá estar generado con sendas señales *cont* y *p_cont* del tipo std_logic_vector y el ancho apropiado (dependerá de la tasa de transmisión). Para poder generar la dirección correcta del dato a transmitir en cada momento también se usará un contador interno debidamente descrito con las señales *dir* y *p_dir* que se incrementará cada vez que se transmite un byte completo.

[5] En el nivel de abstracción RTL que utiliza normalmente en VHDL una máquina de estado un un bloque muy potente y versátil. Se recomienda la utilización de máquinas de estado en los proyectos de los alumnos dentro de lo posible.

A continuación se describe el comportamiento que debe tener la máquina de estados. También se añade un diagrama de bolas para aclarar su funcionamiento. La máquina dispone de un total de 14 estados aunque su funcionamiento es muy sencillo tal y como se aprecia en el diagrama. Se han omitido los estados ***b_1***, ***b_2***, ***b_3***, ***b_4***, ***b_5*** y ***b_6*** por claridad aunque son equivalentes a los estados ***b_0*** y ***b_7*** sin más que cambiar el bit que se envía en cada caso.

1. Reposo

Inicialmente (y si se produce un reset) partimos del estado de ***reposo***. En dicho estado tanto el contador interno como la dirección del dato a leer están a cero.

2. Inicio

Si se activa el puerto button[6] pasamos al estado de ***inicio*** que simplemente consiste en un ciclo de espera para que la memoria ponga el b a la salida. Por el tipo de memoria implementada, una vez que se cambia la dirección a leer es necesario esperar al siguiente ciclo de reloj para que el dato se cargue en el registro de salida que incorpora la memoria y esté disponible para el usuario.

3. Test_data

Después pasamos al estado ***test_data*** donde el nuevo dato ya estará disponible para el usuario. Si el nuevo dato es igual a 0x00 entonces ya hemos terminado de transmitir toda la cadena de caracteres y volveríamos a reposo. Si el dato es distinto de cero entonces pasamos al estado ***b_start*** donde se va a enviar el bit de start. Saldremos de este estado cuando el contador interno llegue a máximo valor de cuenta `VAL_SAT_CONT`, calculado para transmitir a 9600baudios.

4. B_start

En este estado se pone la línea de transmisión a '0' durante el tiempo que indica el protocolo. Iremos incrementando el contador interno hasta que se alcance el valor `VAL_SAT_CONT`. Cuando se cumpla esa condición reiniciaremos el contador interno y pasaremos al siguiente estado ***b_0***. Para reinicializar el contador basta con asignar ceros al próximo valor del contador (el nuevo valor se hará efectivo en el siguiente flanco de reloj y coincidirá con la transición de la máquina de estados al nuevo estado).

5. Bits de datos

En el estado ***b_0*** se transmite el primer bit del dato a enviar. Se empieza por el menos significativo. De nuevo esperamos a que el contador interno alcance el valor `VAL_SAT_CONT`. Cuando se produzca esa condición se reinicia el contador para empezar la cuenta desde cero en el siguiente bit.

En los sucesivos estados ***b_1***,…,***b_7*** se irán transmitiendo cada uno de los bits del dato a enviar. En todos los casos se espera hasta que el contador alcance el valor de `VAL_SAT_CONT`.

6. B_paridad

Después de transmitir todos los bits del dato se manda un bit de paridad para comprobar que no ha habido errores en la comunicación. Dicho bit se genera de forma combinacional en este estado con puertas **xor**. De nuevo esperamos el tiempo correspondiente a un bit.

7. B_stop

[6] Cuando pulsamos el botón correspondiente.

Por último enviamos el bit de stop. La línea de transmisión se vuelve a poner a '1' durante un tiempo de bit. Cuando se cumpla el tiempo de bit se resetea el contador interno y se incrementa la dirección del siguiente dato a leer. Pasamos al estado ***inicio*** para esperar a que el nuevo dato esté disponible.
De nuevo en el estado ***test_data*** volvemos a comprobar si el nuevo dato es distinto de cero para transmitirlo. Cuando todos los datos previstos hayan sido enviados leeremos un cero de la memoria y pasaremos al estado de reposo a esperar a que se vuelva a activar la señal button.
En la Figura 10 aparece representado el diagrama de bolas de la máquina de estados. Se han indicado únicamente las transiciones entre los distintos estados y las asignaciones que hay que realizar en algunas de ellas. Cuando no se cumpla alguna de las condiciones que permiten cambiar de estado se entiende que dicho estado no cambia. En aquellos estados en los que no especifica ningún valor para la señales *p_cont* o *p_dir* se entiende que conservarán su valor previo.

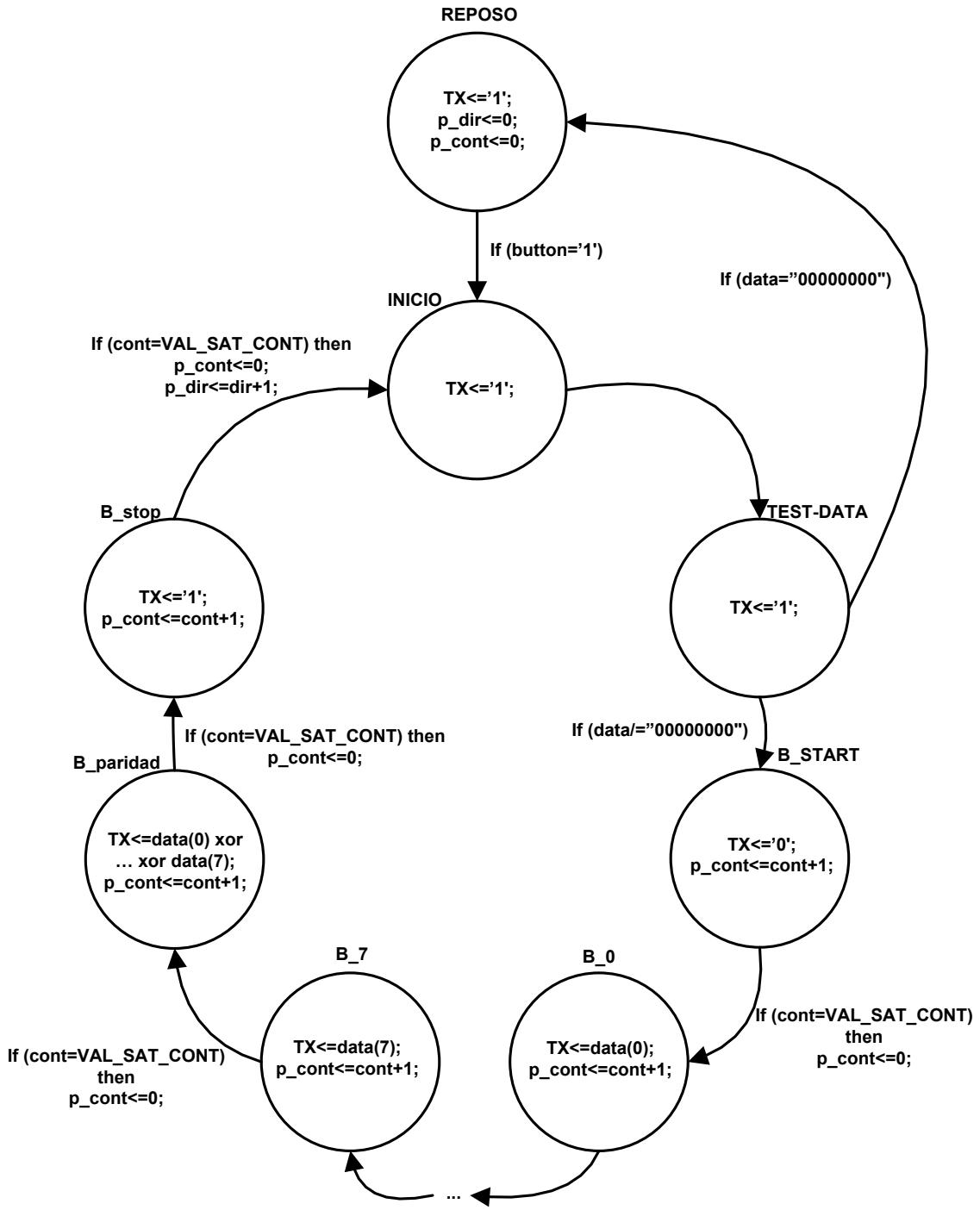

Figura 10. Diagrama de bolas de la máquina de estados que controla la transmisión.

FSM	
Descripción	Circuito de control del proceso de transmisión.

Entidad	`entity fsm is`
	`  Generic (ancho_bus_dir:integer:=4;`
	`  VAL_SAT_CONT:integer:=20000;`
	`  ANCHO_CONTADOR:integer:=20);`
	
	`  Port ( clk : in  STD_LOGIC;`
	`  rst : in  STD_LOGIC;`
	`  button : in  STD_LOGIC;`
	`  data : in  std_logic_vector(7 downto 0);`
	`  direcc : out std_logic_vector(ancho_bus_dir-1`

	downto 0); TX : out STD_LOGIC); end fsm;
Descripción de los generics	
ancho_bus_dir	Indica el ancho del bus de direcciones de la memoria de la que se van a leer los dato.
VAL_SAT_CONT	Valor en el que el contador alcanza un número de ciclos de reloj equivalente al tiempo que debe durar la trasmisión de un bit. Dependerá de la velocidad de transmisión.
ANCHO_CONTADOR	Número de líneas del contador que calcula el tiempo de bit. Debe ser suficiente para poder alcanzar el valor especificado en VAL_SAT_CONT.
Descripción de los puertos	
clk	Reloj de 50MHz
rst	Reset asíncrono activo a nivel alto
button	Pulso externo que inicia la transmisión de los datos.
data	Bus de datos de la memoria que contiene los datos a transmitir.
direcc	Bus de direcciones de la memoria que contiene los datos a transmitir.
TX	Pin de transmisión que va al conector DB9.

El bloque de más alto nivel jerárquico debe contener como componentes la máquina de estados y la memoria. Habrá que definir todas las señales que sean necesarias para conectar estos bloques. Simplemente hay que instanciar cada componente y especificar los valores correctos a los generics. Para una tasa de transmisión de 9600 baudios los valores de las constantes son:

GENERICS PARA 9600 BAUDIOS	
VAL_SAT_CONT	5208
ANCHO_CONTADOR	13

Conexionado de pines de E/S.
EL fichero "USER CONSTRAINTS FILE" debe contener la siguiente información:

```
NET "clk" LOC = "T9";
NET "rst" LOC = "L14";
NET "TX" LOC = "R13";
NET "button" LOC = "L13";
```

3.3. Configuración del Hyperterminal.

HyperTerminal es una aplicación de Windows que permite realizar operaciones de comunicación por diferentes canales, y en particular por el puerto serie, identificado como COM1:. Se encuentra en Inicio -> Programas -> Accesorios -> Comunicaciones -> Hyperterminal. Aparecerá la ventana mostrada en la Figura 11.

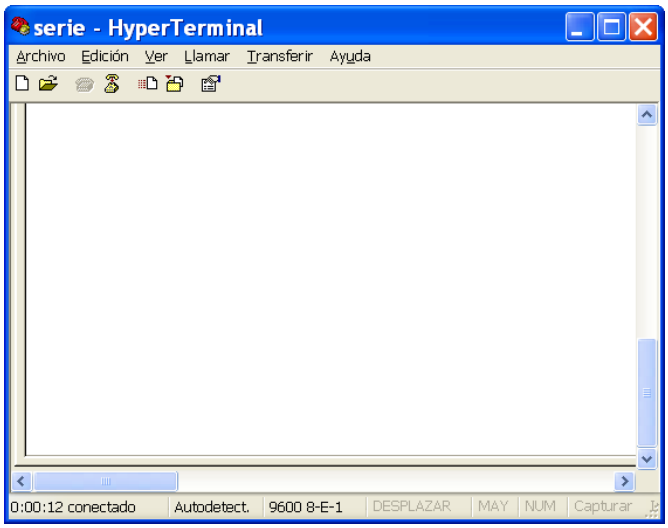

Figura 11. Interfaz básico del HyperTerminal de Windows

Con el programa desconectado (icono del teléfono descolgado), Archivo->Propiedades

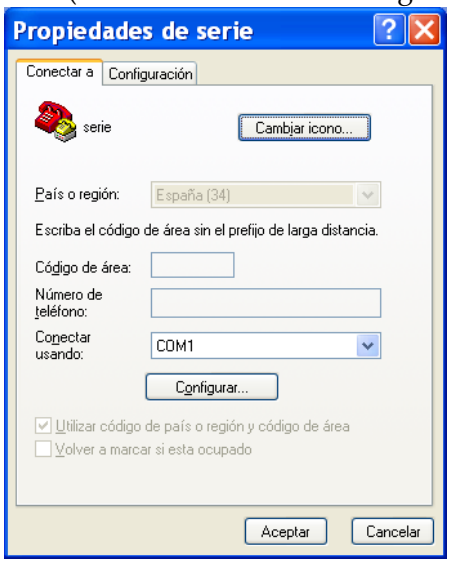

Figura 12. Propiedades de comunicación serie.

Debe aparecer conectar usando COM1. Aquí presionar el botón ·"Configurar". Aquí debe aparecer la configuración de la velocidad del puerto serie:

Figura 13. Configuración del puerto serie.

La Figura 13 muestra la configuración del puerto serie para realizar esta práctica. Una vez configurado la información que se escriba desde teclado en la consola del hyperterminal se transfiere automáticamente al puerto serie, y la información recibida será escrita en la consola. Los caracteres escritos no aparecen en la configuración por defecto de la consola.