Proyecto de Sistemas Informáticos

Título:

Desarrollo de un entorno hardware para la ejecución de aplicaciones de propósito general sobre FPGA's.

Autores:

Javier Fernández Martín
Emilia Mª Rodríguez Alonso
Patricia Ruiz González

Directora:

Hortensia Mecha López

Curso:

2005-2006

Facultad de Informática
Universidad Computense

Resumen

El objetivo global de este proyecto consiste en generar un entorno para la ejecución de tareas sobre dispositivos reconfigurables. Inicialmente comenzamos realizando la implementación sobre una placa de prototipado con una FPGA Virtex II, y posteriormente, sobre otra con una FPGA Virtex II Pro. Este cambio se debió a que la primera no ofrecía la posibilidad física de reconfiguración dinámica.

Aunque este objetivo general no ha llegado a alcanzarse debido a su extensión, sí que hemos desarrollado una serie de módulos necesarios para ello. Estos módulos, que son operativos de manera independiente a los demás, tienen una función definida dentro del proyecto.

El primero de ellos corresponde a una interfaz diseñada para el gestor de memoria correspondiente a la memoria RAM de la placa de la FPGA Virtex II. Esta interfaz de usuario recibe del exterior órdenes de lectura, escritura, o lectura de un bloque consecutivo de direcciones, e interactúa con el interfaz que controla el banco 0 de la memoria RAM para realizar sobre ella dichas operaciones.

A continuación desarrollamos un módulo planificador de tareas, que decide qué tarea pendiente de ejecución debe ser atendida en cada momento. En función de esta decisión, proporcionaría los datos de la tarea correspondiente al gestor de memoria, que se encarga de recuperar el mapa de bits de la tarea que reside en la memoria RAM para que sea ejecutada.

Una parte importante dentro del diseño de cualquier planificador de tareas es decidir la política que va a emplearse en él. En nuestro caso esta política será tipo FIFO, es decir, las tareas van a ser atendidas en el orden en que llegue su petición de ejecución. Por lo tanto la estructura que emplearemos dentro del módulo planificador para gestionar los turnos de las tareas será una cola tipo FIFO encargada de que cada tarea espere a que todas las anteriores hayan sido atendidas.

Para probar que estos módulos funcionan correctamente hemos utilizado un dispositivo de entrada (teclado) para introducir los datos en el sistema, y uno de salida (VGA) para recibirlos del mismo. Para conseguir esta interacción entre el sistema y el usuario a través de estos dispositivos periféricos se nos proporcionaron ya diseñados unos controladores de entrada/salida.

Cuando se sustituyó la primera placa por otra con la FPGA Virtex II Pro el controlador de teclado no precisó ninguna modificación para continuar operativo. Sin embargo el controlador de VGA sí tuvo que ser adaptado a la nueva placa para continuar utilizándolo.

El módulo planificador de tareas también es independiente de la placa utilizada, sin embargo no ocurre así con el gestor de memoria, ya que la memoria de la que dispone la FPGA VIRTEX II es una DDRA SDRAM, que cuenta con una tecnología bastante más avanzada que la anterior.

Por lo tanto el siguiente y último módulo que desarrollamos fue una interfaz para el gestor de memoria de la DDRA SDRAM de esta nueva placa. Aunque las pruebas realizadas mediante simulación para esta interfaz indican que debería funcionar correctamente, no ha sido así a causa de que la interfaz que controla esta memoria y que nos fue facilitado por el MIG 007 (una aplicación que genera este tipo de interfaces) no ha respondido como se esperaba a las órdenes de nuestro módulo.

Overwiew

The initial goal for this project was to create an environment for running tasks on reconfigurable devices. We started by creating the implementation over a prototype board using a FPGA Virtex II and, later, over one using a FPGA Virtex II Pro. This change was due to the fact that the first board didn't have the posibility of dynamic reconfiguration.

While this goal was not reached, we did develop along the year a series of modules needed for it. These modules, which are operative in a stand-alone manner, have a specific function in the project.

The first module is an interface designed for the memory manager used on the FPGA Virtex II board's RAM. This user interface receives read, write, or block read orders from the outside, and interacts with the interface that controls the RAM's bank 0 to make those operations on it.

Then we developed a task scheduler module, which decides which task that is ready for execution gets to be dispatched at each moment. Acording to that decision, it would provide the given task's data to the memory manager, which would be in charge of recovering the bit map of that task from the RAM to be executed.

An important part on every task scheduler's design is to decide the policy to be used in it. In our case this policy will be FIFO, which means that the tasks will be dispatched in the same order their execution request arrived. For this reason, the structure used in the scheduler module for the task's turns would be a FIFO queue in charge of making sure every task waits for the all the previous ones to be dispatched.

To test these modules for correct behaviour, we used an input device (keyboard) to enter the data into the system, and an output device (VGA) to receive the data from it. To achieve this interacion between the system and the user, we were provided with pre-designed input/output drivers.

When the first board was substituted with the one with a FPGA Virtex II Pro, the keyboard driver didn't need any modification to function properly. The VGA driver, however, had to be adapted to the new board to be able to continue using it.

The task scheduler module is also independent from the board used, while this is not true for the memory manager, as the memory in the FPGA Virtex II Pro is a DDRA SDRAM which is technologically superior to the previous one.

Therefore, the next and last module we developed was an interface for this new board's DDRA SDRAM memory manager. While the simulated tests for this interface point to it working perfectly, the fact is it didn't due to the fact that the interface controlling this memory, given to us by the MIG 007 (an application that generates such kind of interfaces), didn't react as expected to our module's orders.

Índice general

Siglas y términos 7
Capítulo 1. Tecnología 9
1.1 Introducción 9
1.2 Sistemas programables. FPGAs 9
1.2.1 Introducción 9
1.2.2 Dispositivos de lógica programable 9
1.2.3 FPGAs 10
1.2.3.1 Introducción 10
1.2.3.2 Estructura interna de las FPGA 11
1.2.3.4 Virtex II 13
1.2.3.5 Virtex II Pro 15
1.3 Xilinx-ISE 6.2 y 7.0 17
1.3 Analizador de señales 20
1.4 MIG 007 21
Capítulo 2. Módulos 26
2.1 Introducción 26
2.2 Comprobador de Tecla 27
2.2.1 Introducción 27
2.2.2 Descripción de señales 27
2.3 Interfaz de SVGA 29
2.3.1 Introducción 29
2.3.2 Descripción de señales 30
2.4 Generador de Relojes 31
2.4.1 Introducción 31
2.4.2 Descripción de señales 31
2.5 Gestor de memoria 32
2.5.1 Introducción 32
2.5.2 Gestor de memoria para la RAM de la FPGA Virtex II 33
2.5.2.1 Introducción 33
2.5.2.2 Modelo de la interfaz del gestor de memoria 33
2.5.2.3 Interfaz del banco 0 de la RAM 34
2.5.2.4 Interfaz del usuario 36
2.5.2.5 Implementación 38
2.5.2.6 Interconexión 42
2.5.2.7 Funcionamiento 43
2.5.3 Gestor de memoria para la SDRAM de la FPGA Virtex IIPro 45
2.5.3.1 Introducción 45
2.5.3.2 Interfaz generado por el MIG007 46
2.5.3.2.1 Inicialización de la memoria 48
2.5.3.2.2 Escritura en memoria 49
2.5.3.2.3 Lectura de memoria 50
2.5.3.3 Unidad de Control del Interfaz 50
2.5.3.3.1 Introducción 50
2.5.3.3.2 Ruta de Datos 52
2.5.3.3.3 Controlador 53
2.5.3.3.4 Simulación 53
2.5.3.3.5 Depuración 58
2.6 Planificador de tareas 60
2.6.1 Introducción 61
2.6.2 Descripción de las señales 61
2.6.3 Implementación 62

2.6.4 Funcionamiento 68
2.6.4 Cola FIFO 70
2.6.4.1 Introducción 70
2.6.4.2 Descripción de señales 71
2.6.4.3 Implementación 72
2.6.4.4 Funcionamiento 74
2.7 Módulo de prueba 77
2.7.1 Introducción 77
2.7.2 Descripción de señales 77
2.7.3 Funcionamiento 78
Capítulo 3. Interconexión de módulos 80
3.1 Introducción 80
3.2 Descripción de señales 80
3.3 Implementación 81
3.4 Unidad de Control Global 83
3.4.1 Ruta de datos 83
3.4.2 Máquina de estados 84
3.5 Funcionamiento 89
Capítulo 4. Conclusiones 94
Anexo: Código fuente 95

Siglas y términos

ACE *"Advanced Configuration Environment".*

ASCII *"American Standard Code for Information Interchange".*

ASIC *"Aplication Specific Integrated Current"*. Son circuitos integrados de aplicación específica (no programables).

BIOS *"Basic Input/Output System".*

CLB *"Configurable Logic Block"*. Es el nombre que recibe cada bloque lógico de la FPGA.

CPLD *"Complex Programmable Logic Device".*

DCM *"Digital Clock Manager*".

DDR SDRAM *"Double Data Rate Synchronous Dynamic RAM".*

DIMMS *"Dual In-line Memory Module".*

EPROM *"Erasable Programmable Read Only Memory".*

EEPROM "*Electrically Erasable Programmable Read Only Memory*".

FIFO *"First In First Out"*.

FPGA *"Field Programmable Gate Array"*. Son dispositivos de lógica programable con gran capacidad.

GAL *"Generic Array Logic"*.

ISE *"Integrated Software Environment"*.

LUT *"Look Up Table"*.

PAL *"Programmable Array Logic"*.

PLA *"Programmable Logic Array"*.

PLD *"Programmable Logic Device"*.

PROM *"Programmable Read Only Memory"*. Es una memoria programable de sólo lectura.

RAM *"Random Access Memory"*. Memoria en la que es posible el acceso directo a una posición de la misma.

SPLD *"Simple Programmable Logic Device".*

VGA *"Video Graphics Array".*

VHDL Estas siglas vienen de VHSIC *“Very High Speed Integrated Circuit” Hardware Description Language*. Se trata de un lenguaje que permite definir circuitos digitales.

Capítulo 1. Tecnología

1.1 Introducción

En este capítulo vamos a realizar una descripción detallada de la tecnología que hemos empleado en nuestro proyecto.

En primer lugar, haremos una pequeña introducción a los dispositivos de lógica programable, centrándonos en las FPGAs, ya que ha sido el dispositivo que hemos utilizado en el proyecto.

Mostraremos aún dos herramientas más. La primera es el analizador de señales Agilent Technologies 16700-Series Logic, que hemos utlizado para la depuración en la última etapa del proyecto. La segunda, es el MIG 007, que genera automáticamente interfaces de memoria para interactuar con DDR SDRAMs.

1.2 Dispositivos programables. FPGAs

1.2.1 Introducción

Dentro de los dispositivos hardware podemos distinguir entre los de lógica programable y los no programables.

Los dispositivos no programables, ASIC (*"Aplication Specific Integrated Current"*), se fabrican a la medida para usos determinados, no siendo programables por el usuario. Su costo es elevado, por lo que resultan rentables cuando se requieren en gran cantidad.

En el siguiente apartado nos centraremos en los dispositivos de lógica programable, por ser el grupo al que pertenecen las FPGAs.

1.2.2 Dispositivos de lógica programable

Un dispositivo de lógica programable es un circuito de propósito general cuya función interna puede modificarse a voluntad para implementar una extensa gama de aplicaciones.

El primer dispositivo de lógica programable fue una memoria PROM (*"Programmable Read Only Memory"*). Una PROM es una memoria programable de sólo lectura, cuya arquitectura generalmente consiste en un número fijo de términos AND que alimenta una matriz programable OR, y que principalmente se usa para

decodificar combinaciones de entrada en funciones de salida. Existen dos tipos básicos: Las programables por máscara, programadas en la fábrica, y las programables por el usuario final. Éstas son las EPROM y las EEPROM, PROMs *borrables* y eléctricamente *borrables* respectivamente, que, aunque proporcionan menos prestaciones, son menos costosas y se pueden programar inmediatamente.

A continuación expondremos un breve resumen de los diferentes dispositivos que se encuentran dentro del grupo de dispositivos de lógica programable.

PAL: Corresponde a las siglas de *"Programmable Array Logic"*. Es un dispositivo de matriz programable. Consiste en una matriz de puertas AND programable, y otra de puertas OR no programable. Es el sistema programable por el usuario más popular.

GAL: Son las siglas de "Generic Array Logic". En un dispositivo de matriz lógica genérica. Son apropiadas para diseños que se implementan utilizando varias PAL comunes.

PLA: Se corresponde con las siglas de "Programmable Logic Array". Es un dispositivo de matriz lógica programable. Realmente son un avance de las PAL, ya que tanto la matriz de puertas AND como la de puertas OR son programables, dotándolas de mayor flexibilidad.

CPLD: Viene de las siglas de "Complex Programmable Logic Device". Si bien los dispositivos que hemos mencionado hasta ahora pertenecen al grupo de los SPLDs ("Simple Programmable Logic Device"), los PLDs complejos pueden verse como grandes PALs (su arquitectura básica es similar) con características de la PLA.

FPGA: Son las siglas de "Field Programmable Gate Array", que puede traducirse como matrices de puertas programables en campo. Consisten en matrices eléctricamente programables con varios niveles de lógica.

Dedicaremos el siguiente apartado a este tipo de dispositivo, ya que ha sido el que hemos empleado durante el desarrollo de nuestro proyecto.

1.2.3 FPGAs

1.2.3.1 Introducción

Las FPGA's son dispositivos de lógica reconfigurable capaces de implementar prácticamente cualquier función que se desee. Esto se consigue interconectando los elementos básicos que realizan las operaciones más simples por medio de una densa red de conexiones. Existe una gran cantidad de elementos básicos que pueden interconectarse de muchas maneras diferentes, lo cual dota a estos dispositivos de una gran flexibilidad y capacidad.

Actualmente existen diversas compañías que desarrollan este tipo de dispositivos, entre ellas Actel, Altera, Atmel, etc. En adelante estudiaremos la estructura de estos dispositivos centrándonos en las familias de los desarrollados por

Xilinx, ya que son los que la Universidad Complutense nos ha suministrado. Por otra parte, se trata de la compañía más destacada en la tecnología FPGA.

Entre las ventajas de las FPGAs destacamos las siguientes:

- Poseen una alta densidad de integración en un chip.
- Elevado rendimiento.
- Bajo coste de prototipado.
- Corto tiempo de producción.
- Alta velocidad en el tratamiento de las señales de entrada.
- Gran número de entradas y salidas definibles por el usuario.
- Esquema de interconexión flexible.
- Entorno de diseño parecido al de matriz de puertas, pero sin estar limitadas a la matriz de puertas AND y OR.

También exponemos los principales inconvenientes de este tipo de dispositivos:

- Los canales de conexión son a menudo largos e introducen retardos mayores de los que puede soportar el diseño.
- La optimización del chip es baja debido a que suelen sobrar recursos sin utilizar por la falta de canales de conexión.
- Actualmente, los entornos de desarrollo son propietarios y las licencias elevadas. Hay que emplear las herramientas del fabricante de las FPGAs, que son caras. No existen a día de hoy alternativas libres.
- El precio de una FPGA, que es bastante más elevado que el de un microcontrolador.

En este apartado introduciremos la estructura general y características de las FPGAs, si bien posteriormente nos centraremos en las FPGA Virtex II y Virtex II Pro, que son las que hemos utilizado en nuestro proyecto.

1.2.3.2 Estructura interna de las FPGA

A diferencia de las PLA o las PAL, las FPGAs no están estructuradas como matrices de puertas AND y OR, sino por bloques lógicos que contienen registros de almacenamiento y lógica programable combinacional capaz de implementar las funciones que se requieran sobre sus variables de entrada. Estos bloques están interconectados mediante los conmutadores necesarios. Así, en función de cómo están establecidas las conexiones entre los bloques, se obtiene un dispositivo u otro.

En los dispositivos Xilinx este elemento básico de procesamiento recibe el nombre de CLBs (*"Configurable Logic Block"*).

Alrededor poseen un anillo de células de entrada/salida (IOBs). Cada una de estas células puede programarse de manera independiente para comportarse como una entrada, una salida, o ambas (bidireccional). Tienen unos flip-flops que hacen de buffers de entrada/salida.

La interconexión se realiza a través de una red de líneas horizontales y verticales que unen las filas y columnas de los bloques lógicos.

En la figura 1.1 se muestra la estructura interna de una FPGA Xilinx..

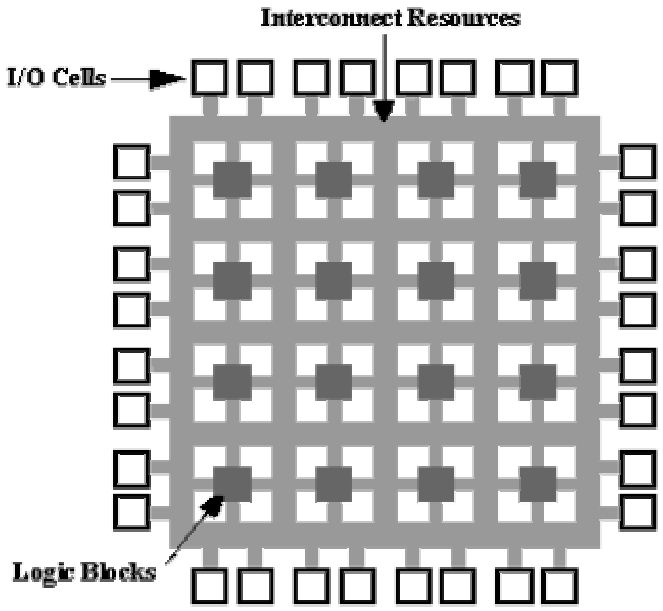


Figura 1.1: Estructura interna de una FPGA (Xilinx)

La forma en que están configuradas las conexiones entre los bloques lógicos se almacena en un fichero llamado bitstream, de manera que cualquier diseño que implementemos con la FPGA tendrá su correspondiente bitstream. Al ser cargado el bitstream en la memoria de configuración de la FPGA, hará que ésta se comporte como el diseño implementado.

En las FPGAs Xilinx, cada CLB tiene dos (CLB simple) o cuatro (en modelos más avanzados) *slices*, y cada slice dos LUT (*"Look Up Table"*). Esto no es más que una jerarquía para manejar la complejidad así como los fenómenos locales y globales. En la figura 1.2 mostramos los dos slices de una CLB simple.

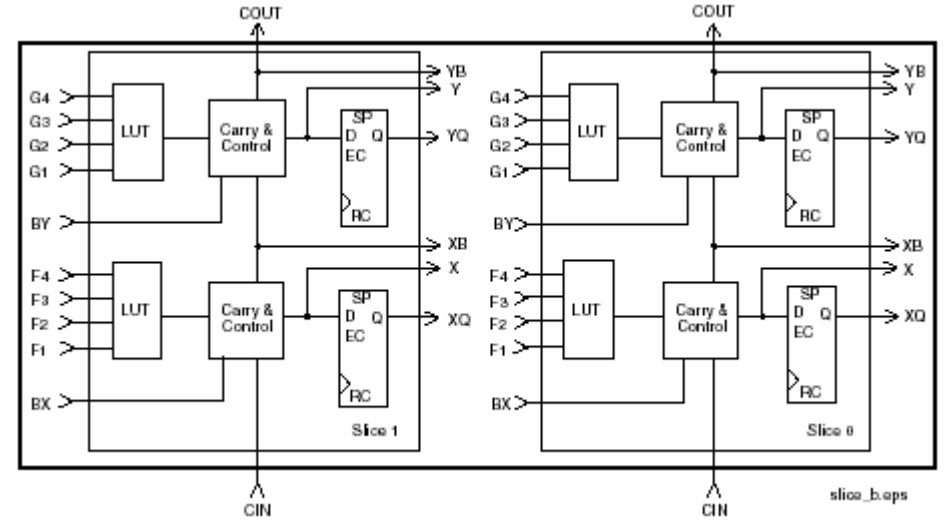


Figura 1.2: Slices de un CLB (Virtex)

En cada slice del CLB observamos dos elementos importantes: Las dos LUTs y los dos registros.

Cuando configuramos el elemento lógico se carga en las LUTs un mapa de la función lógica. De esta manera una LUT puede funcionar como AND, OR, o cualquier función simple con cuatro entradas y una salida. Concatenando LUTs e interconectando salidas puede implementarse cualquier función lógica.

Los registros de los CLB permiten generar dispositivos de lógica síncrona, que permite almacenar resultados intermedios, generar "iteraciones" y realizar pipelines. La introducción de resultados intermedios permite que se separen etapas de computación de manera que el tiempo entre una entrada y su salida es similar o mayor que el caso directo. Sin embargo, la tasa de entrada y salida aumenta notablemente, ya que no hay necesidad de esperar a que se estabilicen las señales del circuito completo, sino únicamente la sección mayor de todas las secciones entre registros (ruta crítica). De ahí la importancia de la cantidad de registros.

Vemos además que un CLB posee bloques de lógica de control, para configuración local de señales de entrada y salida, modo de funcionamiento del slice, etc, y de cuenta, para arrastrar de un slice a otro señales resultantes de operaciones aritméticas como sumas o multiplicaciones.

Otro factor clave en la flexibilidad de las CLBs es la diversidad de buses de interconexión distintos que existen en estas FPGAs. Existen líneas de alta velocidad que conectan CLBs para configurar memorias RAM distribuidas, o llevar los bits de cuenta en operaciones aritméticas de un sector del sumador o multiplicador a otro. Además de estas líneas, cada CLB estará conectado con sus 8 CLBs anexos, y será ésta la línea de flujo de datos más clara. Para conectarse con otros CLBs lejanos, hay buses que conectan uno de cada dos, tres o seis CLBs en la misma columna o fila.

Existen, además, una serie de bloques especiales en algunas FPGAs de gama media/alta, normalmente colocados dentro de la matriz de CLBs, bien en el centro, bien como columnas dentro de ella y maximizando el perímetro de interconexión. En nuestro caso, una de las dos FPGAs empleadas pertenece a la familia de dispositivos VIRTEX II Pro, que posee además de los elementos anteriormente explicados multiplicadores de 18 x 18 bits, bloques de memorias RAM, dos procesadores RISC PowerPC, y bloques de entrada salida de alta velocidad.

1.2.3.4 Virtex II

La primera placa que utilizamos en el proyecto consiste en una FPGA de la familia Virtex II, en torno a la cual se encuentra una serie de componentes periféricos. Vamos a comentar brevemente cuáles son las características básicas de las FPGAs de la familia Virtex II, que pueden verse en forma de tabla en la figura 1.3.

Device	System Gates	CLB (1 CLB = 4 slices = Max 128 bits)			Multiplier Blocks	SelectRAM Blocks		DCMs	Max I/O Pads[1]
		Array Row x Col.	Slices	Maximum Distributed RAM Kbits		18 Kbit Blocks	Max RAM (Kbits)		
XC2V40	40K	8 x 8	256	8	4	4	72	4	88
XC2V80	80K	16 x 8	512	16	8	8	144	4	120
XC2V250	250K	24 x 16	1,536	48	24	24	432	8	200
XC2V500	500K	32 x 24	3,072	96	32	32	576	8	264
XC2V1000	1M	40 x 32	5,120	160	40	40	720	8	432
XC2V1500	1.5M	48 x 40	7,680	240	48	48	864	8	528
XC2V2000	2M	56 x 48	10,752	336	56	56	1,008	8	624
XC2V3000	3M	64 x 56	14,336	448	96	96	1,728	12	720
XC2V4000	4M	80 x 72	23,040	720	120	120	2,160	12	912
XC2V6000	6M	96 x 88	33,792	1,056	144	144	2,592	12	1,104
XC2V8000	8M	112 x 104	46,592	1,456	168	168	3,024	12	1,108

Figura 1.3: Características de las FPGAs Virtex II

El dispositivo que contiene nuestra placa es el **XC2V2000**, correspondiente a la séptima fila del cuadro de la imagen. Puede verse que la FPGA cuenta, entre otras características, con:

- 10752 slices.
- Un máximo de 336Kb de RAM distribuida.
- Un máximo de 1008Kb de RAM (bloques).
- 56 módulos multiplicadores.
- 8 DCMs.

La placa está alimentada de una fuente externa de 5V.

Esta placa cuenta con varios dispositivos periféricos. A continuación resumimos los más importantes para este proyecto:

Salida SVGA. Un triple conversor digital-analógico de 8 bits con un pixel clock de una tasa maxima de 100MHz soporta a una pantalla SVGA de 1024 x 768 pixels con una tasa de refresco vertical de 85Hz.

Ethernet. Esta interfaz soporta transmisión de datos full-duplex y half-duplex a 10Mb/s y 100Mb/s con autonegociación.

Puerto RS-232. La placa cuenta con un puerto RS232 y dos puertos PS2 para teclado y ratón. Estos dos puertos de serie comparten conexiones con la FPGA, no estando los tres disponibles al mismo tiempo. A fin de seleccionar qué puertos están activos hay unos switches llamados “serial port selection switches”.

Memoria. La placa contiene cinco bancos independientes de 512k x 32 RAM ZBT con una tasa de reloj máxima de 130MHz.

Generación de relojes: La FPGA opera en base a los relojes derivados del reloj del sistema, con una frecuencia de 27MHz. El *master_clock* que llega a la FPGA va sincronizado con los relojes de los periféricos. Los DCMs sirven para multiplicar este reloj y proporcionar un sincronismo de manera que el sistema completo pueda operar de forma síncrona. Son generados tres relojes internos, de 27MHz, 53MHz, y 108MHz.

En la imagen 1.4 se ve esta placa que hemos comentado.

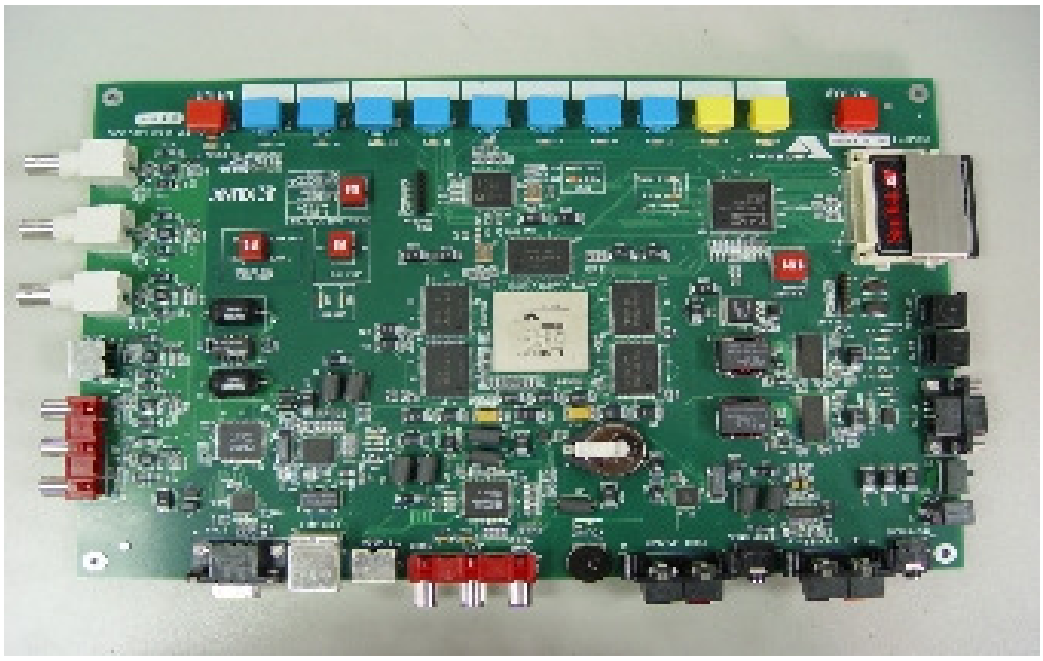


Figura 1.4

1.2.3.5 Virtex II Pro

La segunda placa de prototipado empleada proporciona una plataforma hardware avanzada que consiste en una FPGA de la familia Virtex II Pro rodeada de una serie de componentes periféricos. En esta sección expondremos las características básicas de esta placa y daremos una breve descripción de sus componentes.

La FPGA Virtex II Pro **XC2VP30** cuenta con:

- 13969 slices.
- 2448Kb de RAM (bloques).
- 428Kb de RAM distribuída.
- 136 multiplicadores.
- 8 DCMs.
- 2 RISC PowerPC.

Fuente de energía. La placa está alimentada de una fuente de 5V. Las fuentes de la placa generan 3.3V, 2.5V, y 1.5V para la FPGA.

Componentes de la placa:

RAM. Es posible instalar en la placa un módulo de memoria DDR SDRAM (*Double Data Rate Synchronous Dynamic RAM*). Soporta módulos de memoria de hasta 2Gb de capacidad.

Controlador System ACE de la flash. Esta placa soporta un controlador System ACE (*Advanced Configuration Environment*) que maneja los datos de la configuración de la FPGA.

Ethernet. Esta interfaz soporta transmisión de datos full-duplex a 10Mb/s y 100Mb/s con autonegociación y detección paralela.

Puertos de serie. La placa tiene 3 puertos de serie que se utilizan para la conexión de periféricos como teclado, ratón y pantalla.

Leds y switches. Hay un total de cuatro leds a disposición del usuario, que se iluminan cuando reciben el valor lógico '0'. Hay un switch de cuatro posiciones que envía un '0' lógico a la FPGA cuando está arriba (o en estado *on*).

Conectores de expansión. 80 pins de entrada/salida de la FPGA van a estos conectores cuyo uso puede decidir el usuario. Por ejemplo, son de utilidad para la depuración.

Salida XSGA. Opera con el pixel clock a 180MHz, compatible con una tarjeta gráfica VESA para una salida de 1280 x 1024 y frecuencia de refresco 75Hz y resolución máxima de 1600 x 1200 con una frecuencia de refresco de 70Hz.

AC97 Codec de audio. La placa incluye un codec de audio y un amplificador estéreo.

Interfaz programable USB 2. Un microcontrolador de comunicaciones con hosts USB a 480Mb/s o 12Mb/s. Esta interfaz se usa para programar o configurar la FPGA en modo boundary-scan. Las velocidades de los relojes pueden seleccionarse en un rango desde 750kHz hasta 24MHz.

Generación de relojes: Esta placa soporta los seis relojes que se muestran a continuación.

- Un *system clock* de 100MHz.
- Un reloj de 75MHz (puertos SATA).
- Un reloj alternative *alternate_clock*.
- Un reloj externo para los MGTs.
- Un reloj de 32MHz para la interfaz System ACE.
- Un reloj del modulo de expansión de alta velocidad Digilent.

En la figura 1.5 mostramos una imagen de la placa en la que se han señalado todos sus componentes periféricos.

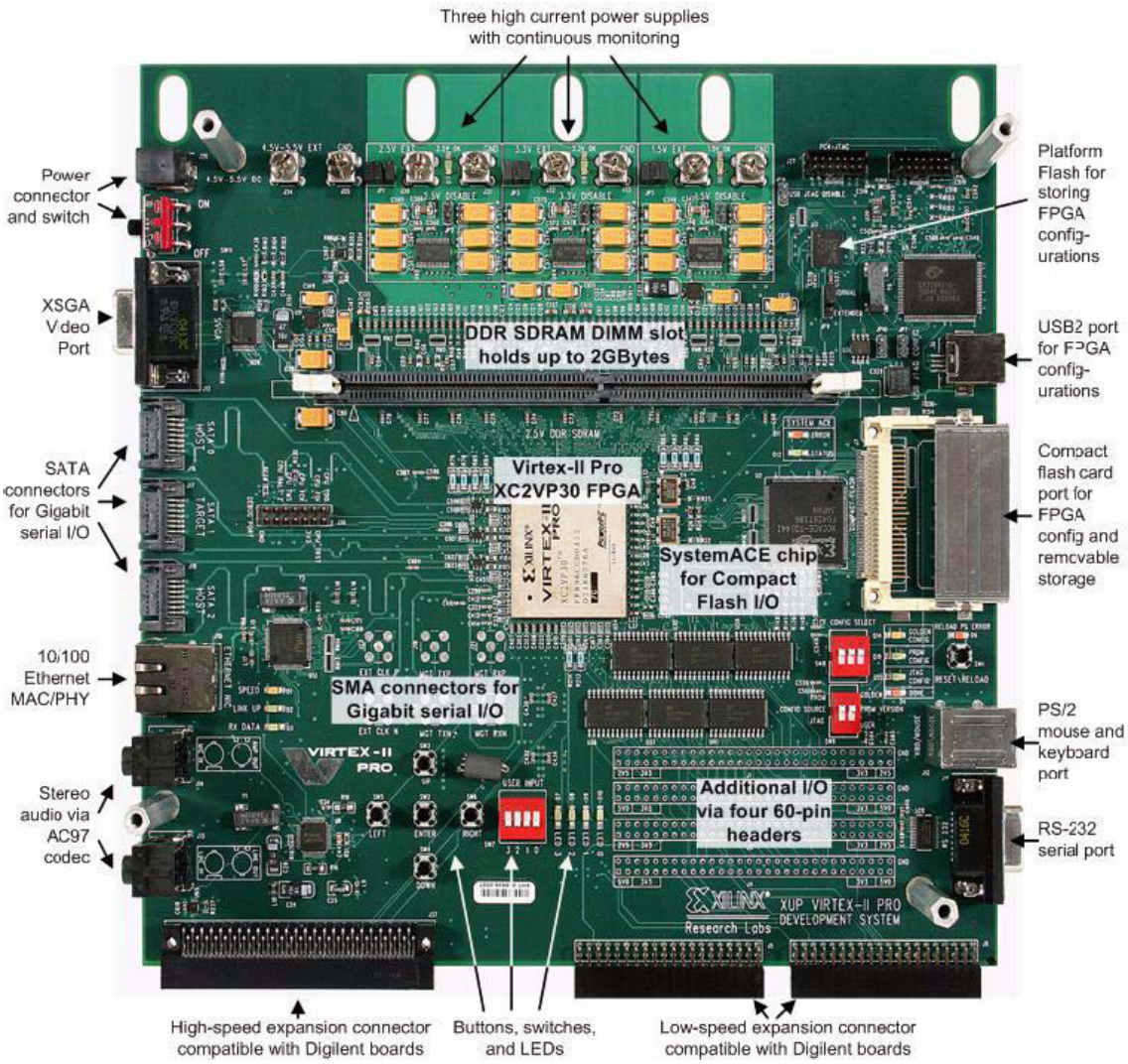


Figura 1.5

1.3 Xilinx-ISE 6.2 y 7.0

Xilinx-ISE (*"Integrated Software Environment"*) es una herramienta de diseño de circuitos que permite realizar y simular diseños de circuitos, ya sea a través de esquemáticos, o bien empleando lenguajes específicos de diseño, como, en nuestro caso, VHDL.

Esta herramienta consta de dos partes diferenciadas: El Project Navigator es la parte que nos permite realizar el diseño del circuito. Con el ModelSim simulamos el circuito que hemos diseñado e implementado con la anterior. Ésta última ha sido de especial utilidad a la hora de verificar la corrección de nuestros algoritmos, ya que la depuración de errores habría sido una tarea prácticamente imposible sin ella. Con este fin, otro dispositivo clave a la hora de la depuración ha sido el analizador de señales, al que dedicaremos el último apartado de este capítulo.

En la figura 1.6 mostramos una captura del entorno del Project Navigator de Xilinx-ISE.

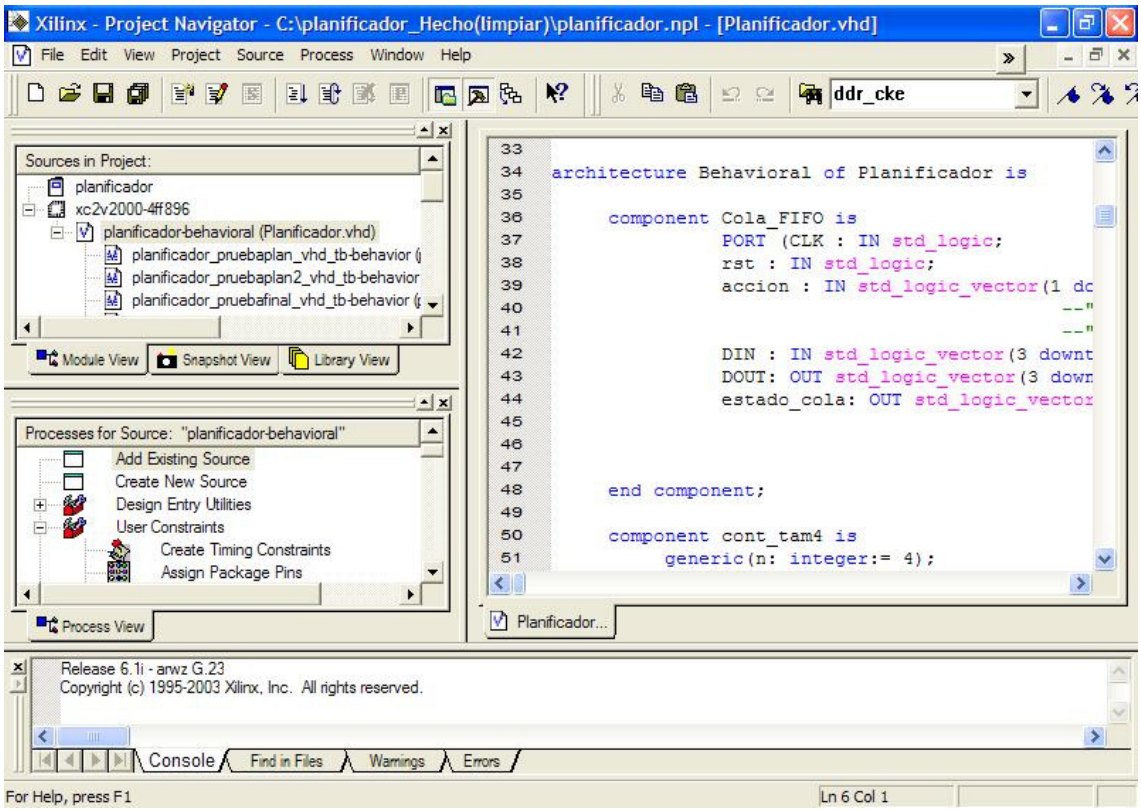


Figura 1.6: Entorno de desarrollo de Xilinx-ISE

En la zona más extensa de la figura (derecha) se encuentra la hoja en la que escribimos nuestro programa. Pulsando la flecha que puede verse arriba a la derecha podemos desunirla del resto de la interfaz del Project Navigator, aumentando el área visible del programa para mayor comodidad.

El área superior a la izquierda posee varias pestañas que seleccionamos en función de lo que nos interese observar. La que empleamos habitualmente es la que se muestra en la figura, en la que se ve la estructura jerárquica de módulos del proyecto que se encuentra abierto. En el mismo área, y seleccionando las otras dos pestañas, podemos tener una vista de las librerías o una captura instantánea del proyecto.

Justo debajo está la vista de los procesos, en la que se encuentran todos los pasos desde añadir un nuevo archivo fuente al proyecto, hasta generar el archivo bitstream del proyecto. Habitualmente utilizaremos las opciones de chequear nuestra sintaxis, sintetizar, o configurar el dispositivo para cargar el bitstream.

Por último, debajo de estas regiones hay una consola en la que visualizaremos el proceso completo de síntesis del proyecto, así como los errores y advertencias que se produzcan en él.

En ocasiones no estaremos interesados en generar un archivo bitstream para volcar en la FPGA, sino en simular el comportamiento del circuito para comprobar la corrección de su funcionalidad. En estos casos crearemos un archivo de pruebas, comúnmente denominado banco de puebas, en el que forzaremos los valores de las entradas de nuestro sistema a conveniencia, con el fin de observar los valores que

toman las salidas en las diferentes situaciones. Cuando seleccionamos en el visor de módulos un archivo de pruebas el contenido de la región que se encuentra justo debajo varía su contenido como puede verse en la figura 1.7.

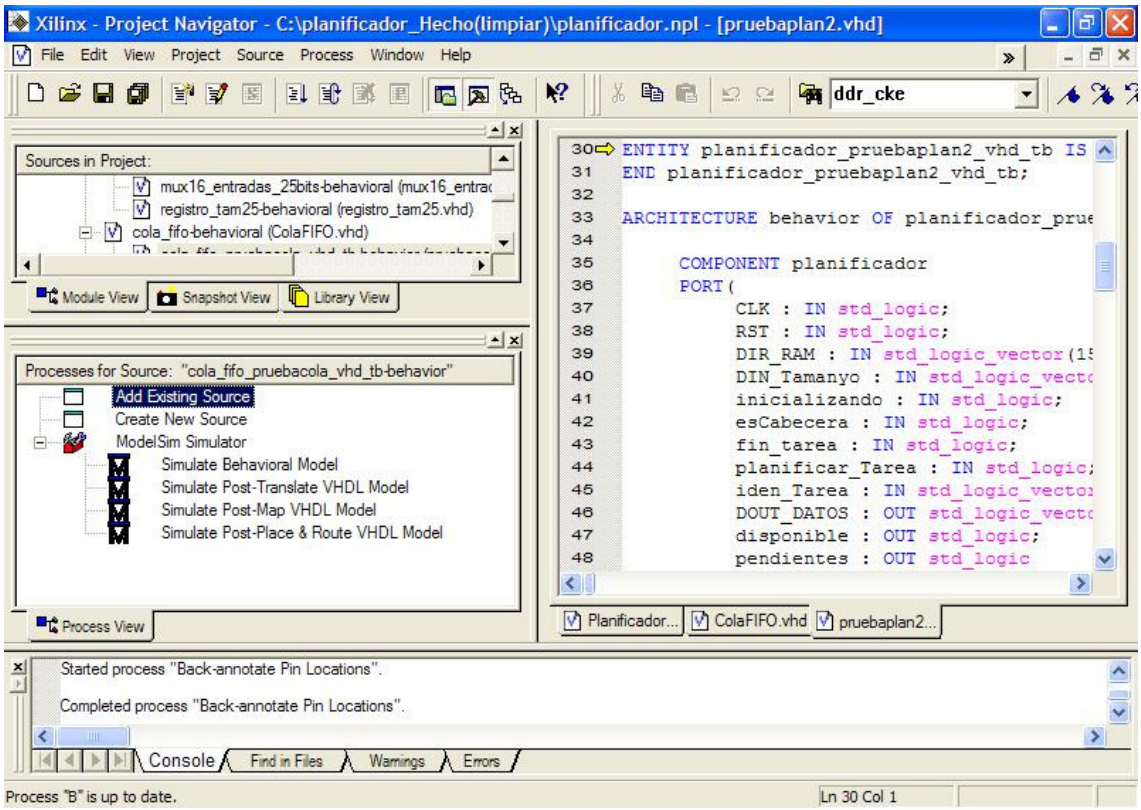


Figura 1.7: Entorno de desarrollo de Xilinx-ISE (banco de pruebas)

Ahora las opciones que aparecen en esta zona son diferentes, podemos observar que se reducen al realizar una serie de simulaciones con el Model Simulator. Si seleccionamos la opción “simulate behavioral model” aparece este programa. A continuación comentamos brevemente la colección de ventanas que aparecen y que pueden verse en la figura 1.8.

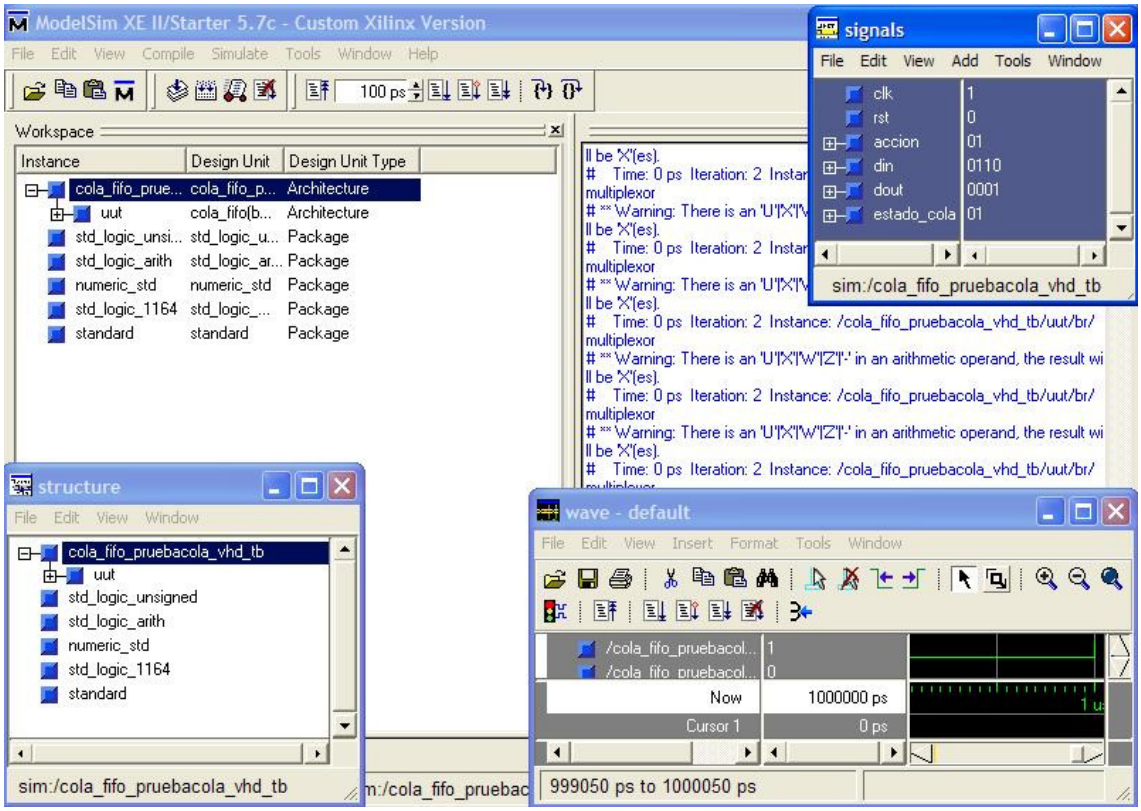


Figura 1.8: Model Simulator

De fondo en la imagen se ve la ventana principal, desde la que se lanzan todas las demás. Cuando se carga el diseño se muestran en la zona derecha de ésta los errores y/o avisos, en caso de que los hubiera. En la parte izquierda, según la pestaña que seleccionemos, podemos ver los archivos que componen el diseño, su estructura jerárquica, o las librerías que se utilizarán.

La ventana que ocupa la parte inferior izquierda en la figura 1.8 muestra la estructura general del diseño cargado. Cuando se selecciona en ella cualquiera de los módulos que aparecen, en la ventana de señales (que aparece en la parte superior derecha de la imagen) aparece un listado de las señales de entrada, salida, e internas que intervienen en el módulo seleccionado. Podremos seleccionar las que consideremos oportunas para incorporarlas a la lista de señales de la tercera ventana, en la que veremos los valores que toman con el paso del tiempo en función de los estímulos aplicados a las señales de entrada en el banco de pruebas.

Esta ventana puede verse en la parte inferior derecha de la imagen, siendo la más interesante de todas. En el próximo capítulo mostraremos el resultado de algunas simulaciones significativas de los módulos que hemos diseñado e implementado para nuestro proyecto, realizadas a través de este programa.

1.3 Analizador de señales

El analizador de señales AGILENT TECHNOLIGIES 16700 es un dispositivo que ha puesto a nuestra disposición la Universidad Complutense de Madrid. Este aparato nos ha servido de mucha utilidad a la hora de depurar nuestros diseños una vez cargados en las FPGAs. Básicamente sirve para visualizar las señales de las que deseemos comprobar sus valores. Estas señales se envían al analizador a través de unas conexiones físicas a la placa.

El mecanismo de depuración consiste en lo siguiente: En primer lugar el usuario debe seleccionar, en el entorno de la aplicación de usuario del analizador, las señales que desea visualizar. Una vez hecho esto, debe configurar el sistema mediante la definición de disparadores que hagan que se muestre en pantalla el estado de las señales que se desea observar. Los tipos de disparadores que ofrece este analizador son muy variados y completos, desde que se produzca un simple flanco de subida en una señal o un cierto valor en un bus, hasta que se den varios valores simultáneos o en un cierto intervalo de tiempo. Cuando se produce el evento esperado, el analizador muestra un cronograma en el que se muestra el instante de tiempo en el que saltó el disparador. Además, se muestra el valor de las señales que se han seleccionado un cierto tiempo antes y después de que ocurra el evento que desencadenó el salto del disparador.

1.4 MIG 007

Ésta es una herramienta capaz de generar interfaces de memoria para diferentes módulos DDR SDRAM y FPGAs. En la figura 1.9 se muestra el interfaz del MIG 007.

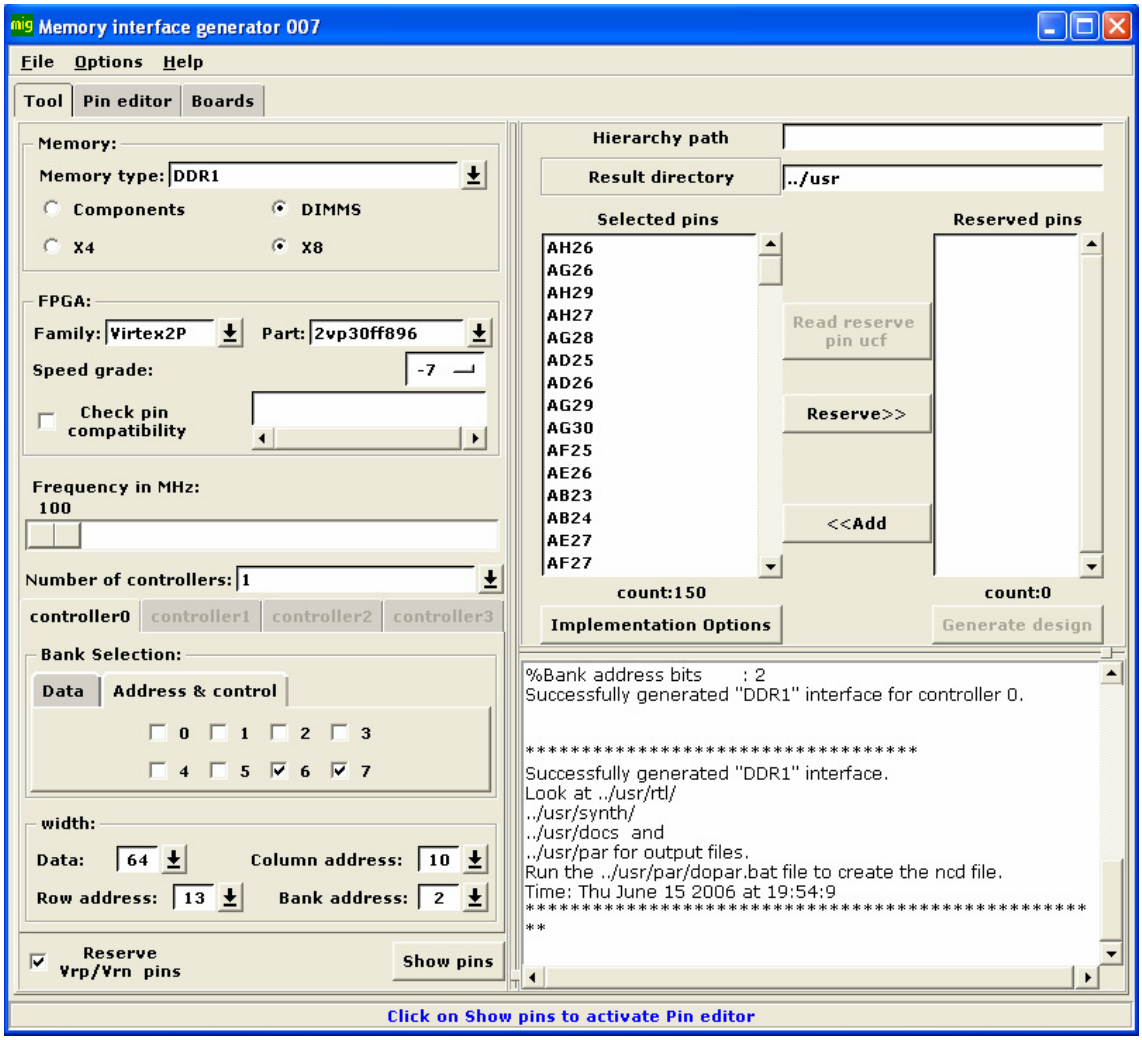


Figura 1.9: Interfaz del MIG 007

Deben configurarse adecuadamente los campos que pueden verse en la imagen. Para empezar, se seleccionará el tipo de módulo para el cual la herramienta generará el interfaz. Marcamos las opciones DIMMS (*Dual In-line Memory Module*), que indica el tipo de memoria RAM que empleamos, y la configuración X8 para la anchura de bus.

La siguiente información que se requiere es la relativa a la FPGA con la que utilizaremos el interfaz. Indicaremos la familia, la FPGA (con su nomenclatura completa), y su *speed grade*. A continuación se especifica la frecuencia de funcionamiento en MHz y el número de controladores. Para cada controlador se habilitará el contenido de una pestaña, dentro de la cual se realiza la selección del banco. Para ello deberán seleccionarse los bancos correspondientes en las pestañas “Address & control” y “Data”, que se encuentran dentro de la pestaña del controlador correspondiente. Para especificar los bancos de la placa a los que está conectado nos hemos fijado en la figura 1.10. En ella se ve la correspondencia de los bancos con los dispositivos periféricos de la placa. Observamos que los bancos correspondientes al módulo de memoria DDR SDRAM son el 6 y el 7, por lo tanto son estos los que seleccionamos.

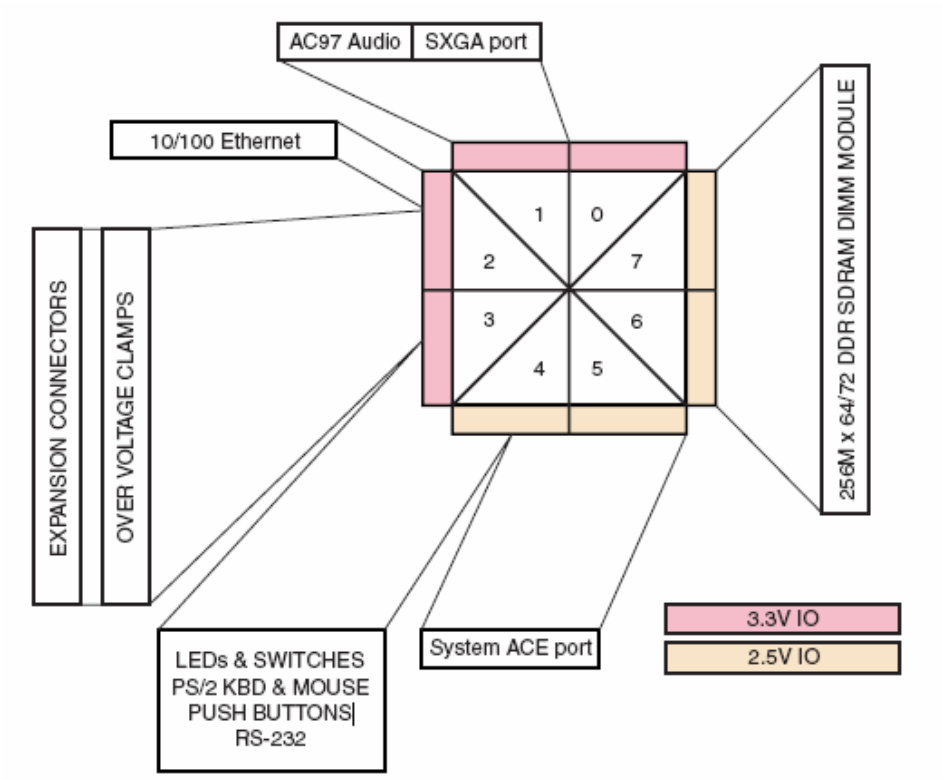


Figura 1.10: Conexiones de los bancos a los dispositivos periféricos

A continuación especificaremos la anchura de los datos, la dirección de la fila y la columna y también la dirección del banco.

Si después de rellenar todos estos campos pulsamos el botón “show pins” aparecerán en la ventana con la etiqueta “selected pins” los pins de la FPGA que han sido seleccionados y la cantidad. Antes de pulsar “generate design”, que generará la interfaz que necesitamos, podemos especificar el directorio en el que queremos que la guarde (en el campo de texto junto a “result directory”).

Finalmente en el cuadro inferior derecho se muestra el resultado del proceso de generación.

Una de las opciones interesantes que ofrece esta aplicación es la de elegir el lenguaje en el que queremos que se genere la interfaz. Basta ir al menú Options (ver figura 1.11), y seleccionar HDL. Se despliega un submenú en el que pueden seleccionarse el lenguaje VHDL o Verilog.

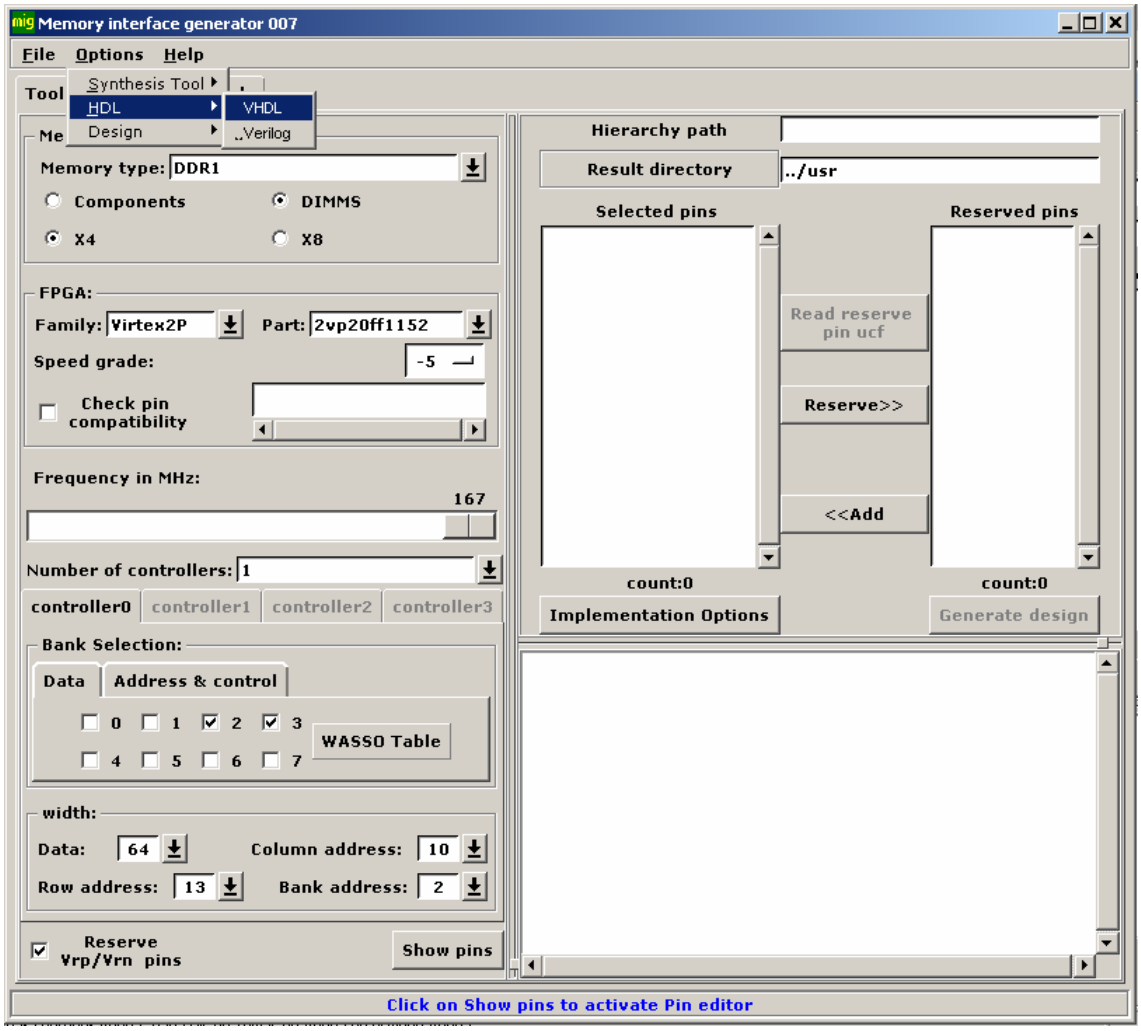


Figura 1.11

Si pulsamos el botón “Implementation Options” aparecerá una ventana en la que, activando los campos correspondientes, podremos decidir si se utilizarán los DCMs, si queremos incluir un banco de pruebas para la interfaz, o el número de canales de escritura. Estas opciones se muestran en la figura 1.12.

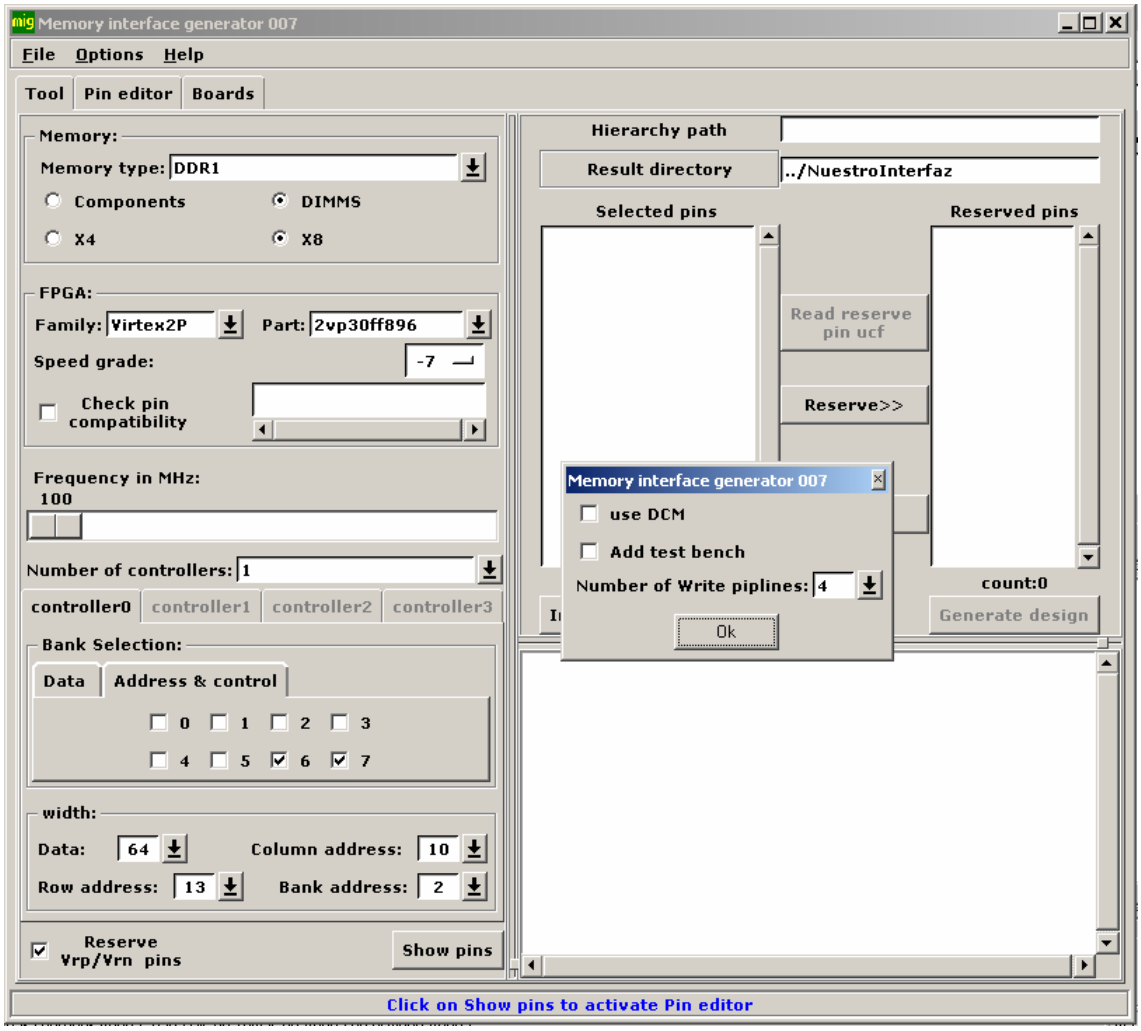

Memory interface generator 007
File Options Help
Tool Pin editor Boards
Memory:
Memory type: DDR1
Components DIMMS
X4 X8
FPGA:
Family: Virtex2P Part: 2vp30ff896
Speed grade: -7
Check pin compatibility
Frequency in MHz:
100
Number of controllers: 1
controller0 controller1 controller2 controller3
Bank Selection:
Data Address & control
0 1 2 3
4 5 6 7
width:
Data: 64 Column address: 10
Row address: 13 Bank address: 2
Reserve Vrp/Vrn pins
Show pins
Hierarchy path
Result directory ../NuestroInterfaz
Selected pins
Reserved pins
Read reserve pin ucf
Reserve>>
count:0
Generate design
Memory interface generator 007
use DCM
Add test bench
Number of Write piplines: 4
Ok
Click on Show pins to activate Pin editor

Figura 1.12

Capítulo 2. Módulos

2.1 Introducción

En la figura 2.1 se muestra el esquemático con la estructura general del proyecto.

En él, la entrada/salida de las tareas se realizaría a través de un interfaz de comunicaciones que diseñaríamos en este sistema, que además permitiera ubicar cada una en la FPGA de manera independiente empleando reconfiguración dinámica y parcial. Se considera una tarea como un mapa de bits que se almacena en memoria y que será leído de la misma cada vez que sea ejecutada.

Los componentes principales del sistema son los siguientes:

Receptor de Aplicaciones: Recibe las aplicaciones desde dos posibles entradas: El almacenamiento local o la red. Almacena las tareas en memoria local y pasa los datos de la aplicación al planificador para que tome las decisiones de planificación.

Planificador de tareas: Controla las tareas recibidas y planifica su ejecución y ubicación siguiendo una política tipo FIFO. Para ello, el planificador incorpora una cola FIFO.

Cargador de inicio: Se encarga de cargar en la FPGA los módulos básicos que permiten el arranque del sistema. Como mínimo incluye un controlador básico para un dispositivo de almacenamiento local no volátil (tarjeta de memoria), y una lista de módulos iniciales (BIOS).

BIOS: Controladores básicos para la E/S: De teclado, vídeo, memoria, y red. Se nos proporcionaron ya diseñados.

Interfaz de comunicaciones: Permite el intercambio de datos y control entre las tareas y el kernel del sistema operativo y los distintos controladores de periféricos.

En este capítulo mostramos detalladamente el diseño e implementación de los módulos que hemos llegado a implementar en este curso: La parte del receptor de aplicaciones que recibe y almacena las tareas en memoria local y transmite los datos de la aplicación al planificador -en adelante nos referiremos a este módulo como gestor de memoria- y el planificador de tareas. El primer gestor de memoria que presentamos fue diseñado e implementado para la FPGA que utilizamos inicialmente, siendo el otro diseñado para la FPGA Virtex II Pro. De éste último únicamente se llegó a implementar su unidad de control, que a día de hoy no es operativa junto al interfaz de la memoria SDRAM que nos fue proporcionado. También presentamos brevemente los módulos de los controladores de entrada/salida (teclado, VGA).

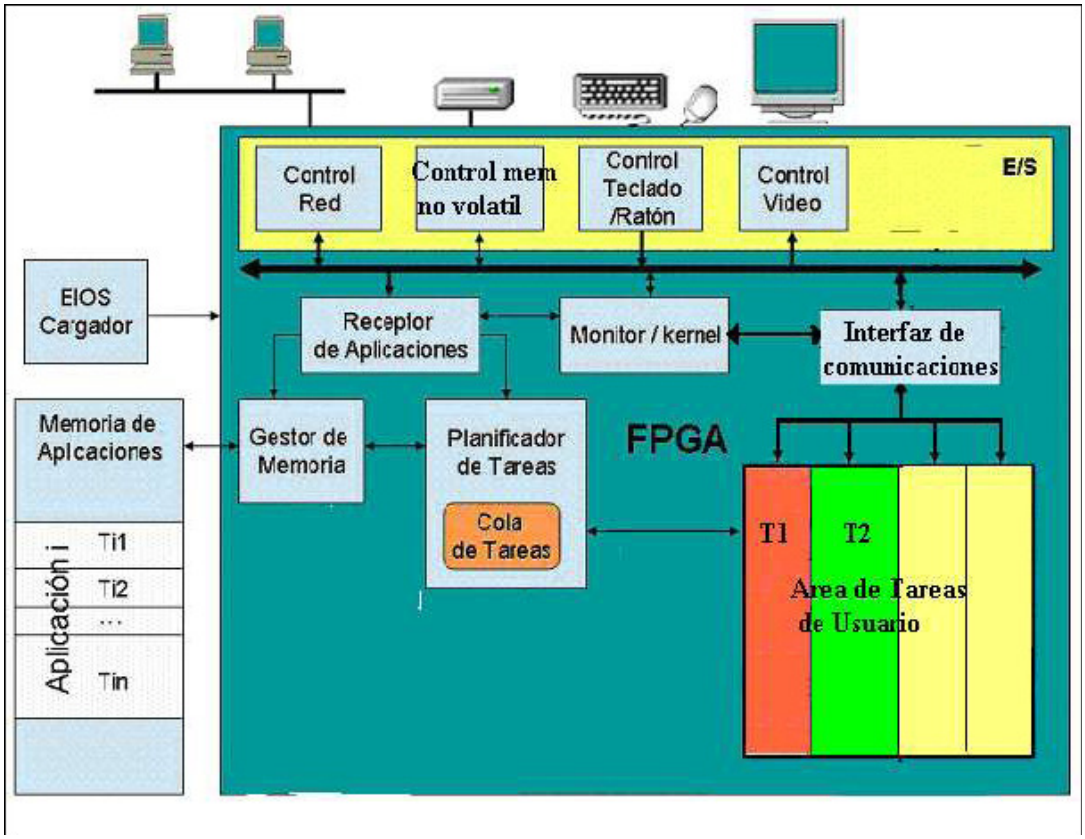


Figura 2.1: Estructura general del proyecto

2.2 Comprobador de Tecla

2.2.1 Introducción

Éste es un módulo que se nos ha proporcionado y que implementa el controlador de teclado. El teclado está conectado físicamente a uno de los dos puertos PS/2 de la placa y usa un bus serie para comunicarse con la FPGA. Este bus incluye el reloj y datos con idéntica temporización de señales y el tamaño de palabra es de 11 bits, que incluyen un bit de start, stop y paridad.

Básicamente, la función de este módulo es la de detectar una pulsación de una tecla y transformar los datos referentes a la tecla que ha sido pulsada a código teclado para ofrecérselo a cualquier otro módulo que quiera utilizar el teclado.

A continuación se describen con más detalle las señales que recibe y envía este módulo.

2.2.2 Descripción de señales

El módulo recibe las siguientes señales:

- rst : se trata del reset del módulo;
- clk: es la entrada de reloj del sistema;
- ps2Clk: es la entrada de reloj del teclado;
- ps2Data: es la entrada de datos del teclado, hemos de decir que es de un solo bit, la información de la tecla pulsada se recibe en serie.

Las señales que se describen a continuación son las que el módulo ofrece tras haber gestionado sus entradas:

- pulsacion: es un bit que cuando está a alta indica que se ha realizado una pulsación de tecla;
- tecla: es un vector de 8 bits que identifica en código teclado la tecla que ha sido pulsada.

En la figura 2.2 se muestra el módulo que representa al Comprobador de Tecla:

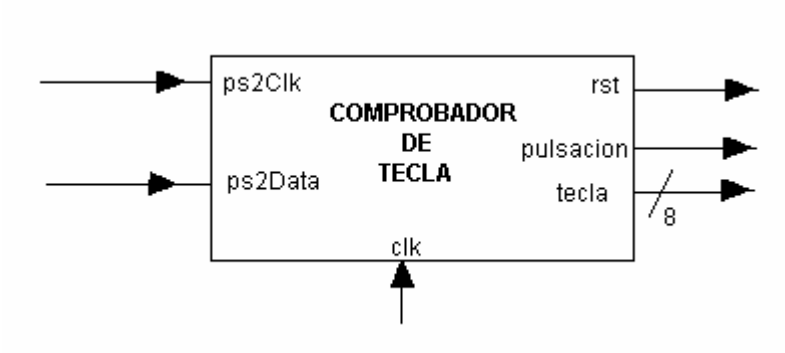


Figura 2.2

Para complementar este módulo hemos realizado un conversor del código de teclado a código ASCII que incluímos en esta sección. Su esquema puede verse en la figura 2.3.

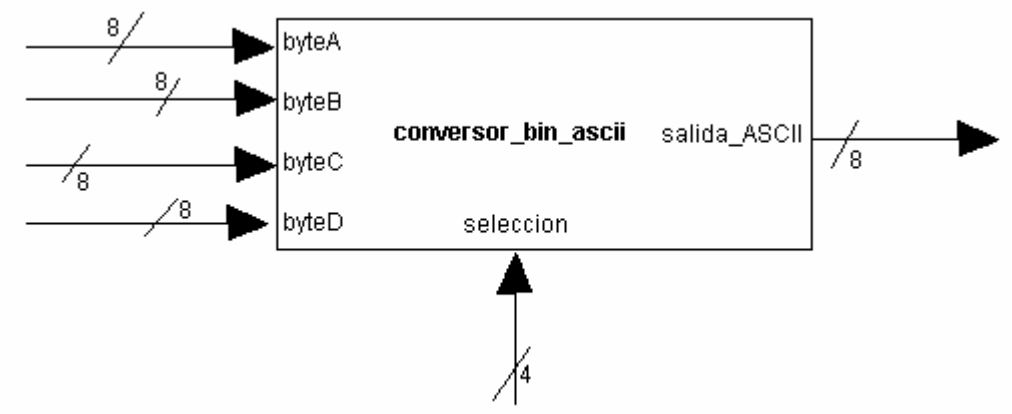


Figura 2.3

Este conversor devuelve en salida_ASCII el código ASCII de parte de los bits de una de las cuatro entradas de datos (byteA, byteB, byteC y byetD) en función del valor de la entrada de selección. Cuando ésta vale “0000” se convierte a código ASCII

el número binario que representan los 4 bits menos significativos de la entrada byteA. Cuando selección vale “0001” se convierte el correspondiente a los 4 bits más significativos de byteA, y así sucesivamente.

2.3 Interfaz de SVGA

2.3.1 Introducción

Éste es un módulo que nos ha suministrado Xilinx para la visualización en pantalla de caracteres a través de la salida XVGA de la placa. A continuación explicaremos el funcionamiento típico de un monitor.

Aunque su funcionamiento es simple desde el punto de vista del usuario, el interior del monitor encierra un sistema complejo. El componente estrella (y el más costoso) es el tubo de rayos catódicos. Éste contiene varios cañones, cuyo cátodo genera electrones, que son acelerados -a través del ánodo- hacia un material fosforescente (la pantalla). El cañón barre toda la pantalla, enfocando cada zona sensible y lanzando un haz de electrones con una cierta intensidad.

La pantalla está formada por una serie de zonas sensibles fosforescentes (píxeles), que al ser excitadas por los electrones, emiten radiación visible hacia el usuario. La intensidad de los haces de electrones condiciona la luminosidad de cada píxel, mientras que la composición del fósforo determina su color.

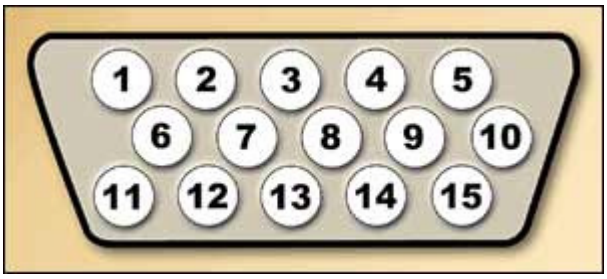
Tras ser excitados, los puntos sensibles de la pantalla son capaces de emitir radiación visible solamente durante un corto periodo de tiempo. Por ello, dichos puntos deben ser excitados de nuevo. Esto se consigue realizando el proceso de barrido multitud de veces por segundo. Si la frecuencia de refresco es apropiada, el usuario percibirá una sensación de continuidad de la imagen en el tiempo. En cambio, si dicha frecuencia es demasiado reducida, la pantalla deja de emitir radiación luminosa entre refresco y refresco, haciendo que el usuario perciba un parpadeo en la imagen. Por otra parte, si la frecuencia de refresco es demasiado elevada, el usuario no va a percibir ninguna ventaja (no hay que olvidar que el ojo tiene su propia frecuencia de muestreo para capturar imágenes) y, además, se requerirá un elevado ancho de banda entre la tarjeta de vídeo y el monitor para mover tanta información por segundo. Por tanto, la elección de la frecuencia de muestreo está sujeta a un compromiso.

Como ya hemos introducido, la misión fundamental del cañón es barrer toda la pantalla, dotando de un color e intensidad luminosa a cada píxel.

Este proceso es imprescindible, y debe repetirse varias veces por segundo (como dato práctico, las frecuencias de refresco estándares son 56, 60, 65, 70, 72, 75, 80, 85, 90, 95, 100, 110 y 120 Hz). En primer lugar, se comienza en el píxel situado en la parte izquierda superior de la pantalla. Entonces, se barren todos los píxeles de la línea superior en sentido horizontal, de izquierda a derecha. A continuación, el haz se desactiva, y el cañón se desplaza hacia el primer píxel de la línea inmediatamente inferior (como si de un “retorno de carro y avance de línea” se tratara). El proceso se repite hasta cubrir toda la pantalla.

Finalmente, el haz se vuelve a desactivar, y el cañón vuelve a enfocar al píxel original, listo para “dibujar” una nueva pantalla.

La tarjeta de vídeo genera señales analógicas (antiguamente digitales) que controlan el funcionamiento del monitor, haciendo posible visualizar imágenes. En otras palabras, dichas señales son las que gobiernan al tubo de rayos catódicos. Por ello se hace necesario establecer una interfaz para la interconexión de tarjetas de vídeo y monitores.



La interfaz empleada en la actualidad se basa en un conector de 15 pines, al que se suele denominar “conector VGA”, puesto que fue el primero en emplearse con dicho estándar. La imagen muestra la disposición de los terminales en dicho conector. Los terminales 1, 2 y 3 contienen las señales de intensidad correspondientes a los colores primarios (rojo, verde y azul), controlando la intensidad de los haces de electrones correspondientes. Los terminales 6 al 8 son las referencias de tensión (masas) correspondientes a dichas señales.

Los terminales 13 y 14 contienen las señales de sincronía horizontal y vertical, respectivamente. La señal de sincronía horizontal contiene pulsos, que indican el final de una línea de píxeles, es decir, el momento en que el de rayos catódicos debe retornar hasta el origen de la línea siguiente.

Por otro lado, la señal de sincronía vertical indica la llegada del haz de electrones al final de la pantalla, o lo que es lo mismo, el momento en que el tubo de rayos catódicos debe retornar al píxel superior izquierdo. Nótese que todas estas señales se envían de forma separada al monitor.

El módulo soporta resoluciones de hasta 800 x 600 (75 filas de 100 caracteres) y usa cuatro bloques RAM para almacenar el código ASCII de los caracteres que se muestran en la pantalla y un bloque RAM como un generador ROM de caracteres. No requiere de ninguna RAM externa. Si se desea incrementar la resolución a 1024 X 768 son necesarios dos bloques RAM adicionales.

Este módulo, que en principio estaba diseñado para la primera placa que utilizamos, tuvo que ser adaptado a los requisitos de temporización de la segunda placa. Básicamente modificamos los parámetros de refresco y sincronismo en el código verilog suministrado, además de utilizar un módulo generador de relojes diferente, que se explica en la sección correspondiente al generador de relojes.

2.3.2 Descripción de señales

En la figura 2.4 se muestra el módulo que representa al controlador de VGA:

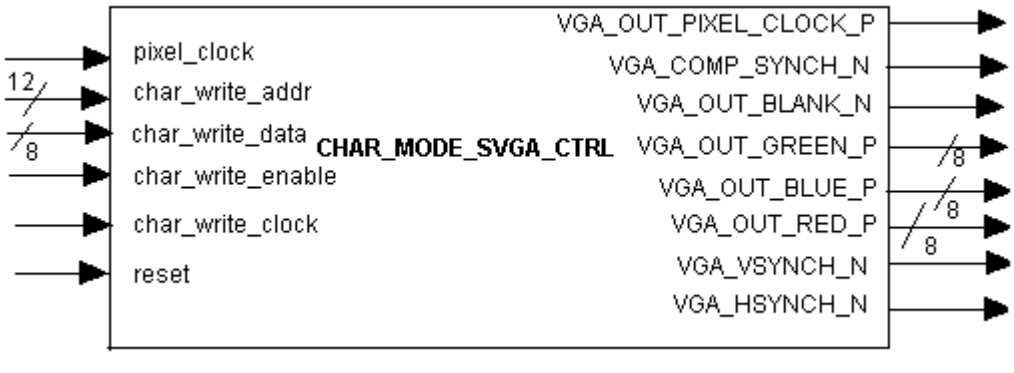


Figura 2.4

Las señales de entrada del módulo son las siguientes:

- pixel_clock: entrada del reloj de píxeles a 25MHz;
- char_write_addres: dirección del carácter que va a ser escrito en la memoria.
- char_write_data: código ASCII del carácter que va a ser escrito en la memoria
- char_write_enable: entrada de capacitación de escritura;
- reset: entrada de reset;

Las señales que envía el módulo a la FPGA son:

- VGA_OUT_PIXEL_CLOCK_P: reloj de píxeles para la pantalla;
- VGA_COMP_SYNCH_N: sincronismo compuesto para la pantalla;
- VGA_OUT_BLANK_N: blanking compuesto para la pantalla;
- VGA_OUT_GREEN_P: señales del color verde de la pantalla;
- VGA_OUT_RED_P: señales del color rojo de la pantalla;
- VGA_OUT_BLUE_P: señales del color azul de la pantalla;
- VGA_VSYNCH_N: sincronismo vertical para el conector XVGA de la placa;
- VGA_HSYNCH_N: sincronismo horizontal para el conector XVGA de la placa;

2.4 Generador de Relojes

2.4.1 Introducción

Éste es un módulo que nos ha suministrado Xilinx para la generación de las señales de reloj necesarias que usan el resto de módulos.

Este módulo logra generar estas señales de reloj usando módulos DCM (*Digital Clock Manager*) cuya función es modular la señal que recibe cambiando su frecuencia y su fase.

2.4.2 Descripción de señales

En la figura 2.5 se muestra el módulo que representa el módulo generador de señales de reloj.

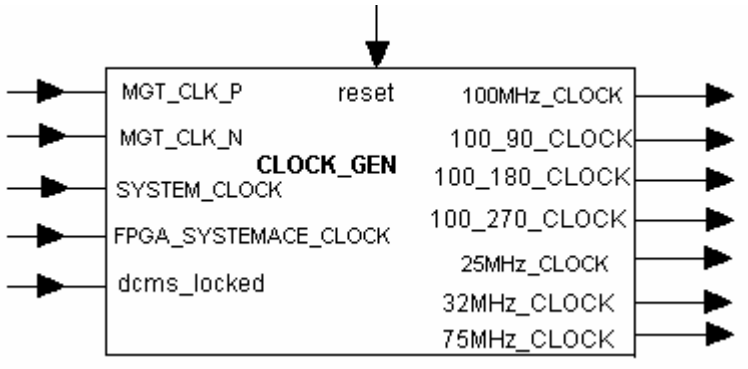


Figura 2.5

Las señales de entrada del módulo son las siguientes:

- MGT_CLKP y MGT_CLK_N: reloj diferencial a 75 MHz;
- SYSTEM_CLOCK: reloj del sistema a 100 MHz;
- FPGA_SYSTEMACE_CLOCK: reloj a 32MHz;
- dcms_locked: indica si los DCM están bloqueados;

Las señales de salida del módulo son:

- 100Mhz_CLOCK: señal de reloj a 100 Mhz;
- 100_90_CLOCK: señal de reloj a 100 Mhz desplazada en fase 90º;
- 100_180_CLOCK: señal de reloj a 100 Mhz desplazada en fase 180º;
- 100_270_CLOCK: señal de reloj a 100 Mhz desplazada en fase 270º;
- 25Mhz_CLOCK: señal de reloj a 25 Mhz;
- 32Mhz_CLOCK: señal de reloj a 32 Mhz;
- 75Mhz_CLOCK: señal de reloj a 75 Mhz;

Cuando cambiamos la placa sobre la que estábamos desarrollando el proyecto fue necesario utilizar otro generador de relojes, simplemente cambiando la frecuencia que se estaba utilizando anteriormente. Estos cambios pueden verse detalladamente en el archivo svga_defines.v que define los parámetros de tiempo requeridos para la VGA. Los parámetros utilizados son para las dimensiones 640 X 480, con una frecuencia de refresco de 60Hz y un pixel clock a una frecuencia de 25.175MHz.

2.5 Gestor de memoria

2.5.1 Introducción

Como ya mencionamos anteriormente, hemos desarrollado este proyecto sobre dos placas de prototipado con dos FPGAs distintas. Cuando se produjo este cambio de placa estaban ya implementados los módulos del gestor de memoria y del planificador de tareas. Así como éste último no fue necesario modificarlo, el primero tuvo que ser completamente rediseñado para adaptarlo a las especificaciones de la nueva memoria (una RAM con tecnología DDR1).

En este apartado expondremos detalladamente el diseño del primer gestor de memoria que realizamos, así como todo lo que llegó a diseñarse e implementarse del segundo. También expondremos las dificultades que nos hicieron imposible la tarea de finalizarlo a tiempo.

2.5.2 Gestor de memoria para la RAM de la FPGA Virtex II

2.5.2.1 Introducción

El gestor de memoria es el módulo que se encarga de almacenar en la memoria RAM las tareas (que se recibirían desde una memoria flash, disco o ethernet) y de enviar estas tareas al módulo planificador de tareas, que se ocupa de controlarlas y planificar su ubicación y ejecución siguiendo una política FIFO. Cuando una tarea llega, a través del almacenamiento local o a través de la red, al receptor de aplicaciones, éste envía sus datos al planificador para que tome las decisiones de planificación, y pide al gestor de memoria que almacene sus datos en la memoria de aplicaciones. En la figura 2.6 se muestran los componentes principales de esta parte del diseño general.

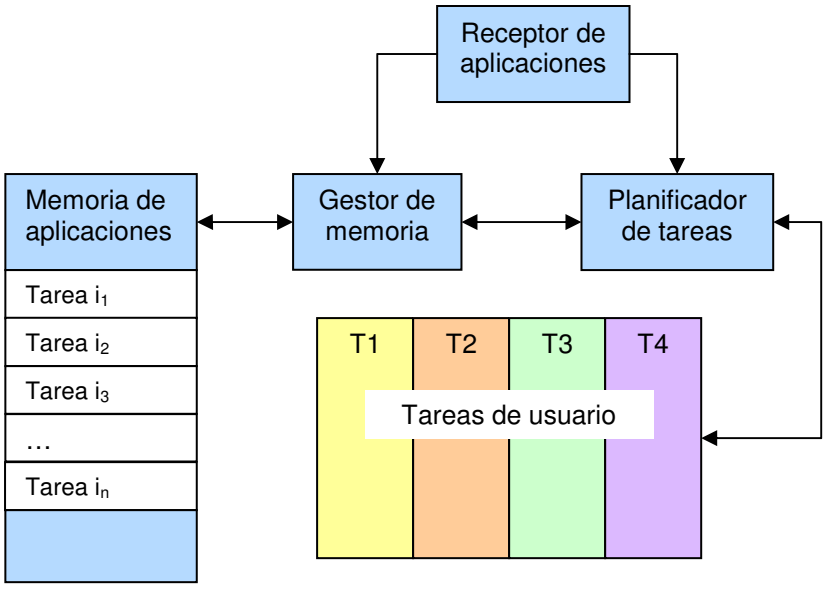


Figura 2.6

2.5.2.2 Modelo de la interfaz del gestor de memoria

La figura 2.7 muestra la estructura básica del gestor de memoria.

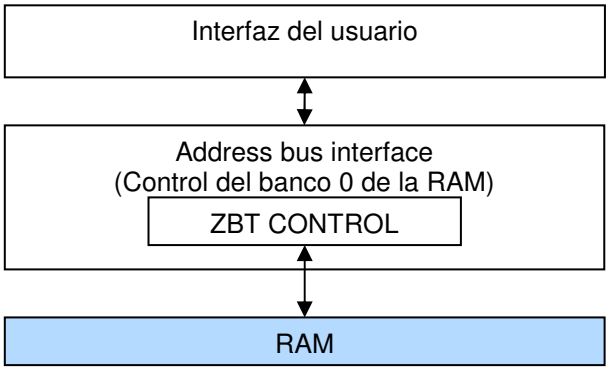


Figura 2.7

El funcionamiento es el siguiente: El usuario indica, a través de su interfaz, cuándo desea realizar una operación de lectura, escritura, o lectura de un bloque de direcciones consecutivas, y proporciona las direcciones y/o datos para realizar dichas operaciones. La comunicación con la memoria RAM se realiza a través del interfaz del banco 0 de la memoria RAM, cuya estructura se detalla en la siguiente sección.

Esta memoria es una ZBT RAM presente en la primera placa que utilizamos. Esta placa contiene cinco bancos independientes de 512k x 32 RAM ZBT con una frecuencia de reloj máxima de 130MHz. Estas memorias pueden utilizarse como buffers de video, mapas de memoria de la SVGA, o para propósito general del usuario. El dispositivo de memoria soporta un bus de 36 bits, pero las limitaciones de pin-out de la FPGA previenen el uso de los cuatro bits de paridad. Las señales de control, dirección, buses de datos y reloj son únicas para cada banco. No hay señales compartidas entre bancos.

El reloj para las RAM ZBT se genera a partir del DCM *memory_dcm*.

2.5.2.3 Interfaz del banco 0 de la RAM

En la tabla 2.1 se muestra un resumen de las señales de entrada y salida que se comunican directamente con la interfaz de usuario.

Nombre de la señal	Dirección	Descripción
user_memory_bank0_clocken	Entrada	Señal activa a alta para capacitación del reloj de la RAM.
user_memory_bank0_ram_clock	Entrada	Señal de reloj de la RAM.
user_memory_bank0_load_addr	Entrada	Señal activa a alta para lectura secuencial.
user_memory_bank0_ce	Entrada	Señal activa a alta para capacitación

		del chip.
user_memory_bank0_wea	Entrada	Señal activa a alta para capacitación de escritura del bloque de bits A.
user_memory_bank0_web	Entrada	Señal activa a alta para capacitación de escritura del bloque de bits B.
user_memory_bank0_wec	Entrada	Señal activa a alta para capacitación de escritura del bloque de bits C.
user_memory_bank0_wed	Entrada	Señal activa a alta para capacitación de escritura del bloque de bits D.
user_memory_bank0_read	Entrada	Señal activa a alta para capacitación de lectura.
user_memory_bank0_write	Entrada	Señal activa a alta para capacitación de escritura.
user_memory_bank0_clock	Entrada	Reloj del banco 0 de la RAM.
user_memory_bank0_addr[18:0]	Entrada	Dirección de escritura del banco 0 de la RAM.
user_memory_bank0_data[31:0]	Entrada	Dato para escribir en el banco 0 de la RAM.
user_memory_bank0_read_data[31:0]	Salida	Dato leído del banco 0 de la RAM.

Tabla 2.1

Descripción de las señales

user_memory_bank0_ram_clock, user_memory_bank0_clock, y user_memory_bank0_clocken se ocupan del reloj de la RAM. La última tiene función capacitadora y está activa a alta.

user_memory_bank0_load_addr se activa para indicar que la lectura se realiza sobre un bloque de direcciones de memoria consecutivas.

user_memory_bank0_ce es la entrada de capacitación del módulo.

Los 32 bits de datos se dividen en 4 grupos de 8 bits cada uno, llamados A, B, C y D de menor a mayor significatividad respectivamente.

Las 4 entradas user_memory_bank0_weX, siendo X A, B, C o D, indican la capacitación de escritura del grupo de 8 bits correspondiente según sea el valor de X.

user_memory_bank0_write y user_memory_bank0_read señalan cuándo se realiza una operación de escritura o lectura sobre la memoria. En un determinado momento sólo una debe estar activa.

Todas estas entradas capacitadoras se consideran activas a alta.

user_memory_bank0_addr[18:0] y user_memory_bank0_write_data[31:0] son, respectivamente, la dirección accedida, ya sea para realizar una operación de lectura o escritura, y el dato que se almacena en el segundo caso.

La salida user_memory_bank0_read_data[31:0] contiene el dato leído de la RAM en la última operación de lectura. Debe tenerse en cuenta que cada operación de este tipo tiene una latencia de 3 ciclos de reloj.

2.5.2.4 Interfaz del usuario

La interfaz del usuario se encarga de recibir del exterior las órdenes, datos y direcciones con las que genera las señales adecuadas para controlar la interfaz de la RAM. El esquemático de ésta se muestra en la figura 2.8.

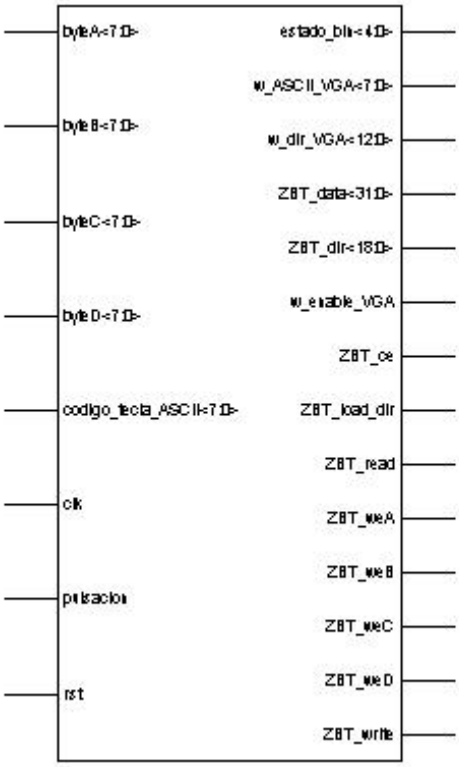


Figura 2.8

Las señales de entrada y salida del módulo se muestran en la tabla 2.2.

Nombre de la señal	Dirección	Descripción
byteA[7:0]	Entrada	8 bits de datos (menor significatividad)
byteB[7:0]	Entrada	8 bits de datos
byteC[7:0]	Entrada	8 bits de datos
byteD[7:0]	Entrada	8 bits de datos (mayor significatividad)
codigo_tecla_ASCII	Entrada	Código de la tecla pulsada
clk	Entrada	Señal de reloj
pulsacion	Entrada	Señal activa a alta que indica que se ha producido una pulsación
rst	Entrada	Señal de reset
w_ASCII_VGA	Salida	Carácter para escribir en la VGA
w_dir_VGA	Salida	Posición de escritura de carácter en la VGA
zbt_data	Salida	Dato que se escribe en la RAM
zbt_dir	Salida	Dirección de lectura/escritura en la RAM
w_enable_VGA	Salida	Señal activa a alta que capacita la escritura en la VGA
zbt_ce	Salida	Señal activa a alta que capacita el chip de la RAM
zbt_load_dir	Salida	Señal activa a alta que indica que se va a leer un bloque de direcciones consecutivas
zbt_read	Salida	Señal activa a alta que indica operación de lectura sobre la RAM
zbt_wea	Salida	Señal activa a alta que capacita la escritura de los bits [7:0] del dato
zbt_web	Salida	Señal activa a alta que capacita la escritura de los bits [15:8] del dato
zbt_wec	Salida	Señal activa a alta que capacita la escritura de los bits

		[23:16] del dato
zbt_wed	Salida	Señal activa a alta que capacita la escritura de los bits [31:24] del dato
zbt_write	Salida	Señal activa a alta que capacita la operación de escritura sobre la RAM

Tabla 2.2

Descripción de las señales

Por las 4 entradas de 8 bits byteA, byteB, byteC y byteD se reciben los 32 bits de datos que se escribirán en la dirección pertinente de la memoria RAM.

ZBT_ce es la salida del gestor de memoria que capacita el funcionamiento de la memoria RAM.

ZBT_weX, para X A, B, C o D, son las 4 salidas del gestor de memoria que permiten la escritura en la memoria RAM. Será necesario que se encuentren activas para que la escritura sea efectiva sobre el grupo de bits que representan.

Las salidas que funcionarán como señales de capacitación se consideran activas a alta.

Cuando ZBT_write y ZBT_read se activan indicarán que se efectúan las operaciones de escritura y lectura respectivamente.

ZBT_load_dir activa indica que la lectura se efectuará sobre un bloque de direcciones de la RAM consecutivas.

Finalmente, ZBT_dir[18:0] y ZBT_data[31:0] son, respectivamente, las líneas de salida de direcciones y de datos que se envían a la RAM.

2.5.2.5 Implementación

Ruta de datos

La ruta de datos del controlador del interfaz de memoria para las ZBT RAMs contiene los siguientes elementos:

- Shift_dir: se trata de un registro de desplazamiento en el que guardamos en binario la dirección introducida por teclado de la que queremos leer o escribir;

- Shift_dato: se trata de un registro de desplazamiento en el que guardamos en binario los datos introducidos por teclado que queremos escribir en la memoria;

- DatoA: nos es más que un módulo que contiene cinco registros denominados Reg_byteA, Reg_byteB, Reg_byteC, Reg_byteD donde guardamos los datos leídos de memoria correspondientes a los bytes ByteA, ByteB, ByteC y ByteD respectivamente que nos envía el interfaz;

- Contador_ciclos_escritura: contador que utilizamos para contar los ciclos de espera en la operación de escritura;

- Conversor: es un módulo que convierte cuatro entradas de ocho bits en una salida en código ASCII de los cuatro bits más significativos o menos significativos en función del valor de la entrada de selección;

- Cont_bytes_leidos: contador que utilizamos para indicarnos cuantas tuplas de cuatro bits hemos mostrado por la pantalla. Es la entrada de selección del conversor;

- Contador_dato: contador que se usa para saber el número de bits que han sido introducidos por teclado al introducir una dirección o un dato;

- Contador: contador usado para avanzar dirección de lectura de la ZBT RAM en las operaciones de lectura de posiciones consecutivas de memoria;

- Contador_caracteres: contador que utilizamos para realizar los saltos de línea de la pantalla para que los datos mostrados e introducidos sean más comprensibles a primera vista;

- Contador_direcciones_vga: este contador lleva la cuenta de la dirección de la VGA donde se deben escribir los caracteres que se quieren mostrar por pantalla.

En la figura 2.9 se muestra la ruta de datos descrita.

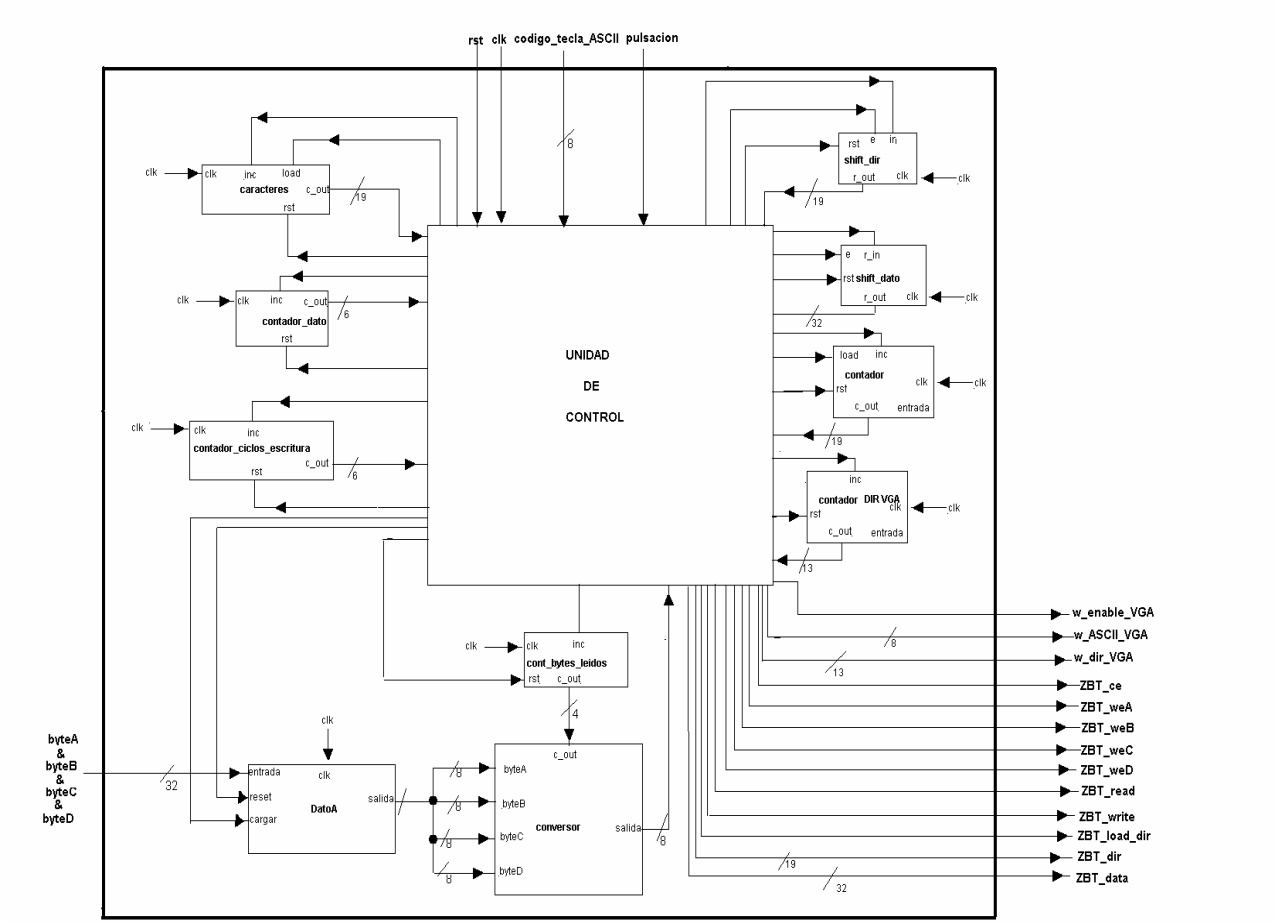


Figura 2.9

Todas las señales de control que se encargan de la transferencia de datos de todos y cada uno de los módulos, las genera una unidad de control que se describe a continuación.

Controlador

Inicialmente, el módulo se encuentra a la espera de recibir una orden a través de la entrada codigo_tecla_ASCII. Las únicas posibles órdenes que pueden ser enviadas se muestran codificadas en la tabla 2.3. En el ciclo de reloj siguiente a la recepción de la orden ya estará preparado para recibir el primer bit de la dirección de la RAM que debe leer, en caso de tratarse de una operación de lectura, o del dato que debe escribir y la dirección en la que debe hacerlo, en caso de que la operación fuera de escritura. Los bits de las direcciones y los datos son introducidos al módulo interfaz de uno en uno, cargándose en un registro de desplazamiento de 32 bits en el caso de los datos, o de 19 bits en el caso de las direcciones. Cuando la orden recibida haya sido la de lectura de un bloque de direcciones consecutivas las dos siguientes entradas esperadas serán, primero, la dirección anterior a la primera que queremos que sea leída, y después, la última dirección que se leerá. De la misma manera, la interfaz espera estos dos valores bit a bit, uno a continuación del otro como si se trataran de un solo dato.

Entrada	Correspondencia ASCII	Operación
"01001100"	L	Lectura
"01010011"	S	Escritura
"01001101"	M	Lectura de un bloque de direcciones consecutivas

Tabla 2.3

La generación de las señales de salida que controlan la interfaz del banco 0 de la RAM se produce de acuerdo con la máquina de estados finita que explicaremos a continuación.

Inicialmente el estado es I0, a la espera de recibir por la entrada codigo_tecla_ASCII alguna de las órdenes válidas (tabla 3). En este estado inicial se activa la señal reset de los registros y contadores, se desactivan las señales de capacitación de lectura y escritura, carga de los registros y cuenta de los contadores. También se inicializan la dirección de escritura en la VGA, en la que se irán mostrando las órdenes del usuario y su resultado, y las demás entradas de la interfaz.

Una vez se han inicializado los valores de todas las señales se realiza un salto incondicional al estado incrementa_dir_vga. Se capacita la señal de carga del contador de direcciones de escritura de la VGA. Después de este paso se realiza un nuevo salto incondicional al estado espera_orden.

La siguiente transición dependerá de la orden que haya sido introducida. Si ha sido de lectura, hacia L1. Si ha sido de escritura, S1. Si ha sido de lectura de bloque de direcciones, M1. En cualquier otro caso se permanece en este estado, a la espera de que se introduzca una orden correcta.

En los estados L1, S1 y M1 únicamente se muestra por pantalla la orden elegida por el usuario. Después se realiza una transición incondicional a los estados Laux0o1, Saux0o1_dir y Maux0o1_dir1 respectivamente.

En el estado Laux0o1 se espera un bit de la dirección de lectura. Cuando se ha recibido, se produce una transición al estado L2, en el que se carga en un registro de desplazamiento y además se escribe en la VGA el bit introducido. Se produce una transición de nuevo hacia el estado Laux0o1, y se repite el proceso hasta que los 19 bits de la dirección han sido introducidos. Se dispone de un contador módulo 19 para determinar si ya han sido introducidos todos los bits. Cuando es así, se produce una transición hacia el estado L3, donde se activa la señal de reset del contador de bits y se prepara el registro de datos para recibir el dato de la RAM, activando su señal de carga. Se produce una transición incondicional al estado L4, en el que el dato ha comenzado ya a leerse.

Hay que tener en cuenta que la operación de lectura tiene una latencia de 4 ciclos de reloj, y que por lo tanto ocurren tres transiciones más a estados en los que ninguna acción va a realizarse. Estos estados de inactividad son L4, L5, L6 y L7. Es en el estado L8 cuando se carga el registro destinado a los datos procedentes de la RAM con el último dato leído. Se produce otra transición al estado L9, en el que se realiza otra lectura del mismo dato, y posterior e incondicionalmente otra transición al estado incrementa_dir_VGA, para volver a empezar con otra orden.

En el estado Saux0o_dir se espera un bit de la dirección de escritura. Análogamente a como ocurría en la lectura, un contador módulo 19 determina cuándo se detiene la iteración de transiciones entre este estado y S2, en el que cada bit entra en un registro de desplazamiento y además es escrito en la VGA.

Cuando se han terminado de introducir los bits de la dirección se produce una transición al estado S3, en el que se activa la señal de reset del contador de bits. Ya está preparado para recibir el dato que va a escribirse en memoria, de forma idéntica a como recibe la dirección: bit a bit. Hay una transición al estado Saux0o1_data. Mientras la salida del contador módulo 32 indique que no se han introducido todos los bits se producen transiciones entre este estado y S4, en el que se carga cada bit del dato en un registro de desplazamiento y además se escribe en la VGA. Cuando se han terminado de introducir todos los bits hay una transición al estado S5. La siguiente transición ya es al estado incrementa_dir_VGA, a la espera de una nueva orden.

En el estado Maux0o1_dir1 se procede de la misma forma que en los anteriores casos. Esta vez, en lugar de una dirección y un dato, se esperan dos direcciones. Las direcciones se reciben en los estados Maux0o1_dir1, M3, Maux0o1_dir2, y M4. Se produce una transición al estado M5, que será el estado al que se vuelve cada vez que se lee una nueva dirección. Las direcciones que quedan por ser leídas se van calculando mediante la salida de un contador.

Así, desde M5 se produce una transición a M6 (primer ciclo de latencia), de éste a M7, posteriormente a M8 y, por último, a M9. En el estado M10 ya se dispone del último dato leído, que se muestra en la VGA. Si no se han visitado todas las direcciones del bloque se realiza una nueva transición a M5. Por el contrario, si ha terminado de leer el bloque de direcciones, vuelve a producirse una transición al estado incrementa_dir_VGA, en el que se espera una nueva orden para ser ejecutada.

2.5.2.6 Interconexión

En la figura 2.10 se muestra el flujo de señales entre la interfaz del banco 0 de la RAM y la interfaz del usuario que hemos explicado en los apartados anteriores.

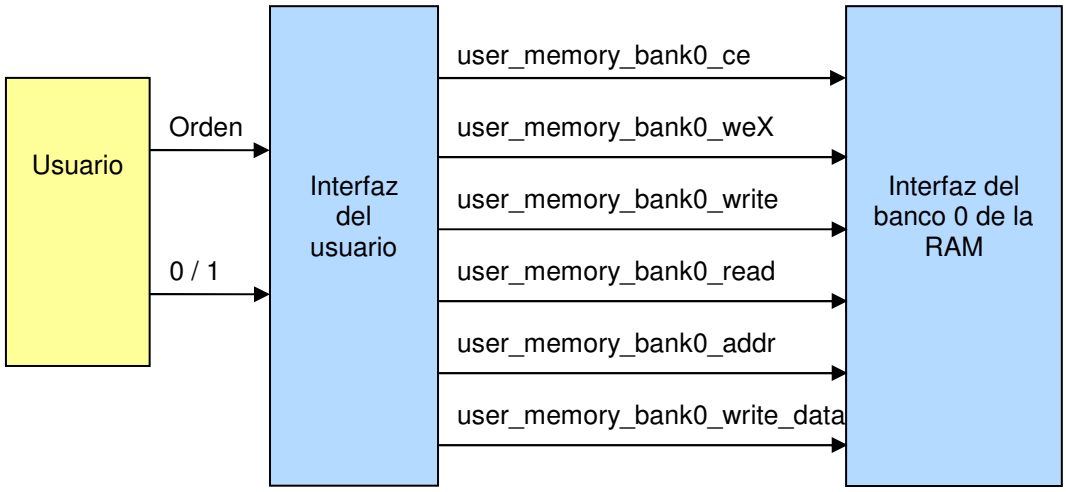


Figura 2.10

2.5.2.7 Funcionamiento

En este apartado vamos a mostrar una simulación del comportamiento de este módulo para una operación de escritura. En la figura 2.11 vemos la primera parte, que pasamos a comentar a continuación.

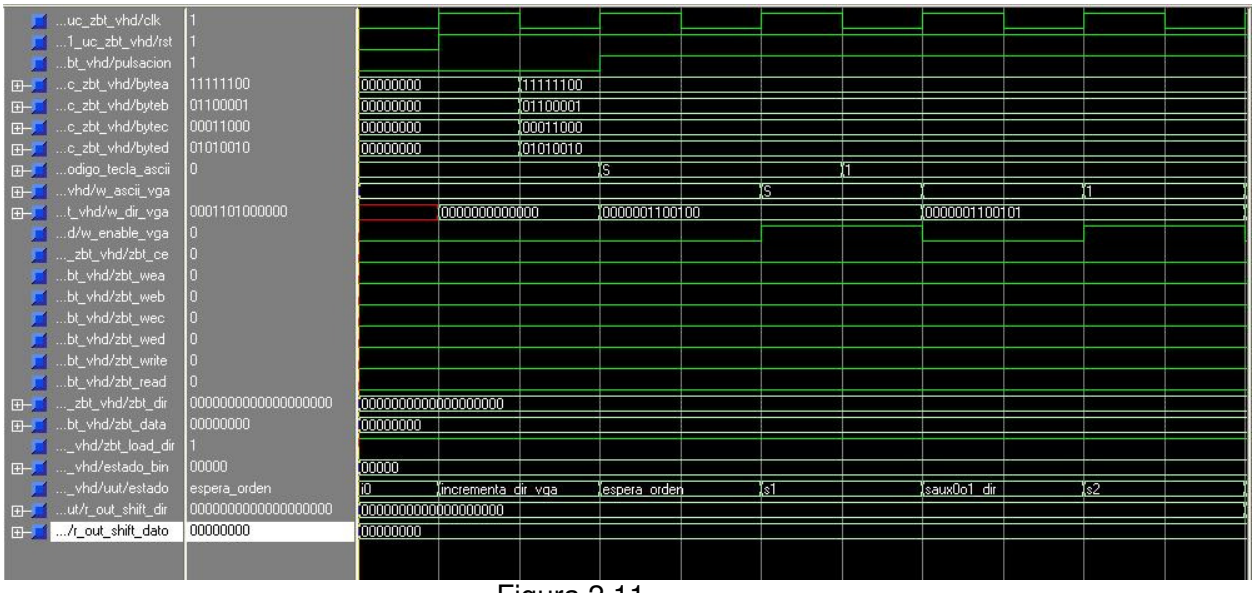


Figura 2.11

Antes de nada, debemos aclarar que para probar este interfaz con la RAM de la placa que utilizamos tuvimos que conectarlo a los módulos de teclado y de VGA. A través del primero introducimos las órdenes y los datos o direcciones, según estemos realizando una operación de lectura o escritura, y por medio del segundo podemos ver en la VGA los datos que nos retorna la RAM tras cada operación.

La señal de entrada código_tecla_ASCII es la que contiene la tecla que se ha pulsado, y pulsacion se activa (a alta) cada vez que se produce el evento de pulsar una tecla. w_dir_vga representa la dirección en la que se escribe cada caracter en la pantalla, y w_enable_vga se activa (a alta) para indicar la habilitación de la escritura.

Vemos en la imagen que inicialmente el estado es I0. Se incrementa la dirección en la que se escribirán los caracteres en la VGA (estado incrementa_dir_vga), y posteriormente se produce una transición al estado espera_orden, en el que se permanece hasta que se reciba una pulsación de las teclas 'S' (escritura), 'L' (lectura), o 'M' (lectura de un bloque de direcciones consecutivas). Se ha forzado la primera pulsación para que se reconozca como el caracter 'S'. Éste se muestra en la VGA (a partir de ahora vamos a asumir que después de cada escritura la dirección de la VGA se incrementa, aunque no se diga explícitamente), mientras los bits de la dirección en la que va a escribirse el dato ya van introduciéndose en serie por la entrada código_tecla_ASCII. En la figura 2.12 se aprecia cómo según van introduciéndose estos bits van almacenándose en el registro de desplazamiento de direcciones (salida r_out_shift_dir).

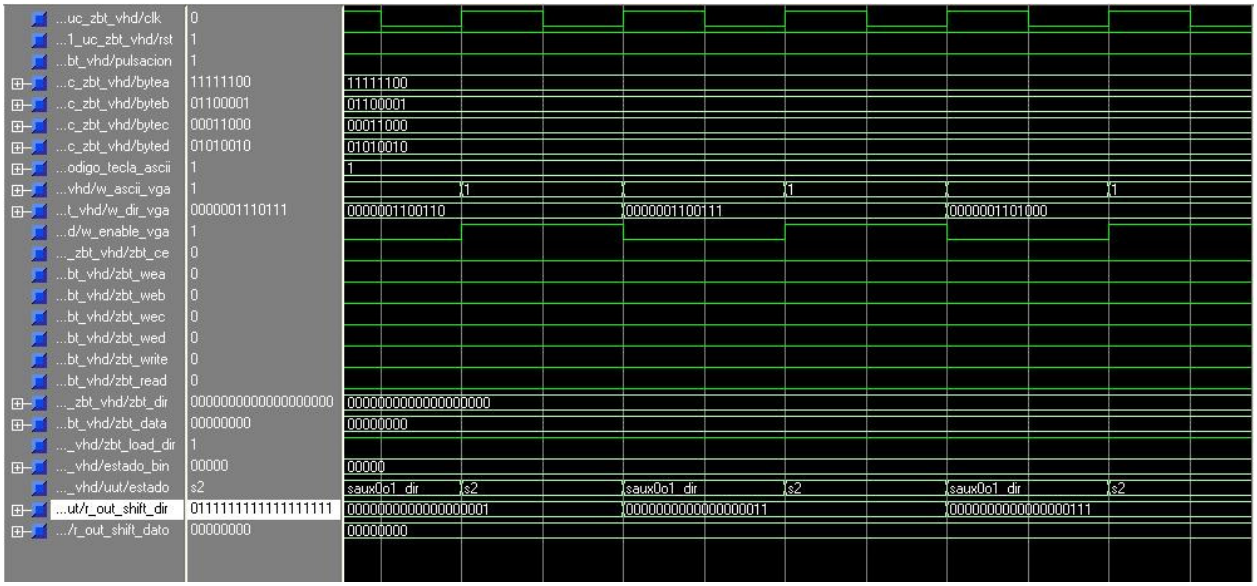


Figura 2.12

En la imagen 2.13 se ve el momento de la simulación en que han terminado de introducirse los bits de la dirección, y se comienza con los 32 bits del dato, de la misma forma que se hizo antes.

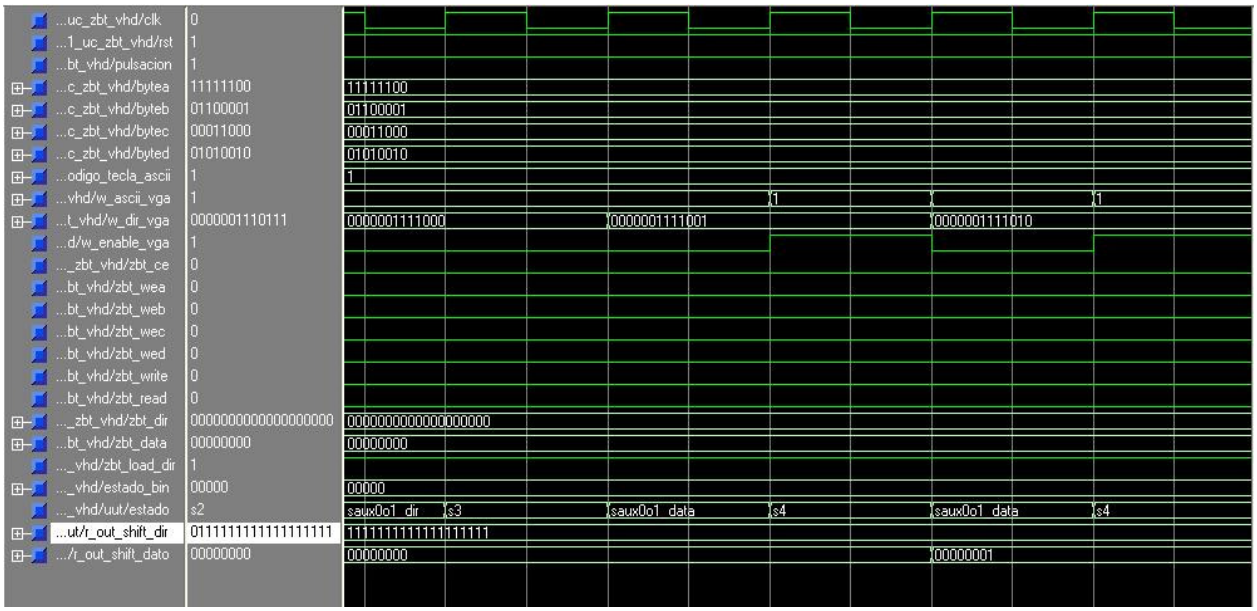


Figura 2.13

Cuando terminan de introducirse estos bits se produce una transición al estado S5, en el que estos datos ya son los que toma el interfaz de la RAM para realizar la operación de escritura del dato que hemos introducido.

Para realizar una lectura se sigue un procedimiento análogo al explicado para la escritura, salvo que en este caso no se introducirán los 32 bits del dato, sino únicamente los 19 de la dirección que quiere leerse.

Finalmente, en el caso de la lectura de un bloque, se procede igual que en la lectura, simplemente iteramos este proceso utilizando la salida de un contador ascendente.

2.5.3 Gestor de memoria para la SDRAM de la FPGA Virtex IIPro

2.5.3.1 Introducción

Debido a que la placa de la Virtex II no dispone de las conexiones que necesitábamos para la reconfiguración dinámica de la FPGA, debimos cambiar a la placa de la Virtex II Pro. Ésta dispone de un módulo de memoria DDR SDRAM en vez de las ZBT RAM que utilizábamos en el anterior dispositivo. De modo que hemos tenido que realizar un nuevo gestor de memoria.

Las memorias DDR son memorias de doble tasa de transferencia de datos. Esto significa que permiten la transferencia de datos en los dos flancos de cada ciclo de reloj, es decir, se aprovechan también los flancos de bajada del reloj. Con esto se consigue enviar dos datos por cada ciclo de reloj, doblando así su frecuencia de funcionamiento y ancho de banda.

En lugar de señales, en las memorias DDR se utilizan comandos. Cada comando está compuesto por varias señales: CS, CAS, RAS, WE. Por ejemplo, el comando READ de lectura se activa poniendo a baja las señales CS y CAS y a alta las demás. Los comandos se activan en el flanco de subida del reloj.

En las DDR síncronas (SDRAM) hay registros de modo programables mediante los cuales podemos seleccionar la latencia CAS (ajustando así la velocidad de nuestra memoria), los tipos y las longitudes de ráfaga. La latencia CAS es el número de ciclos que transcurre entre la orden de lectura (comando READ) y los datos disponibles en DQs. Suele ser de 2 o 3 ciclos.

Poseen varios bancos de memoria, de manera que podemos abrir varias filas en paralelo para hacer posible un mayor número de accesos en modo ráfaga (diversas longitudes de ráfaga secuencial). Antes de utilizar el comando para seleccionar el banco y la fila o los comandos READ o WRITE el banco debe estar cerrado y precargado con el comando PRECHARGE (también AUTOPRECHARGE).

Estas memorias funcionan entre los 66MHz y los 133MHz, con un reloj diferencial, CLK y CLK_Z, considerándose el flanco positivo de reloj el momento en el que CLK sube a alta y CLK_Z pasa a baja.

Podríamos ahondar mucho más en estas memorias y sus ventajas sobre la RAM que utilizamos en la otra placa, pero para no extendernos excesivamente facilitamos más información sobre esta clase de memorias en el siguiente link.

Para facilitarnos la labor de la implementación del gestor de memoria, hemos utilizado un herramienta que genera automáticamente interfaces de memoria para este tipo de módulos, el Mig 007.

A continuación se describe las características principales del interfaz de memoria generado por esta herramienta.

2.5.3.2 Interfaz generado por el MIG007

Para comenzar, hemos de decir las opciones que hemos elegido en el Mig007 para la generación de nuestro interfaz:

- En el campo Memory del Mig007 hemos selecciona el tipo DIMS X8;
- Nuestra FPGA es la correspondiente a la configuración: Virtex2P 2vp30ff896 -7;
- Una frecuencia de funcionamiento de 100 Mhz;
- En el campo Bank Selection hemos elegido los bancos 6 y 7 dado que estos son los bancos a los que está conectada la DDR y la FPGA en nuestra placa.

En la figura 2.14 se muestra de nuevo el interfaz del MIG 007 en el que hemos introducido los datos que hemos indicado arriba:

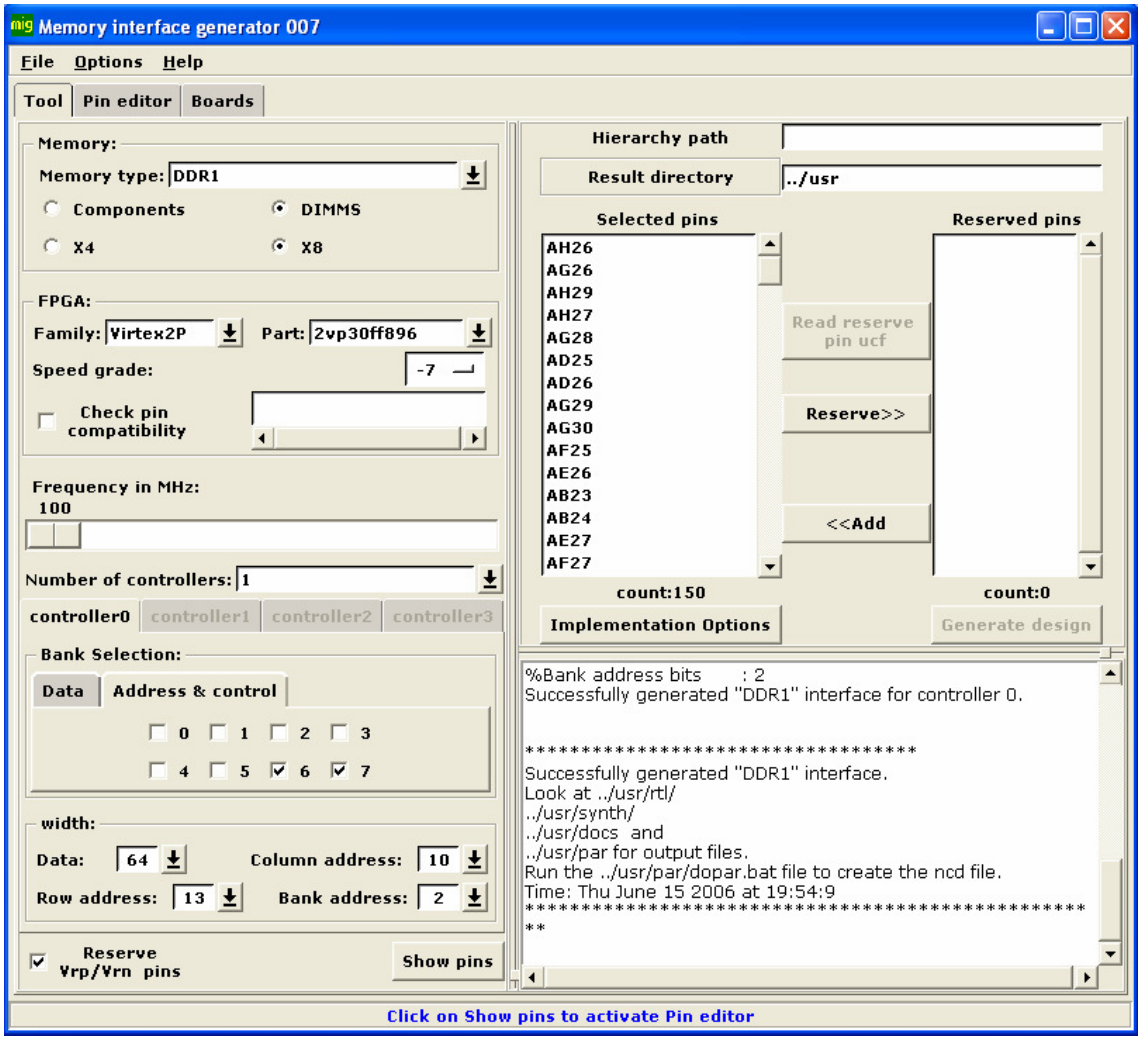


Figura 2.14: Interfaz del MIG

Tras haber elegido esta configuración, se nos generaron una serie de archivos .VHD que implementaban un interfaz con las siguientes entradas y salidas (se muestran en la tabla 2.4).

Señal	Dirección	Descripción
Dip1	Entrada	Sin uso.
Dip3	Entrada	Sin uso.
Sys_CLK y Sys_CLK_b	Entrada	Éstas son las señales del reloj diferencial
Reset_in	Entrada	Es la señal de reset del interfaz. Es activa a baja.
User_input_data[127:0]	Entrada	Estos los datos que se desean escribir en la DDR SDRAM.
User_data_mask[22:0]	Entrada	Es la máscara para la escritura de datos en la DDR SDRAM.
User_bank_address[1:0]	Entrada	Para seleccionar el banco de la DDR SDRAM de donde se quiere leer o escribir.
User_config_reg[9:0]	Entrada	Son los datos con los que se configura la memoria en la inicialización. Lleva información sobre la latencia y la longitud del burst.

User_command_reg[2:0]	Entrada	Entrada de las operaciones que se desean realizar con la DDR. Inicialización, lectura, escritura, NOP.
Burst_done	Entrada	Ésta es la señal que indica que la transferencia de datos ha terminado por parte del usuario.
User_output_data[127:0]	Salida	Son los datos leidos de memoria.
User_data_valid	Salida	Cuando esta señal está a alta los datos contenidos en el bus User_output_data son válidos.
User_cmd_ack	Salida	Esta señal se pone a alta cuando el interfaz reconoce como válido un comando que está en User_command_reg, y permanece a alta hasta que termina la operación que indica este bus.
Init_val	Salida	El interfaz pone a alta esta señal para indicar que la inicialiazación de la DDR SDRAM ha sido realizada.
Ar_done	Salida	Cuando esta señal se pone a alta indica que la memoria DDR SDRAM ha realizado un autorefresco.
Auto_ref_req	Salida	Cuando esta señal se pone alta indica que la memoria entra en un período de autorefresco.
sys_rst_1, sys_rst90_1, sys_rst180_1, sys_rst270_1	Salida	Señales que indican que el interfaz está en el estado de reset durante 200 us.

Tabla 2.4

A continuación se describe el funcionamiento de este interfaz.

2.5.3.2.1 Inicialización de la memoria

Antes de que el interfaz permita el uso de los comandos de lectura y escritura, éste debe inicializar la memoria a través del comando de inicialización. El usuario puede ejecutar este comando después de que el interfaz ponga a baja todas las señales de reset (después de 200 us.).

Tras estos 200 us. el usuario debe poner en el bus user_config_reg los datos válidos para la configuración de la DDR dos ciclos antes de poner en el bus user_command_reg el comando de inicialización. El comando de inicialización debe ponerse en el flanco de subida del reloj clk180 durante un ciclo de reloj, que hace que comience la secuencia de inicialización de la memoria.

La secuencia de inicialización se finaliza cuando el interfaz pone a alta la señal init_val en un flanco de subida del reloj clk180. Una vez que sucede esto se puede mandar al interfaz la ejecución de un comando de lectura o escritura. En la figura 2.15 mostramos el cronograma correspondiente a la inicialización.

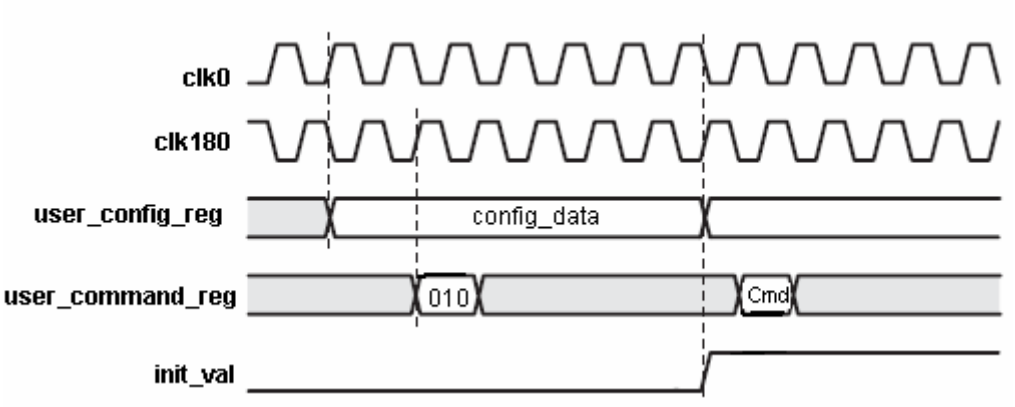


Figura 2.15

2.5.3.2.2 Escritura en memoria

La secuencia de escritura de datos en la memoria DDR se inicia poniendo en el bus user_command_register el comando de escritura y la primera dirección de escritura en el bus user_input_address en un flanco de subida del reloj clk180. El interfaz reconoce el comando poniendo a alta la señal user_cmd_ack en un flanco de subida del reloj clk180. La primera dirección debe estar estable 4.5 ciclos de reloj y los datos a escribir en memoria deben estar disponibles en el bus user_input_data en el primer flanco de subida del reloj clk90 después de que la señal user_cmd_ack se ponga a alta. La segunda dirección debe estar disponible después de los 4.5 ciclos mencionados anteriormente y los siguientes datos a escribir deben estar disponibles en el primer flanco de subida del clk90 después de poner la segunda dirección durante dos ciclos de reloj. La señal que indica que la escritura ha terminado es burst_done, que debe ponerse a alta después de haber mandado los últimos datos. Un tiempo después, el interfaz termina la secuencia de escritura poniendo a baja el user_cmd_ack, permitiendo así la ejecución de nuevos comandos. El cronograma correspondiente a la operación de escritura se muestra en la figura 2.16.

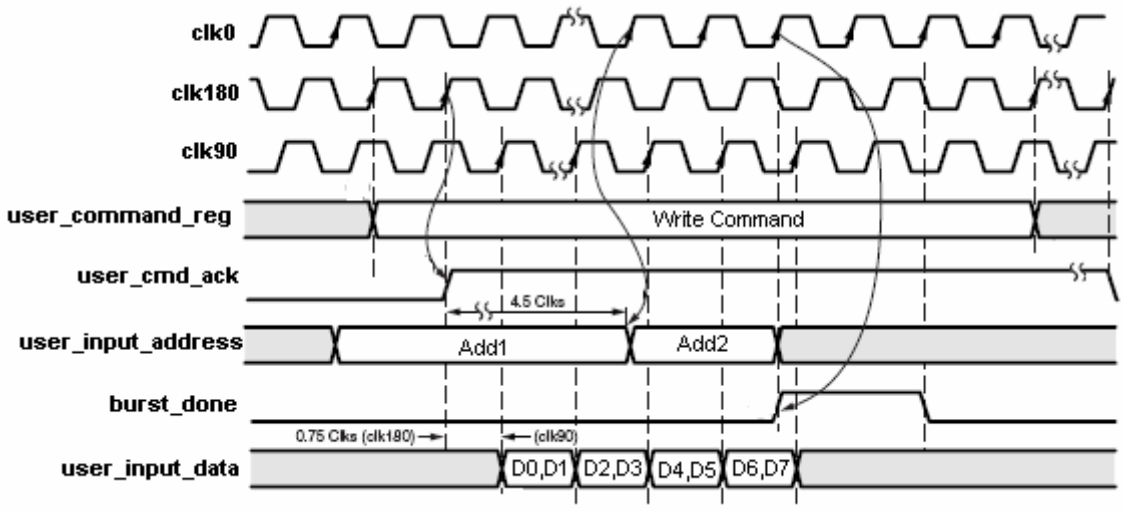


Figura 2.16

2.5.3.2.3 Lectura de memoria

La secuencia de lectura de memoria se inicia poniendo el comando de lectura y la primera dirección que se quiere leer en un flanco de subida del reloj clk180 en los buses user_command_reg y user_input_address respectivamente. Esta primera dirección debe estar durante 4.5 ciclos de reloj antes de poner la siguiente dirección, que debe estar estable durante dos ciclos de reloj. Después de estos dos ciclos, debe ponerse a alta la señal de burst_done durante dos ciclos de reloj coincidiendo con el primer flanco de subida del reloj clk0. Los datos leídos están disponibles cuando la señal user_data_valid se pone a alta, coincidiendo con un flanco de subida del reloj clk90. Durante los siguiente 4 ciclos del reloj clk90 se pueden leer los datos de las dos direcciones deseadas. La secuencia de lectura termina cuando el interfaz pone a baja la señal user_cmd_ack. El cronograma de esta operación puede verse en la figura 2.17.

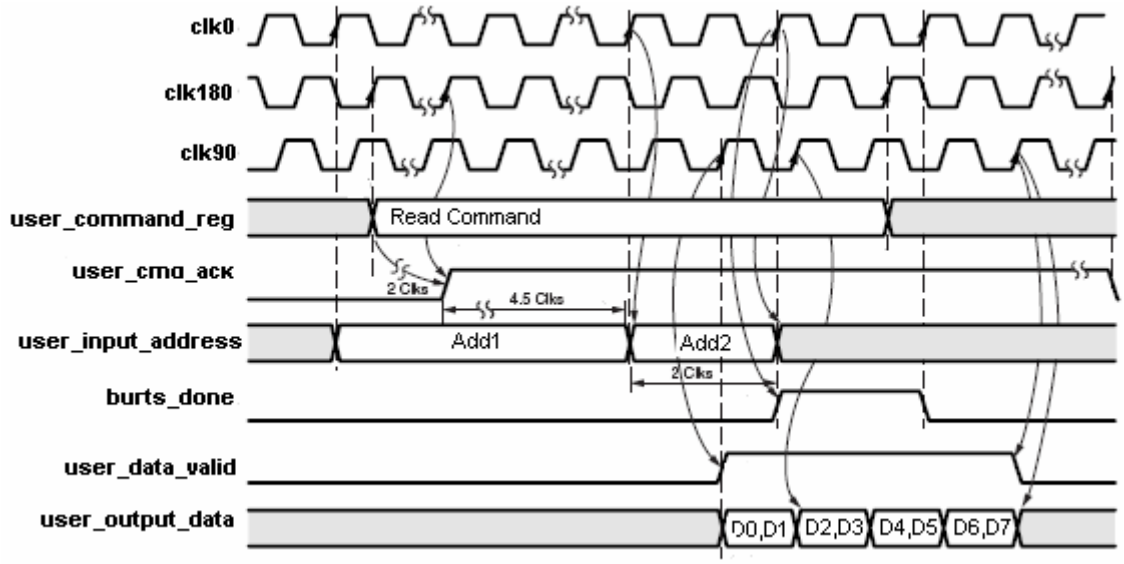


Figura 2.17

2.5.3.3 Unidad de Control del Interfaz

2.5.3.3.1 Introducción

Teniendo en cuenta el funcionamiento del interfaz descrito anteriormente, hemos implementado un módulo para probar el correcto funcionamiento de éste. Este módulo dispone de las siguientes opciones:

Si se pulsa una 'L' en el teclado, se intenta leer una dirección de la memoria que se introduce por teclado;

Si se pulsa una 'S', se almacena en una dirección que se introduce por teclado un dato previamente definido en nuestro módulo por comodidad a la hora de realizar las pruebas.

En la figura 2.18 se muestra el módulo que representa el módulo de la Unidad de Control del interfaz generado por el Mig007.

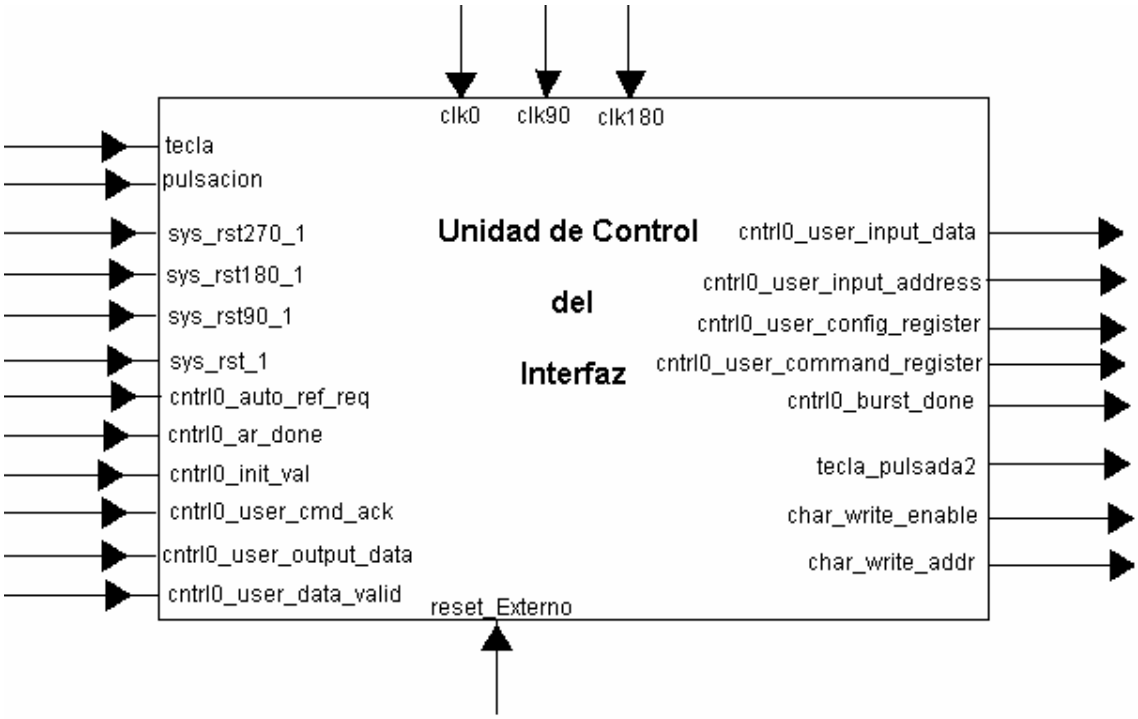


Figura 2.18

Las señales de entrada de este módulo con las siguientes:

- clk0: reloj del sistema a 100 Mhz;
- clk90: es el mismo reloj que el clk0 pero desplazado en fase 90 grados;
- clk180: es el mismo reloj que el clk0 pero desplazado en fase 180 grados;
- tecla: código ASCII de la tecla pulsada en el teclado;
- pulsacion: indica si se ha producido una pulsación de tecla;
- sys_rst_270_1, sys_rst_180_1, sys_rst_90_1,sys_rst_1: señales de reset que envía el interfaz antes de inicializar la memoria;
- cntrl0_auto_ref_req: señal que envía el interfaz indicando que se va a autorefrescar la memoria SDRAM;
- cntrl0_ar_done: señal que envía el interfaz indicando que el autorefresco ha finalizado;
- cntrl0_init_val: señal que envía el interfaz cuyo significado es que la inicialización se ha realizado correctamente;
- cntrl0_user_cmd_ack: señal que indica el comando enviado al interfaz es válido;
- cntrl0_user_output_data: los datos que envía el interfaz correspondientes a los datos leídos de la memoria SDRAM;

- cntrl0_user_data_valid: señal que envía el interfaz indicando que los datos que se encuentran en cntrl0_user_output_data son válidos;
- reset_Externo: señal de reset de la unidad de control;

Las señales de salida de este módulo con las siguientes:

- cntrl0_user_input_data: datos que se envían al interfaz para que los escriba en memoria SDRAM;
- cntrl0_user_input_address: dirección de la SDRAM de la que se quiere leer o en la que se quiere escribir;
- cntrl0_user_config_register: datos de configuración que se envían al interfaz para que los utilice en la inicialización;
- cntrl0_user_command_register: comando que debe ejecutar el interfaz;
- cntrl0_burst_done: señal de finalización de lectura o escritura;
- tecla_pulsada2: carácter en código ASCII que se quiere escribir en la pantalla;
- char_write_enable: indica si se quiere escribir o no en la pantalla;
- char_write_addr: dirección de la VGA donde se desea escribir un carácter.

A continuación se describe con más detalle este módulo.

2.5.3.3.2 Ruta de Datos

En la figura 2.19 se muestra la ruta de datos de la unidad de control que nos sirve para comprobar el funcionamiento del interfaz. El contador Direcciones VGA lleva la cuenta de las direcciones de la pantalla donde se van escribiendo los caracteres. Los registros registro_DatosIN_A y registro_DatosIN_B tienen la labor de guardar los dos datos que se desean escribir en la memoria cuando se ejecute el comando de escritura. El registro shift_dir es un registro con desplazamiento que guarda la dirección de donde se quiere leer o escribir en la memoria SDRAM. Los módulos DatosA y DatoB no son más que la representación de 16 refistros cada uno donde se guardan los datos leídos de la memoria. En los registros de DatosA se guardan los 16 bytes de la primera dirección que se lee y en DatosB los de la siguiente dirección. Es importante el hecho de que estos registros se cargan con el flanco de subida del reloj clk90, ya que es con este reloj con el que los datos leídos de memoria a través del interfaz están disponibles. La salida de estos registros van a unos conversores de binario a código ASCII, que nos sirven para mostrar por pantalla los datos leídos. Hay que tener en cuenta, que como estos registros almacenan 8 bits, estos conversores realizan su labor de 4 en 4 bits. Los cuatro bits que se convierten se eligen en función del valor del contador cont_bytes_leidos que lleva la cuenta de los bytes mostrados por pantalla. La salida de los dos conversores se selecciona a través del multiplexor Mux_Datos_VGA.

Todas las señales que gobiernan estos elementos se generan con tres máquinas de estados implementadas en los módulos Controlador CLK0, Controlador CLK90 y Controlador CLK180 que usan como relojes respectivamente el reloj clk0, clk90 y clk180.

Estos tres módulos interrelacionan a través de las señales habilita0, habilita90 y habilita180 para sincronizarse y lograr la escritura y la lectura de la memoria SDRAM usando los servicios del interfaz.

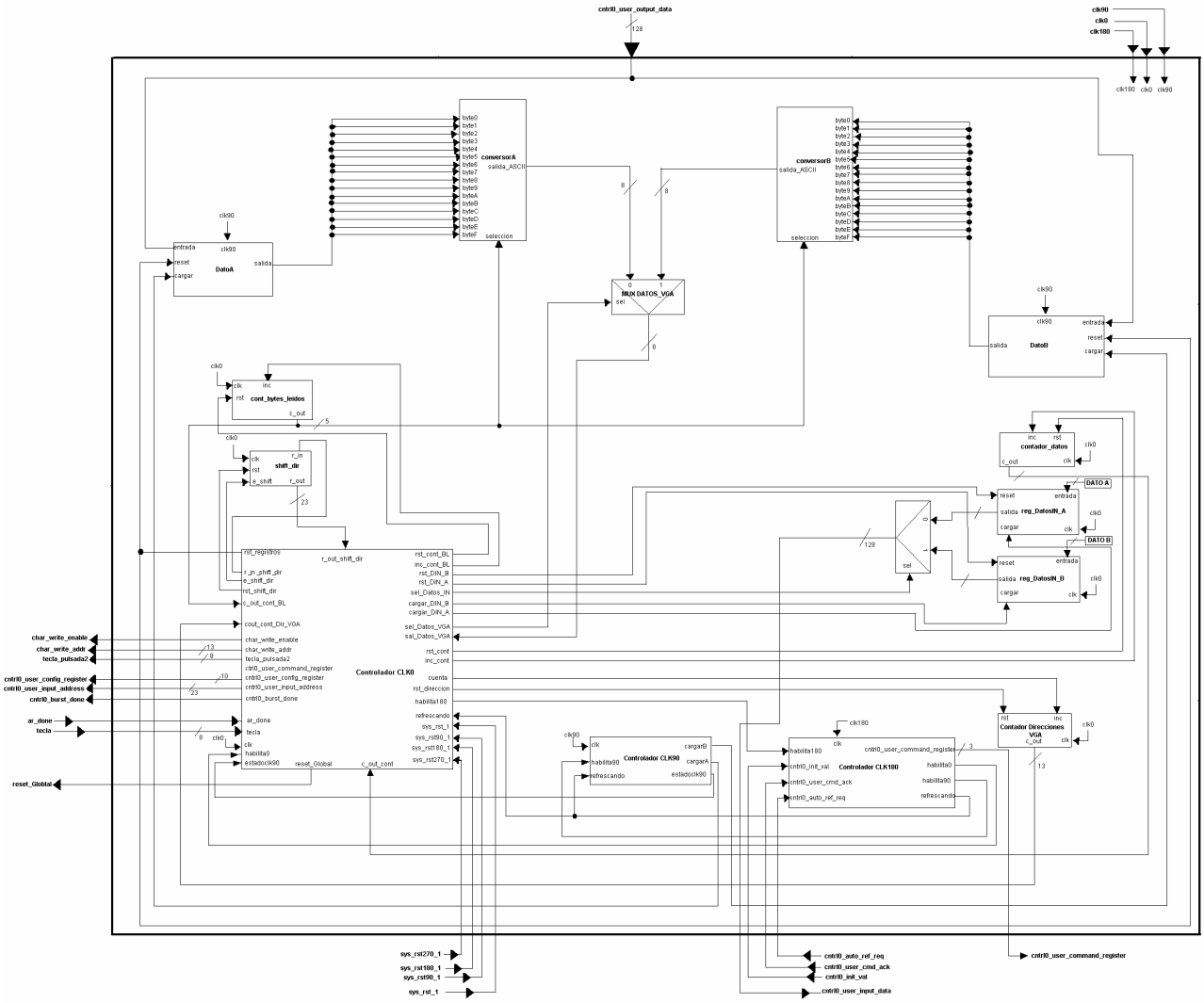


Figura 2.19

2.5.3.3.3 Controlador

A continuación se muestra el comportamiento de las tres máquinas de estados comentadas anteriormente interconectadas entre sí, con la ruta de datos.

2.5.3.3.4 Simulación

Vamos a mostrar en este apartado algunas simulaciones de la unidad de control de nuestro interfaz de memoria, para comprobar que funciona correctamente siempre que los datos de entrada sean los correctos. La primera prueba corresponde a la inicialización, y se muestra en la figura 2.20.

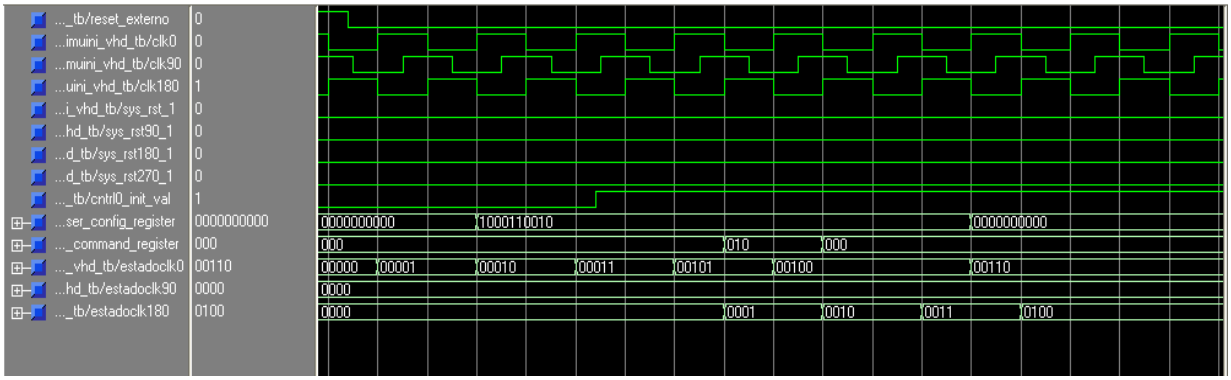


Figura 2.20

Inicialmente, la configuración en la señal user_config_register es “0000000000”. Colocamos en esta entrada la configuración “1000110010”, y dos ciclos del reloj clk180 después colocaremos el comando de inicialización en la entrada user_command_register.

Esto hace que comience la secuencia de inicialización. Cuando la configuración se ha completado, el controlador de la DDR SDRAM nos lo indica activando la señal init_val (que hemos forzado en nuestra simulación). Después de que el controlador haya activado esta señal, podremos colocar el siguiente comando en user_command_register cuando queramos.

A continuación mostramos cómo se realiza una escritura. El comienzo de esta simulación puede verse en la figura 2.21.

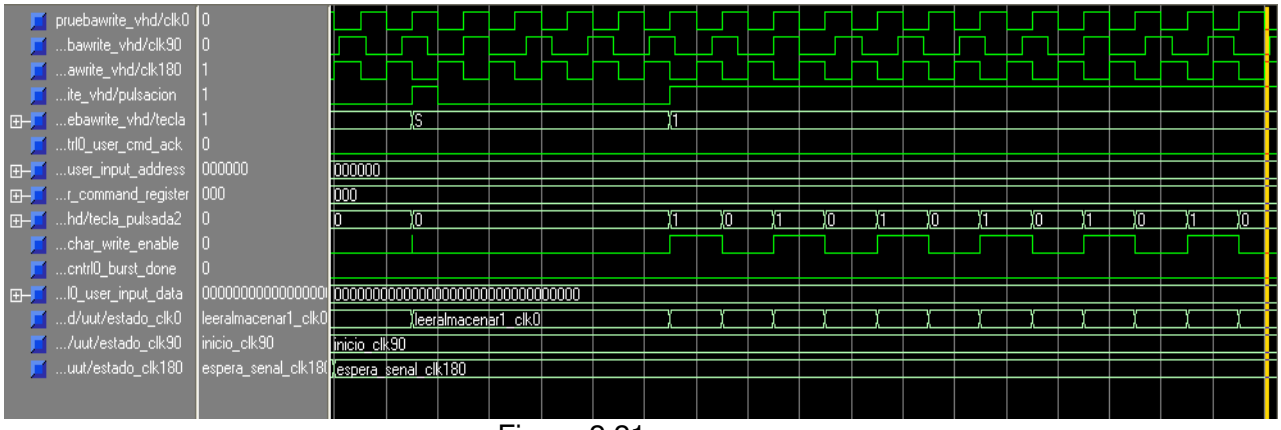


Figura 2.21

Cuando se detecta una pulsación de la tecla ‘S’ se interpreta una orden de escritura. A partir de ese momento, igual que ocurría con el antiguo módulo de memoria, el sistema está recibiendo en serie los bits de la dirección en la que se va a realizar la operación de escritura. Mientras se reciben estos bits únicamente se producen transiciones de estado en la máquina de estados correspondiente al reloj clk0. Las de los relojes clk90 y clk180 permanecen en sus estados hasta que terminen de recibirse los bits de la dirección. En este caso no vamos a introducir por teclado el dato para escribir, debido a su gran tamaño. Lo tomaremos directamente de la salida del registro en el que está siempre almacenado. En la figura 2.22 se observa como

tras la llegada de todos los bits de la dirección se coloca en la señal user_input_adress el valor de la dirección que acabamos de introducir y ha ido almacenándose en un registro de desplazamiento.

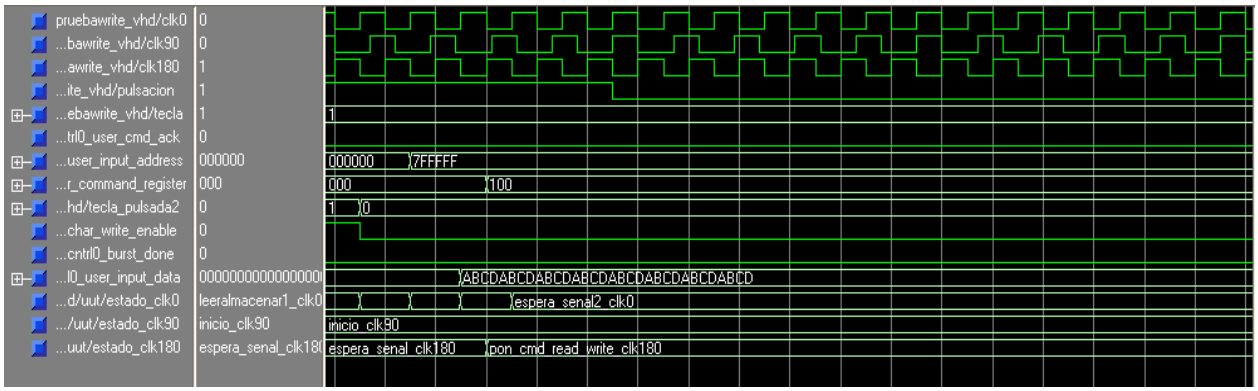


Figura 2.22

Para comenzar a realizar la operación de escritura hay que colocar en la señal user_command_register el comando correspondiente a la escritura, que es “100”. Esto debe hacerse en un flanco de subida del reloj clk180, por lo que la máquina de estados del reloj clk0 es la que se detiene ahora, enviando una señal que activa la del reloj clk180. De esta manera se indica al controlador de la DDR SDRAM la operación que queremos realizar.

En este momento el sistema se encuentra esperando la confirmación por parte del controlador de la DDR de que ha reconocido el comando de escritura. Esto ocurrirá cuando se active la entrada user_cmd_ack, en un flanco de subida del reloj clk180. Esta confirmación se ve ya en la figura 2.23.

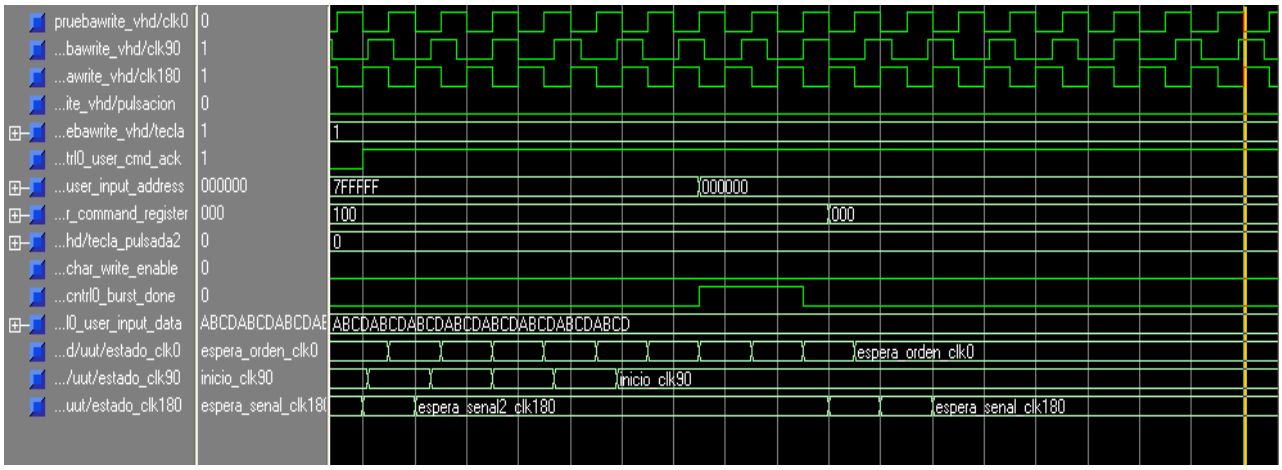


Figura 2.23

El siguiente paso es colocar la dirección de escritura, que como hemos dicho antes, ya había sido colocada en la señal user_input_address. Una vez que ha sido activada la confirmación del comando podemos, además, colocar en la línea de entrada de datos el primer dato, con el reloj clk90. La primera dirección debe permanecer durante 4,5 ciclos de reloj desde la activación de la señal user_cmd_ack. Las siguientes direcciones que se introduzcan (podríamos colocar otra distinta, en

nuestro caso hemos mantenido la misma) lo harán en flancos positivos y alternados del reloj clk0.

Finalmente, el controlador de la DDR activa la señal burst_done durante dos ciclos de reloj. Después de que se active esta señal, user_cmd_ack ya puede ser desactivada en cualquier momento. Después de la operación de escritura quitaremos el comando de la señal user_command_register, dejándolo de nuevo a “000”.

Por último mostraremos una simulación similar a la anterior, esta vez para ilustrar una operación de lectura. Ésta comienza en la imagen 2.24.

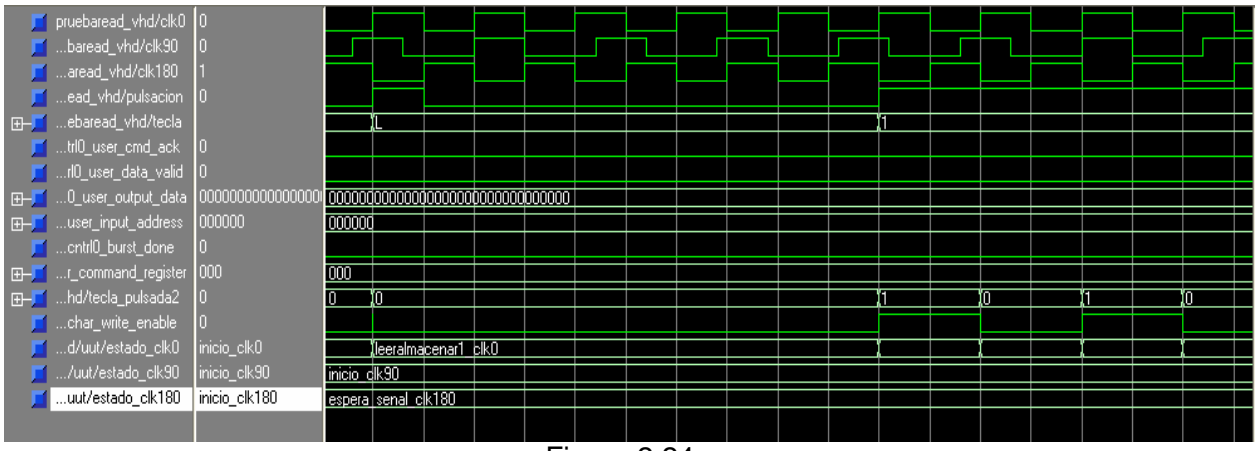


Figura 2.24

Una vez que se ha pulsado la tecla ‘L’ el sistema comienza una nueva lectura. igual que en el caso de la escritura, se irán introduciendo los bits de la dirección que queremos leer, uno a uno. Mientras las máquinas de estados de los relojes clk90 y clk180 están en un estado de espera, es la del reloj clk0 la que realiza las transiciones para ir recibiendo los bits.

A continuación, ya recibida la dirección completa, colocaremos en la entrada user_command_register el comando correspondiente a la operación de lectura, “110”. Esta señal se activa en un flanco de subida del reloj clk180, por lo que, al llegar, produce una transición en la máquina de estados del reloj clk0 a un estado de espera, y otra en la del reloj clk180, que pasa a un estado que recibe este valor del comando. Podemos ver esto en la imagen 2.25.

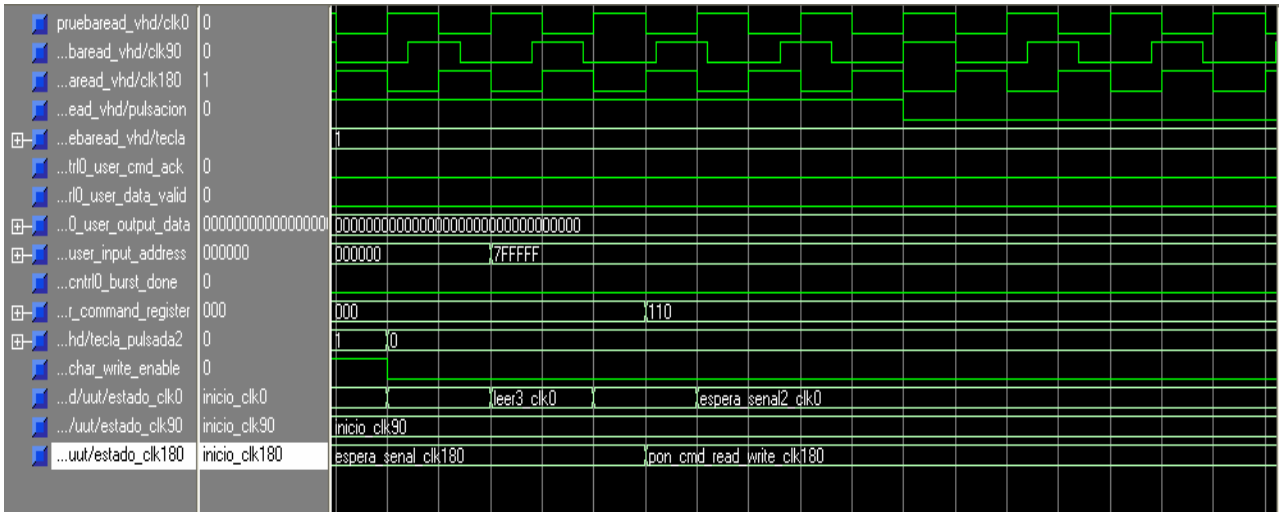


Figura 2.25

La simulación sigue en la imagen 2.26. Deberíamos tener ya colocada en la entrada user_input_address la dirección que fue introducida al principio. Como respuesta al comando de lectura, el controlador de la DDR activa la señal user_cmd_ack en un flanco positivo del reloj clk180. Igual que ocurría en la escritura, la dirección debe estar colocada en la línea de entrada durante 4,5 ciclos de reloj desde que usr_cmd_ack se active. Las siguientes direcciones que se introduzcan lo harán en flancos positivos alternos del reloj clk0. Por lo tanto controlamos el número de ciclos mediante transiciones de la máquina de estados del reloj clk0.

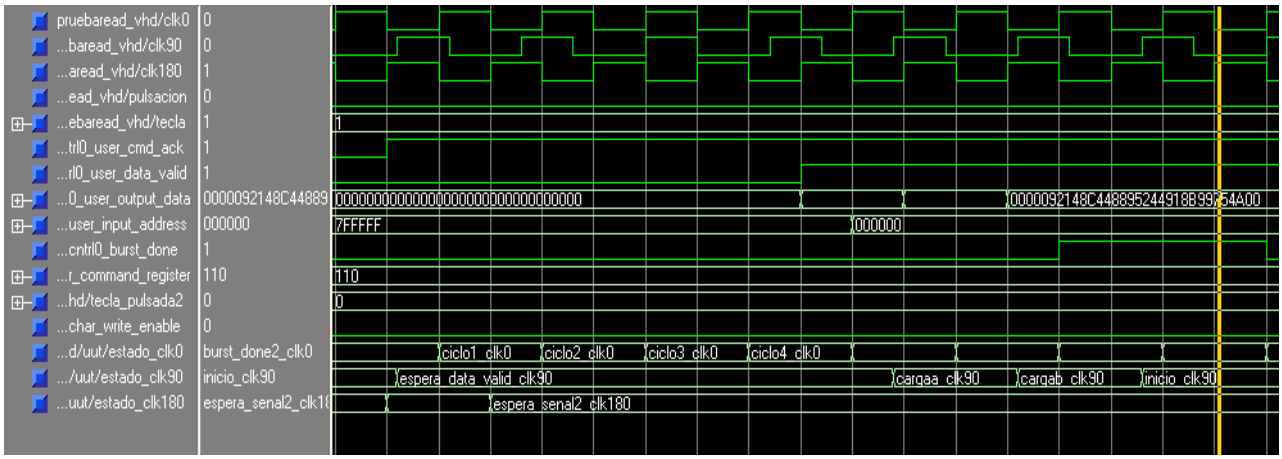


Figura 2.26

El dato que se recibe por user_output_data sólo será válido cuando el controlador de la DDR active la señal data_valid. Como en el caso de la escritura, los datos se cargan con el reloj clk90, por lo que se producen una serie de transiciones en su máquina de estados para cargar los registros.

Tras haber recibido la última dirección, el controlador de la DDR activa la señal burst_done en un flanco de subida del reloj clk0, durante dos ciclos. El comando de user_command_register se puede quitar después de que se desactive burst_done, como siempre lo dejaremos con el valor de no operación “000”. El controlador desactivará la señal user_cmd_ack después de que se haya completado la operación.

En la imagen 2.27 se ve este cambio del comando. Además, se muestra en la VGA el contenido de los registros con los datos leídos en esta operación, para lo cual activamos la señal char_write_enable.

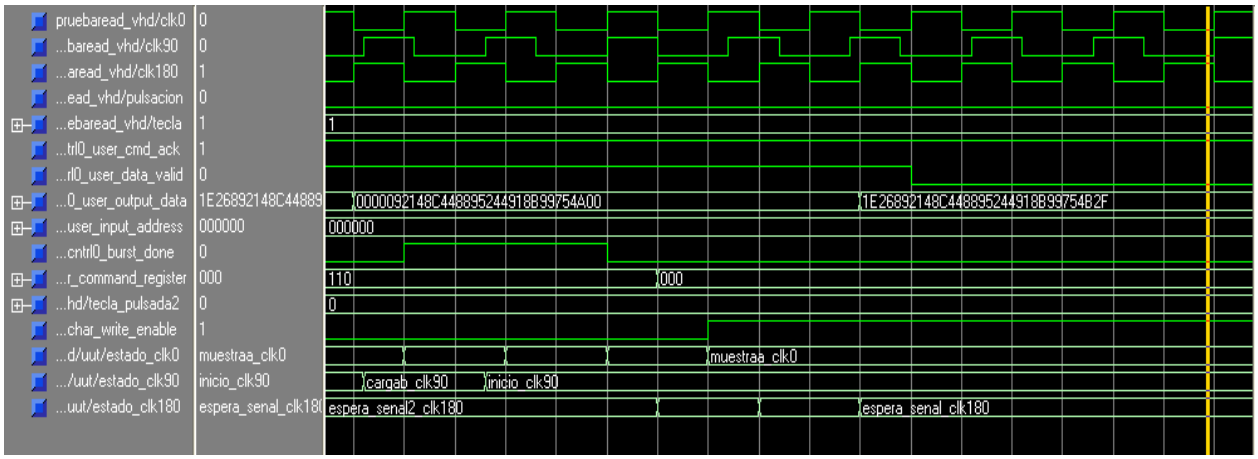


Figura 2.27

2.5.3.3.5 Depuración

Una vez comprobado que, en simulación, funcionaba correctamente nuestra unidad de control del interfaz de memoria para la DDR SDRAM, decidimos generar el archivo bitstream. Pero nos encontramos con una serie de problemas que no nos han permitido usar el interfaz. A continuación realizamos una descripción de estos problemas.

El primer problema surgió fue que el archivo .UCF que genera automáticamente el Mig007 no es específico para nuestra placa, de modo que los números PIN son erróneos. Además hay ciertas restricciones que dan error al intentar generar el archivo bitstream de nuestro proyecto. Así que tuvimos que modificarlo para adaptarlo a nuestra arquitectura del proyecto.

Una vez que solucionamos este problema y generamos el archivo bitstream nos encontramos con que, una vez cargada la FPGA con este archivo, ésta no se comportaba como deseábamos. En principio pensamos que quizás la memoria no se había inicializado correctamente. Pero esta posibilidad fue descartada pronto, ya que mostramos por un LED de la placa el bit que genera el interfaz de memoria cuando la memoria ha sido inicializada correctamente y comprobamos que ése no era el problema.

Tras revisar un largo período de tiempo todo el diseño, tanto nuestro como el del interfaz de memoria, decidimos utilizar el analizador de señales descrito con anterioridad, para observar cuál era el valor real de las señales que gobiernan tanto la DDR como el interfaz de memoria. Recurrimos al analizador debido a que al simular un módulo con el Model Simulator se fuerzan las señales a valores concretos cuando nos interesa, y estos valores pueden que no sean ciertos al cargar el diseño en un sistema real.

En primer lugar, quisimos comprobar si el interfaz de memoria enviaba las señales correctas a la memoria en en las operaciones de escritura y de lectura.

En la captura de la imagen 2.28 se muestra las señales más importantes en la operación de escritura en memoria:

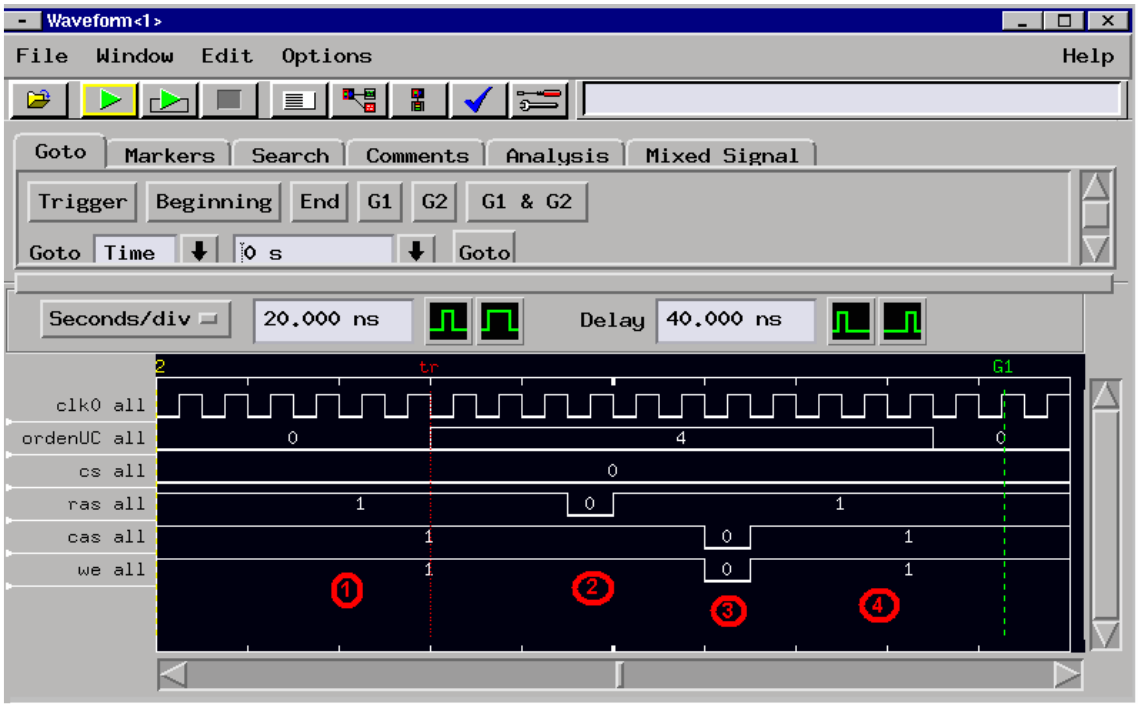


Figura 2.28

En esta captura se muestra el reloj clk0 que genera el módulo generador de señales de reloj. El bus ordenUC es el comando que envía nuestra unidad de control al interfaz generado del Mig007. Las señales cs, ras, cas y we son las señales que genera el interfaz de memoria que especifican la operación a realizar directamente a la DDR.

En esta captura se observa en el lugar marcado con 1 que al principio (mientras se espera una orden de escritura o lectura) el interfaz envía cs=0, ras=1, cas=1 y we=1 a la DDR. Esta configuración es la del operador NOP. De modo que por el momento el funcionamiento es correcto. En el punto 2, después de haber pulsado una 'S' en el teclado, ordenUC tiene un valor de 4, que es el comando de escritura para el interfaz, de modo que el interfaz genera cs=0, ras=0, cas=1 y we=1, que corresponde con el comando ACTIVE para seleccionar el banco de la DDR donde se escribe. A continuación, en el punto 3 el interfaz envía el comando de escritura (cs=0, ras=1, cas=0 y we=0) a la DDR en un flanco de subida del reloj complementario a clk0, es decir el clk180. Tras enviar el comando de escritura, el interfaz envía NOP a la DDR y la máquina de estados de la unidad de control ya no está en un estado de escritura en memoria, ya que ordenUC tiene un valor de 0(punto 4).

Luego los problemas con el interfaz parecen que no están localizados en la operación de escritura ni en la de inicialización, sino en la lectura como se muestra en la captura de la imagen 2.29.

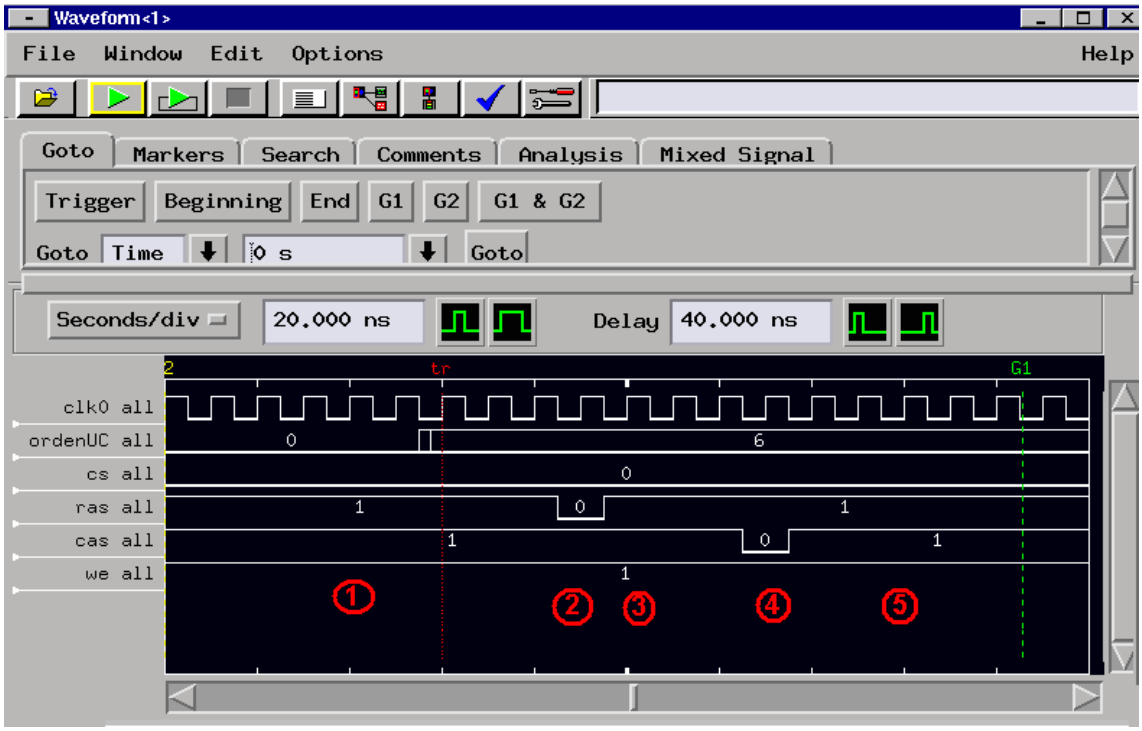


Figura 2.29

La señal orden UC en el punto señalado por 1 está en la configuración de NOP a la espera de recibir una orden por parte de usuario. Una vez que el usuario introduce por teclado una 'L' para proceder a la lectura de memoria, ordenUC se pone en la configuración de lectura (valor 6). En el punto 2 el comando que recibe la unidad de control a través de las señales cs, ras, cas y we es el correspondiente a ACTIVE que selecciona el banco que se quiere leer. Tras unos ciclos en NOP (en el punto 3), el interfaz envía el comando de lectura (cs ras cas we= 0110). Hasta aquí todo ha ido correctamente, pero el problema radica en que nuestra unidad de control se queda bloqueada. Este bloqueo se debe a que por algún motivo, que no hemos logrado subsanar, la señal de user_data_valid que genera el interfaz de memoria cuando lee de ésta, nunca se pone a alta. Así, nuestra unidad de control se queda esperando la llegada de este evento que nunca sucede (punto 5).

Tras haber analizado el diseño generado por el Mig007, creemos que el problema está en unas colas FIFO que utiliza para la lectura y la escritura en memoria, ya que la señal user_data_valid se pone a alta cuando estas colas no están vacías, pero no hemos sido capaces de solventar el problema. De modo que hemos dejado en un segundo plano este problema, para centrarnos en la implementación del módulo del planificador que realmente tiene más importancia en el objetivo del proyecto, ya que ya hemos implementado un gestor de memoria que funciona perfectamente para memorias RAM.

2.6 Planificador de tareas

2.6.1 Introducción

El módulo planificador de tareas es responsable de gestionar la ejecución de las tareas almacenadas en la RAM, que llegan a él por medio del gestor de memoria. La planificación se hará siguiendo una política tipo FIFO. Por supuesto, para esto será necesario que disponga de alguna información de cada tarea, en nuestro caso será la dirección de la RAM en que se ubica el comienzo de la misma, y su tamaño.

Utilizaremos una cola FIFO en la que el planificador irá almacenando los identificadores de las tareas que gestiona en orden de entrada, de manera que cada vez que se desea planificar una nueva tarea se procede a sacar de la cola el primer elemento de ésta y a tratar la tarea correspondiente.

En la figura 2.30 se muestra el planificador de tareas como una caja negra.

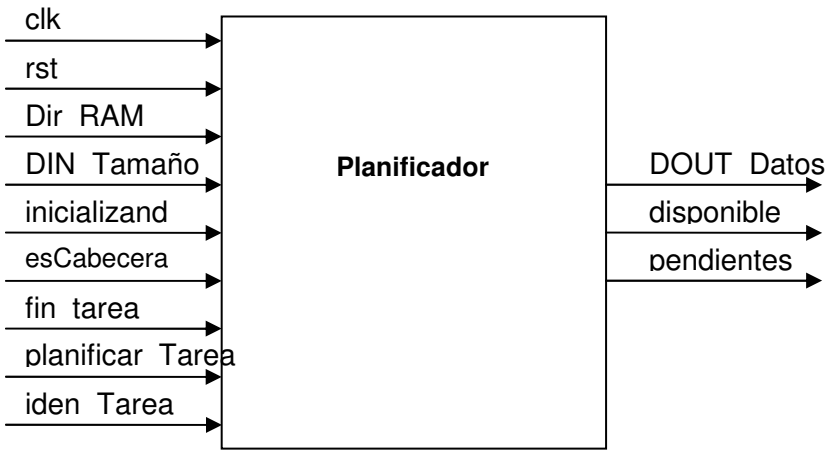


Figura 2.30: Módulo planificador de tareas.

2.6.2 Descripción de las señales

rst y clk corresponden a las entradas de reset y reloj respectivamente.

DIR_RAM[15:0] indica la dirección de la RAM a partir de la cual está ubicada la tarea que va a procesar el planificador.

DIN_Tamaño[8:0] corresponde al tamaño de la tarea. Estos dos últimos parámetros son necesarios para poder incorporar la tarea al planificador desde la RAM a través del gestor de memoria, ya que sin el tamaño de la tarea sería imposible determinar cuándo termina su mapa de bits en la RAM. El tamaño máximo de cada tarea viene limitado por el número de bits de esta señal.

inicializando indica que debe realizarse la carga de la información de cada tarea en el banco de registros destinado a ese fin. La información correspondiente a una tarea se almacena en la posición correspondiente a su identificador, por ejemplo, la información de la tarea cuyo identificador es “0100” se almacena en el registro R4

esCabecera indica que debe cargarse la información de una tarea en el registro correspondiente, sólo en caso de que la señal inicializando se encuentre activa. En cualquier otro caso permanece inactiva.

fin_tarea indica al planificador que una de las tareas que se ejecutaban en la FPGA ha terminado su ejecución, dejando libre uno de los 4 marcos.

planificar_tarea indica que quiere ejecutarse una nueva tarea, cuyo identificador es dado a través de la entrada iden_tarea. Según sea el estado de la cola, o haya marcos libres en la FPGA para su ejecución, el planificador gestionará esta petición como sea necesario.

Tanto inicializando, como esCabecera, fin_tarea y planificar_tarea se consideran activas a alta.

iden_tarea recibe el identificador de la tarea que entra en el planificador.

La señal de salida DOUT_Datos[24:0] determina la dirección y el tamaño de la próxima tarea para ejecutar, siendo los 16 primeros bits de la señal los correspondientes a la dirección y los 9 restantes al tamaño.

disponible se activa para indicar que los datos de la tarea saliente están disponibles, ya que estos no son fiables hasta que esta señal no se active (se considera activa a alta).

pendientes indica si hay tareas pendientes de ejecutar en el planificador, es decir, dentro de la cola. Se considera activa a alta.

2.6.3 Implementación

Ruta de datos

El planificador está implementado a partir de una cola FIFO, cuyo diseño expondremos con detalle en el siguiente apartado. Ésta almacena ordenadamente los identificadores de las tareas que vayan a planificarse, y los va devolviendo por el mismo orden en que entraron cuando sea necesario.

Está claro que almacenar los identificadores de las tareas ordenadamente no es suficiente para planificarlas y enviarlas para su ejecución, sino que para acceder a ellas será preciso además conocer la dirección de memoria en la que están almacenadas, y también su tamaño. A fin de organizar estos datos utilizaremos un banco de registros (información de tareas), cuya salida de datos será a su vez la salida del módulo planificador, ya que, como hemos dicho, contiene toda la información relevante de la tarea.

Se conectará la entrada de datos del banco de registros a la salida de un registro (reg_datos) de 25 bits, en el que se encontrarán almacenadas la dirección de comienzo de la tarea y su tamaño (ambos datos concatenados, uno a continuación de otro). La entrada de selección del banco de registros está conectada a la salida de un sistema de multiplexores (mux_1 y mux_2) con el fin de cubrir los tres casos posibles con los que nos podemos encontrar.

Cuando se está inicializando el planificador es el momento de almacenar en cada registro del banco de registros la información de cada tarea. Como las tareas van ordenadas, utilizaremos la salida de un contador ascendente de 4 bits para indicar el identificador de cada una, y, por lo tanto, la posición del registro en el que vamos a almacenar la información. Conectaremos esta salida a una de las entradas de mux_1.

Sin embargo, cuando termine de inicializarse el planificador podemos introducir tareas en él para que las gestione como corresponda. Por lo tanto la otra entrada de mux_1 se toma directamente de la salida de la cola en la que se almacenan los identificadores de las tareas en espera.

Sin embargo, la salida de este multiplexor no irá conectada directamente a la entrada de selección del banco de registros, ya que no se cubre la posibilidad de que la cola esté aún vacía, y por lo tanto las tareas no tengan que “hacer cola” antes de ejecutarse. Por eso la salida del anterior multiplexor irá conectada a la entrada de un segundo multiplexor, correspondiendo la otra entrada de éste al identificador de tarea que entra directamente al planificador, sin pasar por la cola. Éste identificador se encuentra almacenado en el registro reg_iden. La salida de este multiplexor ya es válida para seleccionar un registro del banco de registros de información de tareas.

Mostraremos, a continuación, la ruta de datos del módulo planificador de forma esquemática. Puede verse en la imagen 2.31.

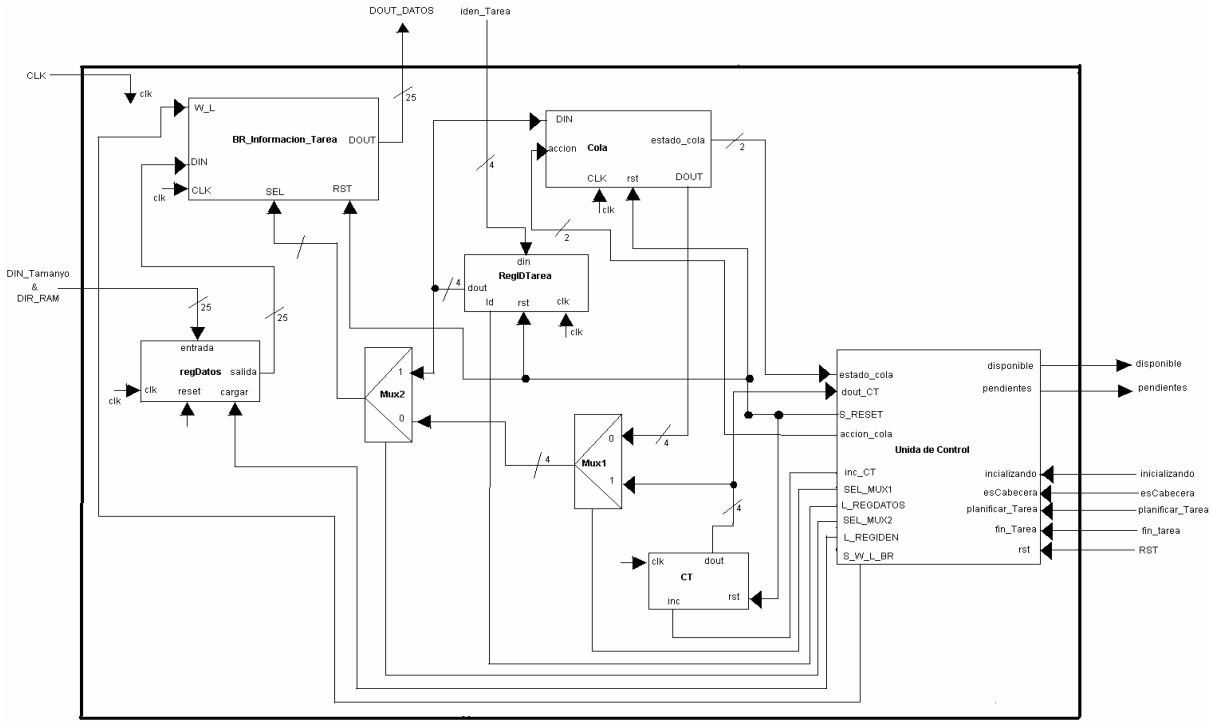


Figura 2.31

Controlador

En la unidad de control del planificador se generan las todas las señales intermedias que intervienen, así como el estado siguiente. Para que resulte más sencilla la consulta de la tabla de transiciones entre estados mostramos en la tabla 2.5 una equivalencia entre los nombres que se les ha dado en la implementación y los que tienen en adelante en este documento.

Nombre de la implementación	Correspondencia para el documento
S0_No_Inicializado	S0
S1_Inicializando	S1
S2_GuardandoBR	S2
S3_Esperando_Eventos	S3
S4_Dame_DirRAM_TAM	S4
S5_Dame_Siguiente_Tarea_Cola	S5_t
S5_Dando_Siguiente_Tarea_Cola1	S5_t_1
S5_Dando_Siguiente_Tarea_Cola2	S5_t_2
S5_Dame_Datos_Iden_Tarea	S5_d
S6_Mete_En_Cola	S6
S6_Metiendo_En_Cola1	S6_1
S6_Metiendo_En_Cola2	S6_2
S7_Demasiadas_Tareas_Posibles	S7
S8_Hay_Que_Resetear	S8
S9_Mete_Saca_Cola	S9
S9_Mete_Cola1	S9_1
S9_Mete_Cola2	S9_2

Tabla 2.5

Siguiendo este convenio, mostramos en la tabla 2.6 la generación del estado siguiente en función del estado actual de nuestra máquina de estados y de las señales de control que se muestran.

Estado actual	inicializando	esCabecera	finTarea	planificar	s_dout_ct	estado cola	Estado siguiente
S0	‘1’	X	X	X	XXXX	XX	S1
S1	‘1’	‘1’	X	X	XXXX	XX	S2
S1	‘1’	‘0’	X	X	XXXX	XX	S1
S1	‘0’	X	X	X	XXXX	XX	S3
S2	X	X	X	X	“1111”	XX	S7
S2	X	X	X	X	≠”1111”	XX	S1
S3	‘1’	X	X	X	XXXX	XX	S0
S3	‘0’	X	X	‘1’	XXXX	“11”	S8
S3	‘0’	X	‘0’	‘1’	XXXX	XX	S6
S3	‘0’	X	‘1’	‘0’	XXXX	“10”	S5_t
S3	‘0’	X	‘1’	‘0’	XXXX	“00”	S3
S3	‘0’	X	‘1’	‘1’	XXXX	“00”	S5_d
S3	‘0’	X	‘1’	‘1’	XXXX	“10”	S9
S5_d	X	X	X	X	XXXX	XX	S3

S5_t	X	X	X	X	XXXX	XX	S5_t_1
S5_t_1	X	X	X	X	XXXX	XX	S5_t_2
S5_t_2	X	X	X	X	XXXX	XX	S3
S6	X	X	X	X	XXXX	XX	S6_1
S6_1	X	X	X	X	XXXX	XX	S6_2
S6_2	X	X	X	X	XXXX	XX	S3
S7	‘1’	X	X	X	XXXX	XX	S7
S7	‘0’	X	X	X	XXXX	XX	S3
S8	X	X	X	X	XXXX	XX	S8
S9	X	X	X	X	XXXX	XX	S9_1
S9_1	X	X	X	X	XXXX	XX	S9_2
S9_2	X	X	X	X	XXXX	XX	S5_t

Tabla 2.6

Para que resulte más sencillo a simple vista, mostramos a continuación la misma información en forma de diagrama de estados. Para mayor claridad hemos separado la máquina de estados en 4 más sencillas. La primera de ellas (figura 2.32) refleja la transición de estados para la etapa de inicialización del planificador.

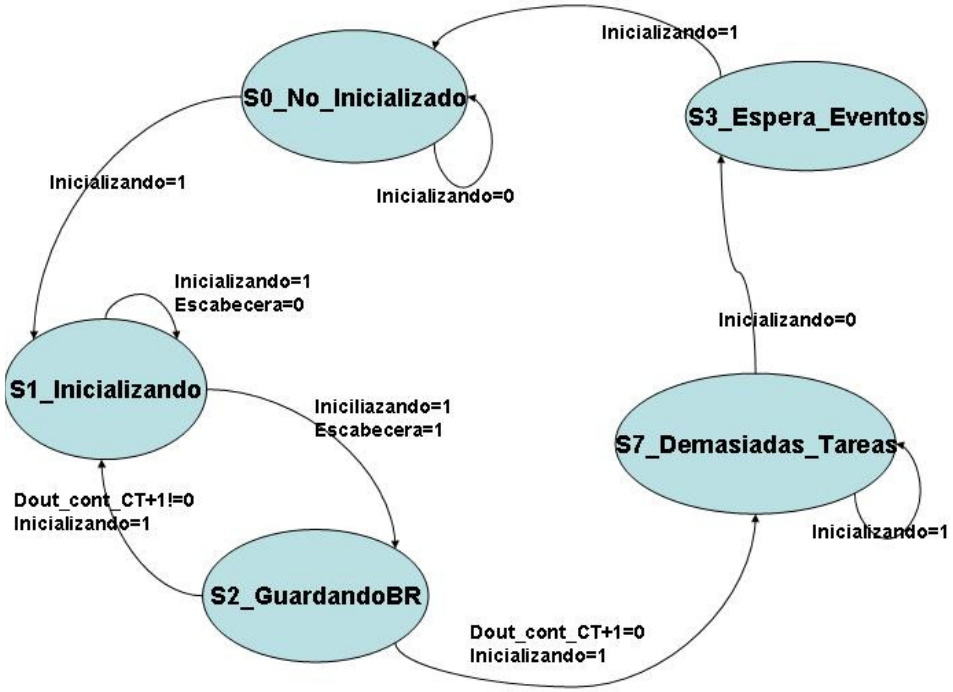


Figura 2.32

En las siguientes se parte del estado de espera, y en función del estado actual se producirán diferentes comportamientos. En la imagen 2.33 se ven las transiciones cuando el estado de la cola indica que no caben más tareas, con lo cual no pueden planificarse más, y para la situación en que se quiere ejecutar una tarea pero no hay ningún marco del área de ejecución de tareas libre.

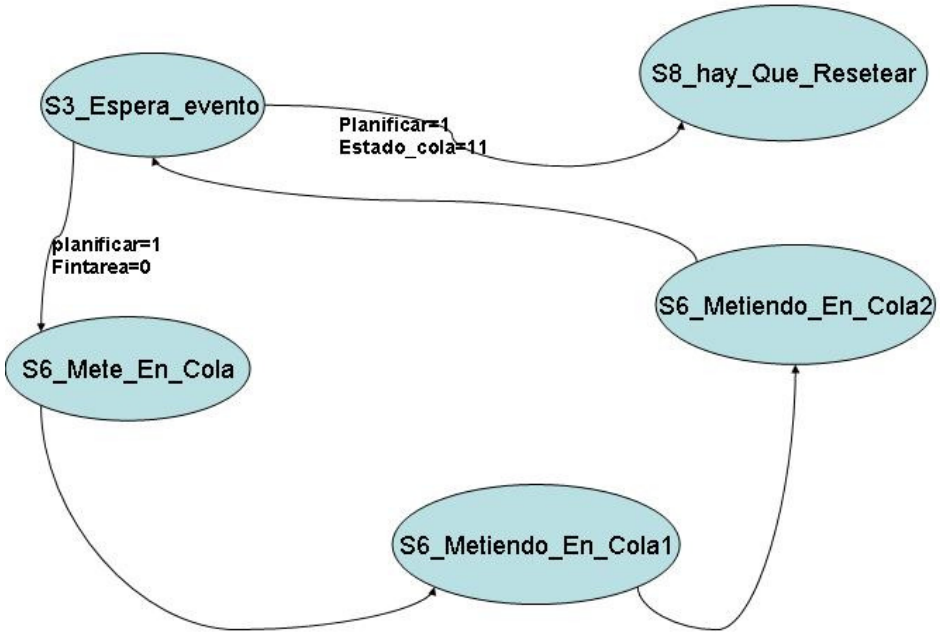


Figura 2.33

En la figura 2.34 se ven las posibles transiciones cuando se quiere planificar una tarea y la cola está vacía, y también cuando la cola está en estado normal, es decir, no está vacía y hay espacio para más tareas.

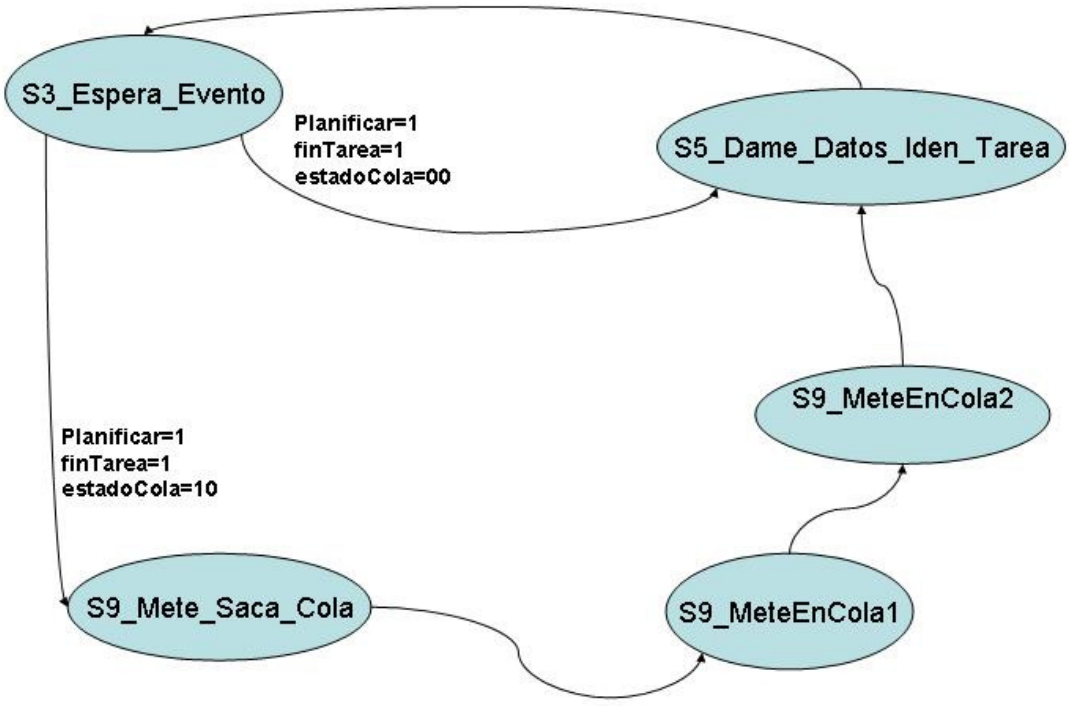


Figura 2.34

Por último, en la figura 2.35 se ve la transición de estados para la situación en que un marco del área de ejecución de tareas se libera.

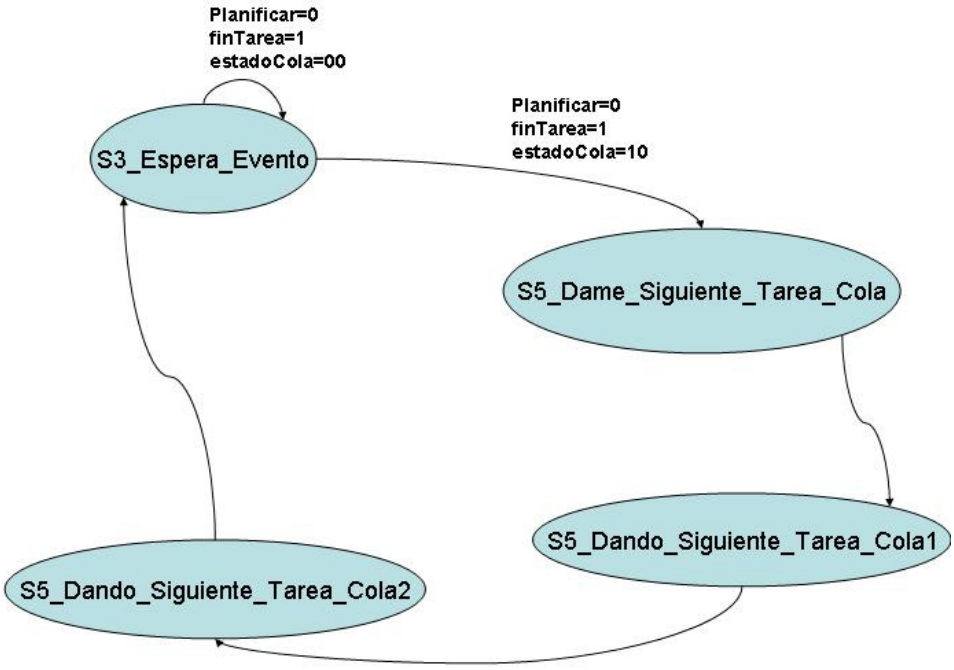


Figura 2.35

En la tabla 2.7 mostramos la generación de las salidas del planificador (los datos de la tarea y el indicador de disponibilidad), las entradas de control para los multiplexores y la señal de incremento del contador.

Estado	s_sel_mux1	s_sel_mux2	disponible	s_inc_ct	s_accion_cola
S0	'0'	'0'	'0'	'0'	"00"
S1	'0'	'0'	'0'	'0'	"00"
S2	'1'	'0'	'0'	'1'	"00"
S3	'0'	'0'	'0'	'0'	"00"
S5_d	'0'	'1'	'1'	'0'	"00"
S5_t	'0'	'0'	'0'	'0'	"10"
S5_t_1	'0'	'0'	'0'	'0'	"10"
S5_t_2	'0'	'0'	'1'	'0'	"10"
S6	'0'	'0'	'0'	'0'	"01"
S6_1	'0'	'0'	'0'	'0'	"01"
S6_2	'0'	'0'	'0'	'0'	"01"
S7	'0'	'0'	'0'	'0'	"00"
S8	'0'	'0'	'0'	'0'	"00"
S9	'0'	'0'	'0'	'0'	"01"
S9_1	'0'	'0'	'0'	'0'	"01"
S9_2	'0'	'0'	'0'	'0'	"01"

Tabla 2.7

2.6.4 Funcionamiento

En este apartado presentaremos algunas simulaciones del módulo planificador de tareas que hemos explicado en este apartado. En la primera de ellas, que se muestra en la figura 2.36, hemos comprobado que la inicialización del banco de registros que contiene la información de tareas se realiza correctamente, y posteriormente hemos activado la entrada planificar y desactivado la entrada fin_tarea. Con esta configuración lo que se pretende es que la tarea obligatoriamente tenga que esperar su turno pasando por la cola FIFO.

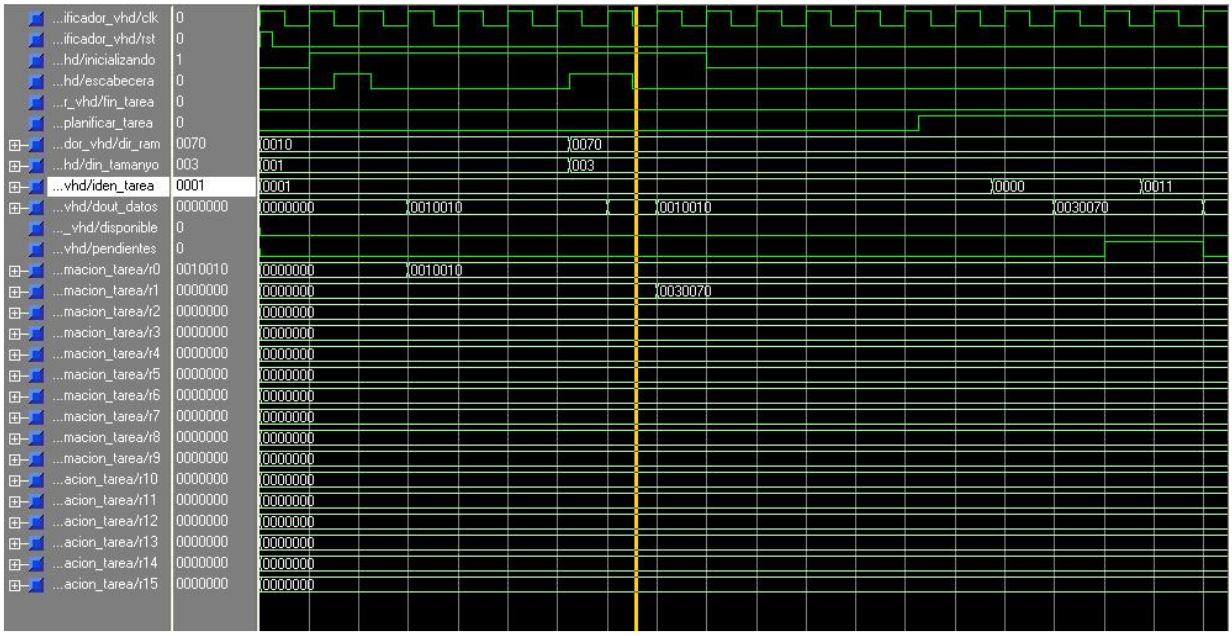


Figura 2.36

En la simulación se ve cómo primero se activa la señal inicializando. Este cambio en la señal desencadena el comienzo de la inicialización del planificador, durante la cual tendrán que almacenarse en el banco de registros la información de las tareas que entra por las entradas DIN_Tamaño y Dir_RAM. Comenzarán a guardarse estos datos cuando se active la señal esCabecera.

Así, cuando se activa la señal esCabecera, se carga en el primer registro R0 del banco de registros la información de la tarea cuyo identificador es “0000”. En la imagen puede observarse como el contenido del registro R0 no es más que la concatenación de los valores de DIN_Tamaño y Dir_RAM. Todos estos valores se muestran en hexadecimal. También se observa este dato en la salida de la información de la tarea del planificador, pero esto no importa ya que la salida disponible está deshabilitada, y por lo tanto el valor de la salida no es relevante. Después de esto la señal esCabecera vuelve a deshabilitarse, a la espera de la información de una nueva tarea.

Varios ciclos de reloj más tarde se puede observar una nueva activación de esCabecera. Además hemos cambiado los valores de DIN_Tamaño y Dir_RAM, ya que, al menos el primero, no debe ser el mismo cuando la ejecución sea real (no puede haber dos tareas almacenadas en la misma dirección de la RAM. Igual que

ocurrió antes, en el siguiente ciclo de reloj se habrá cargado la información de la siguiente tarea (la del identificador “0001”) en el registro R1 del banco de registros.

El siguiente paso que hemos dado ha sido deshabilitar la entrada inicializando, dando por finalizadas las labores de inicialización dentro del planificador. Pasamos a ver cómo se produce la planificación de las tareas.

Para ello activamos la entrada planificar, manteniendo deshabilitada la entrada fin_tarea. En la entrada iden_tarea colocamos el identificador de la tarea que vamos a planificar, “0000”. El planificador interpreta esto como la llegada de una nueva tarea en ausencia de espacio en el área de ejecución de tareas de la FPGA, por lo que deberá introducirla en la cola FIFO. Debido a este cambio en el estado de la cola se activa la salida pendientes, indicando así que hay tareas esperando en la cola.

Vamos a mostrar ahora otro ejemplo de simulación en el que queremos ver cómo funciona el planificador cuando hay espacio en el área de tareas de la FPGA. Mostramos esta situación en la imagen 2.37.

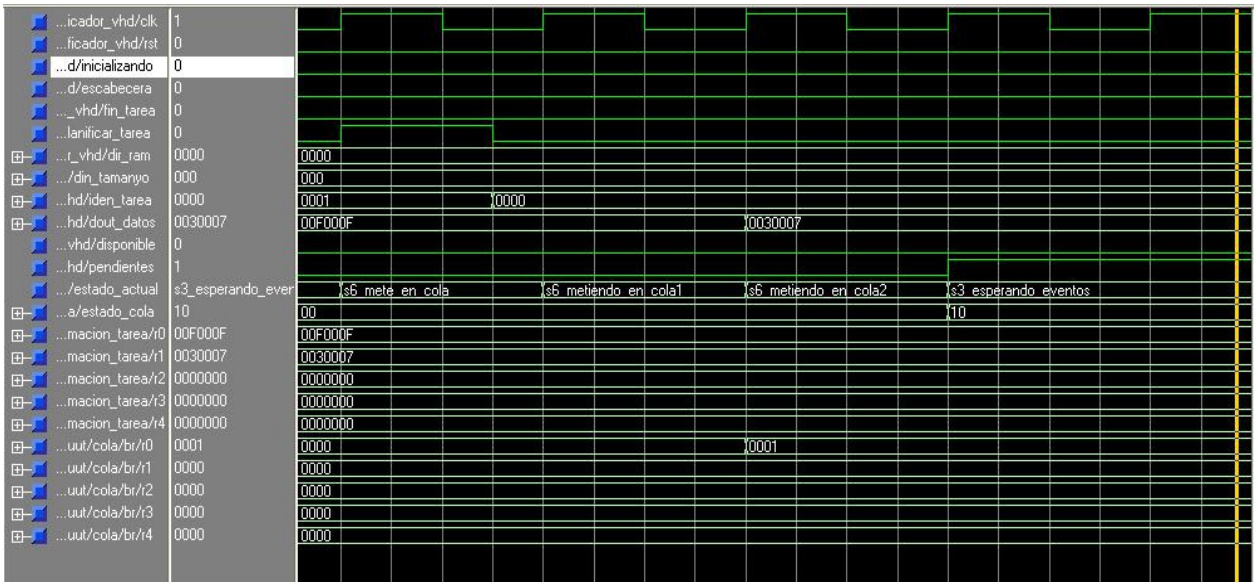


Figura 2.37

Mostramos esta simulación a partir del momento en que el planificador ya ha sido inicializado y por lo tanto está a la espera de recibir alguna orden. Activamos la señal planificar_tarea, y mantenemos inactiva fin_tarea. Como ya hemos comentado, en este caso el planificador interpreta la llegada de una nueva tarea y, al no tener espacio libre en el área de tareas de la FPGA, envía su identificador a la cola. Como veremos en el apartado dedicado a la cola FIFO, las operaciones de lectura y escritura sobre ella tienen una latencia de dos ciclos de reloj, después de los cuales podemos observar que el registro de la cola Rc1 contiene ahora el valor “0001”, que corresponde al identificador de la tarea que fue introducida al planificador, y la señal pendientes se ha activado, ya que hay una tarea pendiente de ejecutarse en el planificador. Después de esto el planificador regresa al estado de espera, y la cola pasa de estar vacía a su estado normal.

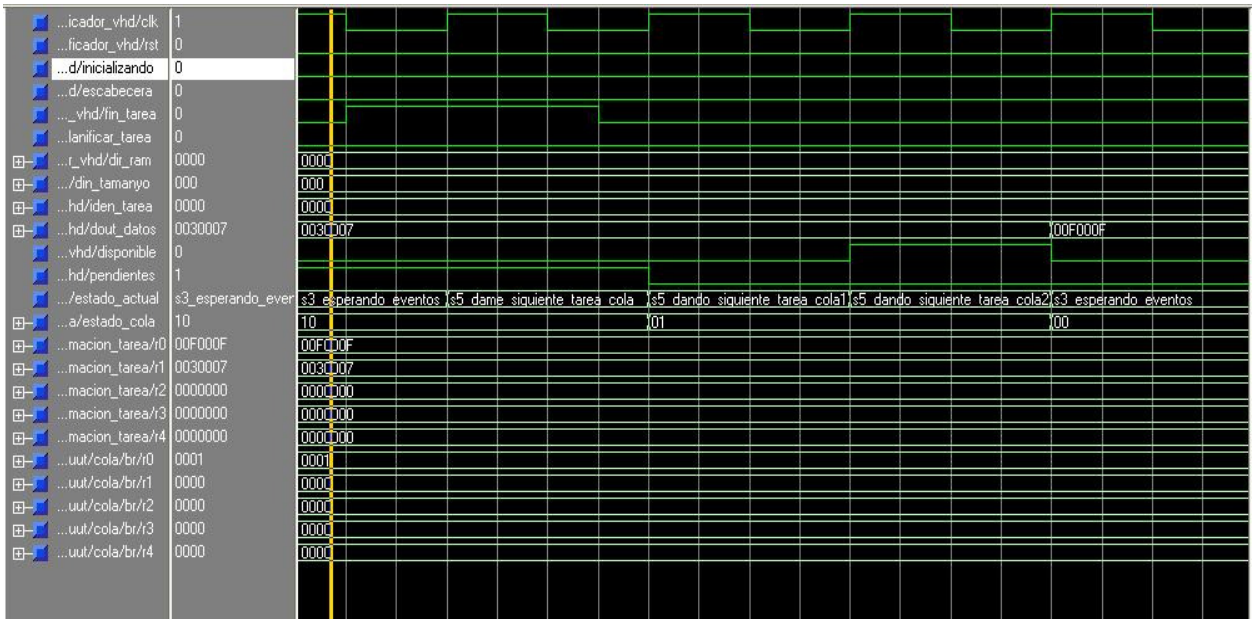


Figura 2.38

En la imagen 2.38 se ve la continuación de esta prueba. Se ha deshabilitado la señal planificar tarea, y a continuación se ha activado fin_tarea. Esto indica que ha finalizado la ejecución de una tarea, y por lo tanto que hay espacio libre en el área de tareas de la FPGA. Ante esta situación el planificador debe coger la primera tarea de la cola, que será la que se insertó primero, y enviarla para su ejecución. Observamos que la información de la tarea se ha colocado en la salida DOUT_datos del planificador, y activamos la señal disponible para indicar que este dato es válido.

Por otra parte, la señal pendientes, que se había activado al planificar e introducir una tarea en la cola, vuelve a desactivarse, ya que tras ejecutar la única tarea que había en la cola ésta se ha quedado vacía y ya no hay tareas pendientes. También se observa como después de los dos ciclos de operación para sacar de la cola, volveremos a su estado de cola vacía. El planificador volverá a su estado de espera.

2.6.4 Cola FIFO

2.6.4.1 Introducción

La parte fundamental de un planificador de tareas es la que se ocupa de decidir el orden en que éstas son atendidas para su posterior ejecución, y los motivos que conducen a esta decisión. Por ejemplo, los procesos pueden estar asociados a niveles de prioridad que haga anteponer la ejecución de unos más prioritarios frente a otros. Sin embargo para nuestro planificador la prioridad de cada tarea simplemente la marcará el tiempo de espera dentro del mismo. Así, una tarea se volverá más prioritaria cuanto más tiempo lleve esperando para ser ejecutada, y de la misma forma

la última tarea entrante será la menos prioritaria. Este mecanismo se implementa por medio de una cola FIFO.

Las colas FIFO (*first in first out*) son una sencilla estructura en la que los elementos son almacenados por estricto orden de llegada, y atendidos por orden de antigüedad dentro de la misma. A continuación mostramos el diseño de la cola FIFO que hemos empleado para la gestión de tareas dentro del módulo planificador.

En la figura 2.39 se muestra el módulo que representa la cola que hemos empleado en nuestro planificador de tareas:

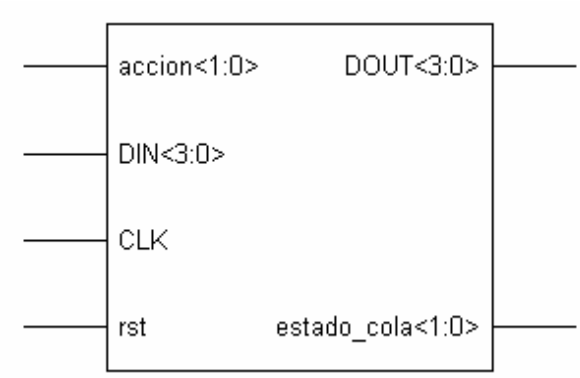


Figura 2.39: Cola FIFO.

2.6.4.2 Descripción de señales

Las entradas **clk** y **rst** corresponden al reloj y al reset, respectivamente. El reset se activa a alta.

accion es la entrada de dos bits que indica qué operación entre las siguientes debe realizar la cola:

> accion = “01” para insertar un identificador de tarea al final de la cola.
> accion = “10” para sacar el identificador de la tarea que más tiempo lleva dentro de la cola.
> Cualquier otro caso: No realizar ninguna acción y mantener el estado de la cola.

Tanto para insertar un identificador de tarea como para sacarlo, la latencia es de dos ciclos de reloj, durante los cuales se indica que la cola está operando mediante la señal de salida estado_cola que se explica más adelante.

La señal de entrada **DIN** recibe el dato a introducir en la cola, en nuestro caso el identificador de la tarea que se quiera insertar en la cola, y la de salida **DOUT**, que muestrea el identificador de la última tarea explusada de la cola.

Tanto **DIN** como **DOUT** son señales de 4 bits, por lo tanto la máxima cantidad de identificadores de tareas que podemos gestionar con esta cola y, por lo tanto, con nuestro planificador, es de 16. Esto parece bastante razonable si tenemos en cuenta que posteriormente sobre la FPGA se tiene previsto ejecutar simultáneamente un

máximo de 4 tareas. Respecto al tamaño de la cola, ésta tiene capacidad para gestionar hasta 32 tareas.

La salida **estado_cola** indica cuál es en cada momento el estado de la cola, dentro de los siguientes posibles:

estado = “00” cola vacía. En este estado pueden ser insertados elementos en la cola, pero no sacarlos.
estado = “01” realizando operación. Se trata de un estado intermedio que indica que la cola se encuentra realizando alguna operación de inserción o borrado, y por lo tanto antes de solicitar realizar otra acción sobre ella deberá esperarse a que abandone este estado.
estado = “10” estado normal. A la espera de recibir alguna acción.
estado = “11” overflow (la cola está llena).

2.6.4.3 Implementación

La cola está implementada con un banco de 32 registros, en cada uno de los cuales se encuentra almacenado el identificador de una tarea. Para ampliar el número de identificadores que podemos almacenar en la cola simplemente emplearíamos un banco de registros de mayor capacidad, en concreto con un número de registros mayor.

La entrada SEL del banco de registros procede de la salida de un multiplexor de dos entradas gobernado por la señal de control S_UC_OP. La primera entrada del multiplexor está conectada a la salida de un contador ascendente cont_ADD. Análogamente, la segunda entrada del multiplexor está conectada a la salida de otro contador ascendente cont_REM.

La salida de cont_ADD indica la posición, dentro del banco de registros, del primer registro disponible para almacenar un identificador de tarea nuevo, que se ubicará al final de la cola. Por lo tanto S_UC_OP seleccionará la salida de este contador cuando pretendamos insertar un identificador en la cola. La salida de cont_REM indica la posición en el banco de registros del registro que almacena el identificador de la próxima tarea a despachar por la cola. Análogamente al caso de la inserción, S_UC_OP seleccionará la salida de este contador cuando lo que queremos sea sacar un identificador de la cola. Inicialmente la cola se encontrará en el estado cola vacía, y por lo tanto la salida de ambos contadores será igual.

De esta manera, cuando la salida del contador ascendente cont_REM más una unidad muestre el mismo valor que la salida del contador cont_ADD,el siguiente identificador que expulsemos será el último de la cola, y por lo tanto el siguiente estado de ésta será “00” correspondiente a la cola vacía. En el extremo opuesto está el caso en que el valor de salida de ambos contadores coincide, dándose la situación de overflow: La cola no puede recibir más tareas porque todos los registros del banco están ocupados. Si en este estado insertáramos un identificador más dentro de la cola empezaríamos a sobreescribir identificadores de tareas que aún no habrían sido despachadas, perdiendo la información de éstas antes de haberlas enviado para su ejecución. Por esta razón la cola permanecerá en este estado hasta que se produzca un evento de la señal reset.

Ruta de datos

La ruta de datos de la cola FIFO se muestra en la figura 2.40:

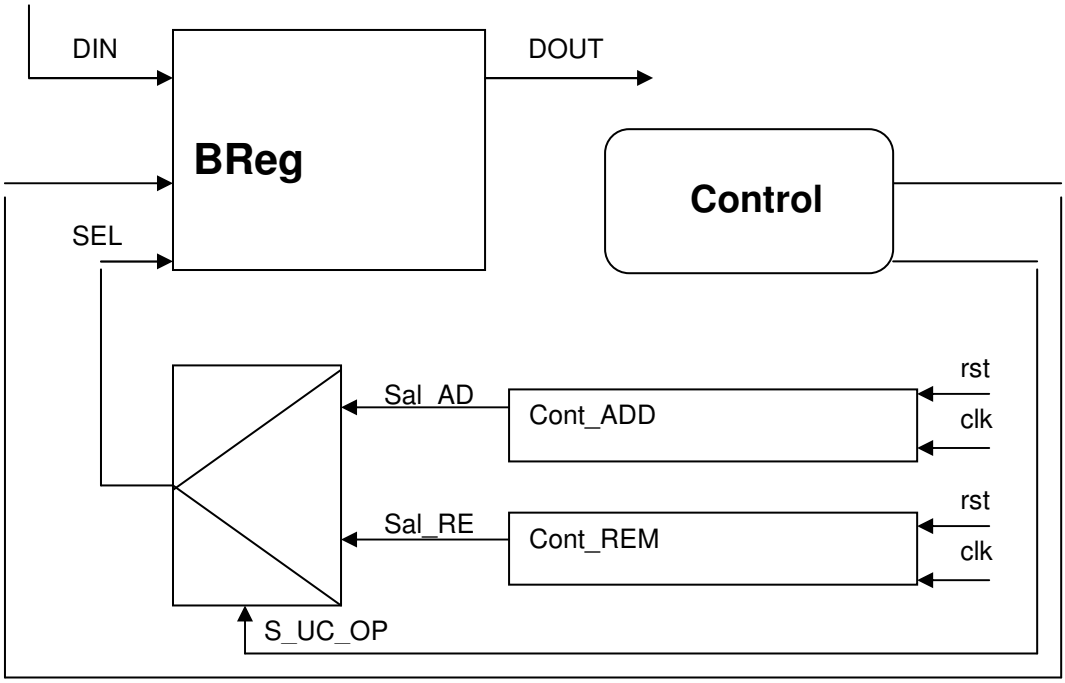


Figura 2.40: Ruta de datos de la cola FIFO

Controlador de la cola

Las señales internas presentes en la figura 2.40 se generan en la unidad de control en función del estado de nuestra máquina de estados finita y otras señales del sistema. El comportamiento de la máquina de estados de la cola FIFO se muestra en la tabla 2.8:

Estado actual	reset	accion	sal_ADD	sal_REM	Estado siguiente
S0_cola_vacía	'0'	"01"	X	X	S6_añade_a_cvacia
S0_cola_vacía	'0'	"00" ó "10" ó "11"	X	X	S0_cola_vacía
S1_estado_normal	'0'	"01"	=sal_REM	=sal_ADD	S5_overflow
S1_estado_normal	'0'	"00" ó "01"	X	X	S1_estado_normal
S1_estado_normal	'0'	"10"	=sal_REM	=sal_ADD	S0_cola_vacia
S1_estado_normal	'0'	"01" ó "10"	≠sal_REM	≠sal_REM	S2_evalua_accion
S2_evalua_accion	'0'	"01"	X	X	S3_add
S2_evalua_accion	'0'	"10"	X	X	S4_rem

S2_evalua_accion	‘0’	“11” ó “00”	X	X	S1_estado_normal
S3_add	‘0’	XX	≠sal_REM	≠sal_ADD	S1_estado_normal
S4_rem	‘0’	XX	1+sal_REM	X	S0_cola_vacia
S4_rem	‘0’	XX	≠1+sal_REM	X	S1_estado_normal
S5_overflow	‘0’	XX	X	X	S5_overflow
S6_añade_a_cvacia	‘0’	“01”	X	X	S7_evalua_accion_cvacia
S6_añade_a_cvacia	‘0’	“11” ó “00” ó “01”	X	X	S0_cola_vacia
S7_evalua_accion_cvacia	‘0’	XX	X	X	S1_estado_normal
SX (cualquier estado)	‘1’	XX	X	X	S0_cola_vacia

Tabla 2.8: Transiciones del diagrama de estados

En función del estado en que se encuentre nuestra máquina finita de estados y de las entradas del sistema asignaremos valor a las señales de control y de salida del módulo de la forma que se muestra en la tabla 2.9:

Estado	estado_cola	s_uc_op	s_inc_add	s_inc_rem
S0_cola_vacia	“00”	‘0’	‘0’	‘0’
S1_estado_normal	“10”	‘0’	‘0’	‘0’
S2_evalua_accion	“01”	‘1’ si accion=“01”	‘0’	‘0’
S3_rem	“01”	‘0’	‘1’	‘0’
S4_rem	“00”	‘0’	‘0’	‘1’
S5_overflow	“11”	‘0’	‘0’	‘0’
S6_añade_a_cvacia	“00”	‘1’	‘0’	‘0’
S7_evalua_accion_cvacia	“00”	‘0’	‘1’	‘0’

Tabla 2.9: Generación de salidas.

Por último, la salida DOUT que muestrea el identificador que se saca de la cola, se toma directamente de la salida del banco de registros que almacena los identificadores de las tareas.

2.6.4.4 Funcionamiento

Para probar el correcto funcionamiento de la cola hemos realizado varias simulaciones significativas en las que intentamos aplicar todas las operaciones y llegar a todos los estados posibles de la cola.

En las figuras 2.41 y 2.42 se muestra una primera simulación que comentamos a continuación.

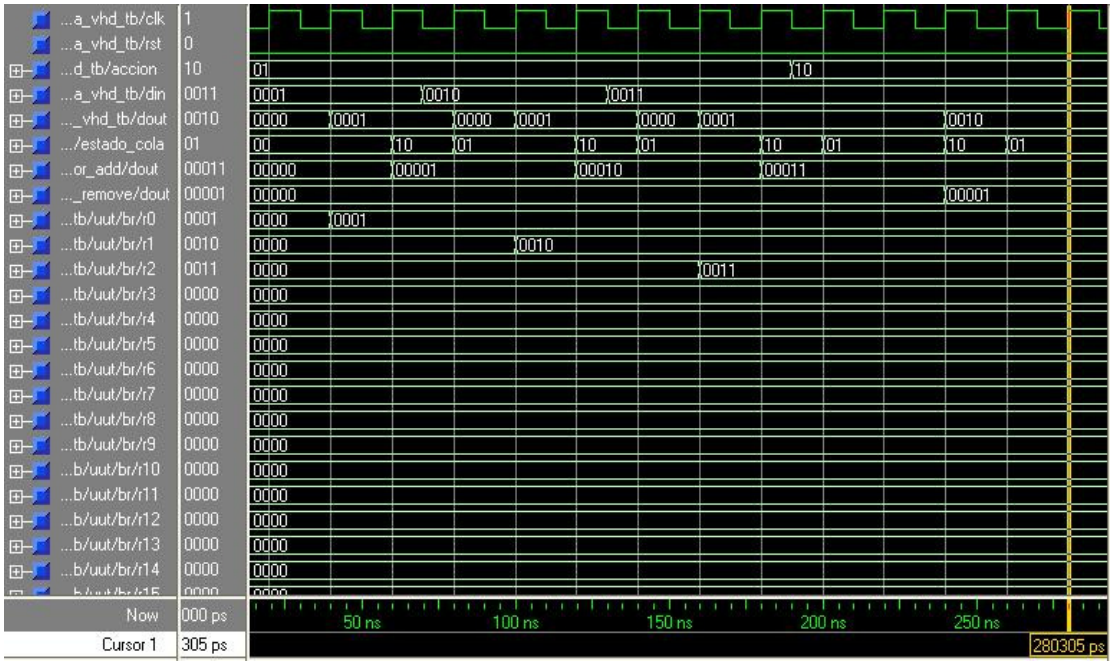


Figura 2.41: Ejemplo 1 de simulación de la cola

Inicialmente la señal estado_cola toma el valor “00”, ya que todavía no se ha añadido ningún identificador en la cola y por lo tanto está vacía. Las salidas de los contadores cont_ADD y cont_REM comienzan tomando valores nulos.

La primera acción que realizamos es una inserción mediante el valor “01” de la señal de entrada acción. El identificador de la tarea que se quiere añadir a la cola, que ya estará colocado en la entrada DIN, es “0001”. Se tarda dos ciclos de reloj cada vez que un elemento nuevo se añade a la cola. Excepcionalmente, la primera vez que esto ocurre no se notifican de ninguna manera los dos ciclos de latencia de la operación añadir. En las sucesivas inserciones la señal estado_cola pasará a valer “01”, indicando así que la cola está operando.

En nuestra simulación observamos que después de los dos ciclos el estado de la cola ha pasado a normal “10”. El identificador de la tarea que se estaba añadiendo a la cola se ha almacenado en el registro 0 del banco de registros, tal como indicaba la salida del contador cont_ADD, que a su vez se ha incrementado en una unidad indicando que ahora el siguiente registro en el que podemos almacenar un identificador de tarea es el R1.

En los siguientes ciclos no variaremos la señal accion, por lo tanto después de los dos ciclos en los que el valor de estado_cola corresponderá al de cola operando insertaremos cada identificador de tarea que vayamos colocando en DIN dentro del registro correspondiente e incrementando el valor de la salida del contador cont_ADD, y así sucesivamente hasta que en un momento cambiamos el valor de accion y lo fijamos en “10”, indicando así que queremos recuperar el primer identificador de la cola.

Igual que ocurría insertando un identificador en la cola, al sacarlo fuera de ella se tardan dos ciclos de reloj en los que la señal estado_cola toma el valor “01” correspondiente a cola operando. Después de este tiempo observamos que el identificador que ha salido de la cola por la salida DOUT es “0001”, justamente el primero que habíamos insertado. Además, el registro que almacenaba este identificador dentro del banco de registros conserva el valor del identificador. Esto no

supone ningún problema, ya que el valor de salida del contador cont_REM ha sido incrementado en una unidad, indicando así que el registro R1 es el que contiene el siguiente identificador que saldrá de la cola, y por lo tanto liberando el registro R0 para ser ocupado posteriormente por otro identificador distinto. Por lo tanto el valor almacenado en el registro R0 del banco de registros es ahora irrelevante.

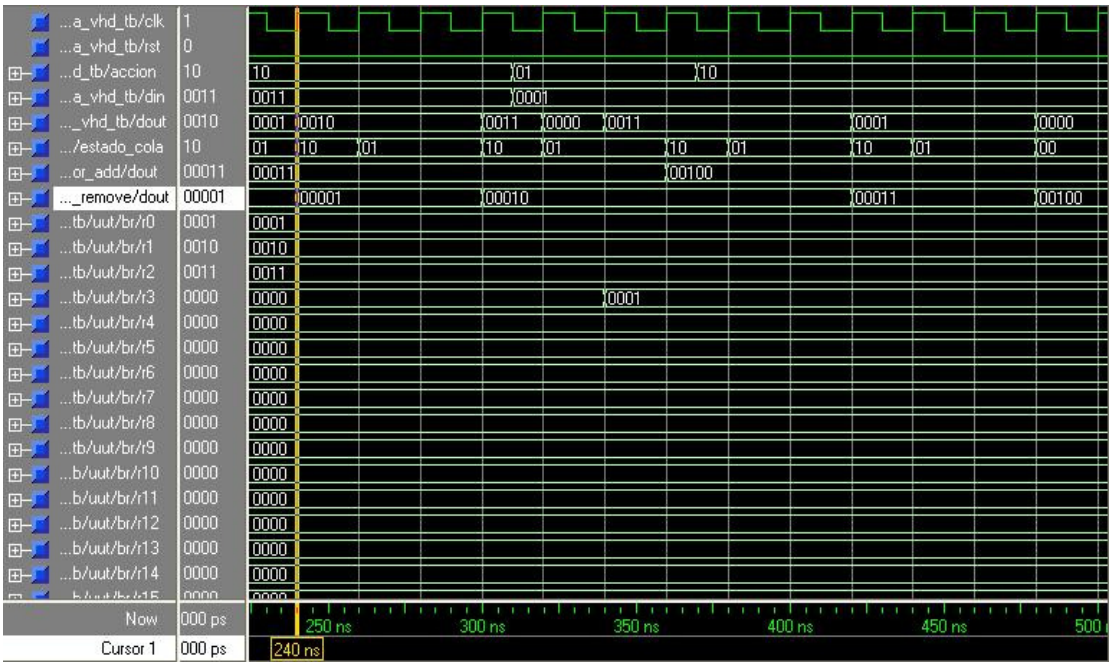


Figura 2.42: Ejemplo 1 de simulación de la cola (continuación)

Todavía sacaremos de la cola el identificador correspondiente a la segunda tarea que insertamos antes de cambiar de nuevo la señal accion para insertar de nuevo la tarea con el identificador “0001”, que esta vez se almacena en el registro 3, ya que es el primero del banco de registros libre según indica la salida del contador cont_ADD en ese momento.

Para terminar con este ejemplo, cambiamos de nuevo el valor de accion para extraer los identificadores que quedaban en la cola. Vemos como la salida del contador cont_REM se incrementa con cada identificador que sacamos de la cola, y finalmente toma el mismo valor que la salida del contador cont_ADD. Esto indica que el primer registro cuyo identificador sería expulsado de la cola se encuentra vacío, ya que es también el primer registro en el que podemos almacenar un identificador, y por lo tanto la situación a la que hemos llegado es a la de cola vacía, como podemos comprobar en la señal de salida estado_cola que es efectivamente “00”. En este punto, aunque la señal accion continua siendo la correspondiente a extraer identificadores de la cola, no se produce ningún efecto sobre ella. De hecho, no ocurrirá nada hasta que el valor de accion cambie y se inserte un nuevo identificador.

Anteriormente hemos especificado un estado overflow de la cola, en el que no pueden insertarse más identificadores debido a que se encuentra repleta y, de seguir realizando inserciones, sobreescribiría identificadores de tareas no enviadas para su ejecución. Para comprobar que la cola también se comporta correctamente evitando que esto ocurra, realizaremos una simulación en la que hemos forzado esta situación. Ésta se muestra en la figura 2.43.

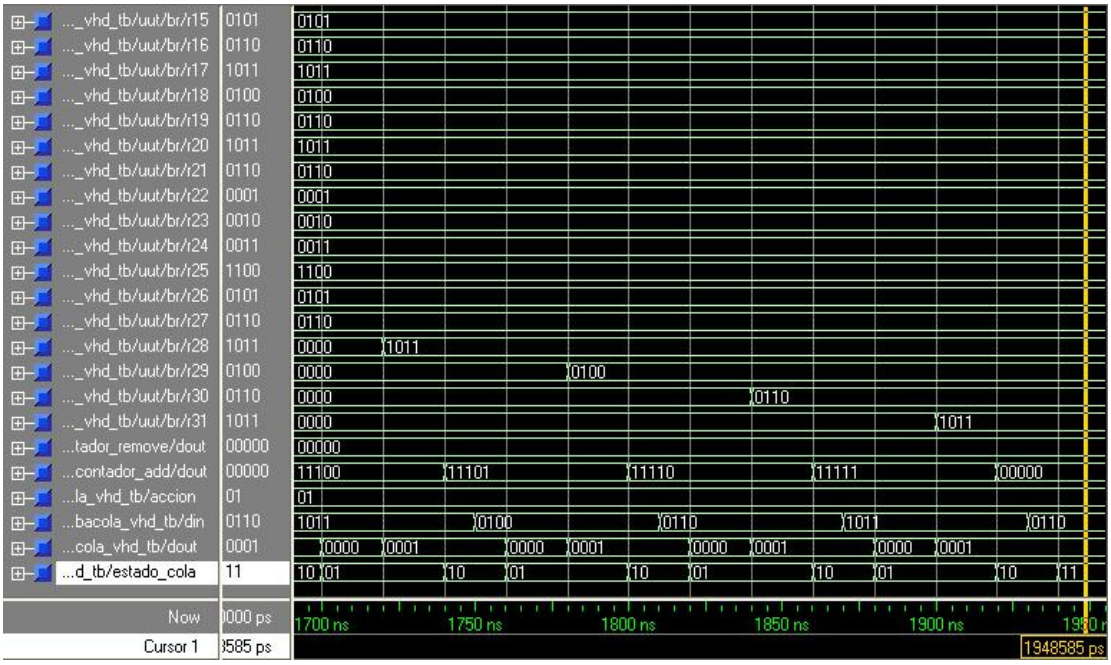


Figura 2.43: Ejemplo 2 de simulación

En la figura sólo vemos la parte final de esta simulación, en la que ya puede verse claramente como la señal estado_cola ha tomado el valor “11” correspondiente a overflow cuando la inserción de una tarea hace que el contador cont_ADD incremente su valor hasta alcanzar el valor de salida del contador cont_REM. A partir de este momento la cola entra en un estado en el que no es posible insertar más tareas, del que no saldrá hasta que no haya sido activada la señal de reset, por lo que es conveniente evitar esta situación no sobrecargando el planificador de tareas.

2.7 Módulo de prueba

2.7.1 Introducción

Este módulo ha sido diseñado con la finalidad de ayudarnos a verificar el correcto funcionamiento del módulo UC_Global (descrito en el capítulo 3). Su funcionamiento imita la transmisión de datos a través del puerto de Ethernet del que dispone la placa.

Por lo tanto, este módulo se limita a enviar los datos de la cabecera de las tareas que se deben guardar en la memoria DDR cuando se solicita.

2.7.2 Descripción de señales

En la figura 2.44 se muestra el módulo que representa el módulo de prueba.

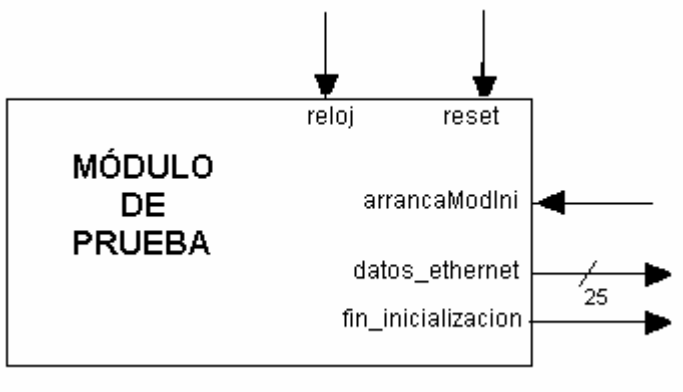


Figura 2.44

Las señales de entrada del módulo son las siguientes:

- reloj: entrada reloj del módulo;
- reset: entrada de reset del módulo;
- arrancaModIni: cuando esta entrada está a alta el módulo empieza a transmitir datos.

Las señales de salida del módulo son:

- datos_ethernet: bus de datos enviados de tamaño de 25 bits;
- fin_inicialización: esta señal se pone a alta cuando el módulo ha terminado de enviar todos los datos.

2.7.3 Funcionamiento

En las siguientes simulaciones se muestra el funcionamiento de este módulo. Cuando recibe la señal de inicio (a los 70 ns. de la simulación de la figura 2.45) el módulo empieza a enviar a través del bus datos_ethernet una serie da datos. Estos datos tienen en los cuatro bits menos significativos una secuencia ascendente desde 0 hasta F (hexadecimal) y en el bit más significativo siempre tiene un 1. Esta configuración de los datos nos servirá más adelante para probar el correcto funcionamiento del planificador.

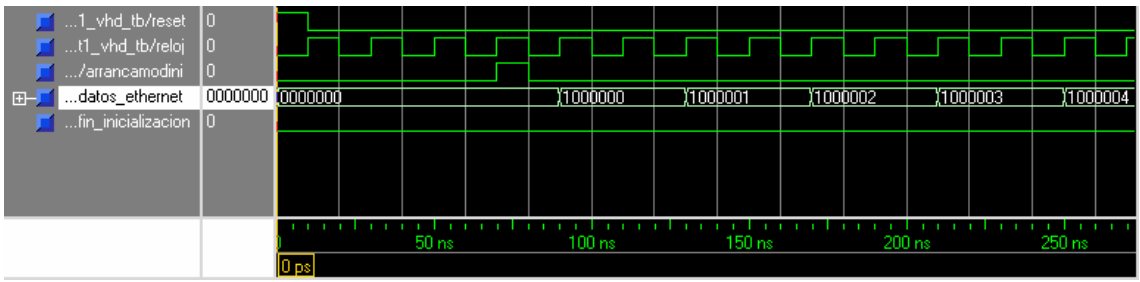


Figura 2.45

En la figura 2.46 se observa cómo, después de haber enviado todos los datos, el módulo pone a alta a los 730 ns .la señal que indica el final de la transmisión de los datos.

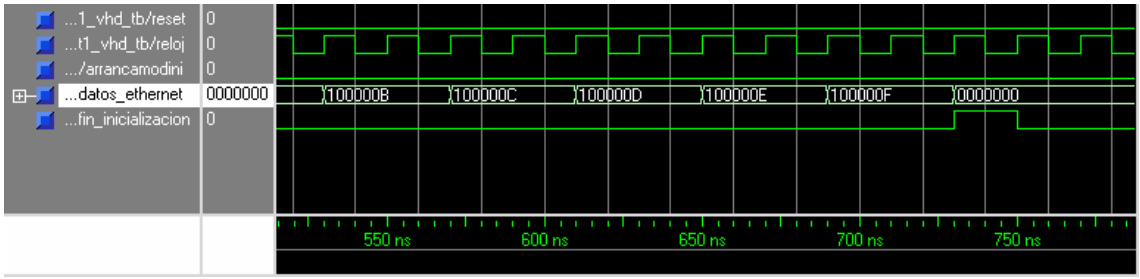


Figura 2.46

Capítulo 3. Interconexión de módulos

3.1 Introducción

En esta sección se va a describir la interconexión entre los diferentes módulos que forman el núcleo del proyecto. A partir de ahora, nos referiremos a esta arquitectura como Controlador Completo,

Hemos de decir, en primer lugar, que en el Controlador Completo no usamos el interfaz de memoria generado por el mig007 debido a los problemas descritos en secciones anteriores, de este modo, favoreceremos la verificación del correcto funcionamiento del planificador de tareas integrado dentro del controlador completo.

El Controlador Completo consiste en la interconexión de los módulos de VGA, Teclado, el Planificador, el Clock_Gen de la FPGA Virtex2Pro y el Módulo de Prueba a través de una unidad de control que gobierna a todos. La unidad de control genera las señales de control de los módulos para lograr que, en función de las órdenes introducidas por el usuario a través del teclado, se planifiquen tareas a ejecutar en la FPGA aprovechando la reconfiguración dinámica que ésta ofrece. Como no hemos aprovechado dicha capacidad de la FPGA, por los problemas ya mencionados del interfaz de memoria, la unidad de control del Controlador Completo muestra en la pantalla el estado en el que debería estar la FPGA, es decir, los espacios de ésta en los que deberían haberse cargado las tareas indicadas por el usuario.

3.2 Descripción de señales

Las únicas señales que salen y entran del Controlador Completo son las correspondientes al teclado y la pantalla.

Hemos de decir, que una vez que el interfaz de memoria funcione correctamente y el módulo de Ethernet esté disponible, habrá que añadir las señales correspondientes a estos dos módulos al Controlador Completo.

El funcionamiento del módulo Ethernet, lo hemos imitado mediante el componente módulo e Prueba, que simula el comportamiento del auténtico Ethernet. Este módulo tendrá la labor de enviar al Controlador Completo la información de las tareas que se deben guardar en la Memoria DDR (el mapa de bits), para posteriormente, según proceda, cargar estos datos de tarea en lugares determinados de la FPGA.

En la figura 3.1 se muestra el módulo que representa el Controlador Completo que hemos empleado en nuestro planificador de tareas:

Figura 3.1: Controlador Completo

Las siguientes entradas son los relojes internos de la placa que utiliza el módulo Clock_Gen para generar las señales de reloj necesarias para la VGA y para nuestra unidad de control:

- mgt_clk_p;
- mgt_clk_n;
- system_clock;
- fpga_system_clock;

Las entradas que recibe que el Contrlador Completo procedentes del teclado son:

- tecladoclk: Reloj de entrada del teclado;
- tecladodata: Línea serie de recepción de datos desde el teclado;

El reset del sistema es entrada con el mismo nombre, que está conectada a switch de la placa.

En cuanto a las salidas, solamente tenemos las correspondientes a la VGA:

- compsynch: Salida de sincronizacion;
- blanking: Blanking del sistema. Nos indica que no leamos el pixel;
- hsync: Señal de sincronismo horizontal;
- vsync: Señal de sincronismo vertical;
- pixelclock_sal: Reloj de cuenta de pixeles;
- videored: Bits del color rojo del sistema, en total son 8 bits;
- videogreen: Bits del color verde del sistema, en total son 8 bits;
- videoblue: Bits del color azul del sistema, en total son 8 bits;

3.3 Implementación

La implementación del Controlador Completo es bastante sencilla, se basa en el uso de los módulos descritos detalladamente en secciones anteriores, los cuales son gobernados por una unidad de control denominada UC_Global. Esta unidad deberá ser ampliada cuando se desee recofigurar dinámicamente la FPGA e integrar en el proyecto los módulos del interfaz de memoria de la memoria DDR y de Ethernet, ya que la unidad de control del Controlador Completo recibe las señales de finalización de tarea mediante un comando específico a través del teclado, en realidad, dicha señal debe ser enviada por la FPGA cuando una tarea previamente cargada desde la DDR en la FPGA haya finalizado de ejecutarse.

A continuación se muestra la ruta de datos del Controlador Completo y la descripción detallada del funcionamiento de la unidad de control que gobierna todo el sistema.

Ruta de datos

En la figura 3.2 se muestra la arquitectura del Controlador Completo, en la que se ven las señales que entran y salen de la FPGA y la interconexión interna de los módulos entre sí.

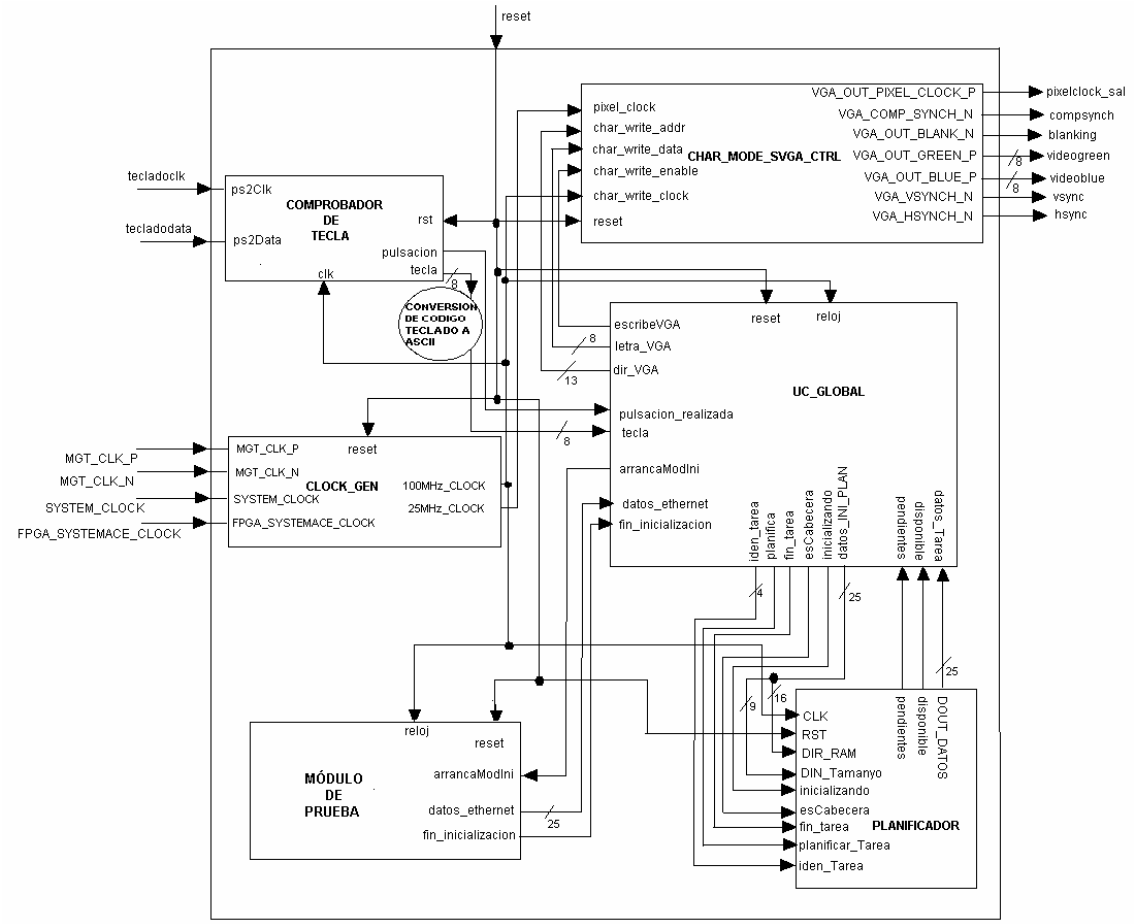


Figura 3.2: Ruta de Datos del Controlador Completo

En el diagrama de bloques se observa que las señales que recibe de la FPGA el Controlador Completo son las del teclado, las señales de reloj interno y el reset del sistema. En cambio, hacia la FPGA sólo salen las señales que gobiernan el monitor.

Hemos de comentar que la salida de datos "tecla" del módulo Comprobador de Tecla es una salida de 8 bits en código teclado, que usamos como entrada de selección en un multiplexor para elegir el código ASCII correspondiente a la tecla pulsada en el teclado. Este código ASCII es el que se envía al módulo UC_Global para identificar la tecla pulsada.

Otro detalle importante es la bifurcación de la señal Datos_IN_Plan del módulo UC_Global. De esta señal se seleccionan los 16 bits menos significativos para especificar en la incialización del planificador la dirección de la memoria DDR donde se alojan los datos de una tarea concreta, y los 9 más significativos para especificar el tamaño de dichos datos, en palabras de memoria, de la misma tarea.

En la secciones que siguen describiremos detalladamente el módulo UC_Global.

3.4 Unidad de Control Global

3.4.1 Ruta de datos

La figura 3.3 muestra la estructura interna de la unidad de control global.

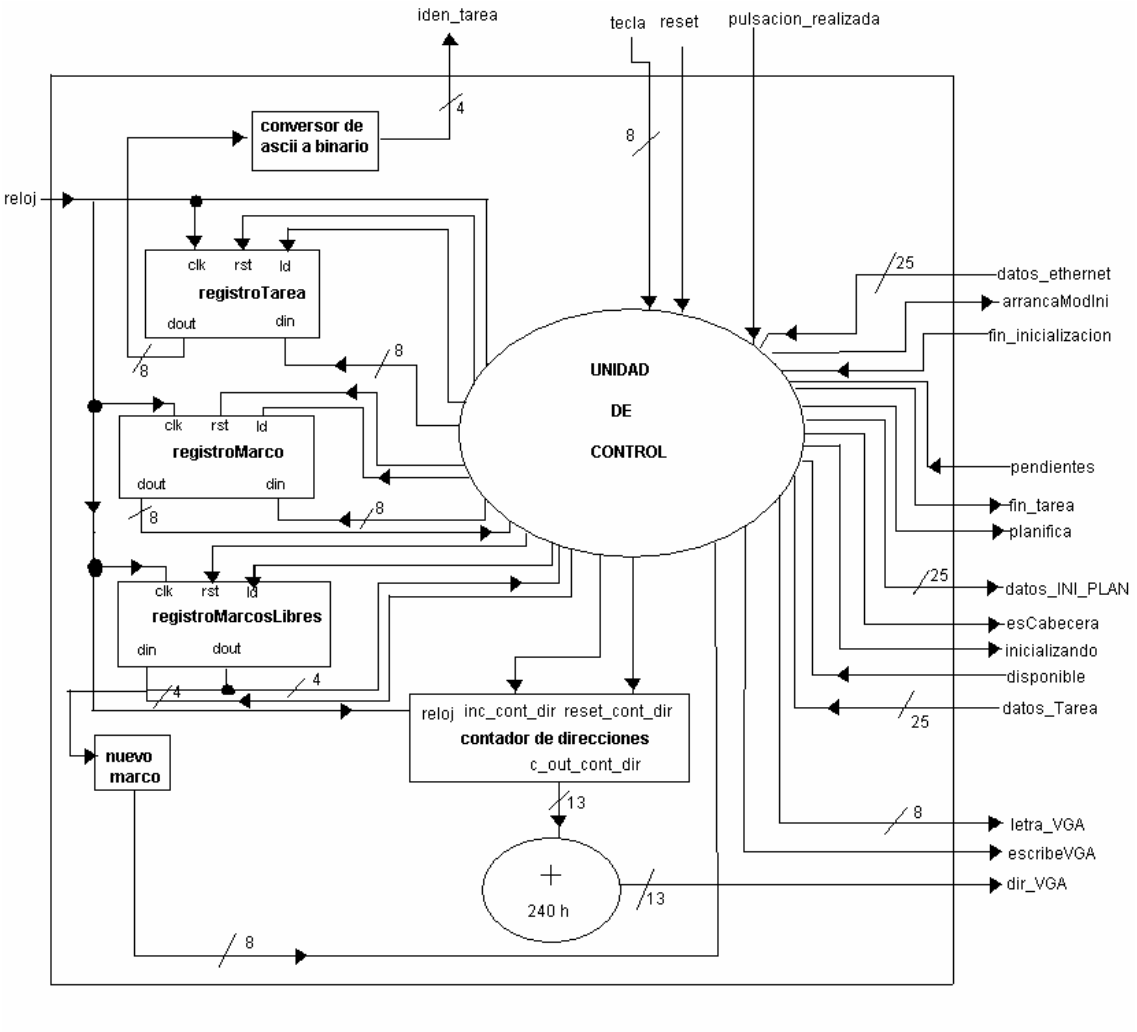


Figura 3.3

En los registros registroMarco y registroTarea se guardan en código ASCII el marco de la FPGA que se libera o se carga y la tarea que se carga en la FPGA respectivamente.

El registro registroMarcosLibres indica que marcos de la FPGA se encuentran libres en cada instante. Así, si el bit menos significativo de este registro se encuentra a uno, sabemos que el marco 1 está ocupado, y si es cero está libre. Para los tres marcos restantes el comportamiento es análogo.

El contador de direcciones nos sirve para determinar la dirección de escritura en la VGA de los caracteres introducidos por teclado. A este valor se le suma 240 (en hexadecimal) para no permitir la escritura en la pantalla en posiciones reservadas para mostrar el estado de los marcos de la FPGA.

El módulo nuevo marco nos da el siguiente marco donde podemos cargar una tarea extraída del planificador en función de los que están libres. El marco libre elegido será el que tenga identificador mayor entre los cuatro posibles.

3.4.2 Máquina de estados

Para que las tablas de generación de salidas y transición de estados sean más comprensibles, hemos cambiando el nombre de los estados según se indica en la tabla 3.1.

Inicio	S0
Pantalla_Inicial1	S1
Pantalla_Inicial2	S2
Pantalla_Inicial3	S3
Pantalla_Inicial4	S4
Pantalla_Inicial5	S5
Pantalla_Inicial6	S6
Pantalla_Inicial7	S7
Pantalla_Inicial8	S8
Inicializar_Plan1	S9
Inicializar_Plan2	S10
Espera_orden	S11
Planifica_sin_espacio_FPGA	S12
Espera_Tarea	S13
Mete_en_cola	S14
Espera1_mete	S15
Espera2_mete	S16
Espera3_mete	S17
Espera4_mete	S18
Finalizar_tarea	S19
Saca_de_cola	S20
Espera_marco	S21
Espera_carga_registro2	S22
Borra_Mi_actualiza_marcos_libres	S23
Espera_disponible	S24
Decidir_marco	S25
Guarda_marco	S26
Planifica_FPGA_Libre	S27
MeteMarcoAmbas	S28
EsperaMarcoAmbas	S29
MeteTareaAmbas	S30
EsperaTareaAmbas	S31
PlanificaFinalizaAmbas	S32
fin_planifica_simultanea	S33
Espera_carga_registro	S34

Tabla 3.1: Renombramiento de estados

En la tabla 3.2, se muestra la transición de estados de la unidad de control en función de las entradas y el estado.

Para más claridad renombramos el nombre de las entradas del modo siguiente:

- reset pasa a ser R;
- fin_inicializacion pasa a ser F;
- pulsacion_realizada pasa a ser P;

- tecla pasa a ser T, que se mostrará como un carácter entre comillas simples, si es X será un valor cualquiera;
- pendientes pasa a ser PD
- disponible pasa a ser D;
- dout_reg_marcos_libres pasa a ser ML;

Estadoactual	R	F	P	T	PD	D	ML	Estado Siguiente
SO	0	X	X	X	X	X	X	S1
S1	0	X	X	X	X	X	X	S2
S2	0	X	X	X	X	X	X	S3
S3	0	X	X	X	X	X	X	S4
S4	0	X	X	X	X	X	X	S5
S5	0	X	X	X	X	X	X	S6
S6	0	X	X	X	X	X	X	S7
S7	0	X	X	X	X	X	X	S8
S8	0	X	X	X	X	X	X	S9
S9	0	X	X	X	X	X	X	S10
S10	0	1	X	X	X	X	X	S11
S10	0	0	X	X	X	X	X	S10
S11	0	X	1	P	X	X	X	S12
S11	0	X	1	F	X	X	X	S19
S11	0	X	1	A	X	X	X	S27
S11	0	X	1	≠A ó ≠P ó ≠F	X	X	X	S11
S11	0	X	0	0	X	X	X	S11
S12	0	X	1	Dígito hexadecimal	X	X	X	S14
S12	0	X	1	No es dígito hexadecimal	X	X	X	S13
S13	0	X	1	Dígito hexadecimal	X	X	X	S14
S13	0	X	1	No es dígito hexadecimal	X	X	X	S13
S13	0	X	0	X	X	X	X	S13
S14	0	X	X	X	X	X	=1111	S34
S14	0	X	X	X	X	X	≠1111	S25
S34	0	X	X	X	X	X	X	S15
S15	0	X	X	X	X	X	X	S16
S16	0	X	X	X	X	X	X	S17
S17	0	X	X	X	X	X	X	S18
S18	0	X	X	X	X	X	X	S11
S25	0	X	X	X	X	X	X	S26
S26	0	X	X	X	X	X	X	S32
S32	0	X	1	'1', '2','3' ó '4'	X	X	X	S28
S32	0	X	1	≠'1', '2','3' ó '4'	X	X	X	S29
S32	0	X	0	X	X	X	X	S29
S29	0	X	1	'1', '2','3' ó '4'	X	X	X	S28
S29	0	X	1	≠'1', '2','3' ó '4'	X	X	X	S31
S29	0	X	0	X	X	X	X	S31
S28	0	X	X	X	X	X	X	S31
S31	0	X	1	Dígito	X	X	X	S30

				hexadecimal				
S31	0	X	0	No es Dígito hexadecimal	X	X	X	S31
S31	0	X	0	X	X	X	X	S31
S30	0	X	X	X	X	X	X	S32
S32	0	X	X	X	X	X	X	S33
S33	0	X	X	X	X	1	X	S11
S33	0	X	X	X	X	0	X	S33
S19	0	X	1	'1', '2','3' ó '4'	X	X	X	S20
S19	0	X	1	≠'1', '2','3' ó '4'	X	X	X	S21
S19	0	X	0	X	X	X	X	S21
S21	0	X	1	'1', '2','3' ó '4'	X	X	X	S20
S21	0	X	1	≠'1', '2','3' ó '4'	X	X	X	S21
S21	0	X	0	X	X	X	X	S21
S20	X	X	X	X	X	X	X	S22
S22	X	X	X	X	1	X	X	S24
S22	X	X	X	X	0	X	X	S23
S24	X	X	X	X	1	X	X	S11
S24	X	X	X	X	0	X	X	S24
S23	X	X	X	X	X	X	X	S11
S27	X	X	1	'1', '2','3' ó '4'	X	X	X	S28
S27	X	X	1	≠'1', '2','3' ó '4'	X	X	X	S29
S27	X	X	0	X	X	X	X	S29

Tabla 3.2: Transición de estados del módulo UC_Global

En la tabla 3.3, se muestra la generación de las salidas del módulo UC_Global, con el renombramiento de estados realizado previamente.

Estado	**Datos_Tarea(3 downto 0)**	**Dout_reg_Marco**	**Disponible**	**Datos_Ethernet(24)**	**EscribeVGA**	**Letra_VGA**	**Dir_VGA**	**Planifica**	**Fin_tarea**	**Inicializando**	**EsCabecera**	**Reset_cont_dir**	**ArrancaModIni**	**Inc_cont_dir**	**Reset_reg_tarea**	**Reset_reg_marco**	**Carga_reg_tarea**	**Carga_reg_marco**	**Din_reg_marcos_libres**	**Carga_reg_marcos_libres**	**Reset_reg_marcos_libres**
S0	X	X	X	X	0	X	X	0	0	0	0	1	0	0	1	1	0	0	X	0	1
S1	X	X	X	X	1	'M'	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0
S2	X	X	X	X	1	'1'	1	0	0	0	0	0	0	0	0	0	0	0	X	0	0
S3	X	X	X	X	1	'M'	A	0	0	0	0	0	0	0	0	0	0	0	X	0	0
S4	X	X	X	X	1	'2'	B	0	0	0	0	0	0	0	0	0	0	0	X	0	0
S5	X	X	X	X	1	'M'	14	0	0	0	0	0	0	0	0	0	0	0	X	0	0
S6	X	X	X	X	1	'3'	15	0	0	0	0	0	0	0	0	0	0	0	X	0	0
S7	X	X	X	X	1	'M'	1E	0	0	0	0	0	0	0	0	0	0	0	X	0	0
S8	X	X	X	X	1	'4'	1F	0	0	0	0	0	0	0	0	0	0	0	X	0	0
S9	X	X	X	X	0	X	X	0	0	0	0	0	1	0	0	0	0	0	X	0	0

S10	X	X	X	1	0	X	X	0	0	1	1	0	0	0	0	0	0	0	X	0	0
S10	X	X	X	0	0	X	X	0	0	1	0	0	0	0	0	0	0	0	X	0	0
S11	X	X	X	X	0	X	X	0	0	0	0	0	0	0	0	0	0	0	X	0	0
S12	X	X	X	X	1	'P'	**(1)**	0	0	0	0	0	0	1	0	0	0	0	X	0	0
S13	X	X	X	X	0	X	X	0	0	0	0	0	0	0	0	0	0	0	X	0	0
S14	X	X	X	X	1	Tecla	**(1)**	0	0	0	0	0	0	1	0	0	1	0	X	0	0
S15	X	X	X	X	0	X	X	1	0	0	0	0	0	0	0	0	0	0	X	0	0
S16	X	X	X	X	0	X	X	0	0	0	0	0	0	0	0	0	0	0	X	0	0
S17	X	X	X	X	0	X	X	0	0	0	0	0	0	0	0	0	0	0	X	0	0
S18	X	X	X	X	0	X	X	0	0	0	0	0	0	0	0	0	0	0	X	0	0
S19	X	X	X	X	1	'F'	**(1)**	0	0	0	0	0	0	1	0	0	0	0	X	0	0
S20	X	X	X	X	1	Tecla	**(1)**	0	0	0	0	0	0	0	0	0	0	1	X	0	0
S21	X	X	X	X	0	X	X	0	0	0	0	0	0	0	0	0	0	0	X	0	0
S22	X	X	X	X	0	X	X	0	1	0	0	0	0	0	0	0	0	0	X	0	0
S23	X	X	X	X	1	'V'	**(2)**	0	0	0	0	0	0	0	0	0	0	0	**(3)**	1	0
S24	**(4)**	**(2)**	1	X	0	**(4)**	**(2)**	0	0	0	0	0	0	0	0	0	0	0	X	0	0
S24	X	X	0	X	0	X	X	0	0	0	0	0	0	0	0	0	0	0	X	0	0
S25	X	X	X	X	0	**(5)**	X	0	0	0	0	0	0	0	0	0	0	1	X	0	0
S26	X	X	X	X	0	X	X	0	0	0	0	0	0	0	0	0	0	0	**(6)**	1	0
S27	X	X	X	X	1	'A'	**(1)**	0	0	0	0	0	0	1	0	0	0	0	0	0	0
S30	X	X	X	X	1	Tecla	**(1)**	0	0	0	0	0	0	1	0	0	0	1	0	0	0
S29	X	X	X	X	0	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S30	X	X	X	X	1	Tecla	**(1)**	0	0	0	0	0	0	1	0	0	1	0	0	0	0
S31	X	X	X	X	0	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S32	X	X	X	X	0	X	X	1	1	0	0	0	0	0	0	0	0	0	0	0	0
S33	**(4)**	**(2)**	1	X	1	**(4)**	**(2)**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S33	X	X	0	X	0	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 3.3: Generación de salidas del módulo UC_Global

Notas:

(1) ***Escribimos en una posición de memoria permitida para órdenes de usuario, es decir:***
240h + c_out_cont_dir

(2) ***En función del marco que esté quedando libre, escribimos en un lugar u otro de la pantalla la letra que corresponda, 'V' (marco vacío) o el identificador de la tarea cargada en el marco. El número de marco que estamos liberando o cargando esta guardando en el registro registroMarco.***

(3) ***Actualizamos los marcos que están libres guardando en el registro registroMarcosLibres la nueva ocupación de los marcos de las FPGA.***

(4) ***Escribimos en la pantalla el carácter correspondiente al identificador de tarea de la que ha sido recuperada del planificador. Este identificador se encuentra en los cuatro bits menos significativos de datos_Tarea. Está en estos bits por la incialización que hemos propuesto para verificar el funcionamiento del planificador, pero como norma general no, ya que en realidad éstos son los cuatro bits menos significativos del campo dirección de DDR de la tarea recuperada.***

(5) ***Cargamos en el registro que guarda el marco con el que estamos operando nuevo_Marco_ASCII, que es el marco donde corresponde guardar una tarea en función de los que están libres.***

(6) ***Actualizamos el registro registroMarcosLibres con el valor adecuado de los marcos libres tras haber liberado o cargado un marco.***

Las primeras cinco columnas son las entradas que desencadenan una transición de estados, y el resto son las salidas que genera la máquina de estados. Hay salidas que no aparecen en la tabla ya que se generan directamente al estar conectadas a la salida de los registros internos del módulo. Estas salidas son:

- iden_tarea: es el identificador de tarea en binario que se envía al planificador cuando se desea planificar una nueva tarea. Se trata del resultado de la conversión del código ASCII que se encuentra guardado en el registro registroTarea.

- datos_INI_PLAN: son los datos particulares de cada tarea que se envían al planificador durante su inicialización. Siempre son los mismos que recibe del Módulo de Prueba. Solamente son válidos para el Planificador cuando éste se está inicializando y si se trata de una cabecera.

- aux_tecla (señal interna): es la entrada de datos de los registros registroMarco y registroTarea que contienen el marco y la tarea sobre los que se quiere operar. Esta entrada es nuevo_Marco_ASCII que es a su vez el marco de la FPGA donde se desea liberar o guardar una tarea en función de los que había libres.

3.5 Funcionamiento

En esta sección vamos a describir el funcionamiento de la unidad de control para una secuencia de órdenes de usuario. La secuencia de órdenes será la siguiente:

P3 PF PA P8 PB F2 F1 A33

El significado de esta secuencia es el siguiente:

- Planificar tarea 3;
- Planificar tarea F;
- Planificar tarea A;
- Planificar tarea 8;
- Planificar tarea B;
- Envía la señal de finalización de tarea del marco 2;
- Envía la señal de finalización de tarea del marco 1;
- Envía la señal de finalización de tarea del marco 3 y planifica la tarea 3;

En la simulación de la figura 3.4 se observa que lo primero que hace a unidad de control es escribir en la VGA la plantilla donde se irán mostrando las tareas que ocupan cada marco de la FPGA.

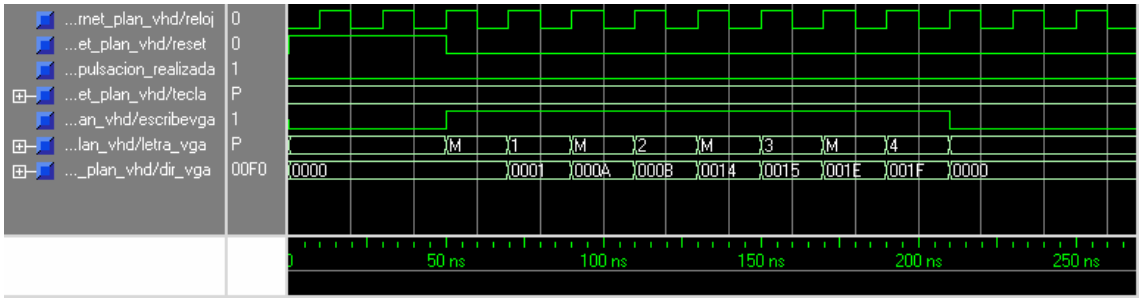


Figura 3.4

En la figura 3.5 se muestra como a partir de los 300 ns. Se inicializan los datos particulares de cada tarea (tamaño y dirección de la RAM) en el planificador. Se han visualizado los registros que contienen dicha información.

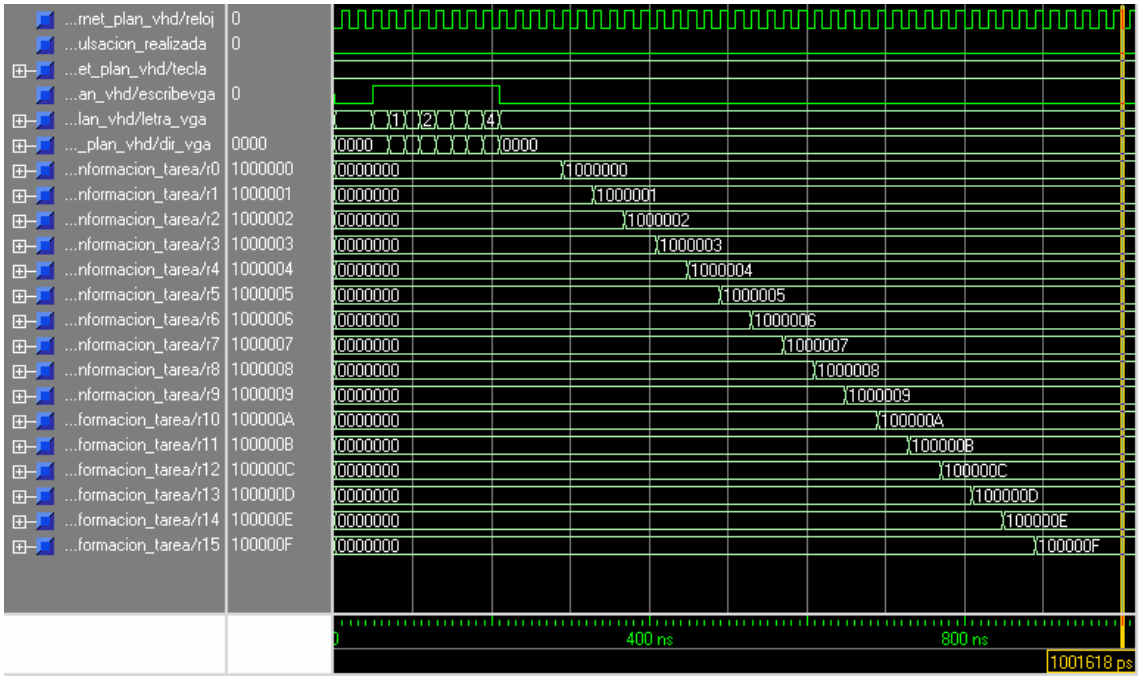


Figura 3.5

En las figuras 3.6 y 3.7 se observa como reacciona la unidad de control ante la pulsación de la tecla 'P'. Esta orden hace que la unidad de control entre en el modo de planificación de una nueva tarea. La unidad de control se queda esperando hasta que se introduzca un identificador de tarea válido, cualquiera desde 0 hasta F (en hexadecimal). Una vez sucede esto, la unidad de control decide qué hacer en función del estado del planificador. Si el éste no tiene tareas pendientes por ejecutar simplemente carga en uno de los marcos que estén libres la tarea indicada. Si no hay marcos libres en la FPGA mete en la cola del planificador la tarea que ha introducido el usuario. En las dos simulaciones que siguen, se observa que debido a que la cola del planificador estaba vacía simplemente se carga en los marcos correspondientes las

tareas 3, F, A y 8. Pero la tarea B no puede ser cargada en ningún marco dado que no queda ninguno libre, por tanto se introduce en la cola del planificador (Simulación 4 entorno a los 2050 ns.)

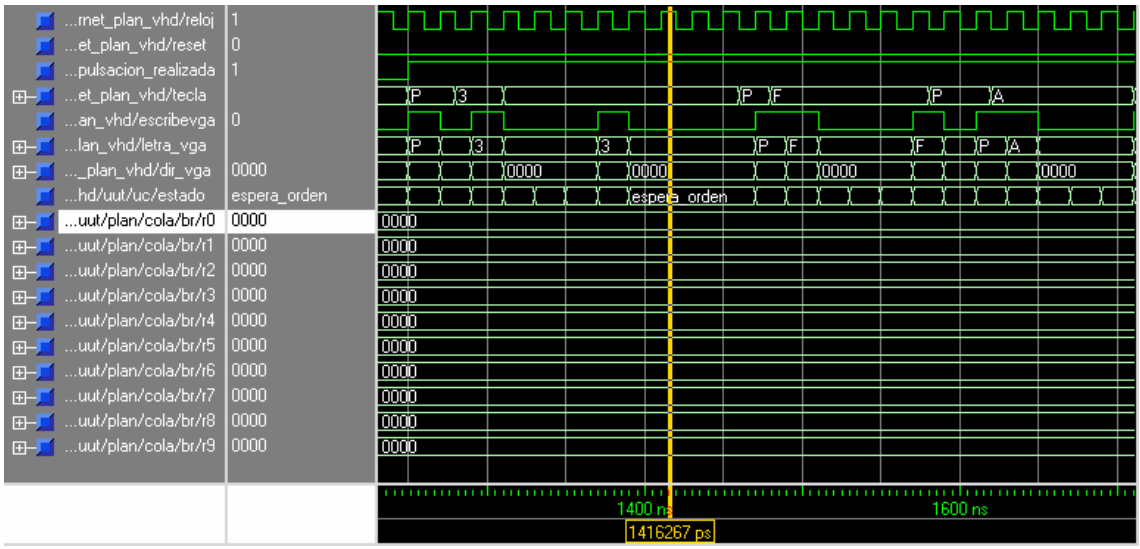


Figura 3.6

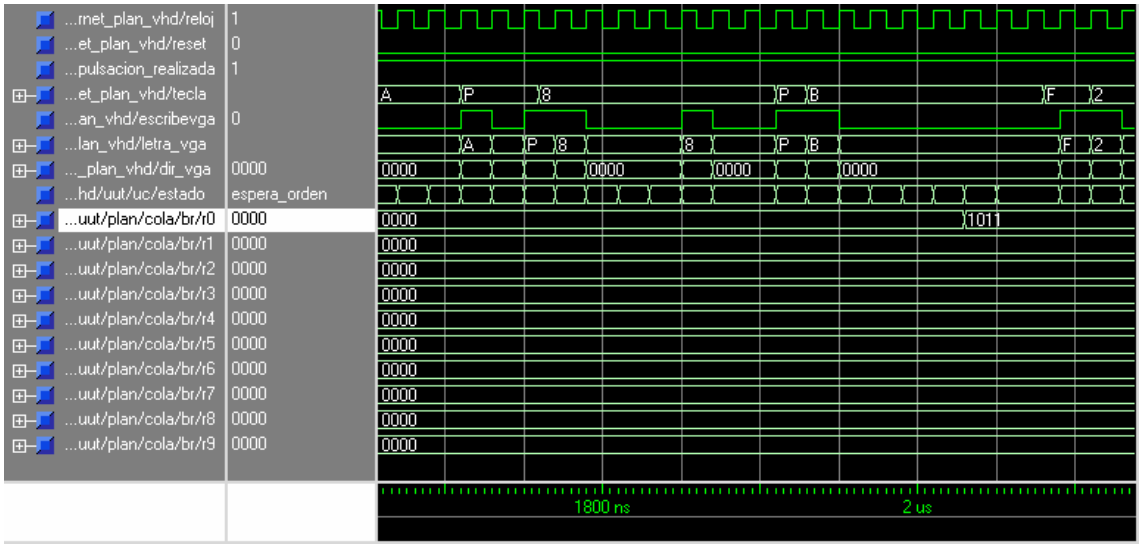


Figura 3.7

En la figura 3.8 se muestra el funcionamiento de la operación finalizar tarea, que se desencadena cuando el usuario pulsa la tecla 'F'. Esta operación consiste en que cuando el usuario entra en este modo, la unidad de control espera a que le indique el usuario el marco de la tarea que ha finalizado. Al indicarle qué marco contenía la tarea que ha finalizado, la unidad de control decide, en el caso de que hubiera tareas pendientes en la cola, cargar una nueva tarea en el marco que ha quedado libre, si no había tareas pendientes, simplemente actualiza el registro que lleva la cuenta de qué marcos quedan libres y escribe una 'V' en la pantalla en el lugar correspondiente al marco liberado para indicar que se encuentra vacío. La primera

alternativa se observa a los 2100 ns. Entramos en el modo finalizar tarea e indicamos que ha finalizado la tarea cargada en el marco 2, como teníamos en la cola la tarea B, ésta es expulsada de la cola y se carga en el marco que ha quedado libre(entorno a los 2200 ns). A continuación volvemos a decirle a la unidad de control que ha finalizado otra tarea, pero esta vez la que estaba en el marco 1. Como no quedan tareas pendientes en la cola simplemente libera el marco y escribe un 'V 'en el lugar correspondiente.

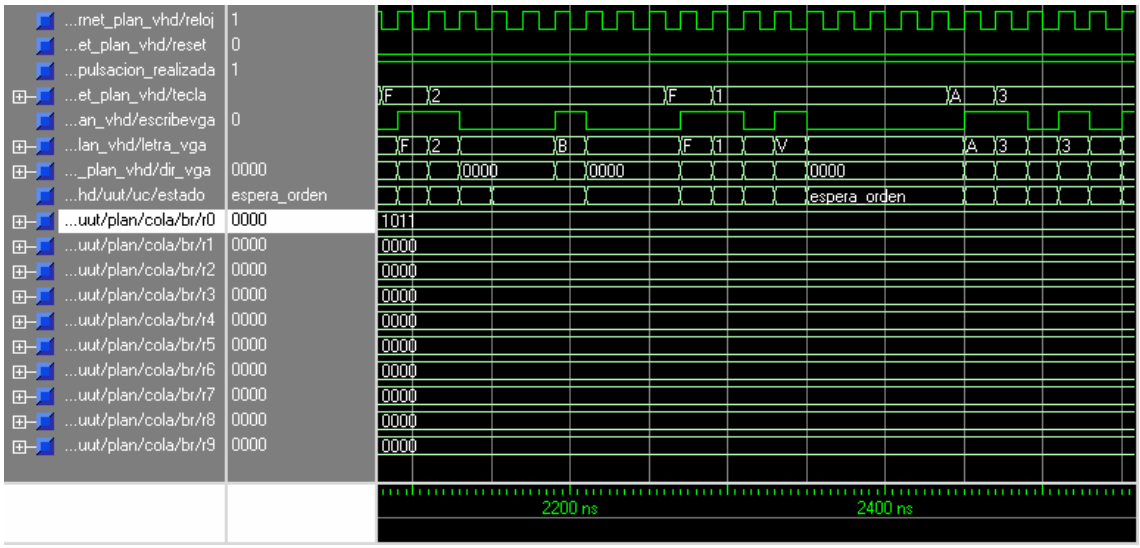


Figura 3.8

En la figura 3.9 se muestra el funcionamiento del tercer modo el que puede actuar el usuario. Se trata del modo de finalización de una tarea y planificación de otra simultáneamente. Este modo tiene sentido dado que puede suceder que una tarea cargada en un marco de FPGA finalice en el mismo instante en que se solicita la ejecución de otra. A este modo se accede pulsando la tecla 'A' y continuación se introduce el marco de la tarea que finalizó y la nueva tarea que se desea ejecutar. En este caso, gracias al planificador, simplemente debemos mandarle a éste la señal de que queremos introducir una nueva tarea en la cola y a la vez queremos sacar la que lleve más tiempo en la cola y esperar a que éste nos dé la siguiente tarea a ejecutar. Si la cola estaba vacía nos da la misma tarea que hemos querido introducir en la cola, si no, nos da la esté en primer lugar de la cola. En el ejemplo de la figura 3.9 hemos dicho a la unidad de control que ha finalizado la tarea del marco 3 y que queremos planificar la tarea 3. Como la cola estaba vacía nos da la misma tarea que intentamos almacenar en la ella.

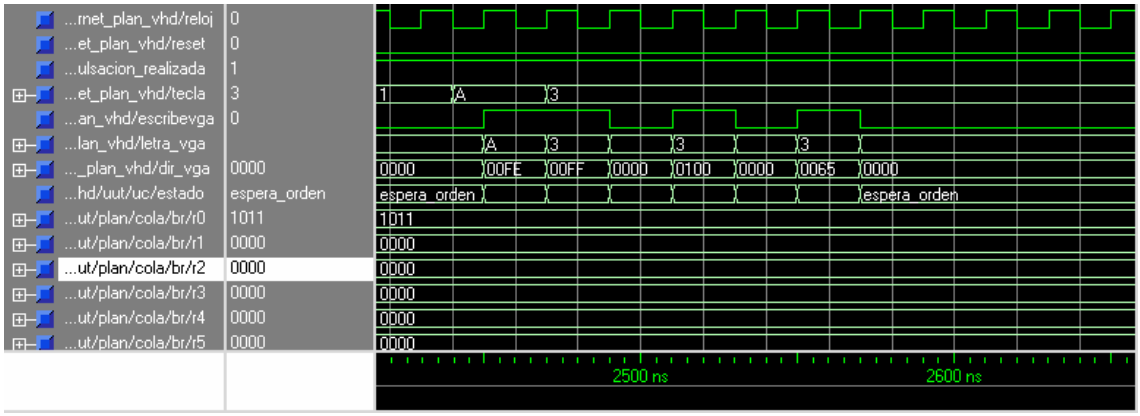

...rnet_plan_vhd/reloj
...et_plan_vhd/reset
...ulsacion_realizada
...et_plan_vhd/tecla
...an_vhd/escribevga
...lan_vhd/letra_vga
..._plan_vhd/dir_vga
...hd/uut/uc/estado
...ut/plan/cola/br/r0
...ut/plan/cola/br/r1
...ut/plan/cola/br/r2
...ut/plan/cola/br/r3
...ut/plan/cola/br/r4
...ut/plan/cola/br/r5
0
0
1
3
0
0000
espera_orden
1011
0000
0000
0000
0000
0000
1
A
3
A
3
3
3
0000
00FE
00FF
0000
0100
0000
0065
0000
espera_orden
1011
0000
0000
0000
0000
0000
2500 ns
2600 ns

Figura 3.9

Capítulo 4. Conclusiones

En este capítulo haremos una revisión de los conceptos más importantes que hemos manejado durante el desarrollo del proyecto y expondremos las dificultades principales con las que nos hemos encontrado.

En primer lugar destacaremos la importancia de las FPGAs cuando hablamos de sistemas digitales. Aunque está claro que su funcionalidad tiene ciertos límites, su flexibilidad y capacidad las convierte en una buena opción para desarrollar cualquier sistema que queramos en un chip.

Estando más acostumbrados a concebir diseños orientados a la programación de software, el primer problema con el que nos hemos encontrado ha sido nuestra propia inexperiencia en el diseño de circuitos digitales. Es indudable que este tipo de diseños debe afrontarse desde otra perspectiva diferente para evitar los problemas que nos hemos encontrado a lo largo del desarrollo de este proyecto.

Una dificultad derivada del uso de FPGAs, inevitable en cierto modo, es la de poder predecir con exactitud la cantidad de recursos que va a emplear nuestro diseño. En nuestro caso el impacto de este problema ha sido decisivo, ya que forzó un cambio en la tecnología fundamental que empleamos en el proyecto (la FPGA), con los cambios que a su vez supuso en el diseño de módulos ya implementados.

El aprendizaje de las tecnologías que eran nuevas para nosotros ha sido rápido y, en general, no nos ha planteado grandes problemas. Sin embargo sí ha supuesto un problema manejar el MIG 007, el generador de interfaces al que se dedica la sección 4 del capítulo 1. La escasa información que se dispone sobre su manejo y la complicación que supone verificar la corrección del interfaz que suministra nos han impedido llegar a generar correctamente un interfaz que interactuara de manera adecuada con el controlador que diseñamos nosotros.

También destacaremos la gran utilidad que ha tenido el analizador de señales empleado en la depuración de nuestro proyecto. Aunque, como hemos comentado, no se consiguiera con ello el buen funcionamiento del interfaz de memoria para la segunda placa utilizada, sí pudimos comprobar los valores de señales a los que no habríamos podido acceder de otra manera, lo cual permitiría acotar rápida y eficazmente el problema en una situación de mal funcionamiento.

Si bien el objetivo inicial del proyecto no se ha cumplido en su totalidad, sí se han hecho una serie de avances que serán de utilidad de cara a una continuación en su desarrollo. El mejor ejemplo de esto es el módulo planificador de tareas: Gracias a la modularidad con la que se planteó el diseño desde el principio podemos decir que ya se cuenta con un módulo capaz de recibir solicitudes de ejecución de tareas desde el exterior a través de un módulo de teclado, de gestionar dichas peticiones y mostrar el estado general del planificador por pantalla, sin que la forma en que ha sido diseñado este módulo influya en cómo deban diseñarse las posteriores ampliaciones del proyecto.

Bibliografía

ACHA, S. y Otros
Electrónica Digital: Lógica Digital Integrada. Teoría, problemas y simulación
RA-MA 2006

Lluis Terés, Yago Torroja, Serafín Olcoz, Eugenio Villar
VHDL: Lenguaje estándar de diseño electrónico
McGraw Hill, 1998

Stephen Brown, Jonathan Rose
Architecture of FPGAs and CPLDs: A tutorial

Y las siguientes páginas web:

Kingston. Empresa fabricante de memorias.
http://www.kingston.com/latinoamerica/newtech/ddr.asp

Micron. Empresa fabricante de chips.
http://www.micron.com/

Xilinx. Empresa fabricante de FPGAs y otros dispositivos.
http://www.xilinx.com/

Anexo. Código fuente

Autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y el prototipo desarrollado.

Javier Fernández Martín

Emilia Mª Rodríguez Alonso

Patricia Ruiz González