Universidad de Valladolid

Escuela Técnica Superior
de Ingenieros de Telecomunicación

Proyecto Fin de Carrera

Ingeniero en Electrónica

Diseño de un microprocesador RISC e implementación mediante la FPGA Xilinx Spartan II

Autores:
Bernardo Carretero Rivera
Fernando Espinosa García

Tutores:
Jesús M. Hernández Mangas
Ruth Pinacho Gómez

Valladolid, 12 de abril de 2005

Información del proyecto

Título:	**Diseño de un microprocesador RISC e implementación mediante la FPGA Xilinx Spartan II**
Autores:	**Bernardo Carretero Rivera** **Fernando Espinosa García**
Tutor:	**Jesús M. Hernández Mangas** **Ruth Pinacho Gómez**
Departamento:	**Electricidad y Electrónica**

Tribunal

Presidente:	**Salvador Dueñas Carazo**
Vocal:	**Jesús Arias Álvarez**
Secretario:	**Jesús M. Hernández Mangas**

Fecha de Lectura:

Calificación:

Resumen del proyecto

En la electrónica moderna, el campo de los dispositivos lógicos programables y, más aún, re-programables, está adquiriendo cada vez más importancia, debido a las ventajas que presenta respecto a otro tipo de dispositivos de *un sólo uso*. Los dispositivos re-programables tienen dos utilidades principales. La primera de ellas es que, dada su característica de *re-programabilidad*, son el medio perfecto para realizar prototipos de diseños que, posteriormente, se implementarán en otros dispositivos definitivos. Además, permiten la implementación de esos diseños definitivos sobre ellos mismos, como si fueran dispositivos dedicados, exclusivamente, a esa función... sin olvidar nunca la posibilidad de modificar ese diseño en cualquier momento.

El objetivo de este proyecto es diseñar un microprocesador RISC de propósito general y embeberlo en uno de estos dispositivos lógicos programables, concretamente, la FPGA Xilinx Spartan II. Además de cumplir las especificaciones propias de un microprocesador RISC, debe poder utilizar los periféricos de que dispone la placa que contiene la FPGA donde está implementado el procesador.

Palabras clave

FPGA, Spartan II, Tarjeta XSA-50, Memoria SDRAM, Memoria Flash, Xilinx ISE, ModelSim, XSTools, Microprocesador RISC.

Abstract

In the world of modern electronic, the field of the programmable logic devices and, even more, the reprogrammable ones, is getting more and more importance because of the advantages in consideration of those others *only one use devices*. Reprogrammable devices have two main utilities. The first one is, thanks to *re-programmability*, they are the perfect way to make prototypes of designs that, afterwards, will be implemented into other definitive devices. Moreover, they allow the implementation of those definitive designs into themselves, as they were dedicated devices, without forgetting the possibility of changing that design at any time.

The goal of this project is to design a general purpose RISC microprocessor and to embed it into one reprogrammable logic devices, concretely into the Xilinx Spartan II FPGA. In addition of verifying the RISC microprocessor specifications, the use of the disposable peripherals of the board where the FPGA is included must be allowed.

Key words

FPGA, Spartan II, XSA-50 Board, SDRAM Memory, Flash Memory, Xilinx ISE, ModelSim, XSTools, Risc Microprocessor.

A nuestros padres y hermanos

y a todos los que han estado a nuestro lado
durante todos estos años.

Agradecimientos

Este Proyecto Fin de Carrera está dedicado de una forma muy especial a mis padres ***Bernardo*** *y* ***María del Pilar****, y hermanos* ***José Luis****,* ***María Concepción*** *y* ***Félix****, que lo son todo para mi, sin olvidarme por supuesto de* ***Cuki*** *que día a día nos alegra la vida a todos con su forma de ser. Y no me gustaría acabar este primer párrafo sin dejar de mencionar a* ***Manfryyyyyyyyy*** *y al borrico que han conseguido que los días sean más fáciles para* ***Sid****.*

A ***Emma****. BLGJNZH*

A los orcos:

- *A Edu, Lui, Vanderlei, Emerson Leao, Rodillas por seguir siendo una de las pocas personas sin limitaciones y prolegómenos.*
- *A Isaac y Isabel, por guiar los designios del grupo con su sabiduría y saber hacer. No nos defraudéis en ese cumpleaños...*
- *A Jaime, Papi, Owen, Rechaces, Casillas, Tobillos, por haber creado una familia numerosa tan bien avenida.*
- *A Jorge, Coke, ese peazo soldado universal galáctico.*
- *A Zenón, Copulito que volvió de las galias a territorio hostil para enseñarnos todo lo aprendido.*

A los titos:

- *A Dani, para que acabe pronto, encuentre trabajo en la NASA y nos lleve para allí al resto de los titos.*
- *A Oscar, Tobilloooooooooooooos, recuerda que tienes una apuesta pendiente conmigo, Comillas 2005... Iuiuiuiuiuiuiuiiiu!!!!*
- *A Raul, el Pollito, que sepas que solo te nombro por ser hermano de Oscar, porque nos tienes abandonaos... bueno o porque a lo mejor me tienes que dejar el portátil.*
- *A Richi, Calzzone, Lechone, Vaporetto...*

A O Mais Grandes:

- *A* ***Alberto****, Taussen, Calcetos, Moby, la Coneja, Hacienda, Megane, Jazmine y al Bakalao... Peón negro a Sebring Plateao...*

- *A **David**, Tolón, Bolinchón, Kabeza Bolo, Rodolfooooooooooo, serás capaz de irte a ver al Longanizas sin mi... no esperaba menos... Peón negro a Cabezudo Eclíptico...*
- *A **Fernando**, Púas, Lumis, Bloomies, por ser mi compi.*
- *A **Rodolfooooooooo**, iuiiuiuiuiuiuiuiui ig fentraugue an dig.*

A la gente del Titamar, en especial Jose A, Edu, y a los entrenadores, para que me saquen de titular. Esperadme que el próximo año regreso con clavícula nueva.

A todos/as aquellos/as que ahora no me vienen a la mente pero que me han hecho sonreír a lo largo de mi vida...

A mi clavícula...

VIVA EL BAKALAO!!!!! GALICIA VOLVEREMOS!!!!!

Bernardo Carretero Rivera

Además de dedicar este proyecto a mis padres, quiero agradecerles públicamente todas las cosas que me han dado, en forma de oportunidades, enseñanzas y orientaciones. Pero, sobre todo, quiero agradecerles todo el esfuerzo que han hecho durante 25 años, durante mis 25 años, para que yo tuviese más fácil cumplir mis sueños e ilusiones. Y, cuanto más mayor me hago, más cuenta me doy de que eso de hacer cumplir sueños e ilusiones no es nada fácil y tiene mucho mérito. ¡¡Muchas gracias, papás!!

También quiero agradecer su ayuda y su dedicación a todos los profesores que he tenido a lo largo de toda mi vida estudiantil, desde EGB hasta la Universidad, sin olvidarme de mis profesores de música e inglés. Todos ellos han puesto su granito de arena que hoy se convierte en una montaña con la presentación de este proyecto. Especialmente, quiero dar las gracias a Jesús M. Hernández Mangas y a Ruth Pinacho Gómez, que nos han dado la oportunidad de realizar este proyecto y que nos han acompañado y ayudado durante este largo año.

Y, por supuesto, la tercera pata del banco de mi vida: mis amigos. A todos, os quiero agradecer vuestra alegría, vuestro ánimo, vuestra ayuda, vuestros consejos, vuestras bromas... en definitiva, vuestra amistad. Han sido... aunque prefiero decir, son, y seguro que serán, tantos los buenos momentos que me habéis regalado, que estos agradecimientos se quedarán muy cortos, pero lo intentaré...

- *A mis compis de colegio... cómplices de tantas risas y bromas...*
- *A Nuria y Fer MJ, mis amigos de música, que tras años de compartir sostenidos y corcheas, ahora compartimos planes y risas.*
- *A mis amigos de la Parroquia de San Juan Bautista. A los mayores, especialmente a Maribel, Roberto y Javi, que siempre me cuidaron, se preocuparon por mi, me trataron con mucho cariño y me ensañaron muchos de los mejores valores que he aprendido. A mi grupo, a Sara, María José, David, Leticia, Begoña, Paco, Rosalía... y a todos los que compartieron mesa y charla conmigo en mi parroquia. A mi grupo de pequeños, a Paco, Fran, Héctor, Pablo, Alba e Izaskun; fue poco el tiempo que compartimos, pero suficiente para sembrar una bonita amistad.*
- *A mis compañeros de la Universidad. A Agustín, a Molino, a Quique, a Pablo, a Judith y a toda la gente de la Politécnica!! Y a mis compañeros de Electrónica!! que, entre todos, habéis sido capaces de convertir una clase en un grupo de amigos que se reunían para aprender... pero también para jugar a fútbol, para ir de cena, o simplemente, para tomar algo... He disfrutado mucho de vosotros estos años.*
- *A mis amigos de internet. A María, Inma, Pilar, Luis, Esther... que hace tiempo que quité el apellido* de internet*, para dejarlo solo, y tanto, en amigos.*
- *A la FIPA, Bendita organización. Al dueño de cortijo, al maldito, a los clones, a mi hermano, al especulador de Nacho, a LC, a AL, a Fco, a Osnofla, a la serie J, a DC, a Tajo... y así hasta los casi 100 amigos que hemos compartido grandes e inolvidables momentos, no solo en torno a un tapete, sino, y sobre todo, en torno a nuestra amistad. Muchas gracias por acogerme en vuestra organización, por tratarme con tanto cariño y enseñarme tantas cosas.*

- *A mi amigo Luis, ejemplo y referencia para mi, que siempre me dio buenos consejos, siempre me ayudó en todo lo que pudo, que ha sido y es mucho, como lo es, por ejemplo, este proyecto.*
- *Por último, a mi compañero de proyecto pero, sobre todo amigo, Bernardo. Muchas gracias por tu trabajo, por tu paciencia, por tu compañerismo, y sobre todo, por tu amistad.*

A todos vosotros, muchas gracias.

Fernando Espinosa García

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción.

El presente documento es la memoria del Proyecto Fin de Carrera **"Diseño de un microprocesador RISC e implementación mediante la FPGA Xilinx Spartan II"** propuesto por los profesores del Departamento de Electrónica de la Escuela Técnica Superior de Ingenieros de Telecomunicación (E.T.S.I.T.) de la Universidad de Valladolid (U.V.A.), **Jesús M. Hernández Mangas** y **Ruth Pinacho Gómez**, y desarrollado por los alumnos de Ingeniería Electrónica **Bernardo Carretero Rivera** y **Fernando Espinosa García**.

1.1. Objetivos del proyecto.

Como se presenta en el título del proyecto, el objetivo final del mismo es el diseño de un *Procesador RISC* para embeberlo en la *FPGA Xilinx Spartan II*. Sin embargo, antes de cumplir este objetivo final, y para poder llegar a él, es necesario cumplir dos objetivos previos:

1. **Familiarizarse con la tarjeta que contiene la FPGA.** Conocer, además de la *FPGA*, los periféricos de que se dispone en la tarjeta, así como sus interfaces de comunicación con el *PC*.

2. **Encontrar y aprender a manejar un software de diseño adecuado.** La *FPGA* es un dispositivo totalmente programable y, para ello, es necesario encontrar un software que permita realizar diseños que, posteriormente, se bajarán a la tarjeta para su implementación práctica.

Una vez cumplidos estos dos objetivos previos, ya conocemos los recursos, tanto hardware como software, de los que disponemos para la realización del proyecto. A partir de este punto, los objetivos inmediatos son:

3. **Realizar el estudio y diseño de un *Procesador RISC.*** El diseño del *Procesador* es totalmente a medida, viniendo definido por la capacidad de la FPGA y la disponibilidad de los periféricos de la tarjeta. La forma de diseño es mediante captura esquemática, aunque se ha utilizado un bloque ya creado mediante lenguaje *VHDL*, que hará la función de controlador de la memoria *SDRAM*.

4. **Descargar el diseño a la tarjeta y probar su funcionamiento.** Con este último objetivo, comprobaremos que se han cumplido todos los objetivos anteriores y, con él, el objetivo final del proyecto.

1.2. Recursos disponibles.

Para la realización de este proyecto, contamos con los siguientes recursos, tanto hardware como software.

1.2.1. Recursos Hardware.

1. Tarjeta *XSA 50*, que contiene:

   - *FPGA Xilinx Spartan II*, modelo XC2S50.
   - *Puerto paralelo* para comunicación con el PC.
   - *CPLD XC9572XL* que gestiona la interface entre el puerto paralelo y el resto de la tarjeta.
   - *Memoria Flash ROM* de 128 Kb.
   - *Memoria SDRAM* de 4 Mb.
   - *Oscilador* de 100 MHz programable para generar la señal de reloj.
   - *Display de 7 segmentos.*
   - *4 DIP switches* accesibles mecánicamente por el usuario.
   - *Puerto PS/2* para la conexión de un ratón o un teclado.
   - *Puerto VGA* para la conexión de un monitor.
   - *Pins libres* que usaremos, a modo de puerto, para entrada/salida de datos.

   Además de la tarjeta *XSA 50*, el lote en que viene también contiene:

   - *Transformador a 9V* para la alimentación de la tarjeta desde la red eléctrica.
   - *Cable paralelo* para comunicación entre el puerto paralelo del PC y el de la tarjeta.
   - *CD-ROM* con el software de instalación (paquete de programas para la comunicación de la tarjeta con el PC).

2. Recursos disponibles en el Laboratorio de proyectos (lab. 21 de la primera planta de la ETSIT):

   - *1 PC (Carpa)* con sistema operativo W-XP, *Procesador* de 350 Mhz, memoria RAM de 160 Mb y disco duro de 4 Gb.
   - *4 Displays de 7 segmentos* con sus respectivos convertidores (hexadecimal a 7 segmentos) 74LS248.

- *Multímetro BLAUSONIC FP-2b.*
- *Inversor integrado 74LS04* y *buffers tri-estado 74LS244.*
- Otros recursos: *LEDs, resistencias, cable, tijeras, pinzas ...*

1.2.2. Recursos Software.

1. Paquete de programas de *Diseño*:

   - **Xilinx Foundation 2.1i.** Es el primer paquete de programas de diseño que probamos, pero tuvimos que descartarlo porque no permite compilar lenguaje VHDL sin licencia. Esta licencia no es gratuita, y nosotros queremos utilizar programas de libre distribución.
   - **Xilinx Integrated Software Environment 6.x.** Este paquete tiene una versión (*WebPACK*) que puede descargarse de forma totalmente gratuita desde la web *www.xilinx.com*, y que permite realizar diseños para la *FPGA Spartan II* y compilar lenguaje VHDL. Nosotros hemos utilizado la versión 6.3, con el Service Pack 2, última versión disponible[1].

2. Programa *Simulador*:

   - **ModelSim XE 5.8x.** También, desde la web de xilinx puede descargarse el simulador Modelsim XE de forma totalmente gratuita, sin más que cumplir el requisito de registrarse en la web. Hemos utilizado la versión de evaluación 5.8c.

3. Paquete de programas para el *Interface con la tarjeta*:

   - **XSTools 4.0.** Este paquete de programas viene en un CD ROM incluido en el lote al comprar la Tarjeta *XSA 50* a la empresa *XESS Corporation.*

Explicamos la obtención e instalación de todos estos programas en el Apéndice A.

1.3. Fases del desarrollo del proyecto.

Como hemos señalado en el apartado 1.1, los objetivos del proyecto deben cumplirse en dos etapas consecutivas. Por tanto, estas dos etapas definen las fases en las que se desarrolla el proyecto.

[1] A la finalización de este proyecto, ya está disponible la versión 3 del Service Pack

FASE I. FASE DE ESTUDIO.

Esta primera fase consta de dos sub-fases que se llevan a cabo de forma simultánea.

I. A. **Estudio de los recursos hardware disponibles.**

1) Estudio de la Tarjeta *XSA 50*, de *Xess Corporation*, y de sus interfaces de comunicación.
2) Estudio de los recursos disponibles en el Laboratorio de proyectos (lab. 21 de la primera planta de la ETSIT).

I. B. **Estudio de los recursos software disponibles.**

1) Estudio del paquete de programas de diseño: *Xilinx Integrated Software Environment WebPACK 6.3*.
2) Estudio del programa Simulador: *ModelSim 5.8c*.
3) Estudio del paquete de programas para el interface con la tarjeta: *XSTools 4.0*.

Una vez que se han completado las dos sub-fases de esta fase, se puede continuar con la siguiente fase.

FASE II. FASE DE DISEÑO.

En esta segunda fase, se realiza el diseño del *Procesador RISC*. Este diseño será jerárquico y estará dividido en dos sub-fases. Todos los puntos de esta fase se realizan de forma secuencial.

II. A. **Sub-fase Dirección descendente (sobre papel).**

- Creación de un set de instrucciones propio que cumpla los requisitos que queremos para nuestro *Procesador*.
- Realización de un esquema de diseño mediante bloques jerárquicos, que cumpla las especificaciones del set de instrucciones, yendo desde los bloques superiores del diseño hacia los bloques inferiores del mismo.

II. B. **Sub-fase Dirección ascendente (sobre PC).**

- A partir de los esquemas obtenidos en la fase anterior, realización del diseño de los bloques inferiores, sobre PC y simulación para comprobar que el funcionamiento es correcto en cada caso.
- Subiendo al nivel superior, unión de los bloques creados en el nivel inferior y, de nuevo, a simulación y comprobación de funcionamiento.
- Repetición del punto anterior hasta el nivel más alto.
- En los niveles superiores, el funcionamiento se comprueba no solo con simulaciones sino también con la descarga los bloques a la tarjeta.

Finalizada esta fase, se pasa a la última.

FASE III. FASE DE CARGA Y COMPROBACIÓN.
En esta fase, se carga de la FPGA con el diseño definitivo del *Procesador*. Este punto es el último paso de la última iteración de la fase anterior.

III. A. **Sub-fase de Pruebas.**

- Creación de varios *Bancos de Pruebas* que, en definitiva, serán varios programas.
- Carga de estos programas en la memoria de instrucciones del *Procesador*.
- Comprobación de los resultados obtenidos con los resultados esperados.

1.4. Estructura de la memoria del proyecto.

Para terminar el capítulo de introducción, presentamos un breve adelanto del contenido de los siguientes capítulos de este documento, con el fin de facilitar su lectura y comprensión, así como la búsqueda de información en el mismo.

El capítulo 1 tiene el nombre de **Introducción**. En este apartado, realizamos la presentación del proyecto: sus objetivos, los recursos (*hardware* y *software*) de que disponemos para la realización del mismo y las fases de su desarrollo. Por último, presentamos un breve resumen del contenido de todos los capítulos de la memoria, que es el apartado actual.

El capítulo 2 tiene el nombre de **Dispositivos Lógicos Programables**. Explicamos qué son los *PLDs*, sus características y tipos principales, centrándonos más en las *FPGAs*. Después, presentamos la familia *FPGA Xilinx Spartan II*, comentando las cualidades de esa familia, su arquitectura y los encapsulados. Por último, presentamos el entorno de la *FPGA* de la que disponemos: la tarjeta *XSA 50*, analizando cada componente de la misma.

El capítulo 3 tiene el nombre de **Procesador**. Presentamos el *Procesador RISC* que hemos diseñado con todas sus características. A continuación, hacemos un desglose de todas las instrucciones de que dispone el *Procesador* y, a modo de resumen, presentamos la *Tabla de Instrucciones* del *Procesador*. Por último, analizamos la estructura del bloque principal de diseño del *Procesador*.

El capítulo 4 tiene el nombre de **Unidad de Control**. En este capítulo, explicamos el diseño de la *Unidad de Control*, la secuenciación de operaciones en su interior y los bloques que la forman con todas sus entradas, salidas y funcionamiento interno de cada uno.

El capítulo 5 tiene el nombre de **Unidad de Proceso**. De la misma forma que en el capítulo anterior, en éste explicamos el diseño de la *Unidad de Proceso*, la secuenciación de operaciones y los bloques que la forman.

El capítulo 6 tiene el nombre de **Implementación y Aplicación Práctica**. Hemos explicado las características especiales del diseño implementado en este apartado. Además, presentamos una aplicación práctica que demuestra el funcionamiento correcto del *Procesador*.

El capítulo 7 tiene el nombre de **Costes y Presupuesto**. En él, estimamos los costes generados durante el desarrollo del *Proyecto*.

El apéndice A tiene el nombre de **Obtención e instalación del software**. El *Software* que hemos utilizado en este *Proyecto* puede obtenerse gratuitamente de la forma que explicamos en este apéndice.

El apéndice B tiene el nombre de **Tutorial de utilización del software**. En este apéndice, hemos desarrollado un pequeño tutorial para comprender la filosofía de funcionamiento del *Software*.

El apéndice C tiene el nombre de **Informes**. Durante la compilación del diseño, *Project Navigator* genera varios informes. En este apéndice, presentamos los más significativos.

Capítulo 2

Dispositivos Lógicos Programables.

Cuando se aborda el diseño de un sistema electrónico y surge la necesidad de implementar una parte con hardware dedicado son varias las posibilidades que existen. En la figura 2.1, se han representado las principales posibilades ordenándolas en función de los parámetros *coste*, *flexibilidad*, *prestaciones* y *complejidad*. Como podemos ver, las mejores prestaciones las proporciona un diseño *full-custom*, consiguiéndolas a costa de enormes *complejidades de diseño* y elevados *costes*. En el otro extremo del abanico de posibilidades, se encuentra la *implementación software*, que es muy barata y flexible, pero que en determinados casos no es válida para alcanzar un nivel de prestaciones relativamente alto.

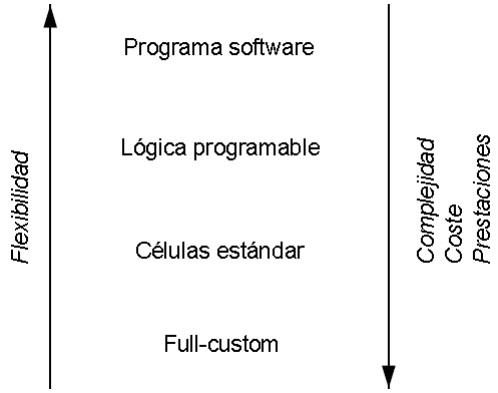


Figura 2.1: Posibilidades de diseño electrónico.

Entre estas dos opciones, se puede elegir la fabricación de un circuito electrónico realizado mediante *diseño semi-custom*, utilizando células estándar, o recurrir a un circuito ya fabricado que se pueda *programar*, como son los *dispositivos lógicos programables* o *PLDs (Programmable Logic Devices)*. De estas dos opciones, el *diseño semi-custom* proporciona mejores prestaciones, aunque es una opción más cara y exige un ciclo de diseño relativamente largo.

Sin embargo, los *dispositivos lógicos programables* constituyen una buena oferta para realizar diseños electrónicos digitales con un buen compromiso *coste-prestaciones*. Y lo que es mejor, permiten obtener una implementación en un tiempo de diseño muy corto, con la consiguiente

reducción del parámetro clave *Time to market*. Otro aspecto que se debe tener en cuenta para decidirse por este tipo de implementación es que el coste de realización es muy bajo, por lo que suele ser una buena opción para la realización de prototipos.

2.1. Dispositivos Lógicos Programables.

Dispositivos Programables son aquellos circuitos de propósito general que poseen una estructura interna pre-definida que puede ser modificada por el usuario (o a petición suya, por el fabricante) para implementar una amplia gama de aplicaciones. Para programar el diseño final, el usuario (o el fabricante) dispone de un conjunto de herramientas de desarrollo.

Los *Dispositivos Lógicos Programables (PLDs)* son dispositivos formados por una o más matrices de puertas *AND*, una o más matrices de puertas *OR* y biestables que se usan como elementos de almacenaje para entrada y salida de datos en un dispositivo. Cualquier circuito lógico puede resolverse como una suma de productos. Por tanto, mediante la estructura de un *PLD*, cualquier circuito lógico puede implementarse como esa suma de productos.

Los *PLDs* son bastante eficientes si implementan circuitos no superiores a unos centenares de puertas. Sin embargo, no dejan de tener una arquitectura rígida, y están limitados por un número fijo de biestables y entradas/salidas.

2.1.1. Características del diseño con *PLDs*.

El diseño con *PLDs* proporciona: facilidad de diseño, prestaciones, fiabilidad, economía y seguridad.

- **Facilidad de diseño:** Las herramientas de diseño con *PLDs* se han desarrollado mucho en los últimos años y facilitan enormemente este proceso de diseño. Las hojas de codificación que se utilizaban en 1975 han dejado paso a los ensambladores y compiladores de lógica programable (OrCadD/PLD, Palasm, Abel, Amaze, CUPL, Xilinx ISE, Altera...). Estas nuevas herramientas permiten expresar la lógica de los circuitos utilizando formas variadas de entrada tales como: ecuaciones, tablas de verdad, procedimientos para máquinas de estados, esquemas, etc. La simulación digital posibilita la depuración de los diseños antes de la programación de los dispositivos. Por tanto, todo el equipo de diseño se reduce a un software de bajo coste que corre en un PC y un programador.

- **Prestaciones:** Los *PLDs TTL* que hay en el mercado tienen tiempos de conmutación tan rápidos como los circuitos integrados de función fija más veloces. Los *PLDs ECL* son todavía más rápidos. Sin embargo, el incremento de velocidad obtenido con los dispositivos *CMOS*, que ya han igualado o superado en prestaciones a los dispositivos *TTL*, está provocando el abandono de la tecnología bipolar por parte de los fabricantes.
  En cuanto al consumo de potencia, los *PLDs*, generalmente, consumen menos que el conjunto de chips a los que reemplazan.

- **Fiabilidad:** Cuanto más complejo es un circuito, más probabilidades hay de que alguna de sus partes falle. Puesto que los *PLDs* reducen el número de chips en los sistemas, la probabilidad de fallo disminuye. Los circuitos impresos con menor densidad de chips son más fáciles de construir, más fiables y las fuentes de ruido también se reducen.

- **Economía:** En este apartado, hay aspectos que resultan difíciles de cuantificar como, por ejemplo, los costes de pérdida de mercado por una introducción tardía de un producto. Otros son más claros; por ejemplo, la reducción del área de las placas de circuito impreso obtenida gracias a que cada *PLD* sustituye a varios circuitos integrados de función fija. Muchas veces se consigue reducir el número de placas de circuito impreso ahorrando en conectores. La reducción de artículos en almacén también aporta ventajas económicas. De la misma manera que para altos volúmenes de producción las memorias *ROM* resultan de menor coste que las *EPROM*, los *PLDs* programados por el fabricante proporcionan ahorros adicionales en grandes cantidades.

- **Seguridad:** Los *PLDs* tienen fusibles de seguridad que impiden la lectura de los dispositivos programados, protegiendo los diseños frente a copias.

Además de los puntos mencionados, podemos añadir que los *PLDs* facilitan el rutado de las placas de circuito impreso debido a la libertad de asignación de patillas que permiten. Además, dan la posibilidad de realizar modificaciones posteriores del diseño e incluso, en algunas ocasiones, la reutilización de circuitos impresos con algunos fallos mediante re-programación de los mismos.

2.1.2. Tipos de *PLDs*.

Los *PLDs* se clasifican según su arquitectura y tamaño. Siguiendo esta clasificación, podemos presentar tres[1] grandes familias de *PLDs*:

1. **Dispositivos Lógicos Programables Simples (Simple Programmable Logic Devices (*SPLDs*)):**
   Los *SPLDs* son los dispositivos lógicos programables más pequeños y, por tanto, los más baratos. Este tipo de dispositivos fue el primero que se utilizó en los diseños de lógica programable.
   Dentro de los *SPLDs*, podemos distinguir dos tipos diferentes:

   1. a. **Matriz de Lógica Programable (Programmable Logic Array (*PLA*)):**
      Las *PLA* se basan en la idea de que todas las funciones lógicas pueden realizarse como *suma de productos*. Para implementar esta idea, disponen de una matriz de puertas *AND* que alimenta una matriz de puertas *OR* y cuyas conexiones son programadas por el diseñador. De esta forma, cualquier término *AND* puede alimentar cualquier término *OR*.

[1] Un cuarto tipo podrían ser las **Memorias Programables de Solo Lectura (Programmable Read Only Memory (*PROM*))**, pero nosotros las consideramos como memorias direccionables, y no como dispositivos lógicos programables.

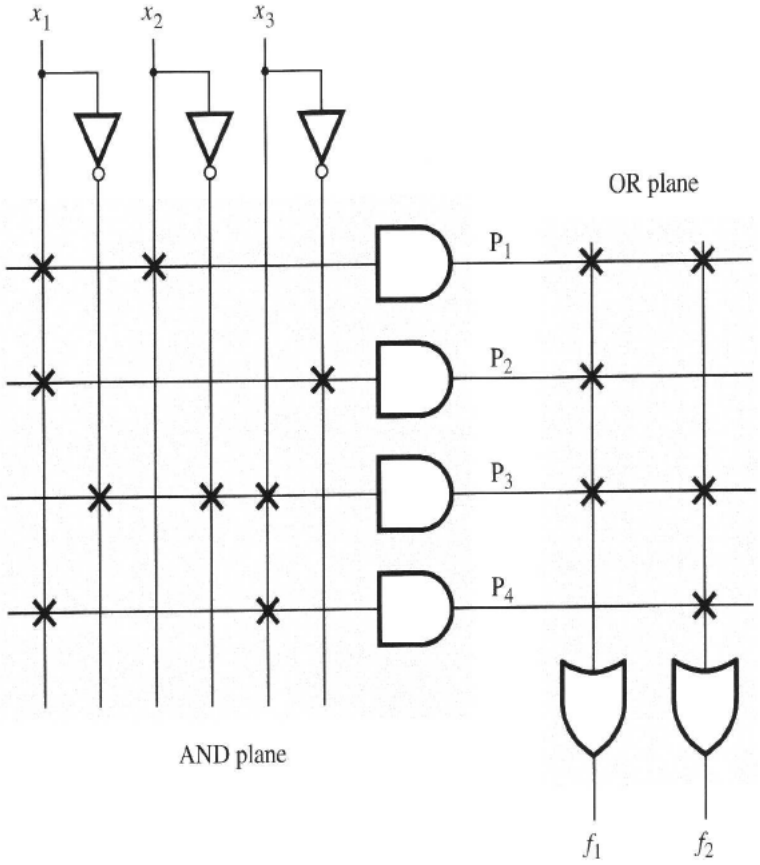


Figura 2.2: Arquitectura de una *PLA*.

Como muestra la figura 2.2, las entradas $x_{1...n}$ pasan a través de un set de inversores, de forma que se disponga de todas las entradas y sus inversiones. Mediante conexiones programables, estas entradas y sus inversiones se introducen en un banco de puertas *AND*. A la salida de cada puerta *AND*, se dispondrá de los términos *producto* $P_{1...n}$. Cada término producto, también mediante conexiones programables, se conecta al banco de puertas *OR* para generar la salida *suma de productos*, que, inequívocamente, determina la función lógica.
Probablemente, las *PLAs* tienen mayor flexibilidad frente a otros dispositivos con respecto a la lógica funcional. Normalmente, poseen realimentación desde la matriz *OR* hacia la matriz *AND*. Y esta realimentación puede usarse para implementar máquinas de estado asíncronas. La mayoría de las máquinas de estado, sin embargo, se implementan como máquinas síncronas. Con esta perspectiva, los fabricantes crearon un tipo de *PLA* denominado *Secuencial (Sequencer)* que posee registros de realimentación desde la salida de la matriz *OR* hacia la matriz *AND*.
Como conclusión, podemos decir que una *PLA* está formada por un plano *AND* y un plano *OR*. El plano *AND* genera los términos *producto* y el plano *OR* genera los términos *suma de productos*.

1. b. **Dispositivos de Matriz Programable (Programmable Array Logic (*PAL*)):** Las *PAL* son dispositivos de matriz programable. La arquitectura interna es muy similar a la vista en las *PLA*, con la única diferencia de que, en esta ocasión, los términos *OR* son fijos.

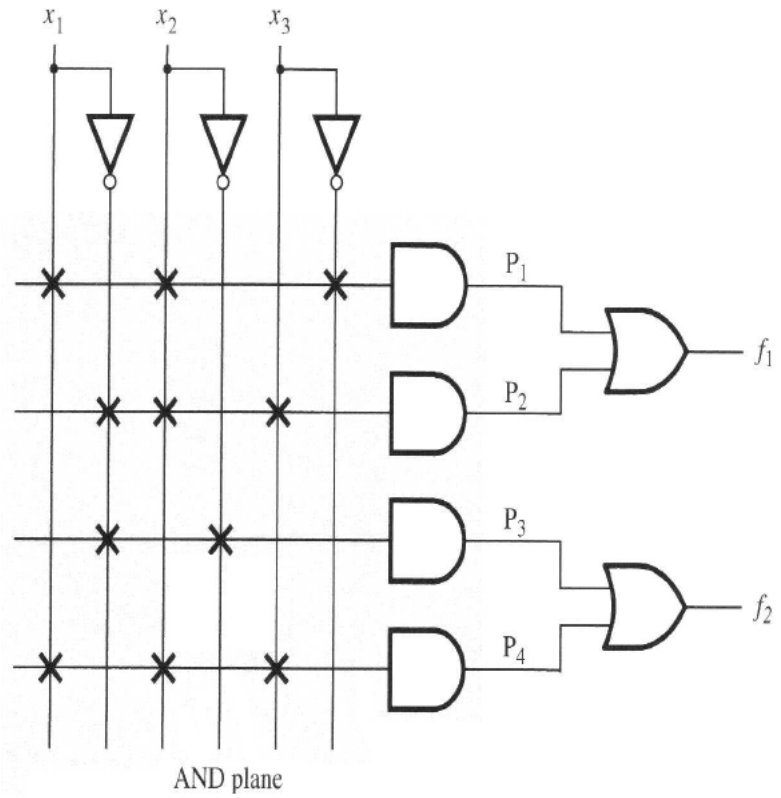


Figura 2.3: Arquitectura de una *PAL*.

Las *PAL* tienen una arquitectura más sencilla de construir y, por tanto, más barata. Además, ofrecen mejores prestaciones. Por estas razones, las *PAL* son los *PLDS* más utilizados.

2. **Dispositivos Lógicos Programables Complejos (Complex Programmable Logic Devices (*CPLDs*)):**
Los *CPLDs* son muy similares a los *SPLDs*. El único aspecto que los diferencia es que los primeros tienen muchísima más capacidad que los segundos. Típicamente, un *CPLD* equivale a unos 64 *SPLDs*.
Los *CPLDs* se consideran *PAL* muy grandes que incluyen algunas características de las *PLA*. La arquitectura básica es muy parecida a la *PAL*, con la posibilidad de modificar la cantidad de términos *AND* para cualquier término *OR* fijo. Esto se consigue quitando términos *AND* adyacentes o empleando términos *AND* desde una matriz expandida. Así, cualquier diseño pueda ser implementado dentro de estos dispositivos.

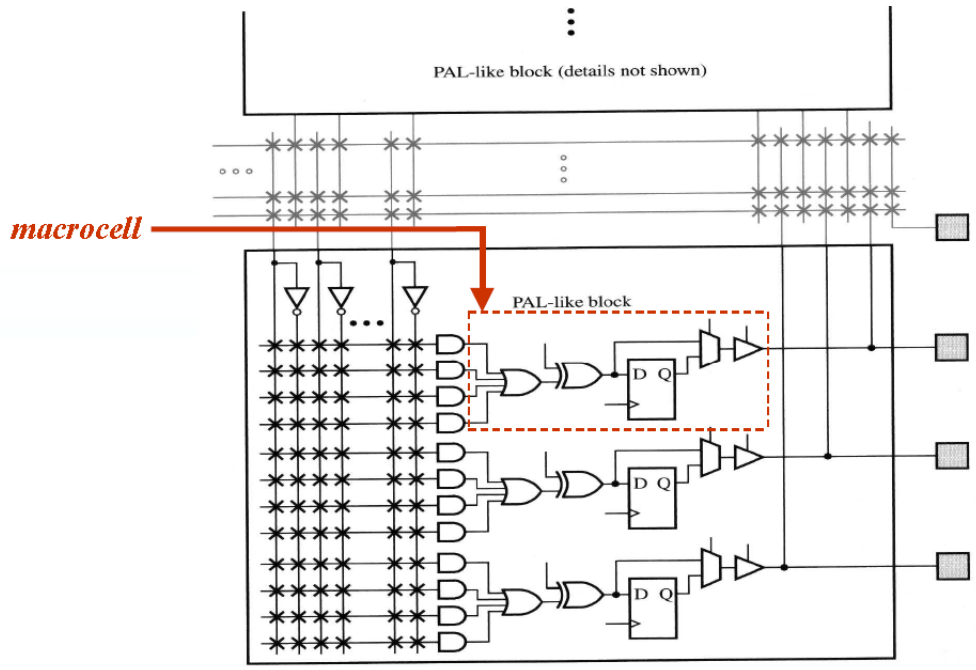


Figura 2.4: Arquitectura interna de un *CPLD*.

Un *CPLD* contiene entre 10 y 100 *macro-celdas* (bloques *PAL*)como las que muestra la figura 2.4. Además, en la figura 2.5, vemos que las *macro-celdas* están inter-conectadas mediante cables de conexión, y también se comunican con el exterior mediante bloques de entrada/salida.

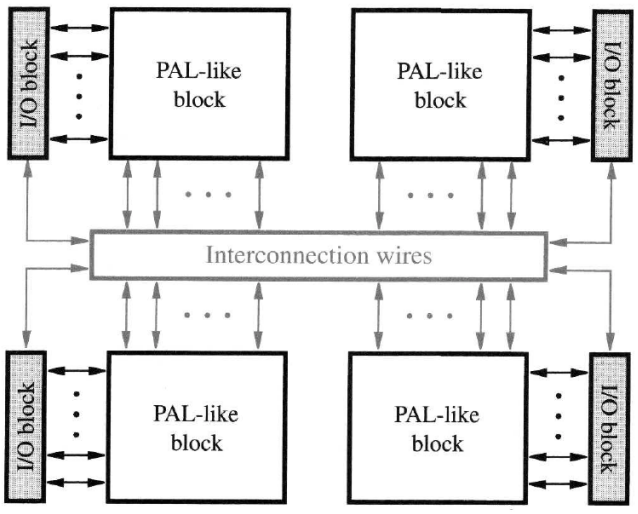


Figura 2.5: Arquitectura global de un *CPLD*.

3. **Matriz de puertas programable por campo (Field Programmable Gate Arrays (*FPGAs*)):**
   Las *FPGA* son dispositivos programables que soportan circuitos lógicos relativamente grandes. Son matrices de puertas eléctricamente programables que contienen múltiples niveles de lógica. Se diferencian de los *SPLDs* y *CPLDs* en que no contienen planos *AND* y *OR*, sino que disponen de bloques lógicos para implementar las funciones lógicas.
   Se caracterizan por:

   - altas densidades de puerta,
   - alto rendimiento,
   - número grande de entradas y salidas definibles por el usuario,
   - esquema de interconexión flexible, y
   - entorno de diseño similar al de los *CPLDs* (pero sin estar limitadas a la típica matriz *AND-OR*).

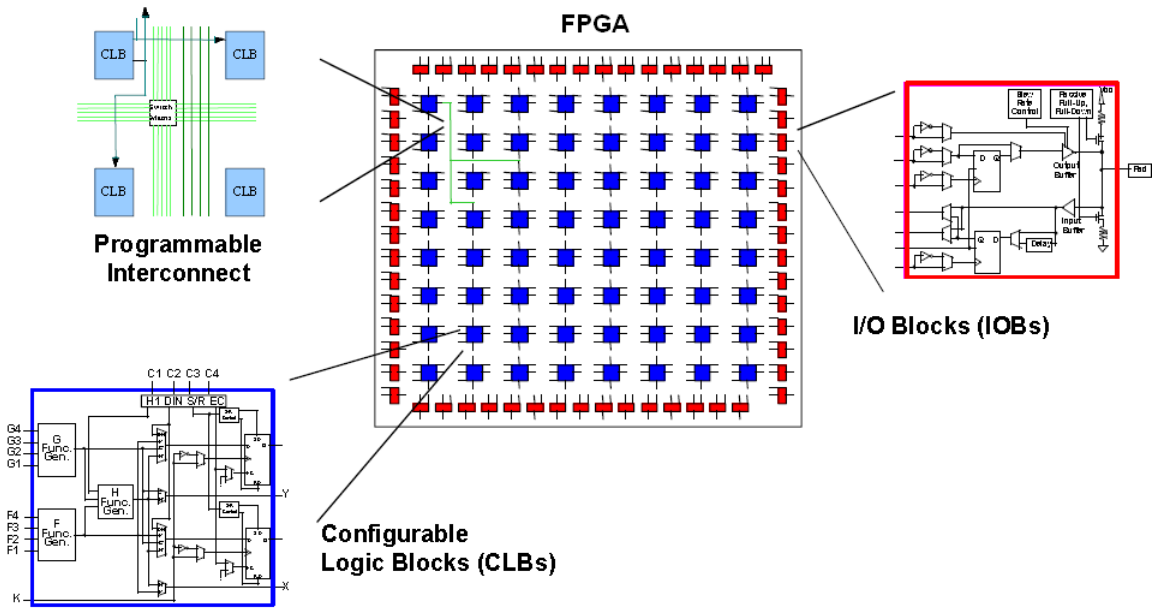


Figura 2.6: Arquitectura de una *FPGA*.

En cuanto a su arquitectura, las *FPGA* están formadas por tres tipos de recursos:

- Bloques lógicos (*CLBs*).
  Cada *CLB* contiene lógica programable combinacional y registros de almacenamiento. La sección de lógica combinacional es capaz de implementar cualquier función booleana de sus variables de entrada. Los registros de almacenamiento mantienen los valores de datos, ya sean datos intermedios o finales.
  Los *CLBs* están organizados, dentro de la *FPGA*, en forma de matriz bi-dimensional

- Bloques de entrada/salida (*IOBs*).
  Este bloque es el encargado de la conexión de los *CLBs* con los pines del encapsulado. Cada *IOB* puede programarse independientemente para ser una entrada y salida con control tri-estado o un pin bi-direccional. También, contiene flip-flops que pueden usarse como buffers de entrada y salida.
  Además, existen conexiones programables entre los *IOBs* y los cables de interconexión.
- Interconexiones.
  Los recursos de interconexión son una red de líneas que corren horizontalmente y verticalmente, en forma de filas y columnas, entre los *CLBs*. Las interconexiones son cables e interruptores programables que permiten interconectar los *CLBs* entre si y con los IOBs de muchas formas.

Los diseñadores que usan *FPGAs* pueden definir funciones lógicas en un circuito y revisar o cambiar estas funciones cuando y como crean conveniente. Además, dividiendo los diseños en en bloques lógicos, las *FPGAs* pueden completarse y verificarse en unos días, a diferencia de las varias semanas necesarias para los diseños con matrices de puertas programables.

2.2. *FPGA Xilinx Spartan II.*

2.2.1. Resumen de *FPGAs.*

Una *Matriz de puertas programable por campo* o *FPGA* es un circuito integrado de propósito general que puede ser programado por el usuario, además de por el fabricante. Al contrario de los *Circuitos Integrados de Aplicación Específica (ASICs)*, que siempre realizan la misma función aunque sea en diferentes sistemas, la re-programabilidad de las *FPGAs* las permite realizar diferentes funciones en diferentes sistemas, e incluso cambiar su funcionalidad cuando ya están incluidas en el nuevo sistema.

Una *FPGA* se re-programa descargando un archivo denominado *bitstream* en una memoria RAM on-chip de que dispone la *FPGA*. Este *bitstream* se genera a partir de herramientas de diseño software. Las herramientas de compilación traducen el diseño en alto nivel creado a un lenguaje de bajo nivel que se descarga en la *FPGA*.

Las *FPGAs* tienen una estructura de matriz bi-dimensional, con bloques inter-conectados entre si y puertos para el interface con el exterior. Dentro de estos bloques, llamados bloques lógicos (*CLBs*), se implementan las funciones lógicas. Cada bloque dispone, típicamente, de los siguientes recursos:

- Look-up tables (*LUTs*)
- Puertos duales de memoria
- Registros
- Buffers tri-estado
- Multiplexores
- Dispositivos para el control del reloj.

2.2.2. Empresa Xilinx.

La empresa Xilinx se dedica, entre otras cosas, al diseño y fabricación de *FPGAs*. Con el tiempo, Xilinx ha ido mejorando sus diseños, mejorando prestaciones, capacidades y demás parámetros característicos de estos dispositivos.

Figura 2.7: Logotipo de la empresa Xilinx.

Xilinx ha desarrollado de 2 familias de *FPGAs*: *Spartan* y *Virtex*, y ha tenido, y tiene, en el mercado los siguiente productos:

- *Spartan*
- *Spartan XL*
- *Spartan-II*
- *Spartan-IIE*
- *Spartan-3/3L*
- *Virtex*
- *Virtex-E*
- *Virtex-E Extended Memory*
- *Virtex-II*
- *Virtex-II Pro*
- *Virtex-4*

Nuestra *FPGA* es modelo *Spartan-II*, es decir, pertenece a la tercera generación de *FPGAs* de la familia *Spartan* de Xilinx.

Dentro de esta familia *Spartan II*, existen varios modelos de dispositivos según su capacidad, número de bloques lógicos, etc. Nuestra *FPGA*, en concreto, es el modelo *XC2S50*, que dispone de las características que se muestran en la figura 2.8.

Device	Logic Cells	System Gates (Logic and RAM)	CLB Array (R x C)	Total CLBs	Maximum Available User I/O	Total Distributed RAM Bits	Total Block RAM Bits
XC2S15	432	15,000	8 x 12	96	86	6,144	16K
XC2S30	972	30,000	12 x 18	216	92	13,824	24K
XC2S50	1,728	50,000	16 x 24	384	176	24,576	32K
XC2S100	2,700	100,000	20 x 30	600	176	38,400	40K
XC2S150	3,888	150,000	24 x 36	864	260	55,296	48K
XC2S200	5,292	200,000	28 x 42	1,176	284	75,264	56K

Figura 2.8: Modelos de la familia *Xilinx Spartan II*.

2.2.3. Presentación de la familia *Spartan II.*

La familia *Xilinx Spartan II* de *FPGAs* tiene una arquitectura regular, flexible y programable de *bloques lógicos (CLBs)*, rodeada por un conjunto de *bloques de entrada/salida (IOBs)*. Tiene cuatro *lazos enganchados en retardo (DLLs) Delay-Locked Loops*, uno en cada esquina del chip para sincronizar las señales del reloj que circulan por la *FPGA*. Además, contiene dos columnas de bloques de memoria *RAM* en sendos lados de la *FPGA*, entre los *CLBs* y los *IOBs*. Todos estos elementos están inter-conectados por un rutado muy potente y versátil.

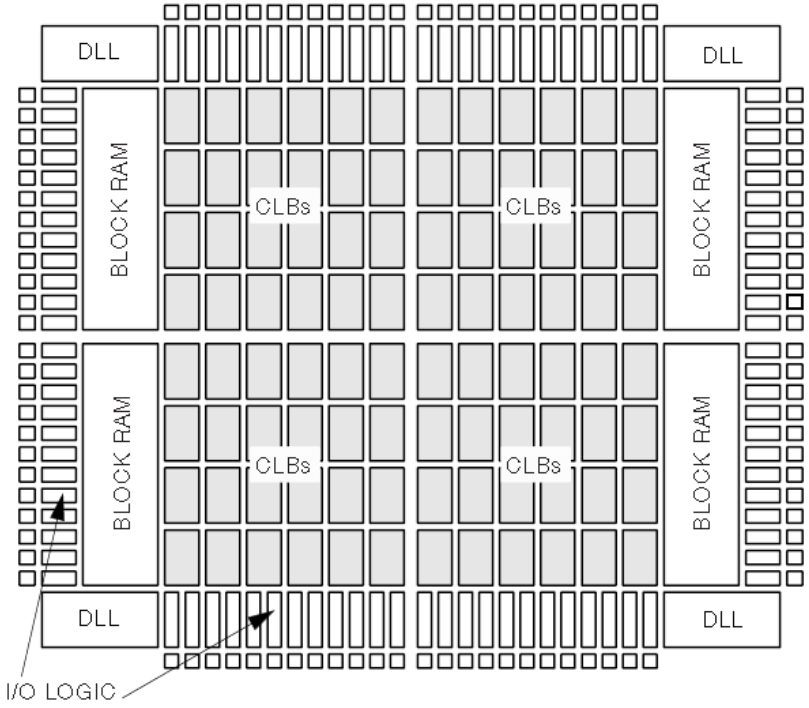


Figura 2.9: Arquitectura de la *FPGA Xilinx Spartan II.*

Las *FPGAs Xilinx Spartan II* están diseñadas para cargar los datos de configuración en celdas internas de memoria estática. Los valores almacenados en estas celdas determinan las funciones implementadas en los *CLBS* y las interconexiones de cada elemento de la *FPGA*. Los datos de configuración pueden leerse de una Memoria *PROM* externa, o pueden ser descargados en la *FPGA* mediante un puerto serie, paralelo o mediante la técnica de *Boundary Scan*. Permite ilimitados ciclos de re-programación.

Las *FPGAs Xilinx Spartan II* se utilizan típicamente en aplicaciones de alto volumen de producción en las que la posibilidad de una programación del dispositivo es un punto decisivo. Son dispositivos ideales para reducir los tiempos de diseño y desarrollo, a la vez que ofrecen una solución económicamente buena para altos volúmenes de producción.

Por último, destacamos que esta familia de *FPGAs* contiene un sistema de gestión de señales de reloj que permite realizar operaciones con una velocidad de hasta $200MHz$.

2.2.4. Encapsulados.

El encapsulado de estas *FGPAs* tiene el aspecto que se muestra en la figura 2.10.

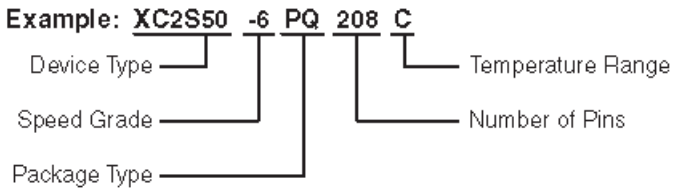


Figura 2.10: Encapsulado típico de la *FPGA Xilinx Spartan II.*

En la figura 2.11, presentamos las diferentes posibilidades que están en el mercado.

Device Ordering Options

Device	Speed Grade		Number of Pins / Package Type		Temperature Range (T_J)	
XC2S15	-5	Standard Performance	VQ(G)100	100-pin Plastic Very Thin QFP	C = Commercial	0°C to +85°C
XC2S30	-6	Higher Performance	CS(G)144	144-ball Chip-Scale BGA	I = Industrial	–40°C to +100°C
XC2S50			TQ(G)144	144-pin Plastic Thin QFP		
XC2S100			PQ(G)208	208-pin Plastic QFP		
XC2S150			FG(G)256	256-ball Fine Pitch BGA		
XC2S200			FG(G)456	456-ball Fine Pitch BGA		

Figura 2.11: Información del Encapsulado.

En las figuras 2.10 y 2.11 , vemos que el encapsulado contiene información de diverso tipo:

- Tipo de dispositivo (*Device Type*).
- Encapsulado (*Package*).
- Velocidad (*Speed*).
- Rango de operación de temperatura (*Operating Range*).
- Código de lote y de fecha (*Data code & Lot Code*).

Para la realización de este proyecto, nosotros hemos trabajado con la *FPGA Xilinx Spartan II XC2S50 -5 TQ 144 C.*

2.2.5. Arquitectura.

La *FPGA Xilinx Spartan II* se compone de los siguientes 5 elementos:

- *IOBs* son el interface entre los pines del encapsulado y la lógica interna.
- *CLBs* donde se implementa la lógica.
- *Bloques de memoria RAM* de 4096 bits.
- *DLLs* para la distribución de la señal de reloj.
- *Interconexiones* versátiles y multi-nivel.

Los *CLBs* forman el núcleo de la *FPGA* con un acceso muy rápido gracias a las interconexiones. Cada *CLB* contiene dos *slices*. Los *IOBs* están alrededor de este núcleo y de los elementos de memoria para facilitar la entrada y salida de las señales del chip. Los valores almacenados de las memorias estáticas controlan la lógica y las interconexiones. Estos valores de control se cargan en las celdas de memoria estática todas las veces que sea necesario, cambiando la función lógica que implementa la *FPGA*.

Para una información más detallada, nos remitimos al documento *Spartan-II 2.5V FPGA Family: Complete Data Sheet*, disponible en la página web de Xilinx y en el *DVD* adjunto a este documento.

2.3. Entorno de la *FPGA*: Tarjeta *XSA 50*.

La *FPGA* que vamos a utilizar para nuestro proyecto está incluida en la tarjeta *XSA 50*, de la empresa *Xess Corporation*. En este apartado, vamos a presentar esta tarjeta, analizando sus componentes, que son los periféricos a los que puede acceder la *FPGA*.

La figura 2.12 muestra el aspecto externo de la tarjeta *XSA 50*. En ella, podemos ver la organización de todos los componentes y su posicionamiento en la tarjeta. En la figura 2.13, presentamos el esquema de conexiones de la tarjeta, con los 4 bloques principales perfectamente identificables. Estos cuatro bloques son:

1. *FPGA Xilinx Spartan II*.
2. *CPLD XC9572XL*.
3. *Memoria Flash ROM*.
4. *Memoria SDRAM*.

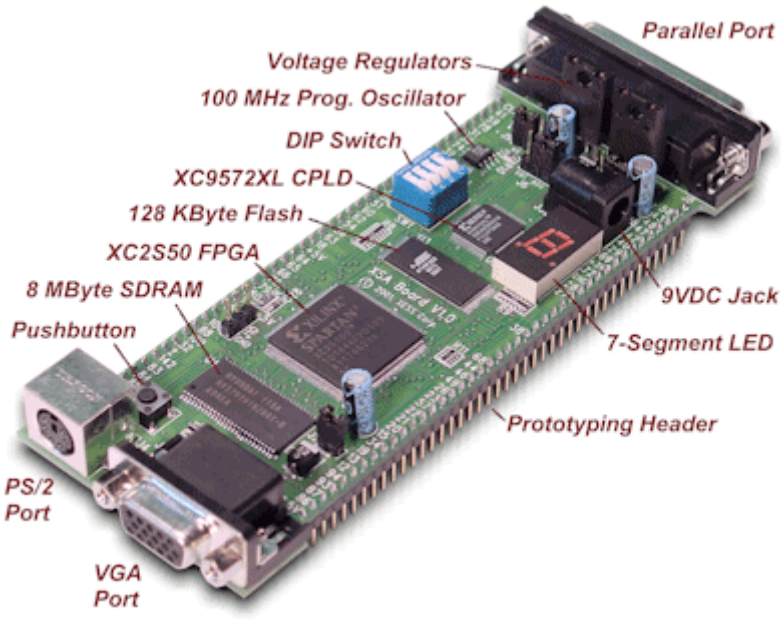


Figura 2.12: Tarjeta *XSA 50*, con la *FPGA Xilinx Spartan II*.

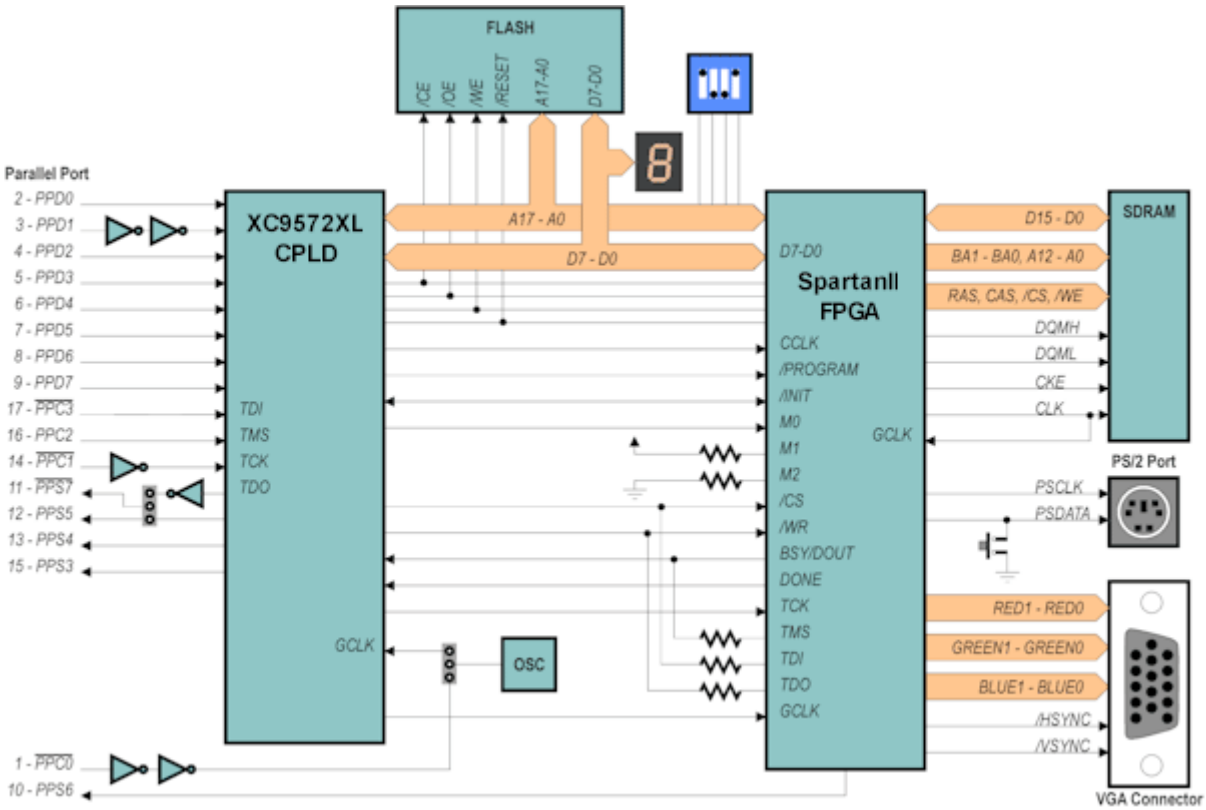


Figura 2.13: Esquema de la tarjeta *XSA 50*.

2.3.1. *FPGA Xilinx Spartan II modelo XC2S50.*

Es el bloque principal de la tarjeta y el resto de recursos lo complementan. Contiene la lógica del diseño y la lógica de las conexiones con los demás dispositivos disponibles en la tarjeta. Nuestro modelo, *FPGA Xilinx Spartan II XC2S50 TQ 144*, tiene un capacidad de 50.000 puertas lógicas en un encapsulado de 144 pins.

Nosotros descargamos nuestros diseños en formato **.bit* (*bitstreams*) sobre la *FPGA Xilinx Spartan II* desde del *Puerto Paralelo* y a través del *CPLD*. Para ello, utilizamos la herramienta *XSLoad*, del paquete *XSTools*, incluida en el *CD-ROM* adjunto a la tarjeta enviado por *Xess Corp.* al realizar la compra.

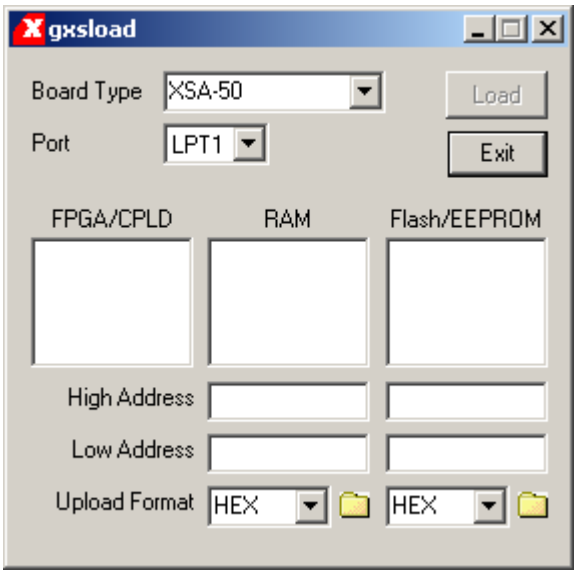


Figura 2.14: Herramienta GXSLoad.

Mediante esta herramienta pueden cargarse datos tanto a la *FPGA* como a la *Memoria SDRAM* y a la *Memoria Flash Rom*, y descargarse datos desde esas dos memorias. Para la descarga de datos, es necesario introducir el intervalo de direcciones que se desea descargar. Al presionar sobre el botón de descarga, el *CPLD* se configurará para realizar la descarga, y ya en nuestro *PC*, en el directorio donde estén instaladas las herramientas del *XSTools*, se creará el archivo de descarga, que, por supuesto, tendrá extensión *.HEX*.

2.3.2. *CPLD XC9572XL.*

El *CPLD XC9572XL* se utiliza para llevar a cabo la configuración de la *FPGA* desde el *Puerto Paralelo*. Además, controla la programación de la memoria *Flash ROM* de la tarjeta.

No es accesible ni programable por el usuario, sino que se configura automáticamente según el tipo de dato que estemos descargando a la tarjeta o subiendo desde la misma.

2.3.3. Memoria *Flash ROM.*

La tarjeta *XSA 50* dispone de una memoria *Flash ROM* de 128 posiciones de 8 bits por posición. Por tanto, dispone de un bus de datos de 8 bits de anchura ($D7 \ldots D0$) y un bus de direcciones de 17 bits ($A16 \ldots A0$). Se trata de la memoria *AT49F001 Flash Ram* de la empresa *Atmel*.

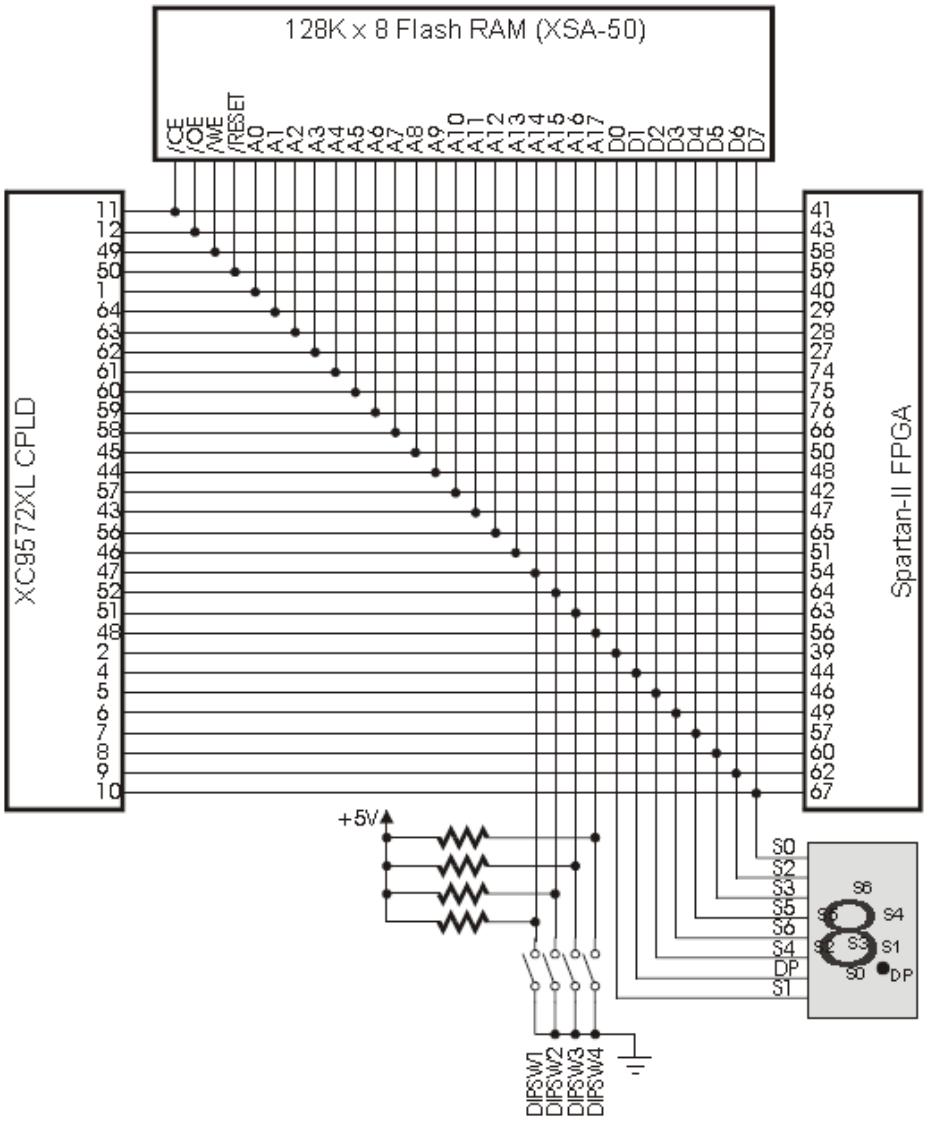


Figura 2.15: Esquema de conexiones *CPLD - FPGA - Flash Rom - Display 7 segmentos.*

Como se aprecia en la figura 2.15, la memoria *Flash Rom* está conectada de forma que es accesible por la *FPGA* y por el *CPLD*. De hecho, cuando se descargan datos desde el exterior sobre la memoria *Flash Rom*, se hace mediante el *Puerto Paralelo* y a través del *CPLD*. Incluso si estos datos son el *bitstream* de la *FPGA*, el *CPLD* se puede configurar para que, cada vez que se alimente la tarjeta, automáticamente se descargue ese *bitstream* sobre la *FPGA*.

La herramienta que se utiliza para cargar datos del exterior sobre la *Flash ROM* y descargar datos de la la *Flash ROM* hacia el exterior es la herramienta *GXSLoad*, mostrada en la figura 2.14. Para una carga correcta, los *DIP Switchs* deben estar colocados en posición *off*. Este punto se analiza en el apartado 2.3.4.

Los datos transferidos deben tener uno de los formatos[2] presentados a continuación:

- *HEX* o *MCS*: Formato Intel hexadecimal.
- *EXO-16, EXO-24, EXO-32)*: Formato de codificación de *Motorola* con direcciones de 16, 24 o 32 bits respectivamente.
- *XESS-16, XESS-24, XESS-32*: Formato hexadecimal de *Xess Corp.* con direcciones de 16, 24 o 32 bits respectivamente.

Formato HEX.

El formato *Intel Hex* permite la codificación de de datos binarios en lenguaje *ASCII*. De esta forma, se pueden interpretar esos datos binarios de una forma muy sencilla facilitando su transmisión. Cada dato *HEX* es una única línea dentro de un archivo formado por muchos datos *HEXs*.

El formato *HEX* se compone de varios campos que especifican tipo de dato, número de datos, dirección de memoria, dato y *checksum*. Cada dato *byte* se codifica como un número hexadecimal de dos dígitos. El primer dígito representa los 4 bits más altos del *byte* y el segundo dígito representa los 4 bits más bajos del *byte*.

En el cuadro 2.1, hemos presentado la codificación *HEX* para los datos binarios que utilizamos. En el campo 4, *Tipo*, nosotros hemos usado el tipo 0, que es el más común. Por tanto, es necesario rellenar ese campo con 00.

El *Checksum* se calcula para cada línea de datos. El método es el siguiente. Se asocian todos los dígitos por parejas, se suman y se hace el complemento a 2. El *Checksum* está formado por los dos dígitos menos significativos del número resultante.

En el siguiente ejemplo, se ilustra el cálculo de un *Checksum*. Estas líneas forman parte de un archivo de datos con formato *HEX*.

```
:10010000214601360121470136007EFE09D2190140
:100110002146017EB7C20001FF5F16002148011988
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
```

[2]Nosotros utilizamos el formato *HEX*.

Campo		**Caracteres**	**Descripción**
1	Código de comienzo	1	Comienzo de cada línea de datos (:).
2	Número de bytes	2	Número de bytes que contiene el campo Dato.
3	Dirección	4	Dirección del primer dato de la línea.
4	Tipo	2	Tipo de dato *HEX*.
5	Dato	0-2^n	Datos.
6	Checksum	2	Verificación de datos.

Cuadro 2.1: Formato *Intel HEX*.

A continuación, analizamos la primera línea de datos:

- **":".** Código ASCII que indica el comienzo de una línea en un fichero de datos *HEX*.
- **10.** Número de *bytes (datos)*, en hexadecimal (16 en decimal), que contiene la línea actual.
- **0100.** Dirección del primer dato del campo de datos.
- **00.** Tipo de Dato.
- **21460136012147013600 7EFE09D21901.** Datos
- **40.** Cálculo del checksum

$$10+01++21+46+01+36+01+21+47+01+36+7E+FE+09+D2+19+01=3C0$$
$$not(3C0)+1=FF\ldots FC3F+1=40$$
$$\boxed{Checksum=40}$$

El cálculo del *Checksum* es importante porque actúa como verificador de datos y su correcto cálculo es indispensable para la carga de un fichero *.HEX* en la *Flash ROM*. Si el *Checksum* no es correcto, *GXSLoad* no permitirá la carga de los datos en la memoria.

Por último, la memoria *Flash ROM* no solo se puede comunicar con el exterior, sino que también puede hacerlo con la *FPGA*. Además del bus de datos y del bus de direcciones, la memoria *Flahs ROM* tiene señales de control ($\overline{CE}$, $\overline{OE}$, $\overline{WE}$ y $\overline{RESET}$) que permiten su escritura y/o lectura desde la *FPGA*.

2.3.4. *DIP switches.*

Como vemos en la figura 2.15, la placa contiene un banco de 4 *switches* que son accesibles manualmente por el usuario. Estos *switches* están conectados a los bits más significativos del bus de direcciones de la memoria *Flash Rom*[3].

Cada *switch* tiene dos posiciones: *on* y *off*. La posición *off* cortocircuita la patilla correspondiente con masa. La posición *on*, a través de la resistencia de *pull-up*, presente en el esquema, permite que la entrada de estas patillas tome cualquier valor, valor que estará definido por el *CPLD*.

Para cargar datos del exterior por el *Puerto Paralelo*, los *switches* deben estar en posición *off*, ya que la carga de la memoria *Flash Rom* exige que los pines del bus de direcciones estén conectados a masa. La descarga de datos hacia el exterior también exige que los *switches* estén en posición *off*. La descarga de datos hacia la *FPGA* permite las dos posiciones. Dependiendo de en qué posición se sitúe el *switch*, la dirección tomará un valor 0 lógico o 1 lógico.

Para nuestro procesador, nosotros aprovechamos esta opción para cargar varios programas diferentes en la memoria *Flash Rom*. Según la posición de los *switches*, al conectar a alimentación, nuestro procesador ejecutará unos programas u otros.

2.3.5. *Display de 7 segmentos.*

Para completar todos los elementos que aparecen en la figura 2.15, presentamos el *Display de 7 segmentos*. Su acceso está compartido por el *CPLD*, la *FPGA* y los 8 bits del bus de datos de la memoria *Flash Rom*. Los segmentos del *LED* son *activos en alta*, es decir, cada segmento se ilumina cuando un *1 lógico* se le aplica.

Por las características de diseño de nuestro procesador, la memoria *Flash Rom* es de solo lectura, y de una lectura constante y continua. Por tanto, del esquema y de esta característica del diseño, se deduce que no tendremos acceso a este display.

2.3.6. Memoria *SDRAM.*

La tarjeta *XSA 50* contiene una memoria *RAM Dinámica y Síncrona* de $4M$ posiciones de 16 bits por posición (memoria *K4S641632F-TC75000* de la empresa *Samsung*), lo que supone un bus de datos de 16 bits ($Q15 \ldots Q0$).

Como la memoria es síncrona y dinámica, no tiene un bus de direcciones similar al caso de la memoria *Flash Rom*, es decir, un bus de direcciones que tendría 22 bits. Este tipo de memoria define la dirección mediante dos ciclos de acceso. Para ello, tiene las direcciones divididas en una matriz de filas y columnas. En el primer acceso, selecciona una fila desde el bus de direcciones, y

[3]Solo los 3 primeros *switches* tienen una función útil, ya que el cuarto *switch* está conectado a la patilla *A17* que no existe en nuestra memoria *Flash Rom*.

en el segundo acceso, modificando este bus de direcciones, selecciona la columna adecuada. Así, reduce el número de bits del bus de direcciones. De hecho, como se puede ver en el esquema de la figura 2.16, el bus de direcciones tiene una anchura de 12 bits ($A11 \ldots A0$), al que se añade las siguientes señales de control: *selección de banco* ($\overline{BA0}$ y $\overline{BA1}$) y *selección de fila y columna* ($\overline{RAS}$ y $\overline{CAS}$).

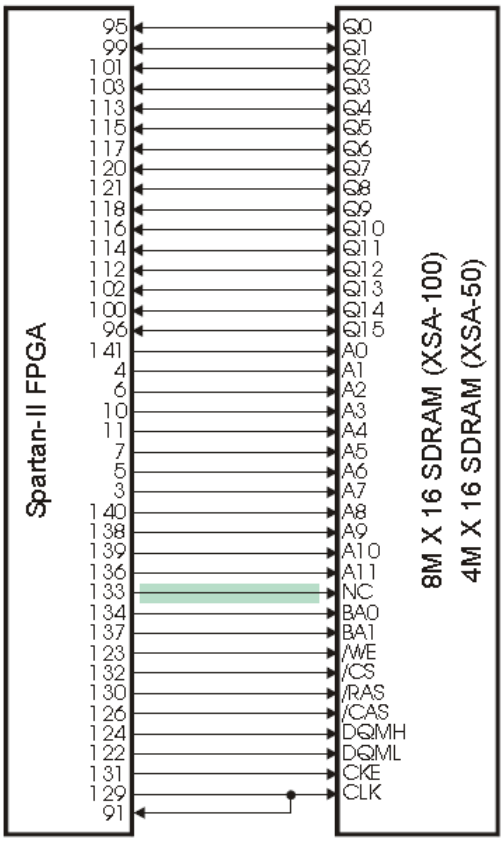


Figura 2.16: Esquema de conexiones *FPGA - SDRAM.*

Además, del problema de acceso en dos ciclos, esta memoria presenta otro inconveniente bastante más importante. Y es que, al ser una memoria dinámica, necesita refresco constantemente. De este refresco debe encargarse la lógica interna de la *FPGA* y este hecho complica mucho su diseño. Por esta razón, hemos usado un controlador para esta memoria *SDRAM*. Dicho controlador está disponible, de forma gratuita, en la web de la empresa *Xess Corp.* y está implementado en lenguaje *VHDL*.

Este controlador actúa como interface entre la *FPGA* y la memoria *SDRAM*, convirtiéndola en una memoria *RAM* tradicional a ojos de la *FPGA*. Acepta peticiones de lectura y escritura, datos y direcciones, una señal de reset y una señal de reloj, y genera las señales necesarias hacia la memoria *SDRAM*, para la escritura o lectura de datos de la *SDRAM*. Si no se accede a la memoria *SDRAM* durante varios ciclos de reloj, el propio controlador se encarga de realizar el refresco de la memoria.

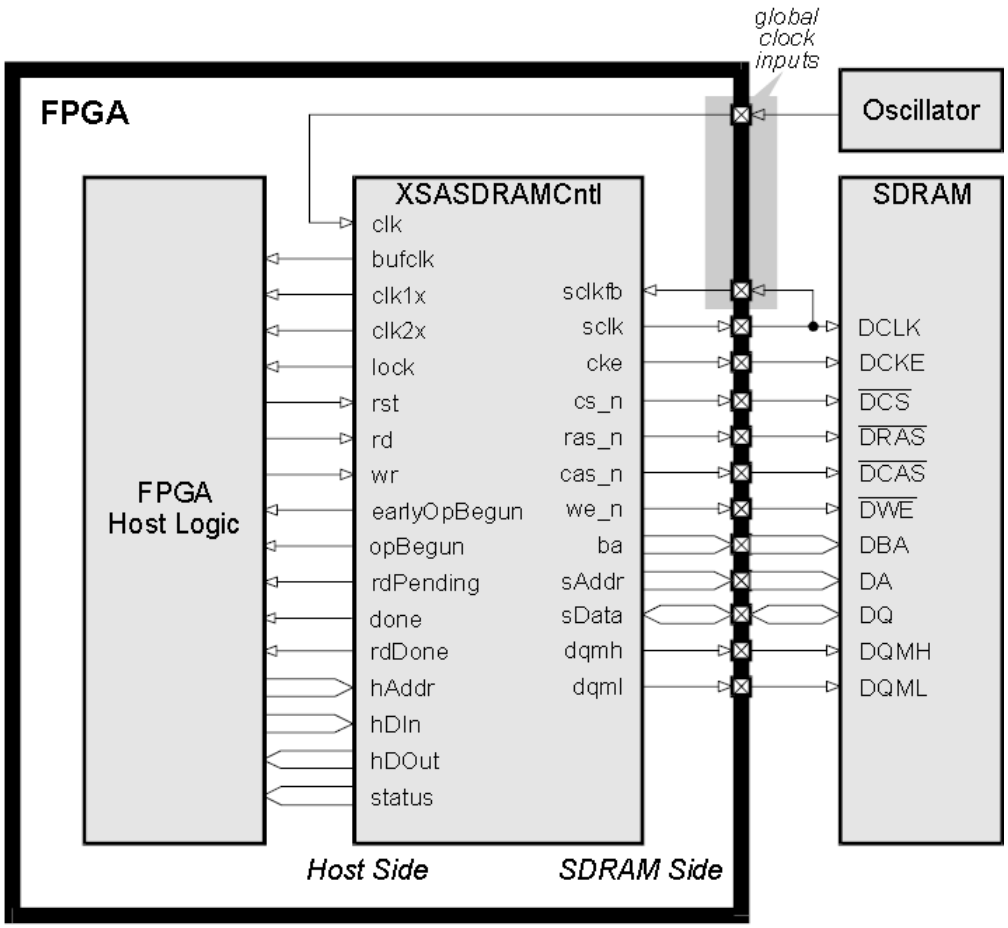


Figura 2.17: Esquema del controlador de la memoria *SDRAM*.

El controlador al que hacemos referencia está diseñado pensando en la tarjeta *XSA 100*, y no en la *XSA 50*, que es la tarjeta de la que disponemos nosotros. La tarjeta *XSA 100* dispone de una memoria *SDRAM* de $8M$ posiciones de 16 bits, es decir, el doble de capacidad que la *SDRAM* de nuestra tarjeta. Como ya hemos comentado, ambas memorias están divididas en filas y columnas, para reducir el número de patillas. Ambas memorias disponen de 4096 filas, pero mientras que la *XSA 100* tiene 512 columnas, la *XSA 50* solo tiene 256. Este punto debe ser modificado en todos los ficheros que implementan el controlador. Explicaremos esta modificación con más detalle, así como numerosos aspectos de su diseño e implementación, en posteriores apartados.

2.3.7. *Puerto Paralelo.*

La placa *XSA 50* dispone de 3 puertos de comunicación con el exterior. El primero es el *Puerto Paralelo*. El *Puerto Paralelo* es el principal interface de comunicación con la tarjeta *XSA 50*. Este puerto tiene 17 líneas: $C0 \ldots C3$ son líneas de control, $S3 \ldots S7$ son líneas de estado y $D0 \ldots D7$ son líneas de datos. Presentamos sus funciones a continuación:

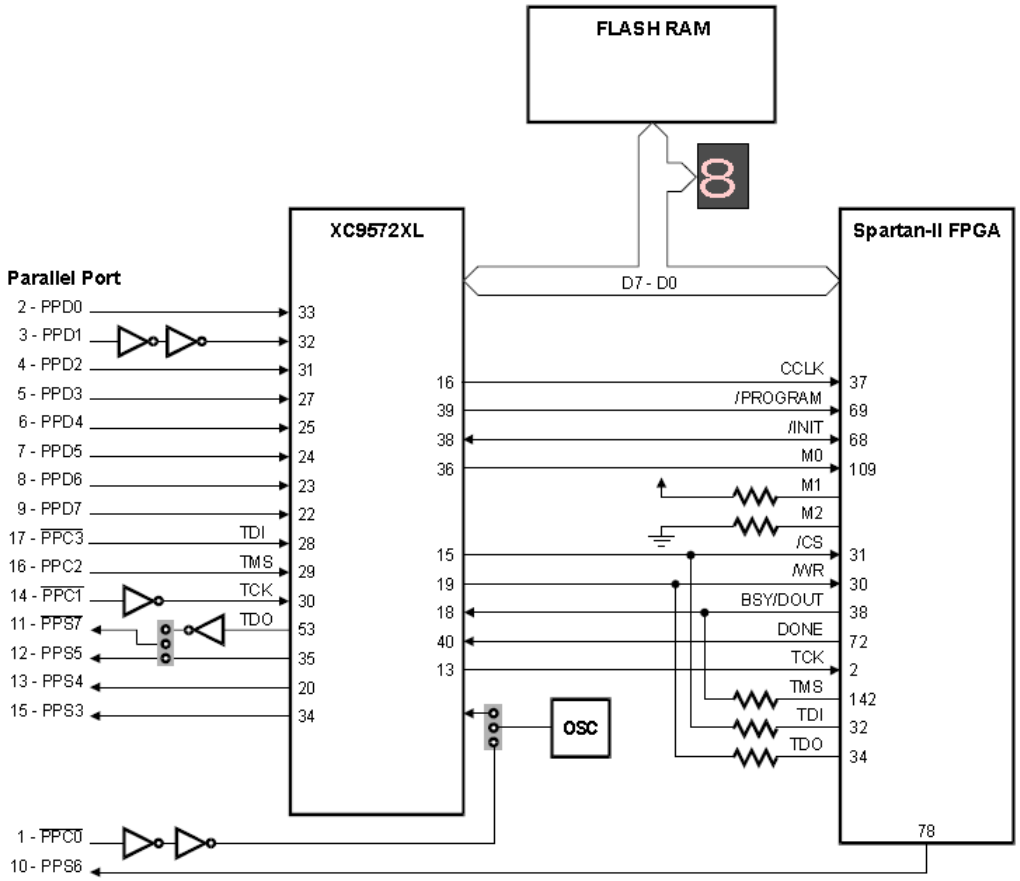


Figura 2.18: Esquema de conexiones del *Puerto Paralelo*.

- ***Puerto Paralelo C0:*** Línea de control que configura el oscilador programable *DS1075*, llevando la información del factor de division. Ver apartado 2.3.10. item ***Puerto Paralelo S6:*** Línea de estado procedente de la *FPGA* hacia el PC que se utiliza para comunicación directa, sin pasar por el *CPLD*.
- ***Puerto Paralelo C1. . . C3:*** Estas líneas de control se utilizan para la configuración del *CPLD*. La linea C1 lleva una señal de reloj para sincronizar los datos que entran por la señal C3 en el *CPLD*. Para que la señal de reloj sea lo más *limpia* posible tiene un *Schmitt-trigger*. La línea C2 controla la máquina de estados interna del *CPLD*.
- ***Puerto Paralelo S7:*** Línea de de estado que informa al PC sobre el estado del *CPLD*.
- ***Puerto Paralelo D0. . . D7:*** Líneas conectadas a los pins de propósito general del *CPLD*.
- ***Puerto Paralelo S3. . . S5:*** Líneas conectadas a los pins de propósito general del *CPLD*.

El *CPLD* puede programarse para que actúe como interface entre el *Puerto Paralelo* y la *FPGA*. En este caso, el *CPLD* está conectado a los pins de configuración de la *FPGA* para pasar los *bitstreams* provenientes del *Puerto Paralelo*. Además, esas líneas también están conectadas a la *Flash Rom* y al *Display de 7 segmentos*. Por otro lado, los pins de configuración (*CCLK,* $\overline{PROGRAM}$*,* $\overline{CS}$ *y* $\overline{WR}$)de la *FPGA* que controlan la carga del *bitstream* también pasan a través del *CPLD*.

El *CPLD* usa la línea *M0* para seleccionar el modo de funcionamiento (*maestro* o *esclavo*) de la *FPGA* y obtiene información del estado del *bitstream* a través de las líneas $overline INIT$*, DONE y BSY{}DOUT* que salen de la *FPGA*.

Antes de finalizar este apartado, es necesario hacer referencia a los pins *JTAG* de la *FPGA* que permiten la comunicación del PC con la *FPGA* directamente a través del *CPLD* y mediante la herramienta *Xilinx iMPACT*. Como nosotros no utilizamos esta herramienta, no entramos más en detalle sobre este punto.

2.3.8. Puerto *PS/2*.

Un segundo puerto que contiene la tarjeta es el puerto *PS/2*. Este puerto es un interface de tipo *PS/2*, con *conector mini-DIN* para la conexión de un ratón o un teclado. Este puerto manda información a la *FPGA* mediante dos señales. La primera es una señal de reloj que entra por el pin 94 de la *FPGA*. Por la otra línea (pin 93 de la *FPGA*), se envía una cadena de datos que la *FPGA* interpretará de forma correcta gracias a la sincronización con la señal de reloj.

En la página web de *Xess Corp.* está accesible un ejemplo que implementa un *driver* para utilizar este puerto conectado a un teclado.

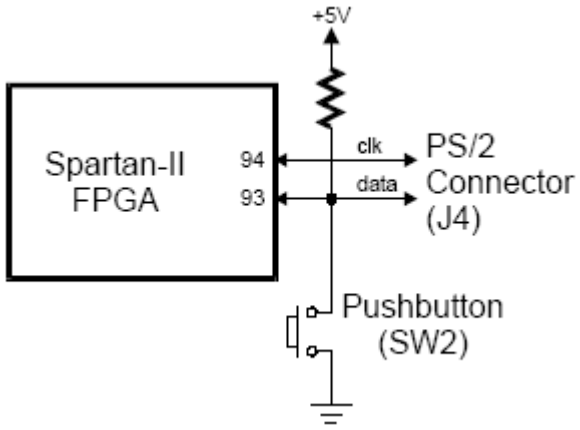


Figura 2.19: Esquema de conexiones del puerto *PS/2*.

2.3.9. Puerto *VGA*.

El tercer y último puerto que tiene la tarjeta es el puerto *VGA*. La *FPGA* es capaz de generar una señal de vídeo para un monitor *VGA*. Cuando la *FPGA* genera señales *VGA*, lleva a su salida dos líneas para cada color: rojo, verde y azul. Estás líneas entran un *convertidor digital-analógico (CAD)*, y las salidas de este *DAC* se envían a la entrada de un monitor *VGA* junto con dos señales de sincronismo (*overline$HSYNC$* y *overline$VHSYNC$*) procedentes de la *FPGA*. En el esquema, vemos los pines de conexión de la *FPGA* y el puerto *VGA*.

En la página web de *Xess Corp.* está accesible un ejemplo que implementa un *driver* para utilizar este puerto conectado a un monitor.

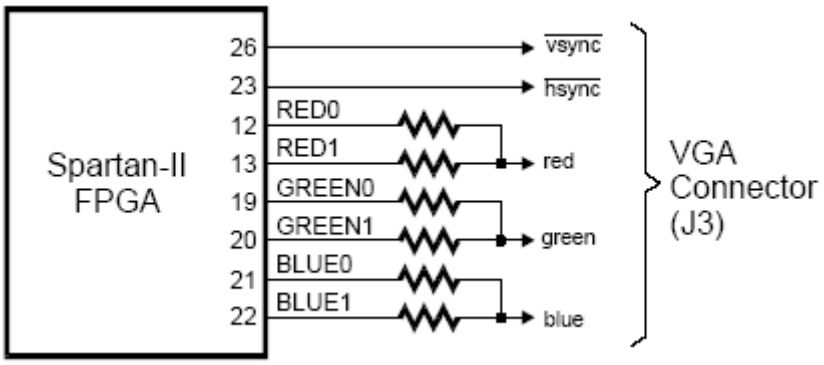


Figura 2.20: Esquema de conexiones del puerto *VGA*.

2.3.10. Oscilador programable.

El oscilador programable *Dallas DS1074* genera una señal de reloj hacia el *CPLD* y la *FPGA* con una frecuencia máxima de 100 MHz. Sin embargo, el oscilador puede programarse de forma que divida esta frecuencia entre números enteros, generando señales de frecuencia 100 MHz, 50 MHz, 33.3 MHz, ..., 48.7 KHz, que corresponde a un factor de división algo superior 2000. La programación del oscilador se realiza mediante la utilidad *GXSSETCLK*, incluida en el pack de herramientas *XSTools*. Como mostramos en la figura 2.21, para programar el oscilador se selecciona el *Puerto Paralelo*puerto paralelo del *PC* al que está conectado la tarjeta, se selecciona el factor de división, o en su caso, si el reloj será externo.

Para programarse, el *DS1075* debe estar en *Modo Programación*, esto es, con la salida del reloj a 5 V, que se consigue cortocircuitando los pines 1 y 2 del *Jumper J6*. Entonces, el oscilador se programa a través del *Puerto Paralelo*. El factor de división se guarda en una memoria *EEPROM* anexa al oscilador, para que el factor de división se mantenga aún cuando se desconecte la alimentación.

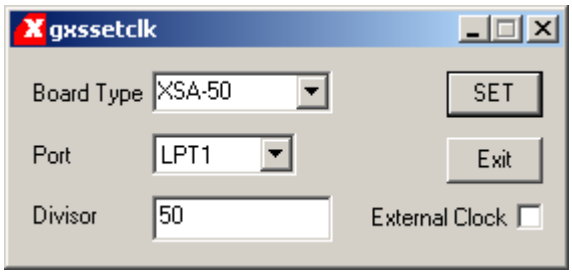


Figura 2.21: Herramienta *GXSSETCLK*.

En funcionamiento normal, el *Jumper J6* debe tener cortocircuitados sus pines 2 y 3. Entonces, la señal de reloj generada por el oscilador está conectada a una entrada dedicada del *CPLD*, y desde el *CPLD* llega hasta la *FPGA*. De esta forma, el *CPLD* controla la señal de reloj que gobierna la *FGPA*.

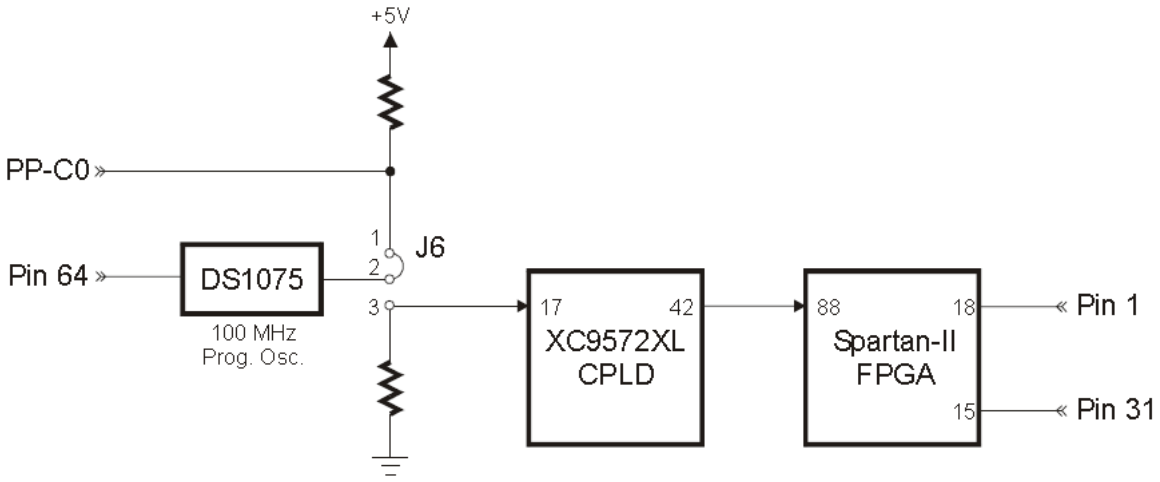


Figura 2.22: Esquema interno para la programación del oscilador.

Por último, para conseguir una frecuencia exacta y diferente a las que se consiguen con la programación del oscilador, se puede introducir una señal de reloj, siempre superior a 50 MHz, a través del *pin 64* de la placa de prototipos. Con la utilidad *GXSSETCLK* y los *Jumpers* (ver apartado 2.3.12), se debe configurar correctamente la tarjeta para activar el uso del reloj externo. Por tanto, señales externas pueden utilizarse como señales de reloj mediante la aplicación directa de las mismas a las entradas dedicadas de la *FPGA* a través de los pines correspondientes en la placa de prototipos.

2.3.11. Alimentación de la placa.

Existen tres métodos para alimentar la tarjeta.

1. **Alimentación de 9 VDC**.
   Esta tensión continua de 9 V se consigue a partir del transformador incluido en el set de la tarjeta. Debemos conectar la entrada del transformador a la línea eléctrica y su salida al *Jack J5* de la tarjeta. Los reguladores de tensión del circuito ajustarán la configuración de la tarjeta para generar los voltajes deseados en todos los lugares de la tarjeta. Estos reguladores, situados entre el *Jack J7* y el *Puerto Paralelo*, cogerán mucha temperatura mientras la placa esté siendo alimentada[4].

2. **Alimentación a partir del puerto *PS/2*.**
   También se puede usar el puerto *PS/2* para alimentar la tarjeta a través del conector *J4*. Para alimentar desde este conector, el *Jumper J7* debe tener cortocircuitados sus pines 1 y 2. La circuitería de regulación de voltaje ajustará las tensiones de toda la placa.
   Sin embargo, algunos puertos *PS/2* no pueden dar más de 0.5 A, con lo que las *FPGA* más veloces no funcionarían.

3. **Alimentación mediante pines de la tarjeta.**
   La tarjeta *XSA 50* puede pincharse en una placa de prototipos, de forma que el acceso a sus pines sea mucho más cómodo. Así, la alimentación puede llegar a la tarjeta a través de los pines dedicados, presentados en la tabla.

Tensión	**Pin**	**Comentario**
+5 V	2	
+3.3 V	22	Si se utiliza una fuente de 3.3 V, quitar el cortocircuito del *Jumper J7*. Si se utiliza una fuente de 5 V, dejar el cortocircuito del *Jumper J7*.
+2.5 V	54	Si se utiliza una fuente de 2.5 V, quitar el cortocircuito del *Jumper J2*. Si se utiliza una fuente de 3.3 V, dejar el cortocircuito del *Jumper J2*.
GND	52	

Cuadro 2.2: Pins de alimentación para la tarjeta *XSA 50*.

Es necesario prestar especial atención a la posición de los *Jumpers*, ya que una configuración incorrecta de los mismos, puede llevar a quemar la tarjeta.

[4]Si es necesario, acoplar un disipador de temperatura.

2.3.12. *Jumpers.*

En la tarjeta, existen 5 *Jumpers* que configuran su arquitectura para adaptarla a la función que deseemos hacer con ella. Además de la situación de funcionamiento normal, hay 3 eventos que obligan a cambiar estos *Jumpers* para modificar la configuración de la tarjeta.

1. Descarga de *bitstreams* a la *FPGA* mediante el software *Xilinx iMPACT* (ver apartado 2.3.7).

2. Re-programación de la frecuencia del oscilador (ver apartado 2.3.10).

3. Cambio de las fuentes de alimentación de la tarjeta (ver apartado 2.3.11).

En la siguiente tabla, se presenta la configuración de cada *Jumper*[5] con la explicación de su funcionalidad:

Jumper	**Posición**	**Comentario**
J2	On*	**Cortocircuito:** si se aplica una fuente de 3.3 V al pin de 2.5 V.
	Off	**No Cortocircuito:** si se aplica una fuente de 2.5 V al pin de 2.5 V.
J6	1-2	**Cortocircuito 1-2:** para programar el oscilador.
	2-3*	**Cortocircuito 2-3:** funcionamiento normal.
J7	1-2*	**Cortocircuito 1-2:** si se aplica una fuente de 5 V al pin de 3.3 V.
	2-3	**Cortocircuito 2-3:** si se aplica la alimentación de 9 VDC.
J9	1-2	**Cortocircuito 1-2:** el *bitstream* se descarga mediante *Xilinx iMPACT*.
	2-3*	**Cortocircuito 2-3:** el *bitstream* se descarga mediante *GXSLoad*.
J10	N/A	Permite acceso de 5 V y GND a la placa.

Cuadro 2.3: Configuración de los *jumpers* de la tarjeta *XSA 50*.

Es muy importante que los *Jumpers* estén en su posición correcta. Si no lo están, la tarjeta tendrá una configuración no adecuada para la alimentación a la que está siendo sometida, corriendo grave riesgo de ser quemada o inutilizada.

[5]* Configuración por defecto

2.3.13. Posicionamiento de los componentes de la tarjeta *XSA 50*.

Analizados todos los componentes en cuanto a su funcionamiento, conexiones y posibilidades, para tener un visión global de la tarjeta, en la siguiente figura presentamos un esquema que ayuda a situar todos los elementos de la tarjeta que hemos ido viendo en los apartados anteriores:

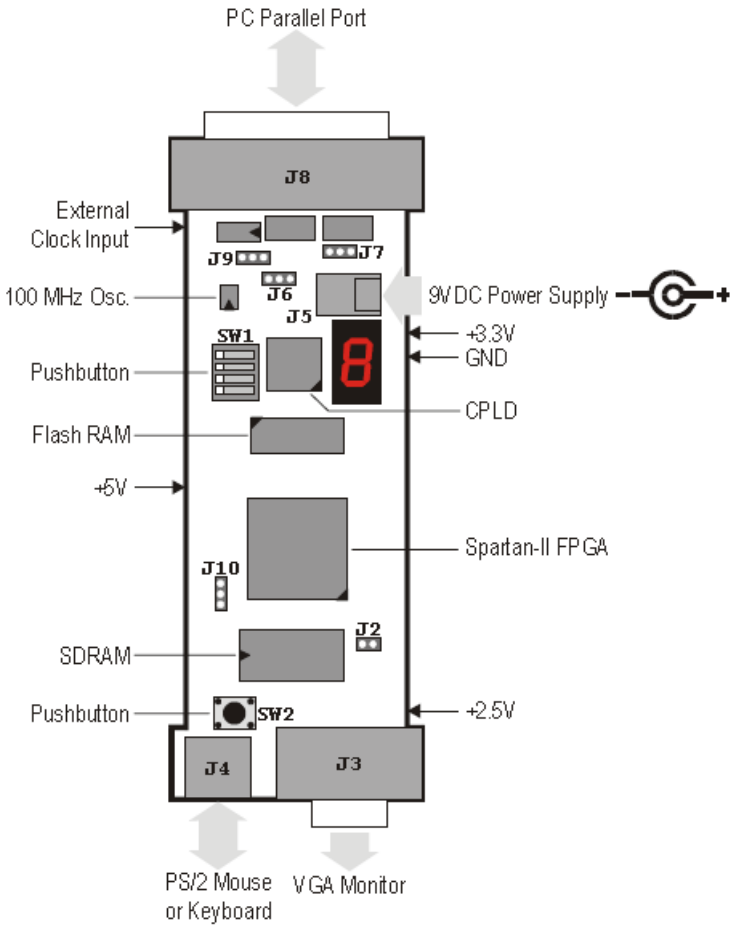


Figura 2.23: Esquema de componentes de la tarjeta *XSA 50*.

En esta figura podemos encontrar los componentes: *FPGA*, *CPLD*, memoria *Flash ROM*, *Dip Switches*, *Display de 7 segmentos*, memoria *SDRAM*, *Puerto Paralelo*, puerto *PS/2*, puerto *VGA*, oscilador programable, circuitería de alimentación y *Jumpers* de configuración.

2.3.14. Comprobación de funcionamiento de la *FPGA*: *GXSTest*.

Para terminar este capítulo y todo lo concerniente a la tarjeta *XSA 50*, presentamos la utilidad del testeado de la placa. Una vez configurados los *Jumpers* de la tarjeta y alimentada, las *GXSTools* contienen una utilidad que comprueba si esta configuración y alimentación es correcta, y si la tarjeta está dispuesta para un funcionamiento normal. Esta utilidad tiene el nombre de *GXSTest*.

El entorno de la utilidad es muy sencillo. Basta elegir el tipo de placa, en nuestro caso la *XSA 50*, y el *Puerto Paralelo* del PC al que está conectada. Entonces, presionando en el botón *Test*, se descarga un *bitstream* que configura la tarjeta para hacer un test de la misma.

Figura 2.24: Utilidad *GXSTest* para comprobar el funcionamiento de la tarjeta.

Si el test es correcto, en el display de la tarjeta se verá la sucesión 0, 1, 2, 3 y 0, y quedará marcado este 0. De no ser correcto, marcará una E. En ambos caso, una ventana informará del éxito o fracaso del test. A partir de este momento y superado el test, la tarjeta está dispuesta para utilizarse.

Capítulo 3

El procesador.

En este capítulo, presentamos el *Procesador RISC* que hemos diseñado. En el primer apartado, comentamos las características más generales del *Procesador RISC*. Después, analizaremos instrucción por instrucción, todas las instrucciones del *set* y presentaremos una tabla-resumen del *Set de Instrucciones*. Por último, presentaremos el bloque principal de diseño del *Procesador RISC*, con sus dos bloques, *Unidad de Control* y *Unidad de Proceso*, y las líneas que los unen.

3.1. Procesador *RISC*.

Un *Procesador RISC* es un *Procesador* con un conjunto de instrucciones reducido, en oposición a los *Procesadores CISC* que tienen un conjunto complejo, o grande, de instrucciones. Considerando este aspecto, a primera vista, parecen mucho mejores los *Procesadores CISC* que los *RISC*. Pero existen otros factores con los que podemos comparar ambos tipos de *Procesadores*.

El hecho de ejecutar instrucciones complejas implica hacerlo en muchos ciclos de reloj, mientras que la ejecución de instrucciones sencillas puede realizarse en pocos ciclos de reloj. Así, los *Procesadores RISC* son más rápidos en cuanto a tiempos de ejecución de instrucciones y más flexibles en cuanto a la ejecución de las mismas. En este aspecto, sin duda, presentan muchas ventajas respecto a los *Procesadores CISC*.

Por otro lado, el diseño de los *Procesadores RISC* es mucho más sencillo que el de los *Procesadores CISC*. Las instrucciones que utilizan los *Procesadores RISC* son más simples que las que utilizan los *Procesadores CISC* y, por tanto, requieren menos recursos para completar su ejecución, lo que implica una mayor sencillez del *datapath*.

Sin embargo, no todo son ventajas a favor de los *Procesadores RISC*. La realización de programas para *Procesadores CISC* es mucho más simple que para *Procesadores RISC*, y la razón es muy obvia. La complejidad de las instrucciones significa que con una sola instrucción se pueden llevar a cabo muchas operaciones. Esta característica simplifica el trabajo de los programadores ya que, para realizar varias operaciones, en un *Procesador CISC* bastará una sola instrucción, mientras que en un *Procesador RISC* serán necesarias varias instrucciones.

Hemos visto que, en algunos aspectos, un *Procesador RISC* presenta ventajas respecto a uno *CISC*, mientras que en otros aspectos es al revés. En cualquier caso, nosotros hemos realizado el diseño de un *Procesador RISC* de propósito general, cuyas características más importantes presentamos en los siguientes puntos.

3.1.1. *Arquitectura Harvard.*

Como hemos visto en el apartado 2.3, la *XSA 50* dispone de dos memorias. La primera de ellas es una memoria *Flash ROM* de $256Kb$ y la segunda es una memoria *SDRAM* de $4Mb$.

La memoria *Flash ROM* es una memoria no volátil, grabable y borrable eléctricamente, y grabable y borrable *on chip*, esto es, mediante programación. Además, la memoria Flash es más rápida, de mayor densidad y consume menos que una memoria *ROM* tradicional.

La memoria *SDRAM* es una memoria volátil, dinámica y síncrona. Con el concepto de dinámica, se representa la degradación que, con el tiempo, sufren los datos almacenados. Para que no se pierdan esos valores, es necesario restaurarlos cada cierto tiempo. Esta operación se denomina refresco. Por otro lado, las memorias síncronas se sincronizan con la velocidad del *Procesador*, pudiendo obtener información en cada ciclo de reloj y evitando estados de espera. Las memorias *SDRAM* tienen costes bajos y prestaciones muy rápidas. Sin embargo, necesitan un control y un refresco constante, razón por la cual, consumen mucha energía.

Entonces, conocidas las propiedades de las dos memorias de las que disponemos, llegamos a la conclusión de que la mejor arquitectura para nuestro *Procesador* es un tipo de arquitectura que separe la memoria de datos y la memoria de programa. Este tipo de arquitectura recibe el nombre de *Arquitectura Harvard*.

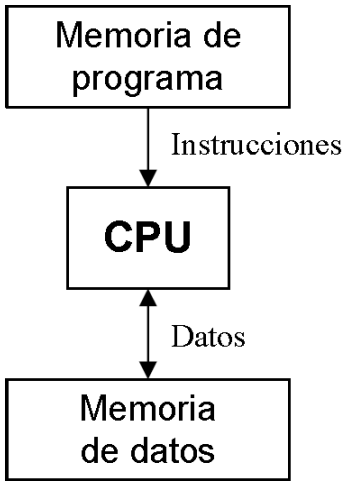


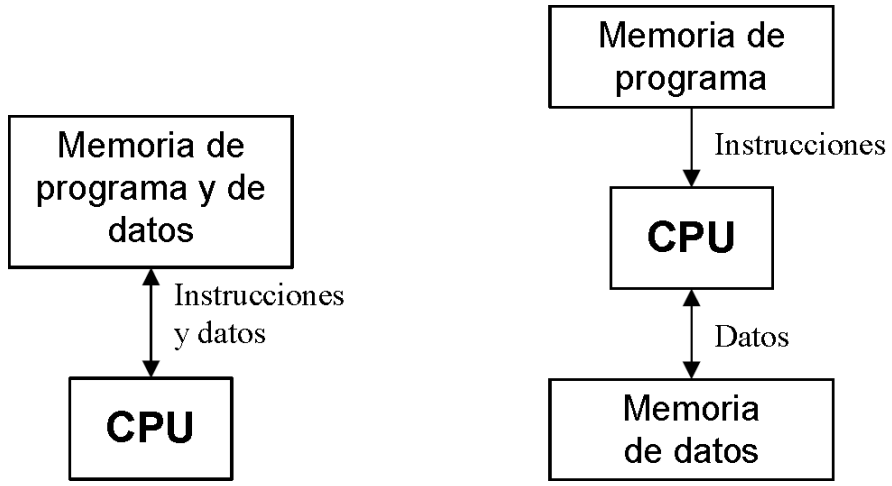
Figura 3.1: Arquitectura *Harvard*.

La memoria de datos será la memoria *SDRAM*, que almacenará resultados intermedios y datos no permanentes generados durante la ejecución de programas.

La memoria de programa será la memoria *Flash ROM*. Cada programa se grabará[1] en ella antes de descargar el diseño del *Procesador* a la *FPGA*, y después no será grabable *on chip* en ningún momento durante la ejecución de programas.

La *arquitectura Harvard* presenta dos ventajas fundamentales respecto a otro tipo de arquitectura de memoria única, como la arquitectura *Von Neumann*. Estas ventajas son:

- Dos buses independientes y de distinto ancho. Permite programas optimizados, más cortos y rápidos.
- El tiempo de acceso a datos e instrucciones puede superponerse. Consigue mayor rapidez de ejecución.



(a) *Arquitectura Von Neumann* (b) *Arquitectura Harvard*

Figura 3.2: Arquitecturas del *Procesador*.

3.1.2. Set de instrucciones.

Como hemos dicho anteriormente, por definición, el set de instrucciones de un *Procesador RISC* es reducido. Para nuestro *Procesador*, hemos diseñado un set de instrucciones pequeño y hemos clasificado cada instrucción dentro de uno de estos tres grandes grupos:

[1]Para la grabación de programas en la memoria *Flash ROM*, se utilizará la herramienta *GXSLOAD* incluida en las *XSTools* de *Xess Corp.*

1. **Instrucciones de tratamiento de datos**
   - Operaciones aritméticas: suma y resta.
   - Operaciones lógicas: and, or, xor, not.
   - Desplazamientos aritmético y lógico y Rotaciones.
   - Comparación de datos.
   - Testeo de bit.
2. **Instrucciones de transferencia de datos**
   - Escritura en la *Memoria de Datos*.
   - Lectura de la *Memoria de Datos*.
   - Escritura en el *Banco de Registros*.
   - Entrada y salida por el *Puerto*.
3. **Instrucciones de control de programa**
   - Salto condicional (Z, NZ, C y NC) o incondicional.
   - Salto relativo o absoluto.
   - Llamada a subrutina.
   - Retorno de subrutina.
   - No operación.

Además, cada una de estas instrucciones tiene varias posibilidades que ya analizamos en profundidad en posteriores apartados.

También, hemos dejado *abierta* la estructura del *Set de Instrucciones* para posibles ampliaciones, ya que no hemos completado las 64 instrucciones que permiten seleccionar los 6 bits que forman el *Código de Operación* y el *Tipo de Operación*.

3.1.3. *Palabra de Instrucción.*

La *Palabra de Instrucción* es la codificación de instrucciones que el *Procesador* es capaz de interpretar. Se guarda en el *Registro de Instrucción (IR)*, y su contenido procede de la memoria de programas.

En nuestro *Procesador*, hemos definido una *Palabra de Instrucción* de 32. Además, hemos diseñado dos formatos para esta palabra según el tipo de instrucción a que hagan referencia.

Si la instrucción toma datos de los registros únicamente, el formato es el siguiente:

***Palabra de Instrucción* (tipo 1)**															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Código				**Tipo**		**Campo 3**				**Campo 1**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Campo 2				**ID 2**		*Vacío*									

Cuadro 3.1: *Palabra de Instrucción* (tipo 1)

Si la instrucción utiliza o toma datos de un literal de entrada, el formato de la *Palabra de Instrucción* tiene el siguiente aspecto:

***Palabra de Instrucción* (tipo 2)**															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Código				**Tipo**		**Campo 3**				**Campo 1**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Literal															

Cuadro 3.2: *Palabra de Instrucción* (tipo 2)

en donde

- **Código (de Operación):** 4 bits que contienen la codificación de la instrucción.
- **Tipo (de Operación):** 2 bits que modifican la instrucción.
- **Campo 3:** 4 bits que seleccionan el registro destino para las operaciones de escritura en registro o la dirección base para las operaciones relacionadas con la memoria de datos.
- **Campo 1:** 4 bits que seleccionan un registro de salida del banco.
- **ID 1:** 2 bits que seleccionan las operaciones de post-incremento y post-decremento del dato contenido en el registro seleccionado por el campo 1.

- **Campo 2:** 4 bits que seleccionan el otro registro de salida del banco.
- **ID 2:** 2 bits que seleccionan las operaciones de post-incremento y post-decremento del dato contenido en el registro seleccionado por el campo 2.
- **Literal:** 16 bits que forman el dato literal que entra en la *Unidad de Proceso*.

Los dos tipos tienen en común los campos **Código**, **Tipo**, **Campo 3**, **Campo 1** e **ID 1**. Los campos **Campo 2** e **ID 2** y **Literal** se interpretan según el tipo de operación realizada.

3.1.4. Modos de direccionamiento.

Los modos de direccionamiento hacen referencia a las distintas posibilidades que tiene el *Procesador* de obtener datos para realizar operaciones.

Como veremos en el apartado 3.1.6, nuestro *Procesador* dispone de un banco de 16 registros, para almacenar los datos con los que trabaja. Además, dispone de una *Memoria de Datos (SDRAM)*, que también permite almacenar datos.

En las operaciones **no relacionadas** con la *Memoria SDRAM*, el origen y el destino de los datos son registros del banco, o en algún caso, el origen puede ser un literal que procede del *Registro de Instrucción*. Los registros son siempre seleccionados por el *Registro de Instrucción*.

Las operaciones **relacionadas** con la memoria siempre tienen como origen o destino el banco de registros. Esto es, al escribir en memoria, el dato o la dirección[2] proviene del banco de registros o de un literal, y al leer de memoria, el dato se almacena en el banco de registros. Por tanto, no es posible tomar los datos de la memoria y realizar operaciones sobre ellos directamente, sino que deben introducirse en un registro, a partir de él, tratarlos y, posteriormente, y si se desea, volver a escribirlos sobre la memoria de datos.

Las formas de acceder a la memoria *SDRAM* son varias:

- Para escritura:
  - Dato y Dirección contenidos en los registros del banco.
  - Dato literal y Dirección contenida en los registros del banco.
  - Dato en los registros del banco y Dirección literal[3].

[2]La dirección de acceso a la memoria *SDRAM* está dividida en dos partes: base y desplazamiento. La base siempre se direcciona desde un registro y el desplazamiento puede obtenerse desde registro o desde un literal.

[3]En adelante, Dirección literal significa desplazamiento de la dirección asociada a la *SDRAM* literal.

- Para lectura:
  - Dirección origen en los registros del banco
  - Dirección origen literal.

Con todas las particularidades que hemos presentado, podemos concluir que, en nuestro *Procesador*, permitimos los siguientes modos de direccionamiento:

1. **Directo**: Los operandos origen y destino se encuentra en el registro especificado en el campo.
2. **Indirecto**: El dato de la memoria se obtiene a partir de la dirección especificada en los registros del campo.
3. **Inmediato**: Uno de los operandos origen es un literal que viene de la *Palabra de Instrucción*, y el destino se encuentra en el banco de registros.

Además, todos los registros accesibles en estos modos de direccionamiento tienen la posibilidad de realizar un post-incremento o post-decremento de su contenido.

3.1.5. *Pipeline.*

La técnica de *Pipeline* busca implementar la idea de trabajo en paralelo en el *Procesador*. Consiste en dividir la ejecución de la instrucción en bloques independientes que se ejecutan en paralelo. Con esta configuración, se aumenta mucho la velocidad del *Procesador*.

Hemos diseñado un *Procesador* que utiliza un *Pipeline* de 2 etapas para mejorar sus prestaciones.

1. La primera etapa es la **fase de búsqueda de instrucción**. Las instrucciones se encuentran en la memoria de programa (*Flash Rom*), cuyo bus de datos tiene una anchura de 8 bits. El *Registro de Instrucción (IR)* tiene una anchura de 32 bits. Por tanto, para tener el *Registro de Instrucción* completo, es necesario realizar 4 accesos a la memoria de programas. Por otro lado, cada acceso a esta memoria *Flash ROM* necesita dos ciclos de reloj. Entonces, considerando estas dos ideas, la conclusión es que la fase de búsqueda de instrucción tiene una duración de 8 (4 x 2) ciclos de reloj.
2. La segunda etapa es la **fase de ejecución de la instrucción**. Como hemos implementado *Pipeline*, y la fase de búsqueda de instrucción dura 8 ciclos de reloj, hemos diseñado la *Memoria de Micro-instrucciones* de forma que la fase de ejecución de todas las instrucciones dure, también, 8 ciclos de reloj.

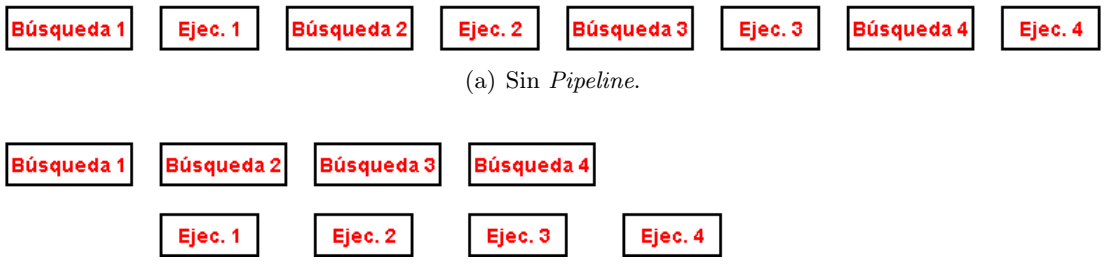
La forma de implementar el *Pipeline* es la siguiente. Utilizamos el *Registro de Instrucción* como *registro tampón*. En un instante dado, la *Unidad de Proceso* está realizando las operaciones que se le señalan en el *IR* y en la *Memoria de Micro-instrucciones*. Mientras tanto, la *Unidad de Control* está leyendo las nuevas instrucciones desde la memoria de programa, y almacenándolas en un registro temporal.

Como ambas etapas están sincronizadas, en el octavo ciclo de reloj se ha terminado la ejecución de la instrucción y se ha completado la lectura del nuevo *IR*. Entonces, en ese punto, se actualiza el nuevo *IR*, y comienza a ejecutarse la instrucción. Mientras tanto, la unida de control ya estará leyendo la siguiente instrucción de la memoria de programas.

Por último, comprendida la filosofía del *Pipeline* que hemos implementado, puntualizamos que, exactamente, las fases de búsqueda y ejecución de instrucciones no comienzan en el mismo instante de tiempo, sino que la fase de ejecución de instrucción comienza un ciclo de reloj después de que comience la fase de búsqueda. La razón es la estructura del *Registro de Instrucción*. Para cargar el *Registro de Instrucción* desde los registros temporales, se necesita un ciclo de reloj, es decir, no es inmediato. Por tanto, la fase de ejecución no comienza instantáneamente sino que está desfasada un ciclo de reloj respecto del comienzo de la fase de búsqueda.

Por último, para comprobar las ventajas del uso de la técnica de *pipeline*, hacemos una pequeña comparativa de los tiempos de ejecución de un programa con ella y sin ella. Consideramos un programa de 4 instrucciones.

En primer lugar, realizamos una comparativa gráfica de las dos situaciones.



(a) Sin *Pipeline*.

(b) Con *Pipeline*.

Figura 3.3: Comparativa gráfica.

Gráficamente, comprobamos que, para la ejecución de un mismo programa de 4 instrucciones, la ejecución usando la técnica del *pipeline* reduce significativamente la duración del programa.

Esta comprobación se puede hacer también de forma numérica. Para este ejemplo, consideraremos que:

- El tiempo de la fase de búsqueda de instrucción son 8 ciclos de reloj.
- El tiempo medio de la fase de ejecución de la instrucción son 6 ciclos de reloj.
- Consideraremos un programa de 4 instrucciones.

Entonces,

Sin *pipeline*. El número de ciclos de reloj que tardaría el programa es:

$$N = \underbrace{8+6}_{Inst,1} + \underbrace{8+6}_{Inst,2} + \underbrace{8+6}_{Inst,3} + \underbrace{8+6}_{Inst,4}$$

$$\boxed{N = 56 \text{ ciclos de reloj}}$$

Con *pipeline*. El número de ciclos de reloj que tardaría el programa es:

$$N = \underbrace{8}_{Busq,1} + \underbrace{8}_{Busq,2} + \underbrace{8}_{Busq,3} + \underbrace{8}_{Busq,4} + \underbrace{6}_{Ejec,4}$$

$$\boxed{N = 38 \text{ ciclos de reloj}}$$

Matemáticamente, también llegamos a la misma conclusión. La técnica del *pipeline* aumenta significativamente la velocidad de ejecución de las instrucciones de un programa. Cuanto más grande sea el programa, más acentuada es esta ventaja.

Sin embargo, la técnica de *pipeline* no es tan efectiva si el programa realiza saltos constantemente. La fase de ejecución de un salto coincide con la fase de búsqueda de la siguiente instrucción escrita en la *memoria de instrucciones*. Sin embargo, esa instrucción no es la siguiente que se va a ejecutar, sino la de la dirección de salto. Ante esta situación, caben dos posiblidades:

Posibilidad 1. **Vaciar la burbuja de ejecución del programa**, realizando una nueva fase de búsqueda sin ejecutar ninguna instrucción, o más estrictamente hablando, ejecutando la operación de *No operación*. Para ello, después de cada instrucción de salto, el programador debe incluir, explícitamente, la instrucción de *No Operación* en el código de su programa.
Esta solución es la más simple e intuitiva, y está orientada a programadores noveles o inexpertos.

Posibilidad 2. **Utilizar la técnica de salto anticipativo**. Esta técnica tiene el objetivo de no desaprovechar la instrucción que está en la burbuja de ejecución del programa al realizar un salto. Para ello, si la instrucción de salto es completamente independiente de la instrucción anterior, como es el caso de un salto incondicional, entonces, dicha instrucción de salto puede escribirse en el código del programa delante de esta otra instrucción independiente. De esta forma, mientras comienza la fase de búsqueda de la instrucción con la nueva dirección de salto, se está ejecutando esa otra instrucción independiente, y así no se pierde el ciclo de instrucción que se había perdido con la ejecución de una instrucción de *No Operación*, que vimos en la *Posibilidad 1*.
Esta solución requiere un mayor conocimiento del *Procesador* y de su programación, pero da mejores prestaciones que la primera posibilidad. Por esta razón, solo la recomendamos para programadores más expertos.

3.1.6. Registros y ortogonalidad.

Los registros son dispositivos de almacenamiento de datos implementados en el interior del *Procesador* y cuyo acceso (lectura o escritura), es mucho más rápido que a la memoria exterior de datos. Hemos diseñado un *Procesador* con un *Banco de 16 Registros de 16 bits* cada uno. Todos son accesibles, ya sea para escritura o lectura, en cualquier instante. El contenido de todos puede salir por cualquiera de los dos canales de salida del banco. Además, todos poseen la capacidad de post-incrementarse o post-decrementarse individualmente con cada acceso.

Por tanto, todos los registros son física y funcionalmente iguales, con lo que el *Banco de Registros* es completamente ortogonal.

3.1.7. Escalabilidad.

Con el término *Escalabilidad*, hacemos referencia a la propiedad que informa sobre la facilidad de aumentar la capacidad y recursos del *Procesador*.

El diseño del *Procesador* se ha realizado mediante bloques jerárquicos.Este hecho permite, de una forma muy sencilla, ampliar la arquitectura para doblar el número de registros o la anchura de los mismos. Por tanto, la arquitectura del *Procesador* es fácilmente escalable.

Sin embargo, para ampliar el número de registros, también sería necesario modificar el formato de la *Palabra de Instrucción*. Los campos de selección los registros actualmente tienen 4 bits, que permiten direccionar 16 registros. Si doblamos el número de registros, será necesario incluir un bit más en los campos de registro, para direccionar esos 32 registros.

En resumen, la arquitectura es fácilmente escalable aunque la *Palabra de Instrucción* no permite que la escalabilidad del *Procesador* sea inmediata.

3.1.8. Puerto de entrada salida.

Los puertos permiten la comunicación del *Procesador* con el mundo exterior durante el mismo proceso de ejecución de programas. Permiten la entrada de datos que pueden utilizarse como los datos tradicionales procedentes del banco de registros o de la memoria de datos. Los puertos de salida permiten la comunicación del *Procesador* hacia dispositivos externos como, por ejemplo, displays, que facilitan, precisamente, la comunicación del *Procesador* con el usuario.

Este *Procesador* dispone de un puerto bi-direccional de entrada/salida de 16 bits. El puerto está asociado al multiplexor de salida (y realimentación) de la *Unidad de Proceso* de la siguiente forma:

- **En dirección entrada,** el dato entra a este multiplexor y, a través suyo, se dirige hacia el banco de registros.
- **En dirección salida,** el dato proviene de la salida de este multiplexor.

Fuera del *Procesador*, gestionamos la dirección del puerto con *Buffers Tri-estado*, que permiten la transferencia de datos en un sentido, en el opuesto, o en ninguno de los dos.

3.1.9. Pila.

Una pila es un bloque de memoria interna, dedicada, de tamaño limitado, diferente de las memorias de datos y programa y organizada en forma circular. Generalmente, se utiliza para guardar las direcciones de retorno de subrutinas e interrupciones.

En nuestro *Procesador*, para poder ejecutar subrutinas, hemos implementado una pila circular de 16 posiciones en la parte más alta de la *Memoria SDRAM*. Esta pila permite anidar hasta 16 niveles de subrutinas. En cada una de estas posiciones, guardaremos el contenido del *Contador de Programa (PC)* actual antes de hacer el salto de subrutina. Y, al realizar el retorno de subrutina, se recuperará el valor del *PC* desde la pila.

Además, la pila tiene asociado un registro *Puntero de Pila (SP)* que guarda la dirección de la primera posición libre.

3.2. Set de instrucciones.

En los siguientes apartados, presentamos el set de todas las instrucciones que hemos diseñado para nuestro *Procesador*.

3.2.1. Instrucciones de tratamiento de datos.

I OPERACIONES ARITMÉTICAS.

- **Suma tipo 1:** Suma sin acarreo. Sumando 1 en registro Campo 1. Sumando 2 en registro Campo 2. Destino en registro Campo 3.

$$R3 = R1(\pm) + R2(\pm)$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**1**	**0**	**0**	**0**	**0**	**Destino**				**Sumando 1**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sumando 2				**ID 2**		Vacío									

Cuadro 3.3: Operación Aritmética. Suma tipo 1.

- **Suma tipo 2:** Suma sin acarreo. Sumando 1 en registro Campo 1. Sumando 2 en literal. Destino en registro Campo 3.

$$R3 = R1(\pm) + literal$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**1**	**0**	**0**	**0**	**1**	**Destino**				**Sumando 1**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sumando Literal															

Cuadro 3.4: Operación Aritmética. Suma tipo 2.

- **Suma tipo 3:** Suma con acarreo. Sumando 1 en registro Campo 1. Sumando 2 en registro Campo 2. Destino en registro Campo 3.

$$R3 = R1(\pm) + R2(\pm) + C$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**1**	**0**	**0**	**1**	**0**	**Destino**				**Sumando 1**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sumando 2				**ID 2**		Vacío									

Cuadro 3.5: Operación Aritmética. Suma tipo 3.

- **Suma tipo 4:** Suma con acarreo. Sumando 1 en registro Campo 1. Sumando 2 en literal. Destino en registro Campo 3.

$$R3 = R1(\pm) + literal + C$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**1**	**0**	**0**	**1**	**1**	**Destino**				**Sumando 1**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sumando Literal															

Cuadro 3.6: Operación Aritmética. Suma tipo 4.

- **Resta tipo 1:** Resta sin acarreo. Minuendo en registro Campo 1. Sustraendo en registro Campo 2. Destino en registro Campo 3.

$$R3 = R1(\pm) - R2(\pm)$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**1**	**0**	**1**	**0**	**0**	**Destino**				**Minuendo**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sustraendo				**ID 2**		Vacío									

Cuadro 3.7: Operación Aritmética. Resta tipo 1.

- **Resta tipo 2:** Resta sin acarreo. Minuendo en registro Campo 1. Sustraendo en literal. Destino en registro Campo 3.

$$R3 = R1(\pm) - literal$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**1**	**0**	**1**	**0**	**1**	**Destino**				**Minuendo**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sustraendo Literal															

Cuadro 3.8: Operación Aritmética. Resta tipo 2.

- **Resta tipo 3:** Resta sin acarreo. Minuendo en registro Campo 1. Sustraendo en registro Campo 2. Destino en registro Campo 3.

$$R3 = R1(\pm) - R2(\pm) - C$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**1**	**0**	**1**	**1**	**0**	**Destino**				**Minuendo**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sustraendo				**ID 2**		Vacío									

Cuadro 3.9: Operación Aritmética. Resta tipo 3.

- **Resta tipo 4:** Resta con acarreo. Minuendo en registro Campo 1. Sustraendo en literal. Destino en registro Campo 3.

$$R3 = R1(\pm) - literal - C$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**1**	**0**	**1**	**1**	**1**	**Destino**				**Minuendo**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sustraendo Literal															

Cuadro 3.10: Operación Aritmética. Resta tipo 4.

II OPERACIONES LÓGICAS.

- **And tipo 1:** And. Dato en registro Campo 1. Máscara en registro Campo 2. Destino en registro Campo 3.

$$R3 = R1(\pm)\ and\ R2(\pm)$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**1**	**1**	**0**	**0**	**0**	**Destino**				**Dato**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Máscara				**ID 2**		Vacío									

Cuadro 3.11: Operación Lógica. And tipo 1.

- **And tipo 2:** And. Dato en registro Campo 1. Máscara en literal. Destino en registro Campo 3.

$$R3 = R1(\pm)\ and\ literal$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**1**	**1**	**0**	**0**	**1**	**Destino**				**Dato**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Máscara Literal															

Cuadro 3.12: Operación Lógica. And tipo 2.

- **Or tipo 1:** Or. Dato en registro Campo 1. Máscara en registro Campo 2. Destino en registro Campo 3.

$$R3 = R1(\pm)\ or\ R2(\pm)$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**1**	**1**	**0**	**1**	**0**	**Destino**				**Dato**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Máscara				**ID 2**		Vacío									

Cuadro 3.13: Operación Lógica. Or tipo 1.

- **Or tipo 2:** Or. Dato en registro Campo 1. Máscara en literal. Destino en registro Campo 3.

$$R3 = R1(\pm)\ or\ literal$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**1**	**1**	**0**	**1**	**1**	**Destino**				**Dato**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Máscara Literal															

Cuadro 3.14: Operación Lógica. Or tipo 2.

- **Xor tipo 1:** Xor. Dato en registro Campo 1. Máscara en registro Campo 2. Destino en registro Campo 3.

$$R3 = R1(\pm)\ xor\ R2(\pm)$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**1**	**1**	**1**	**0**	**0**	**Destino**				**Dato**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Máscara				**ID 2**		Vacío									

Cuadro 3.15: Operación Lógica. Xor tipo 1.

- **Xor tipo 2:** Xor. Dato en registro Campo 1. Máscara en literal. Destino en registro Campo 3.

$$R3 = R1(\pm)\ xor\ literal$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**1**	**1**	**1**	**0**	**1**	**Destino**				**Dato**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Máscara Literal															

Cuadro 3.16: Operación Lógica. Xor tipo 2.

III DESPLAZAMIENTOS Y ROTACIONES.

- **Desplazamiento tipo 1:** Desplazamiento Aritmético. Con Acarreo. A la Derecha. Dato en registro Campo 1. Destino en registro Campo 3.

$$R3 = despA(\rightarrow)\ R1 : C$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**1**	**1**	**0**	**0**	**0**	**Destino**				**Dato**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.17: Desplazamiento tipo 1.

- **Desplazamiento tipo 2:** Desplazamiento Aritmético. Con Acarreo. A la Izquierda. Dato en registro Campo 1. Destino en registro Campo 3.

$$R3 = despA(\leftarrow)\ R1 : C$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**1**	**1**	**0**	**1**	**0**	**Destino**				**Dato**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.18: Desplazamiento tipo 2.

- **Desplazamiento tipo 3:** Desplazamiento Lógico. Con Acarreo. A la Derecha. Dato en registro Campo 1. Destino en registro Campo 3.

$$R3 = despL(\rightarrow)\ R1 : C$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**1**	**1**	**1**	**0**	**0**	**Destino**				**Dato**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.19: Desplazamiento tipo 3.

- **Desplazamiento tipo 4:** Desplazamiento Lógico. Con Acarreo. A la Izquierda. Dato en registro Campo 1. Destino en registro Campo 3.

$$R3 = despL(\leftarrow)\ R1 : C$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**1**	**1**	**1**	**1**	**0**	**Destino**				**Dato**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.20: Desplazamiento tipo 4.

- **Rotación tipo 1:** Rotación. Con Acarreo. A la Derecha. Dato en registro Campo 1. Destino en registro Campo 3.

$$R3 = rot(\rightarrow)\ R1 : C$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**1**	**0**	**0**	**0**	**0**	**Destino**				**Dato**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.21: Rotación tipo 1.

- **Rotación tipo 2:** Rotación. Con Acarreo. A la Izquierda. Dato en registro Campo 1. Destino en registro Campo 3.

$$R3 = rot(\leftarrow)\ R1 : C$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**1**	**0**	**0**	**1**	**0**	**Destino**				**Dato**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.22: Rotación tipo 2.

- **Rotación tipo 3:** Rotación. Sin Acarreo. A la Derecha. Dato en registro Campo 1. Destino en registro Campo 3.

$$R3 = rot(\rightarrow)\ R1$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**1**	**0**	**1**	**0**	**1**	**Destino**				**Dato**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.23: Rotación tipo 3.

- **Rotación tipo 4:** Rotación. Sin Acarreo. A la Izquierda. Dato en registro Campo 1. Destino en registro Campo 3.

$$R3 = rot(\leftarrow)\ R1$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**1**	**0**	**1**	**1**	**1**	**Destino**				**Dato**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.24: Rotación tipo 4.

IV COMPARACIÓN DE DATOS.

- **Comparación tipo 1:** Comparación. Dato 1 en registro Campo 1. Dato 2 en registro Campo 2.

$$Flags = comp\ R1,\ R2$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**1**	**0**	**0**	Vacío				**Dato 1**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dato 2				**ID 2**		Vacío									

Cuadro 3.25: Comparación tipo 1.

- **Comparación tipo 2:** Comparación. Dato 1 en registro Campo 1. Dato 2 en literal.

$$Flags = comp\ R1,\ literal$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**1**	**0**	**1**	Vacío				**Dato 1**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dato Literal															

Cuadro 3.26: Comparación tipo 2.

V TEST DE BIT.

- **Test de Bit:** Test de Bit. Dato en registro Campo 1. Bits de máscara en literal.

$$Flags = TestBit\ R1(\pm),\ literal$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**1**	**0**	**0**	**0**	Vacío				**Dato**				**ID**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Máscara Literal															

Cuadro 3.27: Bit test.

3.2.2. Instrucciones de transferencia de datos.

I ESCRITURA Y LECTURA EN MEMORIA.

- **Escritura en Memoria tipo 1:** Escritura en Memoria. Dato en registro Campo 2. Dirección en registro Campo3:Campo1.

$$Memo(R3 : R1) = Mwrite\ R2$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**0**	**1**	**1**	**0**	**0**	**Dirección alta**				**Dirección baja**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dato				**ID 2**		Vacío									

Cuadro 3.28: Escritura en Memoria tipo 1.

- **Escritura en Memoria tipo 2:** Escritura en Memoria. Dato en literal. Dirección en registro Campo3:Campo1.

$$Memo(R3 : R1) = Mwrite\ literal$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**0**	**1**	**1**	**0**	**1**	**Dirección alta**				**Dirección baja**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dato Literal															

Cuadro 3.29: Escritura en Memoria tipo 2.

- **Escritura en Memoria tipo 3:** Escritura en Memoria. Dato en Registro Campo 1. Dirección en registro Campo3:literal.

$$Memo(R3 : literal) = Mwrite\ R1$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**0**	**1**	**1**	**1**	**0**	**Dirección alta**				**Dato**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dirección baja Literal															

Cuadro 3.30: Escritura en Memoria tipo 3.

- **Lectura de Memoria tipo 1:** Lectura de Memoria. Dirección de memoria en registro Campo 3:Campo1. Destino en Registro Campo 2.

$$R2 = Mread\ R3 : R1$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**1**	**0**	**1**	**0**	**0**	**Dirección alta**				**Dirección baja**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Destino				**ID 2**		Vacío									

Cuadro 3.31: Lectura de Memoria tipo 1.

- **Lectura de Memoria tipo 2:** Lectura de Memoria. Dirección de memoria en registro Campo 3:literal. Destino en Registro Campo 1.

$$R1 = Mread\ R3 : literal$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**1**	**0**	**1**	**1**	**0**	**Dirección alta**				**Destino**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dirección baja Literal															

Cuadro 3.32: Lectura de Memoria tipo 2.

II ESCRITURA EN REGISTRO.

- **Escritura en Registro:** Escritura en Registro. Dato en literal. Dirección en registro Campo3.

$$R3 = Rwrite\ literal$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**0**	**1**	**1**	**1**	**1**	**Destino**				Vacío					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dato Literal															

Cuadro 3.33: Escritura en Registro.

III ENTRADA Y SALIDA POR EL PUERTO.

- **Entrada:** Entrada por el puerto. Destino en registro Campo 3.

$$R3 = Entrada$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**1**	**1**	**1**	**1**	**0**	**Destino**				Vacío					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.34: Entrada por el puerto.

- **Salida:** Salida por el puerto. Dato en registro Campo 1.

$$Salida = R1$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**1**	**1**	**1**	**1**	**1**	Vacío				**Dato**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.35: Salida por el puerto.

3.2.3. Instrucciones de control de programa.

I SALTOS CONDICIONALES (Z, NZ, C y NC).

- **Salto Condicional tipo 1:** Salto condicional si bit $C = 1$. Relativo. Desplazamiento en registro Campo 1.

Si $C = 1$, entonces $PC = PC + R1$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**0**	**0**	**0**	**0**	**0**	Vacío				**Desplazamiento**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.36: Salto Condicional tipo 1.

- **Salto Condicional tipo 2:** Salto condicional si bit $C = 1$. Absoluto. Destino en literal.

Si $C = 1$, entonces $PC = literal$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**0**	**0**	**0**	**0**	**1**	Vacío									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Destino Literal															

Cuadro 3.37: Salto Condicional tipo 2.

- **Salto Condicional tipo 3:** Salto condicional si bit $NC = 1$. Relativo. Desplazamiento en registro Campo 1.

Si $NC = 1$, entonces $PC = PC + R1$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**0**	**0**	**0**	**1**	**0**	Vacío				**Desplazamiento**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.38: Salto Condicional tipo 3.

- **Salto Condicional tipo 4:** Salto condicional si bit $NC = 1$. Absoluto. Destino en literal.

$$\text{Si } NC = 1, \text{ entonces } PC = literal$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**0**	**0**	**0**	**1**	**1**	Vacío									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Destino Literal															

Cuadro 3.39: Salto Condicional tipo 4.

- **Salto Condicional tipo 5:** Salto condicional si bit $Z = 1$. Relativo. Desplazamiento en registro Campo 1.

$$\text{Si } Z = 1, \text{ entonces } PC = PC + R1$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**0**	**0**	**1**	**0**	**0**	Vacío				**Desplazamiento**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.40: Salto Condicional tipo 5.

- **Salto Condicional tipo 6:** Salto condicional si bit $C = 1$. Absoluto. Destino en literal.

$$\text{Si } C = 1, \text{ entonces } PC = literal$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**0**	**0**	**1**	**0**	**1**	Vacío									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Destino Literal															

Cuadro 3.41: Salto Condicional tipo 6.

- **Salto Condicional tipo 7:** Salto condicional si bit $NZ = 1$. Relativo. Desplazamiento en registro Campo 1.

$$\text{Si } NZ = 1, \text{ entonces } PC = PC + R1$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**0**	**0**	**1**	**1**	**0**	Vacío				**Desplazamiento**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.42: Salto Condicional tipo 7.

- **Salto Condicional tipo 8:** Salto condicional si bit $NZ = 1$. Absoluto. Destino en literal.

$$\text{Si } NZ = 1, \text{ entonces } PC = literal$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**0**	**0**	**1**	**1**	**1**	Vacío									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Destino Literal															

Cuadro 3.43: Salto Condicional tipo 8.

II SALTOS INCONDICIONALES.

- **Salto Incondicional tipo 1:** Salto incondicional. Relativo. Desplazamiento en registro Campo 1.

$$PC = PC + R1(\pm)$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**0**	**1**	**0**	**0**	**0**	Vacío				**Desplazamiento**				**ID 1**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.44: Salto Incondicional tipo 1.

- **Salto Incondicional tipo 2:** Salto incondicional. Absoluto. Destino en literal.

$$PC = literal$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	**0**	**1**	**0**	**0**	**1**	Vacío									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Destino Literal															

Cuadro 3.45: Salto Incondicional tipo 2.

III LLAMADA A SUB-RUTINA

- **Llamada a Sub-rutina:** Llamada a sub-rutina. Dirección en literal.

$$PC = Dirección\ Sub\text{-}rutina$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**1**	**0**	**1**	**1**	Vacío									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dirección Literal															

Cuadro 3.46: Llamada a Sub-rutina.

IV RETORNO DE SUB-RUTINA.

- **Retorno de Sub-rutina:** Retorno de sub-rutina.

$$PC = Dirección\ Retorno$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**1**	**0**	**1**	**0**	Vacío									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.47: Retorno de Sub-rutina.

V NO OPERACIÓN.

- **No Operación:** No operación.

No operación

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	Vacío									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vacío															

Cuadro 3.48: No Operación.

3.3. Tabla de instrucciones.

En esta apartado, presentamos la tabla de instrucciones reducida del *Procesador*, con todas las instrucciones ordenadas por códigos.

Número	**Código**				**Tipo**		**Instrucción**
1	0	0	0	0	0	0	No operación
2	0	0	0	0	0	1	—
3	0	0	0	0	1	0	—
4	0	0	0	0	1	1	—
5	0	0	0	1	0	0	Comparación
6	0	0	0	1	0	1	Comparación
7	0	0	0	1	1	0	—
8	0	0	0	1	1	1	—
9	0	0	1	0	0	0	Test de bit
10	0	0	1	0	0	1	—
11	0	0	1	0	1	0	Retorno de Sub-rutina
12	0	0	1	0	1	1	Llamada de Sub-rutina
13	0	0	1	1	0	0	—
14	0	0	1	1	0	1	—
15	0	0	1	1	1	0	—
16	0	0	1	1	1	1	—
17	0	1	0	0	0	0	Suma
18	0	1	0	0	0	1	Suma
19	0	1	0	0	1	0	Suma
20	0	1	0	0	1	1	Suma
21	0	1	0	1	0	0	Resta
22	0	1	0	1	0	1	Resta
23	0	1	0	1	1	0	Resta
24	0	1	0	1	1	1	Resta
25	0	1	1	0	0	0	And lógica
26	0	1	1	0	0	1	And lógica
27	0	1	1	0	1	0	Or lógica
28	0	1	1	0	1	1	Or lógica
29	0	1	1	1	0	0	Xor lógica
30	0	1	1	1	0	1	Xor lógica
31	0	1	1	1	1	0	Entrada Puerto
32	0	1	1	1	1	1	Salida Puerto

Cuadro 3.49: Tabla de instrucciones 1.

Número	**Código**				**Tipo**		**Instrucción**
33	1	0	0	0	0	0	Salto si C
34	1	0	0	0	0	1	Salto si C
35	1	0	0	0	1	0	Salto si NC
36	1	0	0	0	1	1	Salto si NC
37	1	0	0	1	0	0	Salto si Z
38	1	0	0	1	0	1	Salto si Z
39	1	0	0	1	1	0	Salto si NZ
40	1	0	0	1	1	1	Salto si NZ
41	1	0	1	0	0	0	Salto
42	1	0	1	0	0	1	Salto
43	1	0	1	0	1	0	—
44	1	0	1	0	1	1	—
45	1	0	1	1	0	0	Escritura en Memoria
46	1	0	1	1	0	1	Escritura en Memoria
47	1	0	1	1	1	0	Escritura en Memoria
48	1	0	1	1	1	1	Escritura en Registro
49	1	1	0	0	0	0	Rotación
50	1	1	0	0	0	1	—
51	1	1	0	0	1	0	Rotación
52	1	1	0	0	1	1	—
53	1	1	0	1	0	0	Lectura de Memoria
54	1	1	0	1	0	1	Rotación
55	1	1	0	1	1	0	Lectura de Memoria
56	1	1	0	1	1	1	Rotación
57	1	1	1	0	0	0	Desplazamiento
58	1	1	1	0	0	1	—
59	1	1	1	0	1	0	Desplazamiento
60	1	1	1	0	1	1	—
61	1	1	1	1	0	0	Desplazamiento
62	1	1	1	1	0	1	—
63	1	1	1	1	1	0	Desplazamiento
64	1	1	1	1	1	1	—

Cuadro 3.50: Tabla de instrucciones 2.

3.4. Estructura del procesador.

El *Procesador* que hemos diseñado tiene dos unidades principales: *Unidad de Control* y *Unidad de Proceso*. Presentamos el esquema del bloque superior de este *Procesador* en la figura.

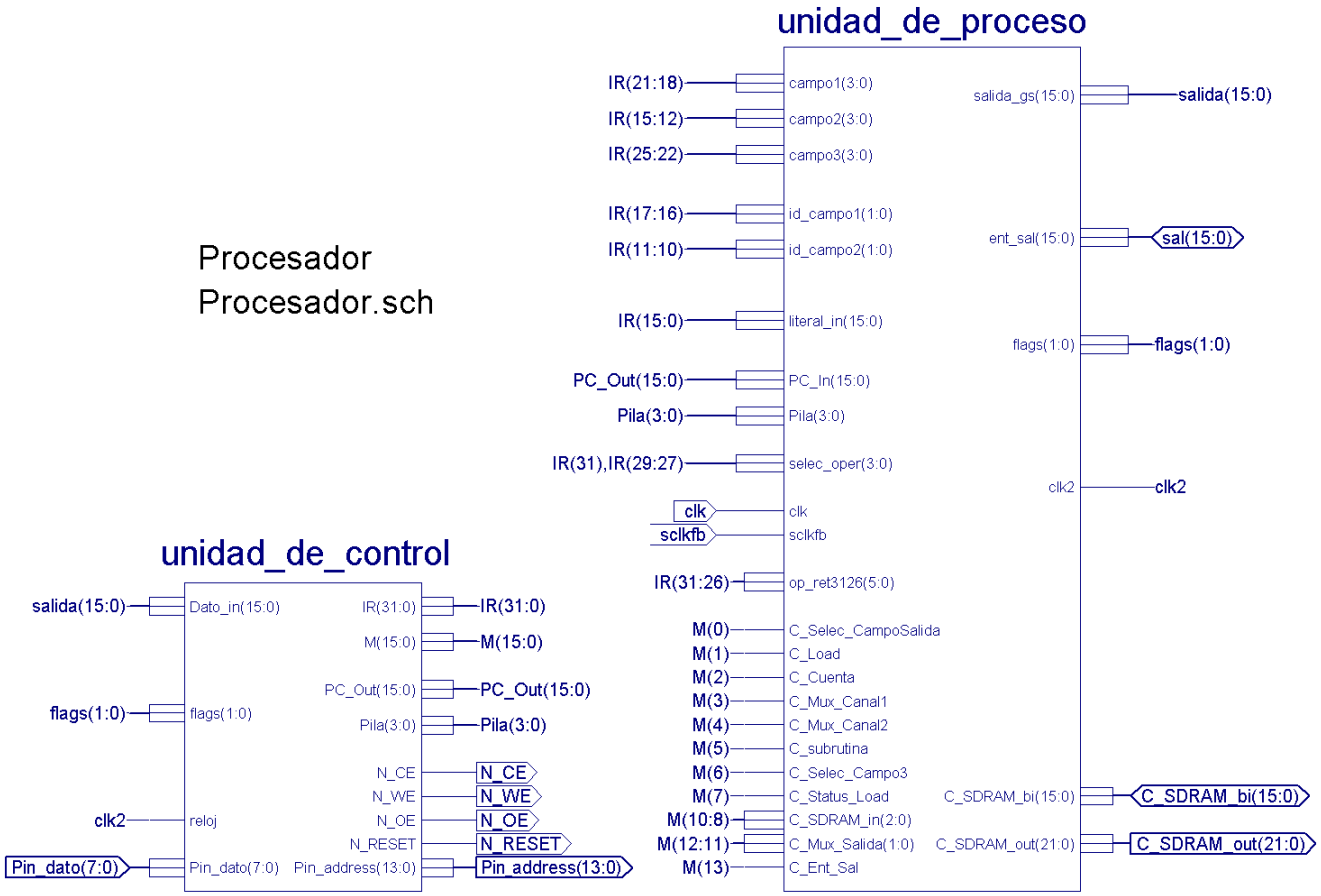


Figura 3.4: Esquema principal del *Procesador*.

Como su propio nombre indica, la *Unidad de Control* se encarga de controlar el funcionamiento del *Procesador*. Lleva el contador de programa (*PC*) y el *Registro de Instrucción IR*. También, contiene la *Memoria de Instrucciones* que permite almacenar las señales de control que gobernarán el funcionamiento del *Procesador*. Además, la *Unidad de Control* también contiene un bloque encargado de la gestión de saltos así como de las llamadas y retornos de subrutinas.

La *Unidad de Proceso* realiza las operaciones con datos que le ordena la *Unidad de Control*. Contiene de un banco de 16 registros de 16 bits cada uno, con posibilidad de auto-post-incremento y auto-post-decremento del contenido de cada registro en cada acceso que se realiza a él. Dispone de las siguientes entradas de datos de 16 bits:

- dato literal procedente del *IR*
- contenido del *PC* y contenido del puntero de pila *SP*
- puerto de entrada de 16 bits.

Tiene una única salida de 16 bits que puede ir:

- hacia la *Unidad de Control* (en caso de que sea el contenido del *PC*) en una instrucción de retorno de subrutina
- hacia el puerto de salida.
- hacia el propio banco de registros.

Así mismo, existe una *Unidad de Función* que realiza todas las operaciones aritméticas, lógicas y de desplazamiento, con un *Registro de Banderas (Flags)* asociado. Y, por último, un acceso a la memoria de datos, que, como hemos visto, es la *Memoria SDRAM*.

Como hemos visto, ambos bloques están conectados mediante bastantes líneas. Las siguientes líneas van desde la *Unidad de Control* hasta la *Unidad de Proceso*:

- *IR(31:0)*, donde
  - *IR(21:18):* selección del campo 1.
  - *IR(15:12):* selección del campo 2.
  - *IR(25:22):* selección del campo 3.
  - *IR(17:16):* selección del incremento/decremento del campo 1.
  - *IR(11:10):* selección del incremento/decremento del campo 2.
  - *IR(15:0):* dato literal.
  - *IR(31),IR(29:27)*: selección de la operación para la *Unidad de Función*.
- *PC_ Out(15:0):* valor del *PC* para guardarlo en caso de salto de subrutina.
- *Pila(3:0):* valor del puntero de pila.
- *M(13:0):* señales de control para la correcta sincronización de operaciones.

Las líneas que van desde la *Unidad de Proceso* hasta la *Unidad de Control* son, únicamente, estas dos:

- *Dato_ in(15:0):* dato de entrada para la recuperación del *PC*.
- *flags(1:0):* registros de banderas para la gestión d saltos condicionales.

También, hay numerosas líneas que están conectadas con los diversos periféricos disponibles en la tarjeta:

- Líneas relacionadas con la *Unidad de Control*:
  - *N_CE, N_WE, N_OE* y *N_RESET:* líneas procedentes de la *Unidad de Control* para el interface con la memoria de instrucciones (*Flash Rom*).
  - *Pin_address(13:0):* línea procedente de la *Unidad de Control* para el bus de direcciones de la memoria de instrucciones (*Flash Rom*).
  - *Pin_dato(7:0):* línea hacia la *Unidad de Control* para el bus de datos de la memoria de instrucciones (*Flash Rom*).
- Líneas relacionadas con la *Unidad de Proceso*:
  - *ent_sal(15:0):* línea bi-direccional conectada con los puertos de entrada y salida.
  - *C_SDRAM_bi(15:0):* línea bi-direccional para el bus de datos de la memoria de datos *SDRAM*.
  - *C_SDRAM_out(15:0):* línea procedente de la *Unidad de Proceso* para el bus de direcciones de la memoria de datos *SDRAM*.

Además, hacemos referencia a las dos señales de reloj que conviven y gobiernan el *Procesador*.

- *CLK:* señal de reloj principal, procedente del pin 88 de la tarjeta.
- *SCLKFB:* señal de reloj de realimentación de la *Memoria SDRAM*, procedente del pin 91 de la tarjeta.

En los siguientes capítulos, analizamos en profundidad las *Unidad de Control* (Capítulo 4) y *Unidad de Proceso* (Capítulo 5). Por último, debido a la reducida capacidad de la *FPGA Xilinx Spartan II*, no hemos podido implementar todo este diseño completo en la *FPGA*. Por esta razón, en el Capítulo 6, presentamos las reducciones que nos hemos vistos obligados a hacer para su implementación en nuestra *FPGA Xilinx Spartan II*. Si dispusiéramos de una a una *FPGA* de mayor capacidad, como la *FPGA Xilinx Spartan II XC2S100*, el diseño cabría completo. En cualquier caso, las reducciones de arquitectura que hemos realizado no son significativas para el funcionamiento del *Procesador*.

Capítulo 4

Unidad de Control.

4.1. Introducción.

La *Unidad de Control* es el *cerebro* del procesador. Se encarga de controlar y coordinar toda la actividad de procesamiento de datos:

- operaciones a realizar con datos,
- almacenaje y recuperación de datos de la memoria y
- entrada y salida de datos por los puertos.

Además de gestionar el tratamiento de los datos, la *Unidad de Control* también controla el flujo del programa:

- lee los códigos de operación desde la memoria de programa para obtener la palabra de instrucción,
- genera las *Señales de Control* para la ejecución de instrucciones,
- gestiona los saltos y las llamadas y retornos de subrutinas y
- sincroniza todo el procesador.

Como vemos en la figura 4.1, la *Unidad de Control* tiene estos 6 bloques inter-conectados:

1. *Registro PC*: es el registro que almacena el *Contador del Programa*.
2. *Interface con la Memoria de Instrucciones*: se trata de la memoria *Flash Rom*.
3. *Registro IR*: es el registro que almacena la *Palabra de Instrucción*.
4. *Memoria de Micro-instrucciones*: memoria que almacena las *Señales de Control* que forman las *Micro-instrucciones*.

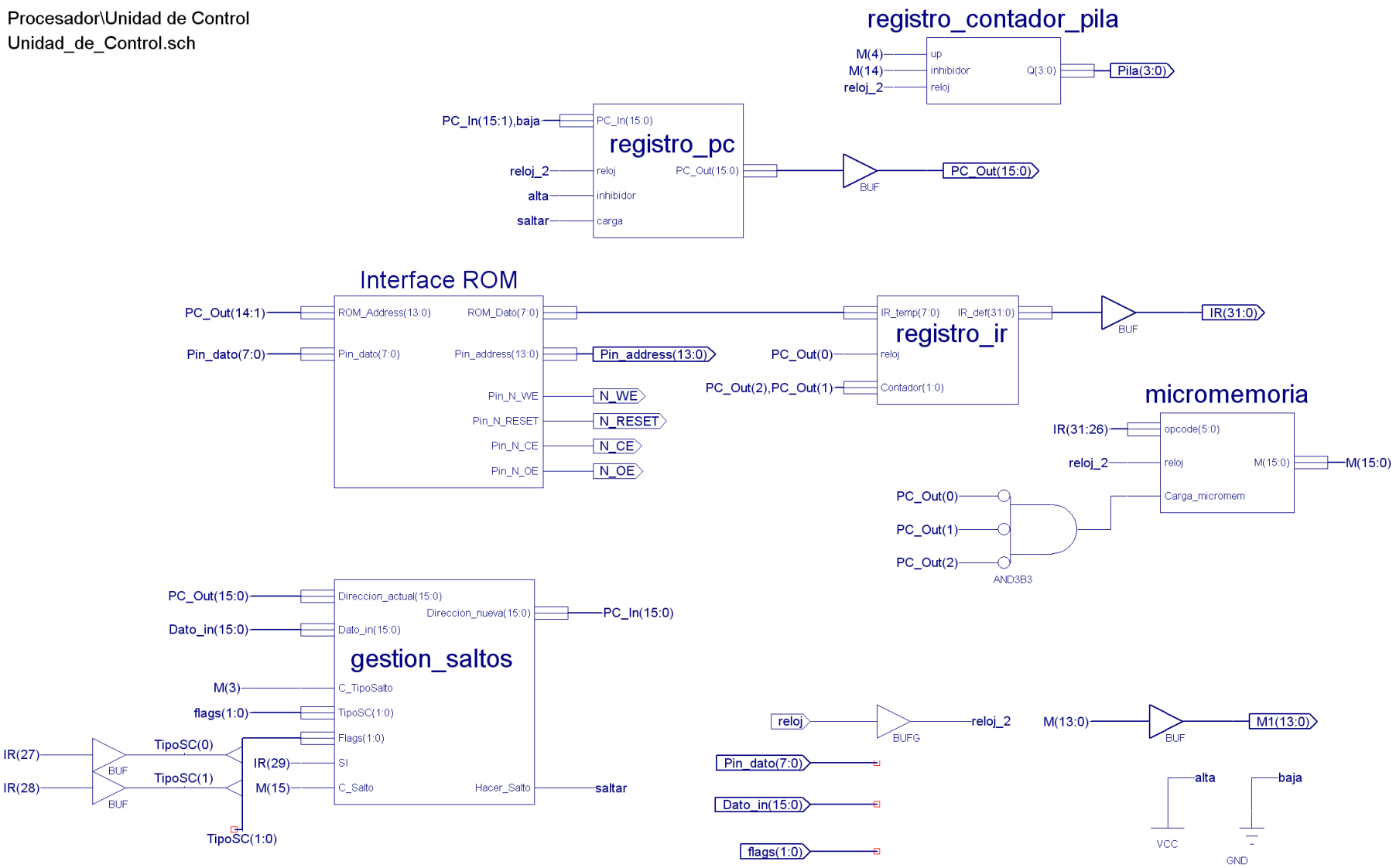


Figura 4.1: *Unidad de Control.*

5. *Gestión de Saltos*: bloque que se encarga de actualizar el contenido del *PC* si se realiza un salto, una llamada a subrutina o un retorno de subrutina.
6. *Registro SP*: es el registro *Contador de Pila* que almacena la primera posición libre de la misma.

En los siguientes apartados, analizamos en profundidad cada uno de estos bloques. Pero antes, explicamos la secuencia de operaciones que realiza la *Unidad de Control* en cada fase de búsqueda y ejecución de una instrucción.

4.2. Secuenciación de operaciones en la Unidad de Control.

Para realizar la ejecución de una instrucción, se deben completar las dos fases que ya hemos comentado: *Fase de Búsqueda* y *Fase de Ejecución*. El objetivo de la *Fase de Búsqueda* es obtener la *Palabra de Instrucción* que permitirá la ejecución de la instrucción en la *Fase de Ejecución*.

A partir de esta *Palabra de Instrucción*, la *Unidad de Control* obtiene:

- **Código de la instrucción (*IR(31:26)*).** Con este código, direcciona la *Memoria de Micro-instrucciones* para extraer las *Señales de Control* que gobernará la *Unidad de Proceso* y, en su caso, la *Unidad de Control*, durante la *Fase de Ejecución* de la instrucción.
- **Direccionamiento de los Campos 1, 2 y 3 (*IR(21:18)*, *IR(15:12)* e *IR(25:22)*).** Estos 3 bloques de 4 bits direccionan las entradas del *Banco de Registros* de la *Unidad de Proceso*, para seleccionar los registros sobre los que tendrán lugar las operaciones (lectura o escritura) de datos.
- **Incremento y Decremento de los Campos 1 y 2 (*IR(17:16)* e *IR(11:10)*).** Estos 2 bloques de 2 bits posibilitan el post-incremento o post-decremento del contenido de los registros a los que se accede en cada instrucción.
- **Dato Literal(*IR(15:0)*).** Este dato de 16 bits sale hacia la *Unidad de Proceso* y se utiliza en el caso de que la instrucción necesite un dato literal para ejecutarse.

En este apartado, vamos a ver la secuencia de operaciones que tienen lugar desde el comienzo de la *Fase de Búsqueda*, la posterior obtención de la *Palabra de Instrucción* y, para finalizar, el gobierno de la *Unidad de Proceso* (o la *Unidad de Control*) en la *Fase de Ejecución*. En la figura 4.2, mostramos las relaciones que existen entre cada bloque.

El contenido de la *Palabra de Instrucción* se encuentra en la memoria de instrucciones (*Memoria Flash Rom)*. Como vimos en el apartado 2.3.3, esta memoria dispone de un bus de datos de 8 bits. La *Palabra de Instrucción* tiene una longitud de 32 bits. Para completarla, por tanto, es necesario hacer 4 accesos secuenciales a la memoria de instrucciones. Para leer de la memoria *Flash Rom* basta con cambiar el direccionamiento para obtener el dato en el bus de datos.

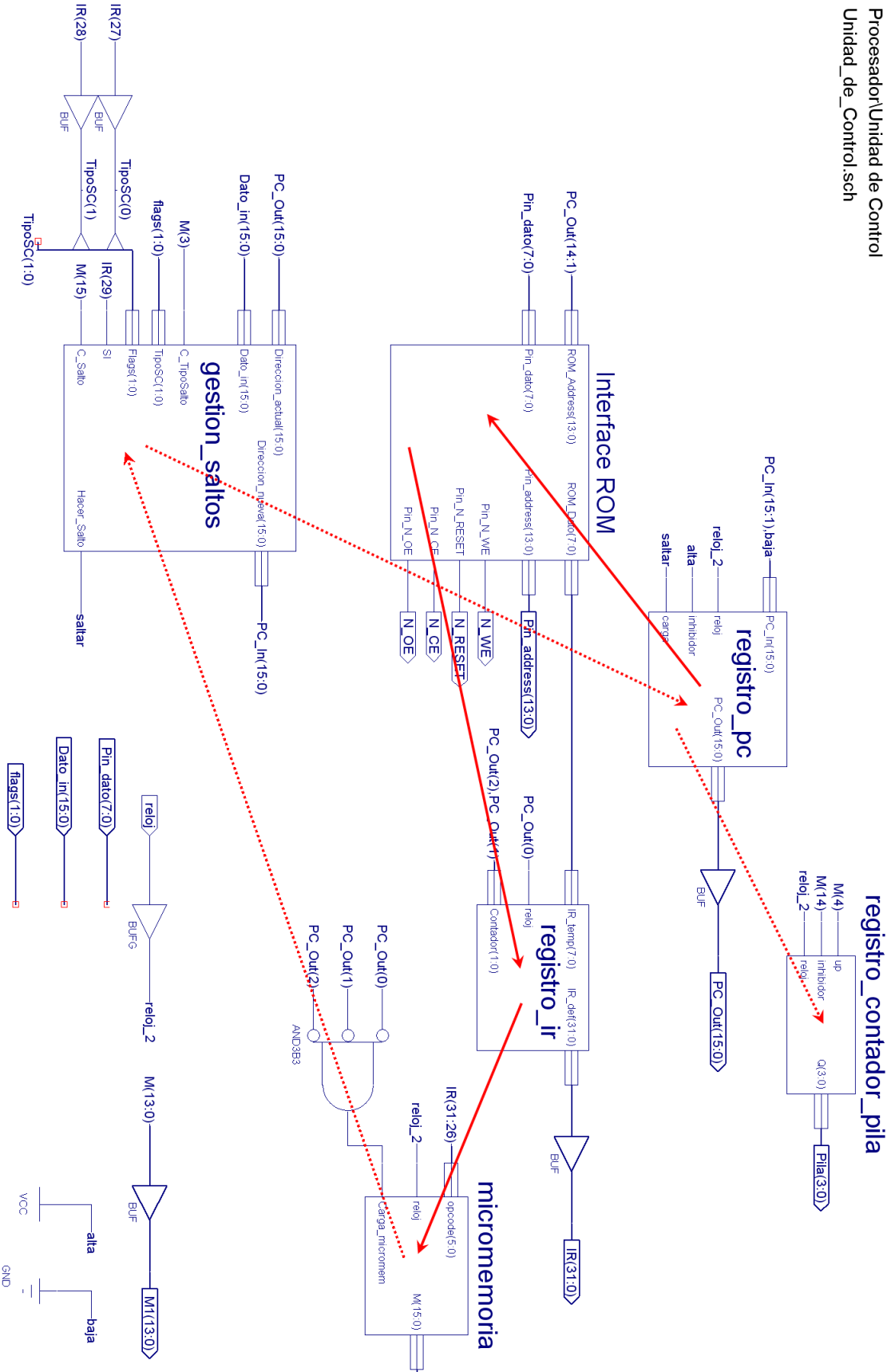


Figura 4.2: Secuencia de operaciones de la *Unidad de Control*.

Entonces, para implementar este proceso en nuestro diseño, necesitamos los siguientes 3 bloques:

- *Registro PC.* Genera las direcciones que entran en la *Memoria de Instrucciones* (*Flash Rom*) secuencialmente.
- *Interface con la Memoria de Instrucciones.* A partir de las direcciones que provienen del *PC*, presenta, de forma sucesiva, el contenido de la *Memoria de Instrucciones*, esto es, los códigos que forman la *Palabra de Instrucción.*
- *Registro IR.* Almacena los datos procedentes de la *Memoria de Instrucciones*, y cuando almacena el cuarto bloque de 8 bits, ya tiene 32 bits y actualiza la *Palabra de Instrucción.* Mientras mantiene la *Palabra de Instrucción* actual, sigue almacenando los datos para la siguiente *Palabra de Instrucción.*

Este *Registro IR* actúa como *Registro Tampón* para el *Pipeline.* Cuando tiene la *Palabra de Instrucción* completa, ésta se envía por el resto de la *Unidad de Control* y por la *Unidad de Proceso* para realizar la ejecución de la instrucción. Mientras tanto, el *PC* y la *Memoria de Instrucciones* comienzan la *Fase de Búsqueda* de la siguiente instrucción.

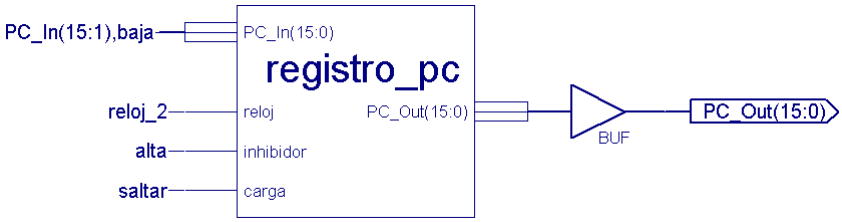
Una vez que el *Registro IR* tiene la *Palabra de Instrucción* completa, dispone de 8 ciclos de reloj para completar la *Fase de Ejecución.* Para ello, el primer paso es direccionar la *Memoria de Micro-instrucciones* con los 6 bits del campo *Código* de la *Palabra de Instrucción.* Tras un retardo despreciable (menor de un ciclo de reloj), están disponibles las *Señales de Control* para la ejecución de la instrucción. A partir de este punto, es la *Unidad de Proceso* quien, con los códigos del *IR* y las *Señales de Control*, finaliza la ejecución de la instrucción.

En el caso de un salto, la *Unidad de Control* dispone de un bloque que gestiona la realización o no realización de dicho salto. A este bloque entran el contenido del *PC actual* y un dato de 16 bits que actuará como dirección de salto definitiva o relativa al *PC* dependiendo del tipo de salto que se realice (absoluto o relativo). Además de estos dos datos, entran varias señales más que gestionarán la realización y el tipo del salto. Las salidas de este bloque son el nuevo *PC* y una línea de control que informará de si se realiza el salto o no al bloque *Registro PC* para que cargue el nuevo valor del *PC* o siga con el que tenía antes.

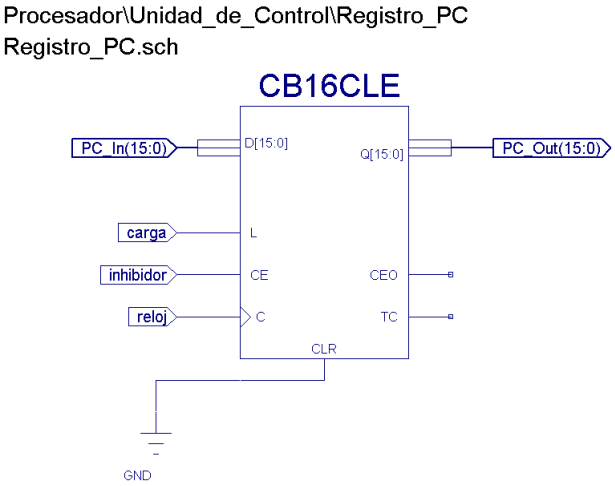
Vista la secuenciación de operaciones de la *Fase de Búsqueda* y en la *Fase de Ejecución* para la implementación del *Pipeline*, en los siguientes apartados, analizamos el diseño de cada bloque de la *Unidad de Control* individualmente. Para cada bloque, presentamos: entradas, salidas, esquema interno y funcionamiento.

4.3. *Registro PC.*

El *Registro PC* almacena el *Contador del Programa*. Este contador direcciona la *Memoria de Instrucciones* para obtener los códigos de la *Palabra de Instrucción*.



(a) Entradas y Salidas



(b) Esquema interno

Figura 4.3: *Registro PC.*

Las entradas del *Registro PC* son:

Entrada 1. **PC_In(15:0).** Es la entrada del *PC* proveniente del bloque de *Gestión de Saltos*. En caso de que se haga un salto, el *Registro PC* debe cargarse con un nuevo *PC*, y este nuevo *PC* llega por esta entrada. Para tener la seguridad de que el *PC* es el correcto, el bit menos significativo de esta entrada se fuerza a 0, y los otros 15 bits, se toman de la salida *PC_ Out(15:1)* del bloque de *Gestión de Salto*.

Entrada 2. **reloj.** Por esta entrada, llega el reloj global que circula por todo el procesador.

Entrada 3. **Carga.** Por esta entrada de *Carga*, llega la orden procedente del bloque de *Gestión de Saltos*, de que se realice un salto. Cuando se activa esta orden, se debe realizar un salto, y un salto consiste en cargar el *Registro PC* con un nuevo valor, que será el de la dirección destino.

Entrada 4. **Inhibidor.** Esta entrada está pensada para futuras actualizaciones del procesador. Para permitir el *pipeline* es estrictamente necesario que la *Fase de Ejecución* dure menos o, a lo sumo, lo mismo, que la *Fase de Búsqueda*. Sin embargo, si se diseña alguna instrucción cuya ejecución dure más de los 8 ciclos de reloj de la *Fase de Búsqueda*, es necesario parar el *Contador de Programa* para que no comience la ejecución de una nueva instrucción, hasta que no termine la anterior.
En nuestro caso, hemos diseñado las instrucciones de forma que su *Fase de Ejecución* dure lo mismo que su *Fase de Búsqueda*. Por esta razón, esta entrada de inhibición no tendrá ninguna funcionalidad.

Las salidas del *Registro PC* son:

Salida 1. **PC_Out(15:0).** La única salida del *Registro PC* es este bus de 16 bits que actúa como direccionamiento de la *Memoria de Instrucciones*.
Además, también tiene una entrada al bloque de *Gestión de Saltos* porque, para realizar un salto relativo, es necesario conocer el *PC*.
Y, por último, también está conectado, a través de un multiplexor, con la *Memoria de Datos* cuyo acceso se realiza desde la *Unidad de Proceso*.

Su núcleo se basa en un *Registro Contador* que está disponible en la librería de la *Spartan 2* de nuestro programa de diseño: *ISE WebPack 6.3*. El nombre de este contador es *CB16CLE*. A partir de este nombre, podemos explicar sus características:

CB. Significa *Contador Binario*, es decir, un contador que utiliza lógica binaria (0 y 1) para sus operaciones.

2, 4, 8 ó 16. Este contador tiene una capacidad de 16 bits, es decir, puede contar desde el número 0000_{16} (0_{10}) hasta el número $FFFF_{16}$ (65535_{10}). Al llegar a este valor, el siguiente número vuelve a ser el 0000_{16}.

C. Esta C hace referencia a la existencia de una entrada de *Clock Enable* (activación del reloj). Si su valor es 0, esta entrada inhibe los pulsos de reloj que le lleguen al contador, y si vale 1, permite la cuenta.
Así, esta entrada da la posibilidad de conectar en cascada varios contadores de este tipo. Para ello, el contador dispone de la salida *CEO*, que conectada a la entrada *CE* de otro contador, los conecta en cascada.

L. Esta L viene de *load* (carga), y es la señal que se activa para cargar un dato en el contador.

E. Representa la señal de *Enable* (activación) de que dispone el sistema. Se trata de una activación asíncrona. Si E vale 1, la salida del contador es 0000_{16}, y cuando E pasa a 0, el contador comienza a incrementarse con cada flanco de reloj.

Entradas					**Salidas**		
CLR	**L**	**CE**	**C**	**Dz-D0**	**Qz-Q0**	**TC**	**CEO**
1	X	X	X	X	0	0	0
0	1	X	↑	Dn	Dn	TC	CEO
0	0	0	X	X	No cambio	No cambio	0
0	0	1	↑	X	Inc	TC	CEO

Cuadro 4.1: Tabla de la Verdad del *CBxCLE*.

* *Nota.* Este contador no dispone de la posibilidad de cuenta hacia atrás.

A este *Registro Contador* conectamos las entrada y las salidas de la forma que muestra la figura 4.3(b). Las salidas *CEO* y *TC* no están conectadas[1], porque no nos son de ninguna utilidad.

4.4. *Interface con la Memoria de Instrucciones.*

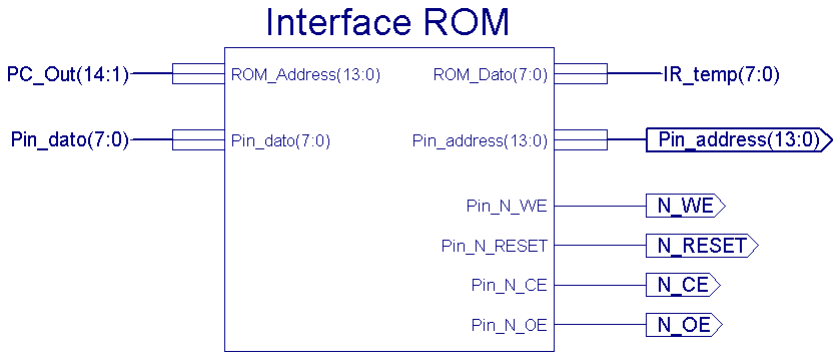
Este bloque genera las señales necesarias para leer los datos de la *Memoria de Instrucciones*. Recordamos que no se trata de una memoria implementada en la propia *FPGA*, sino de la memoria *Flash Rom* disponible en la tarjeta.

Las entradas de la *Interface con la Memoria de Instrucciones* son:

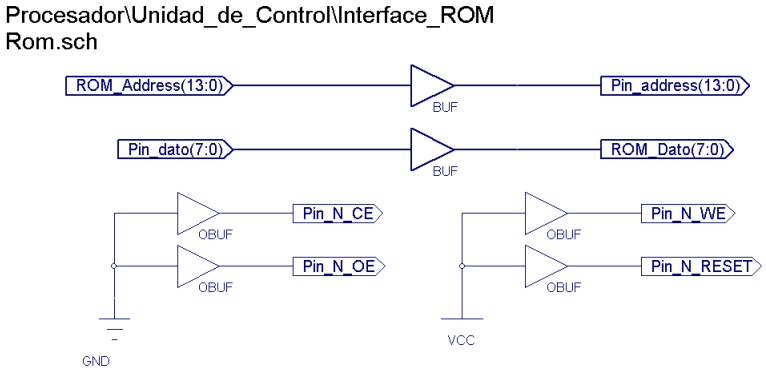
Entrada 1. **ROM_Address(13:0).** Esta es la entrada para el bus de direcciones que se conecta a la memoria *Flash Rom*. Como veremos en el apartado 4.5, para almacenar un dato en un flip flop, es necesario que este dato esté estable durante un flanco de subida del reloj. Entonces, para asegurarnos esta sincronización, utilizaremos como reloj del *Registro IR* la señal *PC_Out(0)*. Por esta razón, las direcciones de la *Memoria de Instrucciones* vienen definidas por las líneas *PC_Out(14:1)*, y forzamos al valor correspondiente a la entrada *PC_Out(0)* a un 0 lógico.

Entrada 2. **Pin_dato(7:0).** Esta entrada es necesaria para indicar a la *FPGA* los pines a los que está conectado el bus de datos de la memoria *Flash Rom*.

[1] En la compilación, provoca un *Warning* o *Aviso*, pero no genera ningún error.



(a) Entradas y Salidas



(b) Esquema interno

Figura 4.4: *Interface con la Memoria de Instrucciones.*

Las salidas de la *Interface con la Memoria de Instrucciones* son:

Salida 1. **ROM_Dato(7:0).** Es el bus de datos procedente de la *Memoria de Instrucciones*. A través de este bus, circulan los códigos que formarán la *Palabra de Instrucción* al juntarse, de forma correcta, en el *Registro IR*.

Salida 2. **Pin_address(13:0).** Esta línea es indispensable porque indica a la *FPGA* los pines a los que está conectado el bus de datos de la *Memoria de Instrucciones*.

Salida 3. **Pin_N_WE.** Línea que indica a la *FPGA* el pin al que está conectado la señal $\overline{WE}$ de la *Memoria Flash Rom*.

Salida 4. **Pin_N_RESET.** Línea que indica a la *FPGA* el pin al que está conectado la señal $\overline{RESET}$ de la *Memoria Flash Rom*.

Salida 5. **Pin_N_CE.** Línea que indica a la *FPGA* el pin al que está conectado la señal $\overline{CE}$ de la *Memoria Flash Rom*.

Salida 6. **Pin_N_OE.** Línea que indica a la *FPGA* el pin al que está conectado la señal $\overline{OE}$ de la *Memoria Flash Rom*.

El contenido de este bloque es el interface que permite la comunicación de la *FPGA* y la *Memoria Flash Rom*. Todos los pines se definen en el fichero *procesador.ucf*, como veremos en el Apéndice B.

4.5. *Registro IR.*

El bloque *Registro IR* almacena los datos que van llegando, en bloques de 8 bits, desde la *Memoria de Instrucciones* y, cuando tiene los 4 bloques que forman una *Palabra de Instrucción*, la genera y la mantiene a su salida mientras comienza a almacenar los bloques que formarán la *Palabra de Instrucción* de la siguiente instrucción.

Las entradas del *Registro IR* son:

Entrada 1. **IR_temp(7:0).** Los datos procedentes del bus de datos de la *Memoria Flash Rom* llegan a través de este bus de entrada.

Entrada 2. **reloj.** Para asegurar la sincronización del *PC*, la *Memoria de Instrucciones* y el *Registro IR*, la señal de reloj que gobernará este bloque es la señal *PC_Out(0)*. De esta forma, garantizamos que los datos que formarán la *Palabra de Instrucción* se almacenan correctamente.

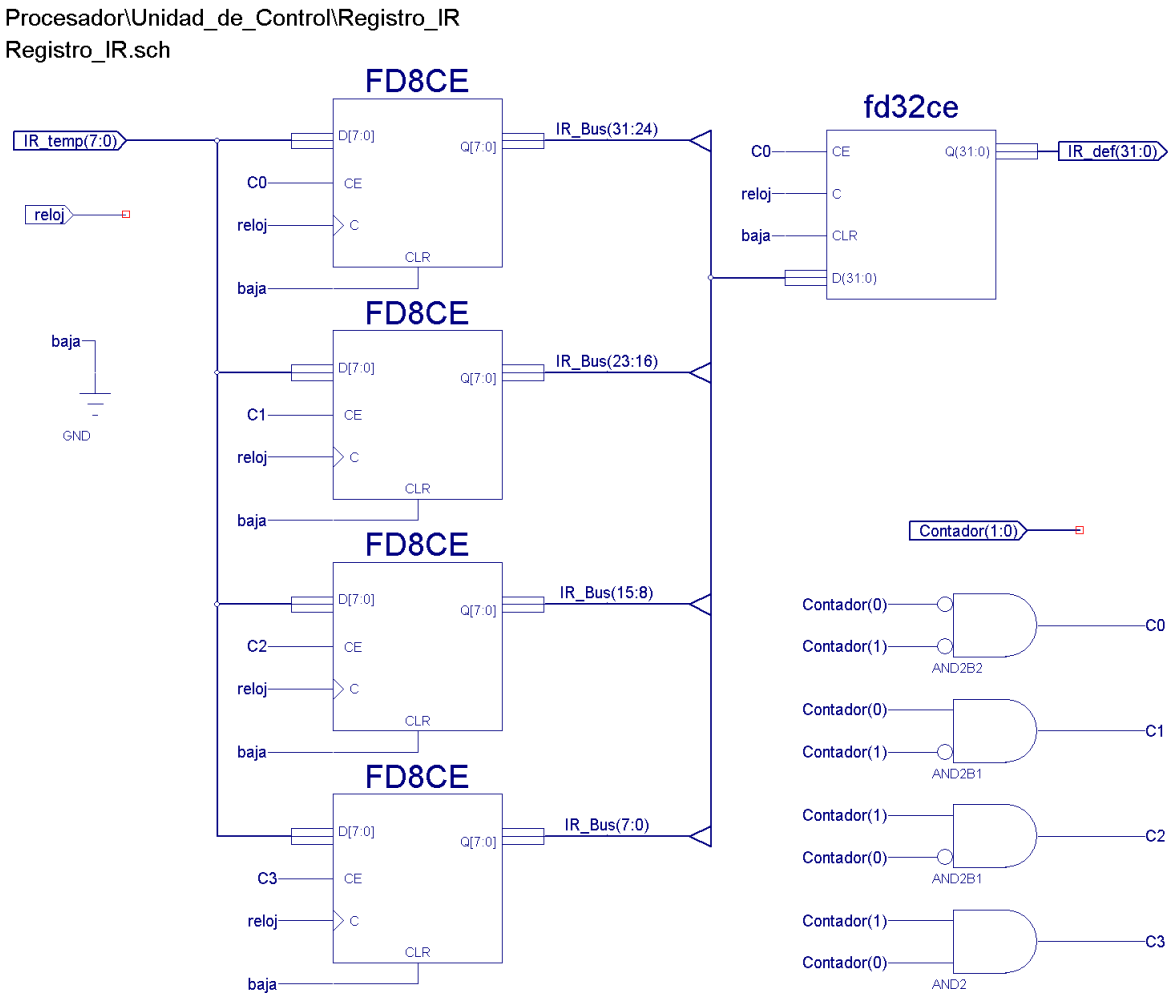
Entrada 3. **Contador(1:0).** Este bus selecciona los 4 registros internos donde se almacenan los bloques de 8 bits que formarán, posteriormente, la *Palabra de Instrucción*. La escritura en cada registro debe estar sincronizada con la lectura de los datos de la *Memoria de Instrucciones*. Para garantizarlo, se utilizan las señales *PC_Out(2)* y *PC_Out(1)* como entradas de este bus. Así, cuando leamos de la *Memoria de Instrucciones*, el primer bloque correspondiente a una *Palabra de Instrucción* será con los códigos *PC_Out(2:1)=00*, y este código activará la escritura del dato en el primer registro. Para el resto de los registros, el proceso es análogo.

La salida del *Registro IR* es:

Salida 1. **IR_def(31:0).** La *Palabra de Instrucción* sale al exterior del *Registro IR* a través de esta línea y se mantendrá durante todo el ciclo de ejecución de la instrucción.

En el interior del *Registro IR*, por tanto, hay 4 registros de 8 bits que almacenan los datos que vienen de la *Memoria de Instrucciones*. Un quinto registro de 32 bits que almacena la *Palabra de Instrucción*, una vez se tienen los 4 registros anteriores completos. Como vemos en la figura 4.5(b), el bus de entrada está conectado a los 4 registros de 8 bits y las 4 salidas de esos registros forman el bus de entrada del registro de 32 bits.

(a) Entradas y Salidas



(b) Esquema interno

Figura 4.5: *Registro de Instrucción.*

En la figura 4.5(b), vemos los 2 tipos de registro que hemos utilizado: los de 8 bits (*FD8CE*) y el de 32 bits (*FD32CE*). Lo único que los diferencia a ambos es su capacidad, que en el primero es de 8 bits, y en el segundo es de 32. Presentamos sus características a partir de su nombre:

FD. Significa *Flip flop Tipo D*. Es el tipo de registro más común. Vemos su funcionamiento a partir de sus *Señales de Control*

8 ó 16 Número de bits que es capaz de almacenar el contador.

C. Dispone de una señal *Clock Enable* (activación del reloj). Si su valor es 0, esta entrada inhibe los impulsos de reloj que le lleguen al contador. Sin embargo, si esta entrada vale 1, el registro almacena los nuevos datos, es decir, tiene funcionamiento normal.

E. Representa la señal de *Enable* (activación) de que dispone el sistema. Se trata de una activación asíncrona. Si E vale 1, se borran los datos almacenados, presentando un 0_8 (ó 0_{32}) en la salida. Si vale 0, el registro tiene funcionamiento normal, almacenando y sacando datos en cada flanco ascendente del reloj.

Entradas				**Salidas**
CLR	**CE**	**Dz-D0**	**C**	**Qz-Q0**
1	X	X	X	0
0	0	X	X	No cambio
0	1	Dn	↑	Dn

Cuadro 4.2: Tabla de la Verdad del *FDxCE*.

4.5.1. *Registro de 32 bits.*

ISE WebPACK 6.3 solo incluye los *registros tipo D* con estas características de 4, 8 y 16 bits (*FD4CE*, *FD8CE* y *FD16CE*. EL registro de 32 bits *FD16CE* es un diseño propio que hemos realizado conectando dos registros *FD16CE* en paralelo, como vemos en la figura 4.6(b).

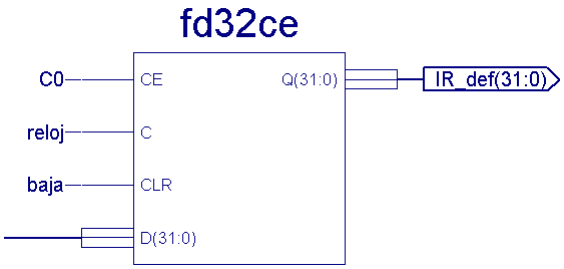
Las entradas del *Registro de 32 bits* son:

Entrada 1. **D(31:0).** Bus de entrada de datos de 32 bits.

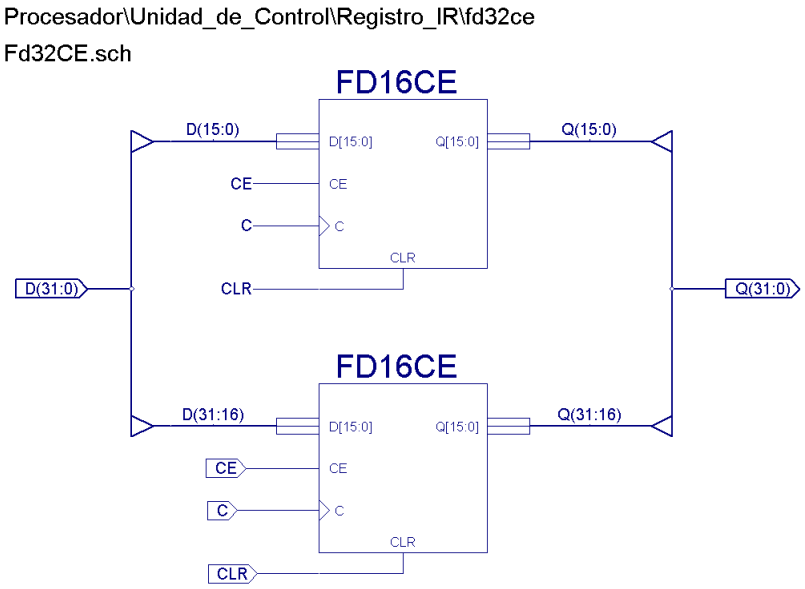
Entrada 2. **CE.** Señal de inhibición de reloj.

Entrada 3. **C.** Por esta entrada, llega el reloj global que circula por todo el procesador.

Entrada 4. **CLR.** Señal de activación asíncrona del registro.



(a) Entradas y Salidas



(b) Esquema interno

Figura 4.6: *Registro de 32 bits* (*FD32CE*).

La salida del *Registro de 32 bits* es:

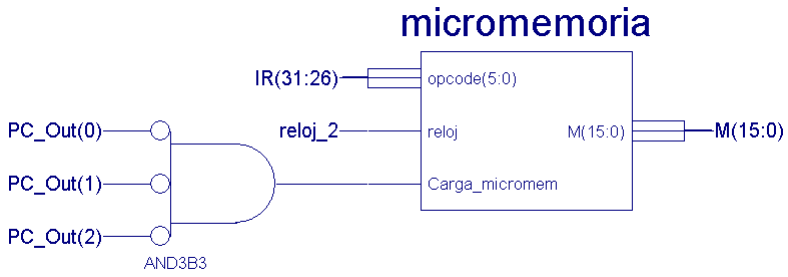
Salida 1. **Q(31:0).** Bus de salida de datos de 32 bits.

La tabla de la verdad de este registro es la tabla que se muestra en el Cuadro 4.2.

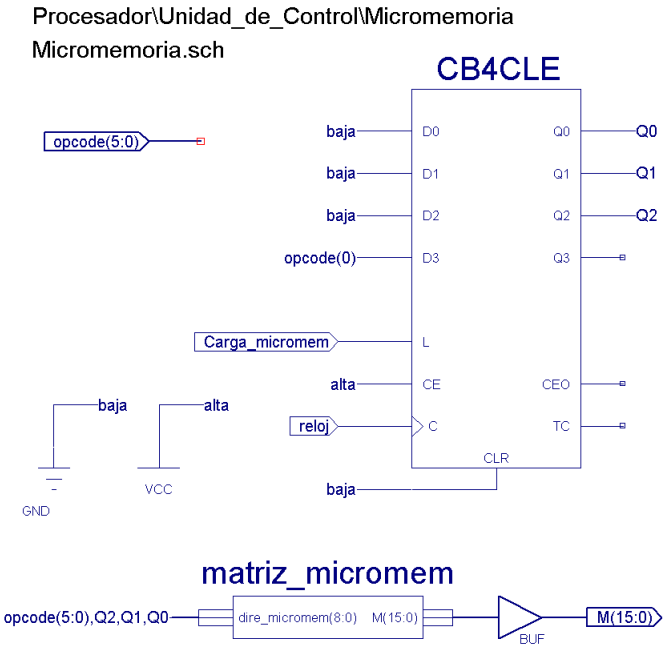
Para terminar todo lo concerniente al *Registro IR*, hacemos referencia a la lógica de activación de los registros. Cada registro debe activarse al mismo tiempo que está activa la dirección correspondiente de la *Memoria de Instrucciones*. Como hemos visto, la señal de activación de un registro es una línea que activa el registro si vale 1, y lo desactiva si vale 0. Por tanto, con puertas lógicas *AND de dos entrada* la resolución de este problema es inmediata.

4.6. *Memoria de Micro-instrucciones.*

La *Memoria de Micro-instrucciones* almacena las *Señales de Control* necesarias para el gobierno de la *Unidad de Control* y la ejecución de instrucciones en la *Unidad de Proceso*. Acepta los 6 bits del *Código de Operación* de la *Palabra de Instrucción* (*IR(31:26)*) y genera las *Señales de Control* de la instrucción que indica dicho código. Como cada instrucción está formada por 8 ciclos de reloj, este bloque genera las *Señales de Control* durante cada uno de esos 8 ciclos.



(a) Entradas y Salidas



(b) Esquema interno

Figura 4.7: *Memoria de Micro-instrucciones.*

Las entradas de la *Memoria de Micro-instrucciones* son:

Entrada 1. **opcode(5:0).** A través de este bus entran los 6 bits del *Registro IR* que forman el *Código de Operación*. Son los bits *IR(31:26)*.

Entrada 2. **reloj.** Por esta entrada, llega el reloj global que circula por todo el procesador.

Entrada 3. **Carga_micromem.** Esta señal se utiliza para inicializar el contador interno del bloque en cada instrucción. Cada nueva instrucción comienza cuando los valores de *PC_ Out(2)*, *PC_ Out(1)* y *PC_ Out(0)* son 000. Entonces, con la puerta lógica *nand* que se muestra en la figura 4.7(a), se puede controlar el valor de la entrada de carga del contador. Cuando las tres entradas de esta puerta lógica sean 0, su salida es 1, y se inicializa el contador interno.

La salida de la *Memoria de Micro-instrucciones* es:

Salida 1. **M(15:0).** Son las 15 *Señales de Control* que gobiernan el funcionamiento del procesador durante la ejecución de instrucciones.

El esquema interno de este bloque se muestra en la figura 4.7(b). Básicamente, consta de dos bloques: un contador interno y una *Matriz de Memoria de Micro-instrucciones*.

El contador interno tiene el nombre de *CB4CLE*. Como vimos en el apartado 4.3, se trata de un contador binario de 4 bits con carga en paralelo. El objetivo de este contador es generar la parte de baja (3 bits menos significativos) de las direcciones que entran en la *Matriz de Micro-instrucciones*. Para comprender la necesidad de este contador, es necesario comprender la estructura de la *Matriz de Memoria de Micro-instrucciones* que vamos a presentar en el apartado 4.6.1.

4.6.1. *Matriz de Memoria de Micro-instrucciones.*

La *Matriz de Memoria de Micro-instrucciones* es una memoria ROM implementada en el interior de la *FPGA Xilinx Spartan II*[2]. Es una memoria de 512 posiciones de 16 bits cada una. Estas 512 posiciones se corresponden con las 8 posiciones que tiene cada una de los 64 (2^6) *Códigos de Instrucción* diferentes.

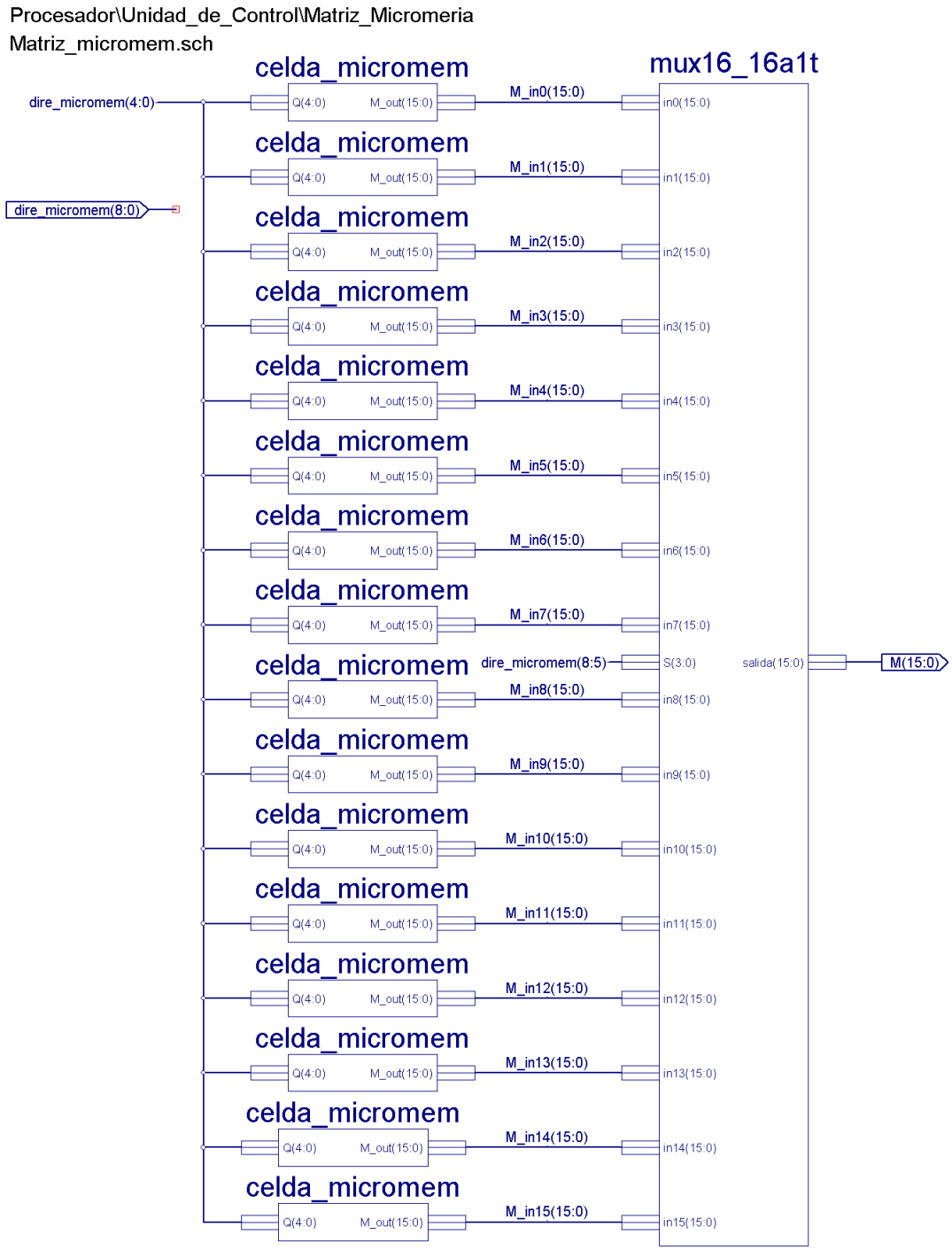
$$2^6 \text{ instrucciones} \cdot 8 \text{ ciclos de reloj/instrucción} = \boxed{512 \text{ posiciones}}$$

La entrada de la *Matriz de Memoria de Micro-instrucciones* es:

Entrada 1. **Dire_micromem(8:0).** Estos 9 bits direccionan las 512 posiciones de la *Matriz de Memoria de Micro-instrucciones*.
Los 3 bits menos significativos del bus proceden del contador interno, de forma que, al comenzar la ejecución de una instrucción, estos bits valen 000, ya que se acaba de realizar la inicialización del contador interno de la *Micromemoria*. Los otros 6 bits proceden del *Código de Operación* de la *Palabra de Instrucción*.

[2]No confundir con la *Memoria de Instrucciones* (*Memoria Flash Rom*)

(a) Entradas y Salidas



(b) Esquema interno

Figura 4.8: *Matriz de Memoria de Micro-instrucciones.*

La salida del *Matriz de Memoria de Micro-instrucciones* es:

Salida 1. **M(15:0).** Este bus de 16 líneas lleva las *Señales de Control* hacia el bloque de la *Unidad de Control*. Desde allí, se repartirán por la propia *Unidad de Control* e irán hacia la *Unidad de Proceso*.

La *Matriz de Memoria de Micro-instrucciones* está diseñada de forma que la primera *Micro-instrucción* de cada instrucción está en la posición *IR(31:26),000*, donde *IR(31:26)* es el *Código de Operación* de la instrucción. Por esta razón, en la *Matriz de Memoria de Micro-instrucciones* se introducen las direcciones de la forma en que se ve en la figura 4.7(a). Así, la parte *IR(31:26)* de la dirección selecciona la operación a realizar, mientras que los 3 bits menos significativos seleccionan cada *Micro-instrucción* de la instrucción. Por esta razón, estos 3 bits menos significativos proceden de un contador.

Conforme el contador va avanzando, direccionamos las sucesivas *Micro-instrucciones* de cada instrucción. Al llegar a la séptima *Micro-instrucción*, se vuelve a inicializar el contador y se actualiza el *Código de Operación*, comenzando la ejecución de una nueva instrucción. Este bucle se repite constantemente durante la ejecución de un programa.

Por tanto, con esta arquitectura, la *Matriz de Memoria de Micro-instrucciones* está sacando las *Señales de Control* que gobiernan el funcionamiento del procesador en cada ciclo de reloj.

El esquema que presentamos en la figura 4.8(b) es el esquema interno de la *Matriz de Memoria de Micro-instrucciones*. En él, tenemos una matriz de 16 celdas y un multiplexor que selecciona la salida de una de las 16 celdas. En los siguientes sub-apartados, analizamos estos bloques.

4.6.2. *Celda de la Matriz de Memoria de Micro-instrucciones.*

Cada una de estas *Celdas de Memoria*, contiene 16 bloques de *memoria ROM* de 32 posiciones de 1 bit cada una. Todos los bloques están conectados en paralelo, y sus salidas forman un único bus de salida de 16 bits, como mostramos en la figura 4.9(b).

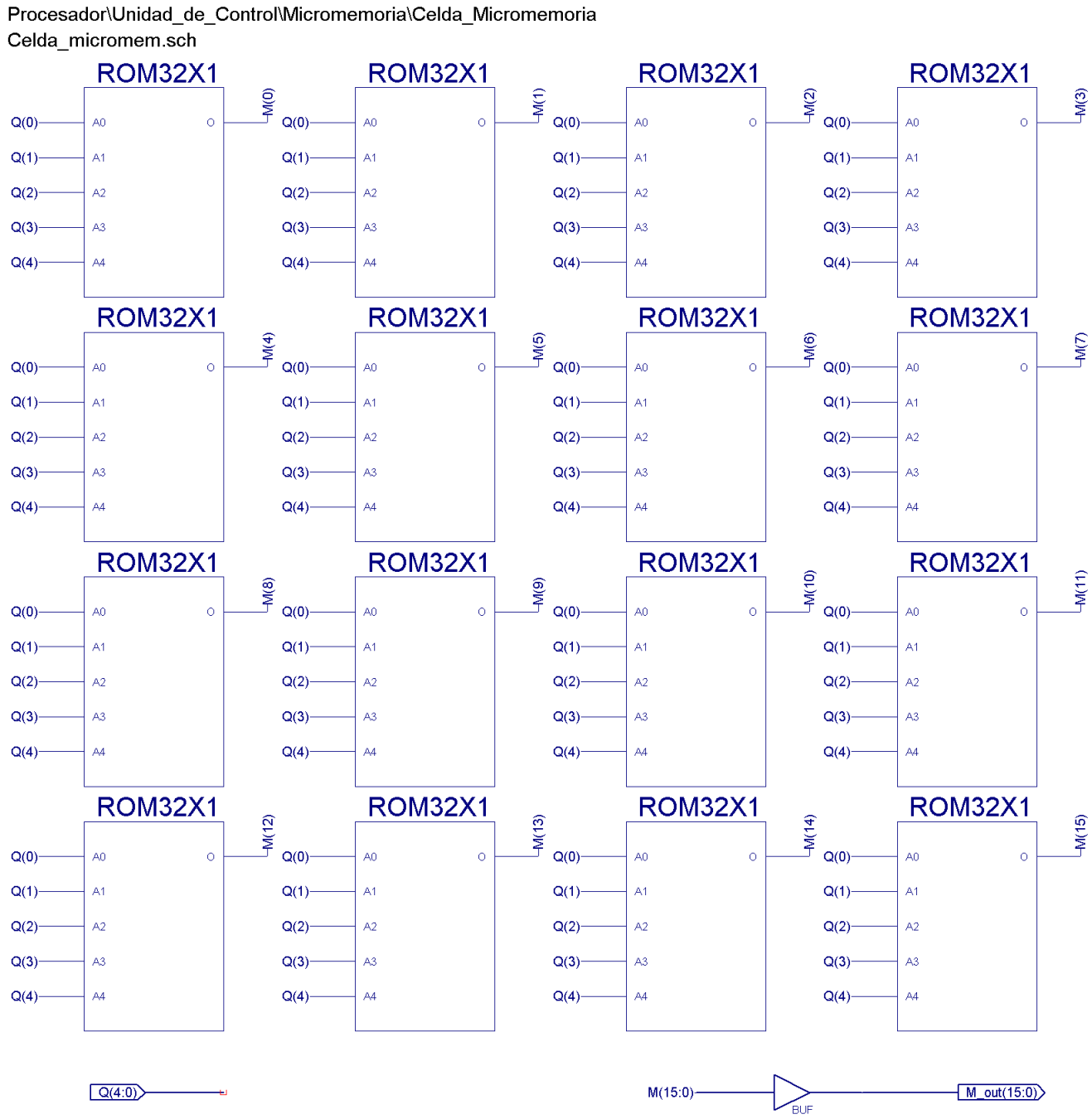
La entrada de la *Celda de la Matriz de Memoria de Micro-instrucciones* es:

Entrada 1. **Dire_micromem(4:0).** Son 5 bits que direccionan las 32 posiciones de cada bloque *Memoria Rom*.

La salida de la *Celda de la Matriz de Memoria de Micro-instrucciones* es:

Salida 1. **M_out(15:0).** Este bus de 16 líneas lleva las *Señales de Control* hacia el bloque de la *Unidad de Control*.

(a) Entradas y Salidas



(b) Esquema interno

Figura 4.9: *Celda de la Matriz de Memoria de Micro-instrucciones.*

Los bloques de *Memoria Rom* tienen 32 posiciones de un bit. Para direccionar 32 posiciones son necesarios 5 bits.

Cada bloque de *Memoria Rom* está asociado con una *Señal de Control*. La salida de todos los bloques se conectan a un bus para que, de esta forma, este bus de salida lleve todas las *Señales de Control*

Para una sola *Señal de control*, la ejecución completa de una instrucción necesita 8 ciclos de reloj. Por otro lado, cada ciclo de instrucción se corresponde con una posición en la *Memoria de Micro-instrucciones*. Cada *Memoria Rom* tiene 32 posiciones. Por tanto, cada *Memoria Rom* contiene las *Señales de Control* de 4 instrucciones diferentes.

Entonces, de los 5 bits de entrada, los 3 bits menos significativos seleccionan uno de los 8 ciclos de reloj en el que se encuentra la ejecución de la instrucción y los 2 bits más significativos seleccionan una de las 4 instrucciones.

4.6.3. *Multiplexor Tri-estado de 16 a 1 de 16 bits.*

Este bloque toma 16 entradas de 16 bits cada una y una entrada de selección de otros 4 bits y, a partir de ese bus de selección, saca en su salida una de las señales de entrada. Es decir, este bloque funciona como un multiplexor. Sin embargo, no es un multiplexor tradicional, ya que utiliza *Buffers Tri-estado*.

Las entradas del *Multiplexor Tri-estado de 16 a 1 de 16 bits* son:

Entrada 1. **in0...15(15:0).** 16 entradas de 16 bits.

Entrada 2. **S(3:0).** 4 bits que seleccionan una de las 16 entradas para llevarla a la salida.

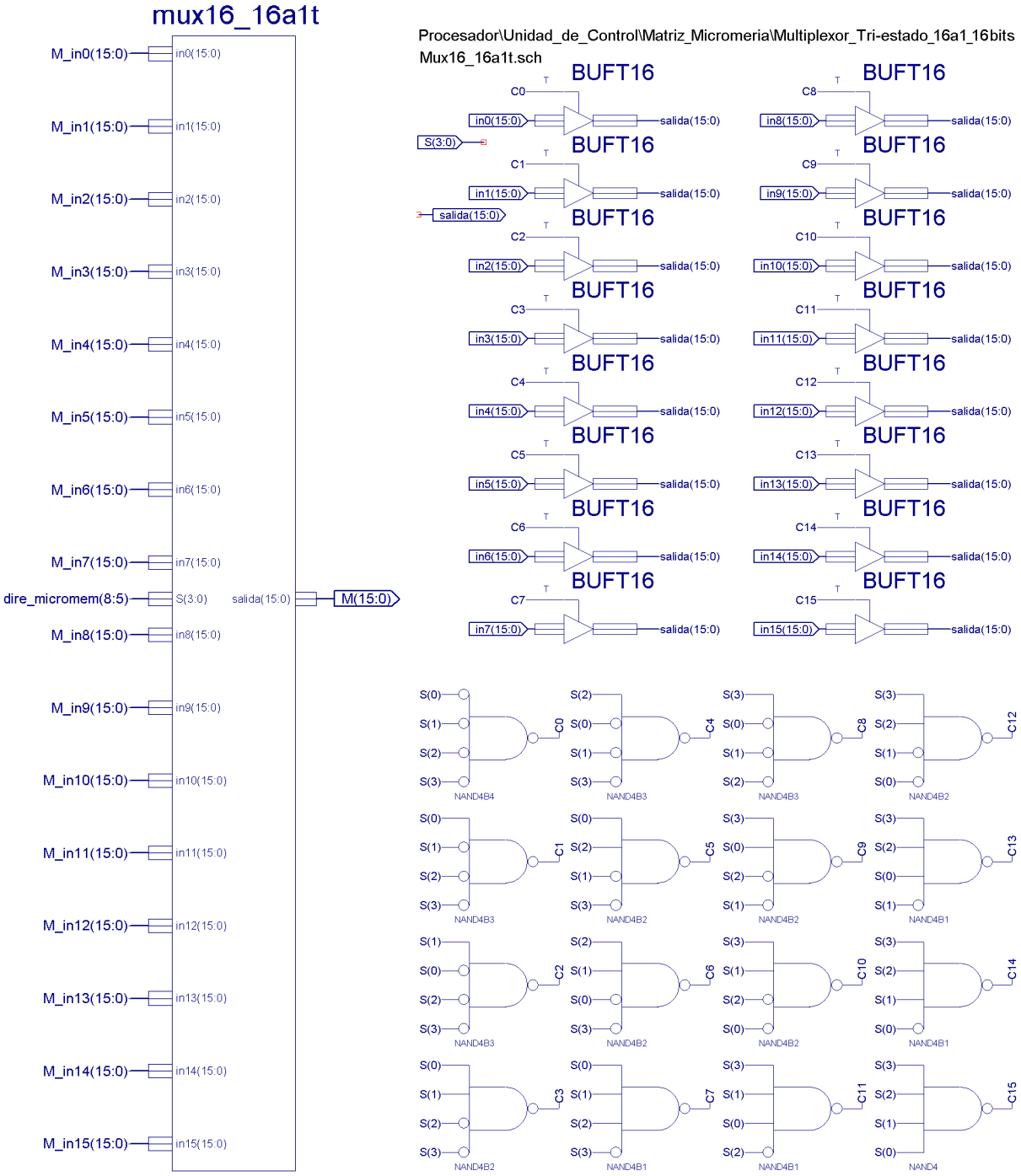
La salida del *Multiplexor Tri-estado de 16 a 1 de 16 bits* es:

Salida 1. **salida(15:0).** Bus de 16 bits que saca una de las entradas.

La estructura interna del bloque se muestra en la figura 4.10(b). Las 16 entradas de 16 bits van a 16 *Buffers Tri-estado* de 16 bits cada uno. Las salidas de todos los *Buffers* están conectadas a un mismo bus de salida[3]. Entonces, en todo momento, en la salida está presente una, y solo una, de las entradas. Hemos diseñado la lógica de selección de las salidas mediante puertas lógicas, como también mostramos en esta figura 4.10(b).

A lo largo del procesador, hemos diseñado varios bloques *Multiplexor* con esta arquitectura de *Buffers Tri-estado*, ya que ocupan muchos menos recursos de la *FPGA* que un multiplexor tradicional. Sin embargo, tampoco podemos implementar todos los *Multiplexores* con *Buffers Tri-estado*, ya que disponemos de un número limitado de los mismos.

[3]Este tipo de multiplexores da un aviso al realizar la compilación, pero no genera ningún error.



(a) Entradas y Salidas (b) Esquema interno

Figura 4.10: *Multiplexor Tri-estado de 16 a 1 de 16 bits.*.

Una vez analizados estos dos bloques, *Celda de la Matriz de Memoria de Micro-instrucciones* y *Multiplexor Tri-estado de 16 a 1 de 16 bits*, vamos a ver las conexiones que se muestran en la figura 4.8(b). De los 9 bits de entrada, los 5 menos significativos direccionan una posición de cada una de las *Celdas de Memoria*. Recordamos que cada posición contiene 16 bits, ya que cada posición consta de 16 bloques de *Memoria Rom* conectados en paralelo. Entonces, de cada *Celda* obtenemos un bus de 16 bits, que entran en el *Multiplexor Tri-estado*. Además de esas 16 entradas, al *Multiplexor Tri-estado* le llegan los 4 bits más significativos de la entrada del bloque, de forma que estos bits sirven para la selección de las entradas.

Con esta arquitectura, a partir de los 9 bits de entrada al bloque *Matriz de Memoria de Micro-instrucciones*, obtenemos una línea de salida de 16 bits con el contenido de la posición de memoria direccionada con esos 9 bits.

Para terminar el sub-apartado de la *Memoria de Micro-instrucciones*, presentamos cada *Señal de Control* con la función que realiza en el procesador. Como veremos a continuación, algunas de ellas realizan varias funciones de control.

Señal de Control 0. **C_Selec_CampoSalida.** La señal *M(0)* selecciona el canal por el que salen los datos de los registros que direccionan los *Campos*.

- Si *M(0) = 0*, el contenido del registro seleccionado por *Campo1* sale por *Canal1*.
- Si *M(0) = 1*, el contenido del registro seleccionado por *Campo2* sale por *Canal2*.

Señal de Control 1. **C_Load.** La señal *M(1)* controla el instante en el que se cargan los datos en los *Registros* del *Banco de Registros*.

- Si *M(1) = 0*, el *Registro* mantiene su valor.
- Si *M(1) = 1*, el *Registro* se carga con un nuevo valor.

Señal de Control 2. **C_Cuenta.** La señal *M(2)* gobierna el momento en el que se realizan las operaciones de post-modificación de los datos de los *Registros* del *Banco*.

- Si *M(2) = 0*, el *Registro* mantiene su valor.
- Si *M(2) = 1*, el *Registro* realiza las modificación de su contenido.

Señal de Control 3. **C_Mux_Canal1.** La señal *M(3)* está asociada con la señal *C_Subrutina*. Solo se considera cuando *C_Subrutina = 0*. Con *C_Subrutina = 0*,

- Si *M(3) = 0*, en el *Canal1*, tenemos la salida del *Banco de Registros*.
- Si *M(3) = 1*, en el *Canal1*, tenemos el *Dato Literal*.

Señal de Control 3. **C_TipoSalto.** La señal *M(3)* también gobierna la realización de *Saltos*. Controla si el *Salto* que puede tener lugar, o no , será absoluto o relativo. La realización o no de este salto no depende de esta *Señal de Control*.

- Si *M(3) = 0*, el *Salto* sería relativo.
- Si *M(3) = 1*, el *Salto* sería absoluto.

Señal de Control 4. **C_Mux_Canal2.** La señal *M(4)* está asociada con la señal *C_Subrutina*. Solo se considera cuando *C_Subrutina = 0*. Con *C_Subrutina = 0*,

- Si *M(4) = 0*, en el *Canal2*, tenemos la salida del *Banco de Registros*.
- Si *M(4) = 1*, en el *Canal2*, tenemos el *Dato Literal*.

Señal de Control 4. **C_UpPila.** La señal *M(4)* también gobierna la dirección de modificación del *Puntero de Pila*. No implica que se vaya a modificar, sino que, en caso de que me modifique, gobierna la dirección.

- Si *M(4) = 0*, el *Puntero de Pila* se decrementa.
- Si *M(4) = 1*, el *Puntero de Pila* se incrementa.

Señal de Control 5. **C_Subrutina.** La señal *M(5)* permite el acceso del contenido del *PC* y del *Puntero de Pila* a la propia *Pila* implementada en la *Memoria de Datos (SDRAM)*.

- Si *M(5) = 0*, por los *Canales* circulan datos para realizar operaciones con ellos.
- Si *M(5) = 1*, por los *Canales* circulan el *PC* y el *Puntero de Pila* hacia la *Pila*.

Señal de Control 6. **C_Selec_Campo3.** La señal *M(6)* decide si se realizarán las operaciones con el dato del *Registro* direccionado pos el *Campo1* o por el *Campo3*.

- Si *M(6) = 0*, el dato será el del *Registro* direccionado por el *Campo1*.
- Si *M(6) = 1*, el dato será el del *Registro* direccionado por el *Campo3*.

Señal de Control 7. **C_Status_Load.** La señal *M(7)* activa o desactiva la actualización del *Registro de Estado* que modifican las operaciones de la *Unidad de Función*.

- Si *M(7) = 0*, el *Registro de Estado* mantiene su valor anterior.
- Si *M(7) = 1*, el *Registro de Estado* se actualiza.

Señal de Control 8. **C_SDRAM_in(2:0).** Las señales *M(10:8)* gobiernan el interface de lectura y escritura de la *Memoria de Datos*.

- Si *M(10) = 0*, el *Registro* adjunto a la *Memoria de Datos* actúa como una *memoria* y mantiene su valor en la salida, independientemente del dato que tenga en su entrada.
- Si *M(10) = 1*, el *Registro* adjunto a la *Memoria de Datos* almacena las direcciones altas que permitirán el direccionamiento completo de la *Memoria de Datos*.
- Si *M(9) = 0*, no realiza ninguna función.
- Si *M(9) = 1*, el dato presente en el bus de datos del controlador de la *Memoria de Datos* se escribe en la posición de la *Memoria de Datos* seleccionada por el bus de direcciones (escritura).
- Si *M(8) = 0*, no realiza ninguna función.
- Si *M(8) = 1*, en la salida del bloque *Interface con la* Memoria de Datos, el dato presente es el que contiene la posición de la *Memoria de Datos* seleccionada por el bus de direcciones (lectura).

Señal de Control 11. **C_Mux_Salida(1:0).** Las señales *M(12:11)* deciden el dato que saldrá del *Multiplexor de Salida*.

- Si *M(12) = 0* y *M(11) = 0*, la salida de este *Multiplexor* es el dato que proviene de la *Unidad de Función*.
- Si *M(12) = 0* y *M(11) = 1*, la salida de este *Multiplexor* es el dato que proviene de la *Memoria de Datos*.
- Si *M(12) = 1* y *M(11) = 0*, la salida de este *Multiplexor* es el *Dato Literal* que proviene de la *Palabra de Instrucción*.
- Si *M(12) = 1* y *M(11) = 1*, la salida de este *Multiplexor* es el dato que proviene del *Puerto de Entrada*.

Señal de Control 13. **C_Ent_Sal.** La señal *M(13)* activa o desactiva los *Buffers Tri-estado* que permiten la comunicación con el *Puerto de Entrada/Salida*.

- Si *M(13) = 0*, está activo el *Puerto de entrada*.
- Si *M(13) = 1*, está activo el *Puerto de salida*.

Señal de Control 14. **C_Inhibidor.** La señal *M(14)* permite la modificación del *Puntero de Pila* en las instrucciones de *Llamada y Retorno de Sub-rutina*.

- Si *M(14) = 0*, la *Pila* no modifica su contenido.
- Si *M(14) = 1*, la *Pila* modifica su contenido.

Señal de Control 15. **C_Salto.** La señal *M(15)* controla el instante en el que, dependiendo del tipo de *Salto*, se puede realizar el mismo.

- Si *M(15) = 0*, no se realiza *Salto*.
- Si *M(15) = 1*, se realiza *Salto*.

Con todas estas *Señales de Control*, gobernamos el funcionamiento del *Procesador* permitiendo y controlando la ejecución de todas las instrucciones del *Set de Instrucciones* del *Procesador*. En los siguientes cuadros, presentamos las *Señales de Control* que implementan las *Micro-instrucciones* de cada instrucción.

- Operaciones aritmético-lógicas.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0

Cuadro 4.3: *Señales de Control* de las *Operaciones Aritmético-Lógicas*.

- Rotación y Desplazamiento.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0(1)	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Cuadro 4.4: *Señales de Control* de las *Operaciones de Rotación y Desplazamiento*.

- Saltos Condicionales e Incondicionales.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0	1

Cuadro 4.5: *Señales de Control* de las *Operaciones de Salto.*

- Escritura en *Memoria de Datos*: Dato en registro. Dirección en registro.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Cuadro 4.6: *Señales de Control* de *Escritura en* Memoria de Datos.

- Escritura en *Memoria de Datos*: Dato literal. Dirección en registro.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Cuadro 4.7: *Señales de Control* de *Escritura en* Memoria de Datos.

- Escritura en *Memoria de Datos*: Dato en registro. Dirección literal.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0

Cuadro 4.8: *Señales de Control* de *Escritura en* Memoria de Datos.

- Escritura en *Registro*.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Cuadro 4.9: *Señales de Control* de *Escritura en Registro.*

- Lectura de *Memoria de Datos*: Dirección en registro.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0

Cuadro 4.10: *Señales de Control* de *Lectura de* Memoria de Datos.

- Lectura de *Memoria de Datos*: Dirección literal.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0
0	1	0	0	1	0	0	0	1	0	0	1	0	0	0	0
0	1	0	0	1	0	0	0	1	0	0	1	0	0	0	0

Cuadro 4.11: *Señales de Control* de *Lectura de* Memoria de Datos.

- Llamada a *Sub-rutina*.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0
0	0	0	1	1	1	0	0	0	1	0	0	1	0	1	1

Cuadro 4.12: Llamada a *Sub-rutina*.

- Retorno de *Sub-rutina*.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	1	0	0	1	0	0	1	1

Cuadro 4.13: Retorno de *Sub-rutina*.

- Comparación y *Bit Test*.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0(1)	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0(1)	0	0	0	0	0	0	0	0	0	0	0

Cuadro 4.14: Comparación y *Bit Test*.

- *In*.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0

Cuadro 4.15: *In*.

- *Out*.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Cuadro 4.16: *Out*.

- *No Operación.*

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Cuadro 4.17: *No Operación.*

Estas *Señales de Control*, junto con los bits de la *Palabra de Instrucción*, implementan las *Micro-instrucciones* que forman cada instrucción del set de *Procesador*. Estas *Señales de Control* también se pueden ver en el fichero *Procesador.ucf*, que presentamos en el *Apéndice C*.

4.7. Bloque de *Gestión de Saltos.*

Este bloque se encarga de calcular las direcciones de salto y gestionar la realización o no realización de los saltos.

Las entradas del bloque de *Gestión de Saltos* son:

Entrada 1. **Direccion_actual(15:0).** Por esta entrada llega el contenido del *PC* en cada instante.

Entrada 2. **Dato_in(15:0).** El desplazamiento, en caso de salto relativo, o la dirección definitiva, en caso de salto absoluto, llegan por esta entrada. Esta línea proviene de la salida de la *Unidad de Proceso*.

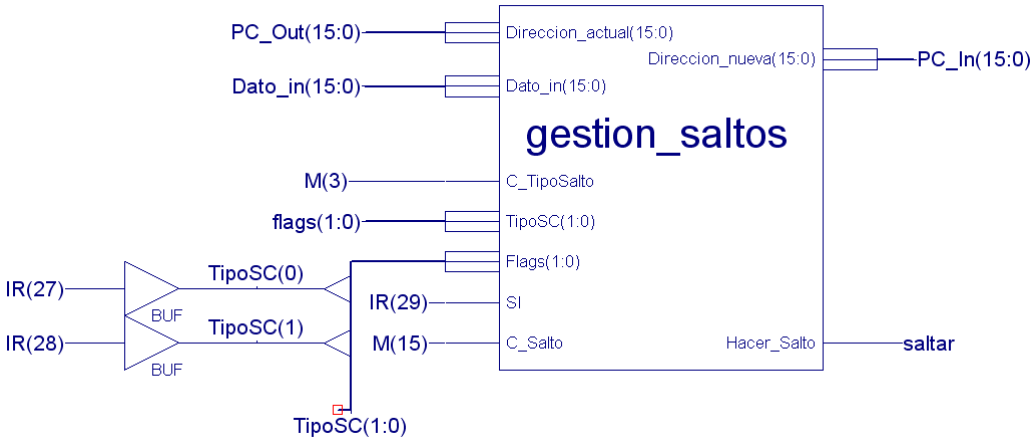
Entrada 3. **C_TipoSalto.** Por esta entrada llega la señal que selecciona si el salto a realizar es absoluto o relativo.

Entrada 4. **TipoSC(1:0).** Estas dos *Señales de Control* gobiernan la realización de los saltos condicionales.

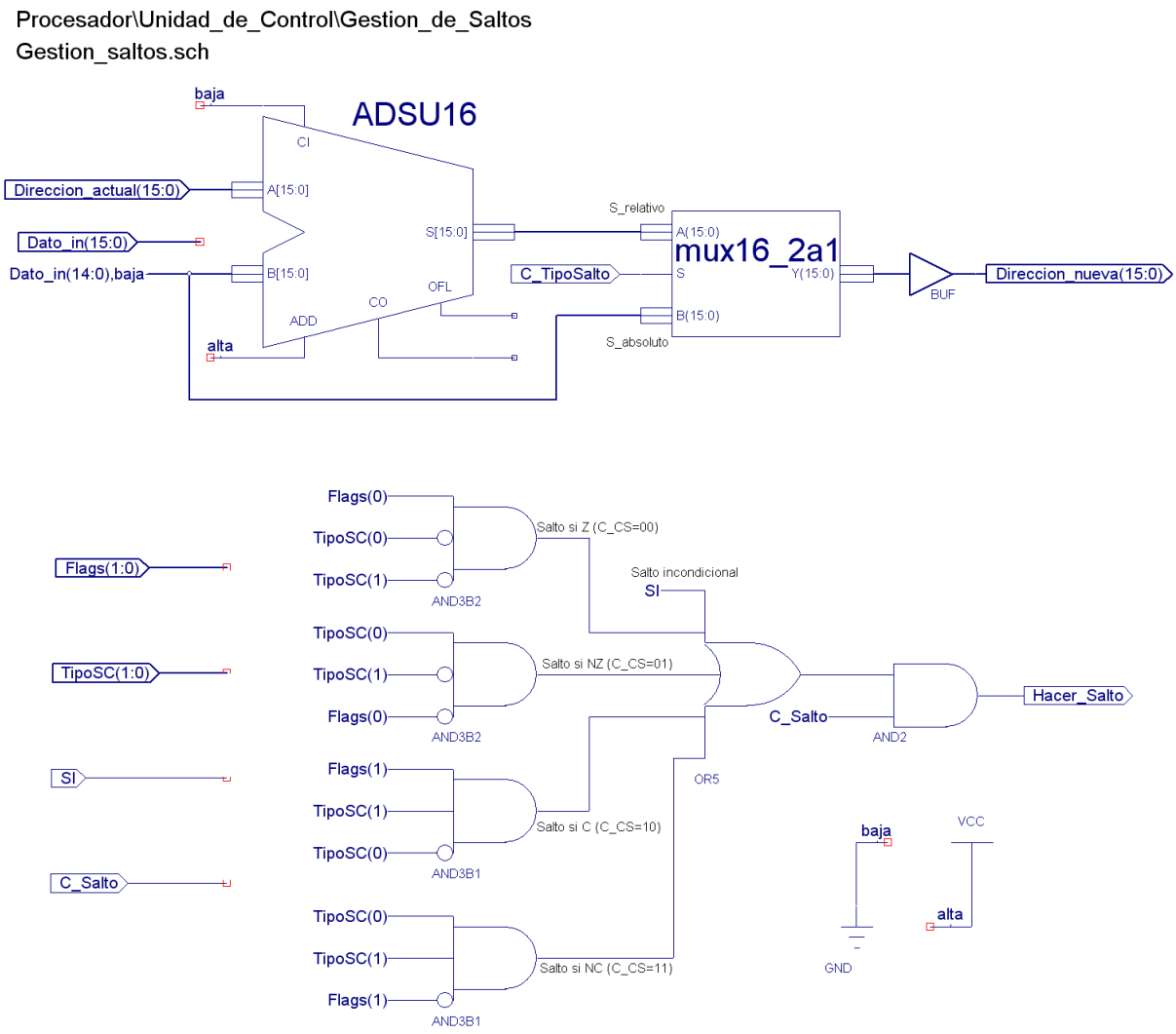
Entrada 5. **Flags(1:0).** El *Registro de Banderas*, o *Flags*, contiene el estado del procesador en cada momento. Tiene los bits correspondientes a *bit de 0* y *bit de acarreo*.

Entrada 6. **SI.** Activamos esta señal cuando el salto a realizar es incondicional.

Entrada 7. **C_Salto.** Esta señal controla el instante en el que se va a realizar el salto, es decir, el instante en el que se actualiza el contenido del *PC*.



(a) Entradas y Salidas



(b) Esquema interno

Figura 4.11: Bloque de *Gestión de Saltos.*

Las salidas del bloque de *Gestión de Saltos* son:

Salida 1. **Direccion_nueva(15:0).** Por este bus, sale la nueva dirección calculada para el *PC* que se cargará en el *Registro Contador de Programa*.

Salida 2. **Hacer_Salto.** Esta señal indica si se debe realizar el salto y en qué ciclo de reloj se hace. Está conectada a la señal de carga del *Registro PC*.

Como vemos en la figura 4.11(b), el esquema interno de este bloque está dividido en dos partes bien diferenciadas. En la parte superior, calculamos el nuevo valor del *PC*, mientras que en la parte inferior implementamos la lógica que activa la señal de carga, esto es, que permite o no el salto.

Con el fin de calcular las nuevas direcciones del *PC*, necesitamos un sumador para el cálculo de la dirección total, en el caso de saltos relativos, y un *Multiplexor de 2 a 1 de 16 bits*, para discriminar entre el salto relativo y el salto absoluto.

El sumador *ADSU16* se muestra en la figura 4.12. Se trata de un sumador/restador de 16 bits, con acarreo de entrada y de salida y con bit de desbordamiento. La entrada *CI* es la entrada de *acarreo*, y para este caso, debe ser 0. La entrada *ADD* controla la operación que realiza el bloque. En nuestro caso, es siempre una suma, y por esto, la entrada *ADD* vale 1.

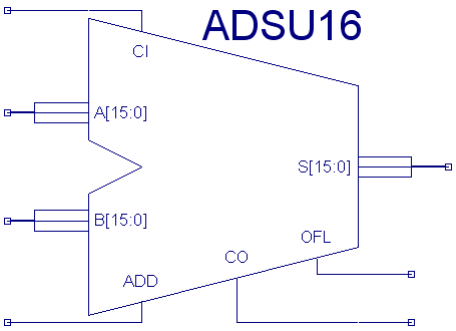


Figura 4.12: Sumador interno del Bloque de *Gestión de Saltos*.

El *Multiplexor de 2 a 1 de 16 bits* utiliza la tecnología clásica de *Multiplexores*, no la tecnología de *Buffers Tri-estado* que vimos en el apartado 4.6.3. El esquema interno está formado por 16 bloques como el que se muestra en la figura 4.13. Son 16 bloques de puertas lógicas conectados en paralelo, donde, según la *Señal de Control*, se selecciona una entrada u otra.

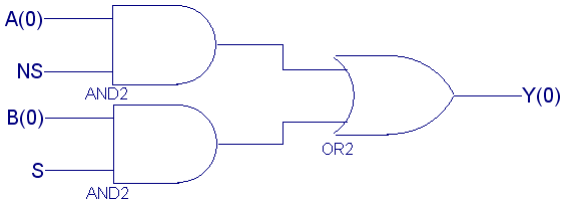
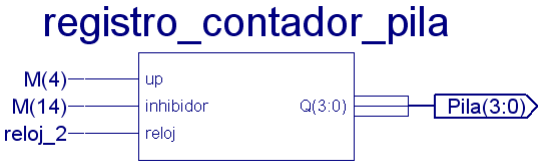


Figura 4.13: *Multiplexor* de un bit.

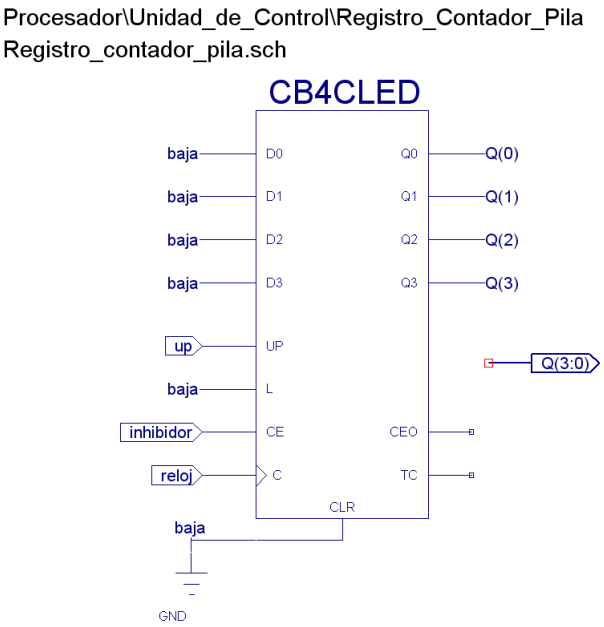
La lógica que controla la realización, o no, del salto se muestra en la parte inferior de la figura 4.11(b). Hemos definido 5 tipos posibles de salto: salto condicional si bit Zero = 0, salto condicional si *bit Zero = 1*, salto condicional si *bit Acarreo = 0*, salto condicional si *bit Acarreo = 1* y salto incondicional. Las *Señales de Control TipoSC* controlan el tipo de salto condicional que se va a realizar. Las 4 puertas *And* toman como entradas las *Señales de Control TipoSC* y el contenido del *Registro de Banderas (flags)* y activan su salida si se cumplen las condiciones de salto necesarias. Todas estas salidas, junto a la procedente de la entrada *SI* de *Salto Incondicional* entran en una puerta *or* que las unifica todas en una única señal de salto. Por último, el bit *C_Salto* gestiona el momento en que se debe realizar el salto.

4.8. *Registro de Contador de Pila.*

El bloque *Registro de Contador de Pila* está directamente relacionado con el *Registro PC* y las llamadas y retornos de subrutina. Se trata de un contador que almacena la primera dirección libre de la *Pila*.



(a) Entradas y Salidas



(b) Esquema interno

Figura 4.14: *Registro de Contador de Pila.*

Las entradas del *Registro de Contador de Pila* son:

Entrada 1. **Up.** Esta señal controla la dirección de paso del contador. Si se introduce un dato en la *Pila*, la dirección es ascendente, y si se saca un dato de la *Pila*, la dirección es descendente.

Entrada 2. **Inhibidor.** La señal de inhibición no permite avanzar al contador a no se que se haya hecho un acceso a la *Pila*.

Entrada 3. **reloj.** Por esta entrada, llega el reloj global que circula por todo el procesador.

La salida del *Registro de Contador de Pila* es:

Salida 1. **Q(3:0).** La *Pila* tiene una capacidad de 16 posiciones que se direccionan con los 4 bits que salen de este *Registro Contador de Pila*.

La *Pila* se utiliza en las instrucciones de *Llamada* y *Retorno de Sub-rutina*. El *Puntero de Pila* apunta a la primera posición libre. En una *Llamada a Sub-rutina*, el proceso es el siguiente. Introducimos el valor actual del *PC* en la posición que apunta el *Puntero de Pila* e incrementamos la posición de este *Puntero de Pila*. Para el *Retorno de Sub-rutina*, el proceso es el inverso. En primer lugar, decrementamos el *Puntero de Pila* y, en segundo lugar, leemos la posición que señala dicho *Puntero*.

Capítulo 5

Unidad de Proceso.

5.1. Introducción.

La *Unidad de Proceso* está encargada de realizar las operaciones con los datos en el procesador e informar a la *Unidad de Control* de los resultados y, en su caso, proveerle de los datos que necesite para gestionar el flujo del programa. Las operaciones que realiza la *Unidad de Proceso* con los datos son las siguientes:

- Almacenamiento de poca capacidad de datos en dispositivos temporales para el acceso inmediato a ellos.
- Operaciones aritméticas, lógicas, desplazamientos y rotaciones.
- Control del estado del procesador.
- Almacenamiento de mucha capacidad de datos en dispositivo externo con acceso lento a ellos.
- Interface de comunicación con el exterior.
- Interface de comunicación con la *Unidad de Control*.

Cada una de estas operaciones se lleva a cabo por uno de los siguientes bloques de la *Unidad de Proceso*:

1. *Banco de Registros*: banco de 16 *Registros* de 16 bits que almacena los datos durante la ejecución de un programa.
2. *Unidad de Función*: esta unidad realiza las operaciones aritméticas, lógicas, desplazamientos y rotaciones con los datos.
3. *Registro Flags*: el *Registro de Banderas* almacena información sobre la última operación realizada en la *Unidad de Función*.

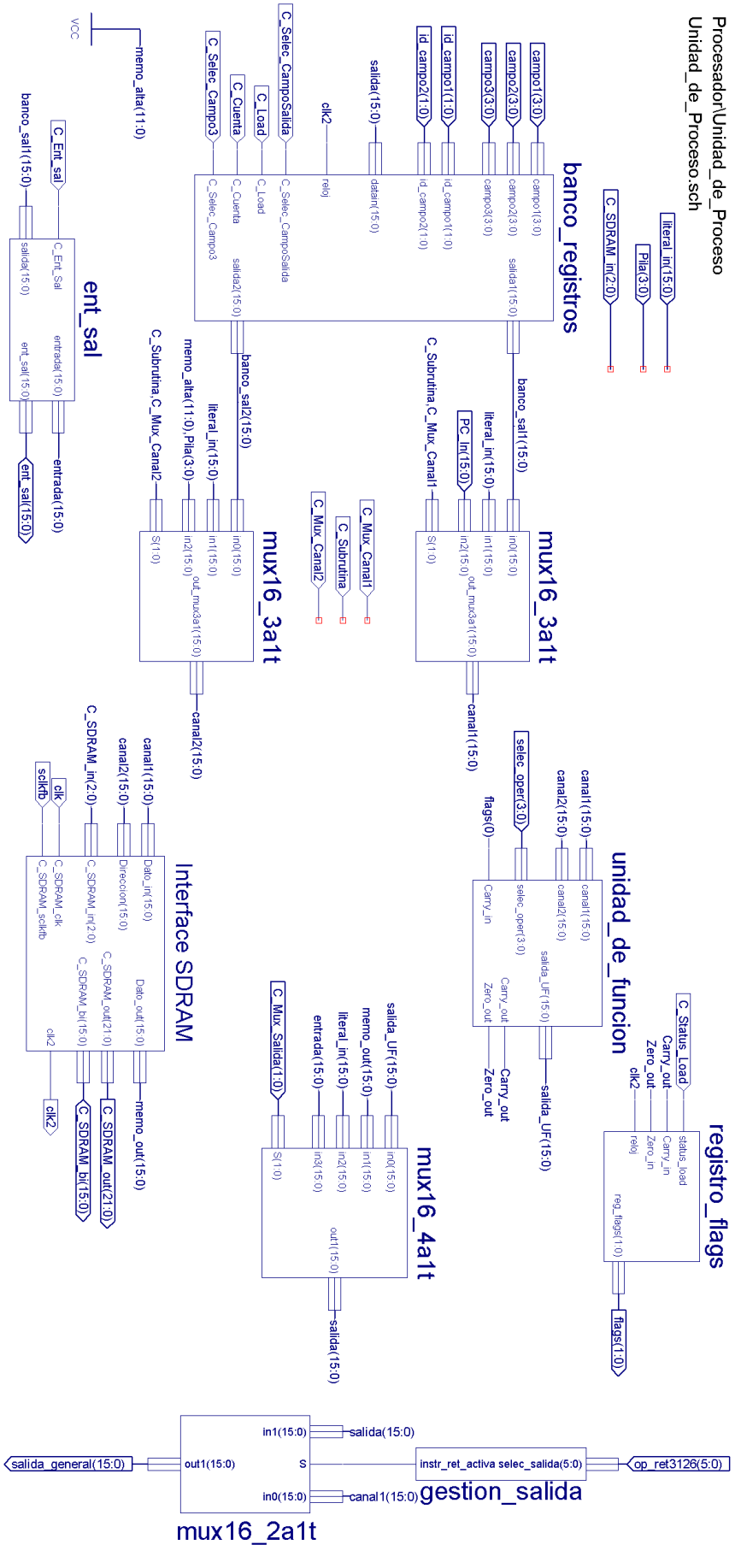


Figura 5.1: *Unidad de Proceso.*

4. *Interface con la Memoria de Datos (SDRAM)*: este bloque contiene el controlador que genera las señales para acceder a la *Memoria de Datos*

5. *Puerto de entrada/salida*: Este bloque permite tomar datos del exterior para realizar operaciones y sacar datos al exterior para su visualización o tratamiento en otros dispositivos.

6. *Multiplexores de Canal*: Permite seleccionar el dato que circulará por cada canal.

7. *Multiplexores de Canal*: Permite seleccionar el dato que saldrá hacia el exterior de la *Unidad de Proceso*.

En los siguientes apartados, analizamos en profundidad cada uno de estos bloques. Pero antes, explicamos la secuencia de operaciones que tienen lugar en la *Unidad de Proceso* para el tratamiento de los datos.

5.2. Secuenciación de operaciones en la Unidad de Proceso.

Además de las instrucciones de control de programa, hemos definido dos tipos de instrucciones. El primer tipo son las instrucciones que realizan el **tratamiento de datos** (línea contínua) y el segundo tipo son las instrucciones para **comunicación con el exterior** de la *Unidad de Proceso* (línea de puntos) a través de los puertos. Dentro de las instrucciones que realizan operaciones con los datos, también distinguimos dos sub-tipos. El primer sub-tipo está formado por las instrucciones que hacen operaciones aritméticas, lógicas, de rotación y de desplazamiento. Su camino circula a través de la *Unidad de Función*. Y el segundo sub-tipo lo forman las instrucciones que guardan o recuperan datos de la *Memoria de Datos*. Su camino pasa por el *Interface de Memoria de Datos*.

El *Banco de Registros* tiene dos buses de salida que van conectados a sendos multiplexores. Estos multiplexores discriminan los datos que van a circular por cada canal. Según la selección del multiplexor, por el canal 1 puede circular un dato procedente del *Banco de Registros*, un dato procedente de la propia *Palabra de Instrucción* (dato literal) o el propio contenido del *PC*, para las llamadas a subrutina. Por su parte, los datos que pueden circular por el canal 2 son un dato procedente del *Registro*, un dato literal o el contenido del puntero de pila. Los dos *Multiplexores Tri-estado de 3 entradas a 1 salida* son los encargados de seleccionar unas u otras entradas. Las combinaciones de datos que circulan a través de la *Unidad de Proceso* debe ser una de estas tres:

- Dos datos procedentes del *Banco de Registros*.
- Un dato procedente del *Banco de Registros* y otro dato procedente la *Palabra de Instrucción* (dato literal).
- Un dato procedente del *Registro PC* y otro procedente del *Registro Contador de Pila*.

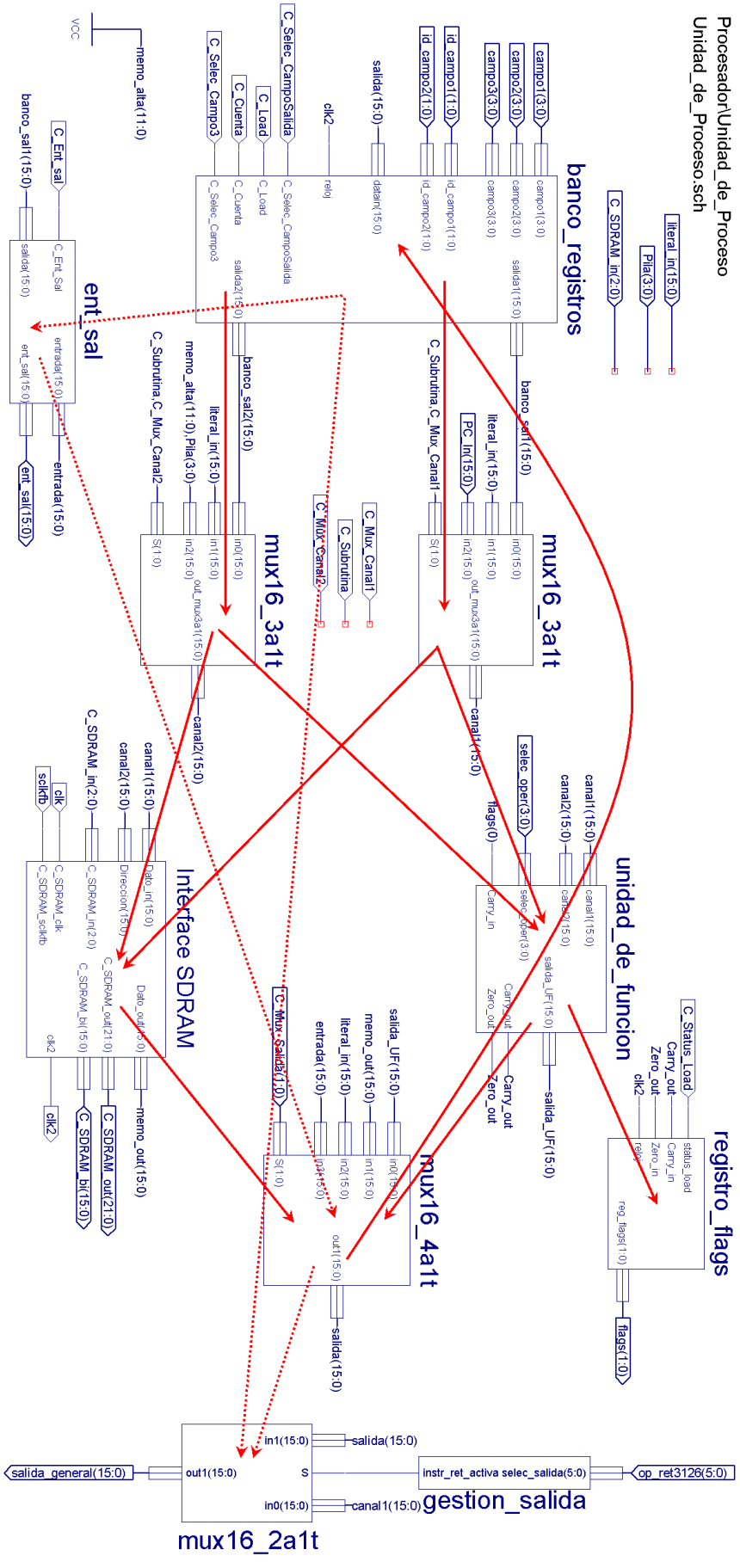


Figura 5.2: Secuencia de operaciones de la *Unidad de Proceso*.

(1)	**(0)**	**Salida**
0	0	Salida del *Banco de Registros*
0	1	Dato *Literal*
1	X	Dato para *Sub-rutinas*

Cuadro 5.1: Tabla de Verdad de los *Multiplexores Tri-estado* de *Canal*.

Las salidas de los dos *Multiplexores Tri-estado* determinan el contenido de los datos de sus respectivos canales. Estos canales están conectados a dos bloques "paralelos": la *Unidad de Función* y el bloque de *Interface con la Memoria de Datos*. Las *Señales de Control* generadas en la *Unidad de Control* y las provenientes de la *Palabra de Instrucción* determinan las operaciones que se realizan con los datos en cada bloque. Por su parte, la *Unidad de Función* tiene asociado el *Registro de Banderas* que almacena datos de la última operación que ha realizado la propia *Unidad de Función*.

A la salida de estos dos bloques, hemos colocado otro *Multiplexor Tri-estado de 4 entradas a 1 salida*. Este *Multiplexor* selecciona el dato que se realimentará hacia el el *Banco de Registros*, que saldrá del bloque de la *Unidad de Proceso* o que irá hacia el *Puerto de Salida*. El dato que sale de este *Multiplexor* puede ser: el dato procedente de la *Unidad de Función*, el dato procedente de la *Memoria de Datos*, el dato literal procedente de la *Palabra de Instrucción* o el dato procedente del *Puerto de Entrada*.

(1)	**(0)**	**Salida**
0	0	Dato de la *Unidad de Función*
0	1	Dato de la *Memoria de Instrucciones*
1	0	Dato *Literal*
1	1	Dato del *Puerto de Entrada*

Cuadro 5.2: Tabla de Verdad del *Multiplexor Tri-estado* de *Salida*.

Además, hemos implementado un bloque que actúa como *Puerto de Entrada/Salida*. Los datos que entran por el *Puerto de Entrada* se introducen en el *Multiplexor Tri-estado de 4 entradas a 1 salida* de forma que esos datos se realimentan hacia el *Banco de Registros* para guardarlos en uno de ellos. Los datos que se llevan hacia el exterior por el *Puerto de Salida* proceden de la salida 1 del *Banco de Registros*.

Por último, el *Bloque de Salida* selecciona el dato que saldrá de la *Unidad de Proceso* en dirección a la *Unidad de Control*. Este dato puede venir del *Banco de Registros* para las instrucciones de salto o del *Multiplexor Tri-estado de 4 a 1* para la instrucción de retorno de sub-rutina.

En los siguientes apartados, analizamos el diseño de cada bloque de la *Unidad de Proceso* individualmente. Para cada bloque, presentamos: entradas, salidas, esquema interno y funcionamiento.

5.3. *Banco de Registros.*

Para nuestro procesador, hemos diseñado un *Banco de Registros* de 16 *Registros* de 16 bits cada uno. Estos *Registros* se utilizan como almacenamiento inmediato de datos durante la ejecución de programas y su acceso es muy rápido. Los datos almacenados solo están disponibles mientras el programa está corriendo.

Las entradas del *Banco de Registros* son:

Entrada 1. **Campo1(3:0).** Línea de control que selecciona un *Registro* para la realización de operaciones con su contenido.

Entrada 2. **Campo2(3:0).** Línea de control que selecciona un *Registro* para la realización de operaciones con su contenido.

Entrada 3. **Campo3(3:0).** Línea de control que selecciona un *Registro* para la realización de operaciones con su contenido.

Entrada 4. **Id_campo1(1:0).** Línea de control para el gobierno de la post-modificación del contenido del dato del *Registro* direccionado por el campo 1.

Entrada 5. **Id_campo2(1:0).** Línea de control para el gobierno de la post-modificación del contenido del dato del *Registro* direccionado por el campo 2.

Entrada 6. **Datain(15:0).** Bus de datos de 16 bits utilizado para introducir un dato en el *Banco de Registros*.

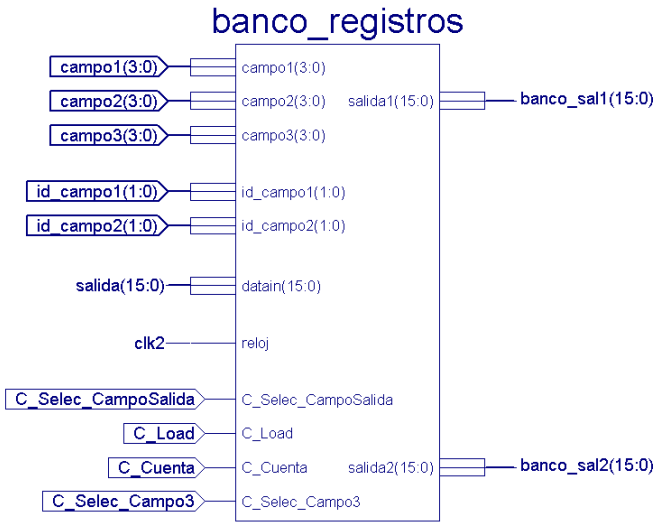
Entrada 7. **reloj.** Por esta entrada, llega el reloj global que circula por todo el procesador.

Entrada 8. **C_selec_camposalida.** *Señal de Control* que selecciona el canal por el que sale el contenido de los *Registros* direccionados por las líneas de entrada *Campo1(3:0)* y *Campo2(3:0)*.

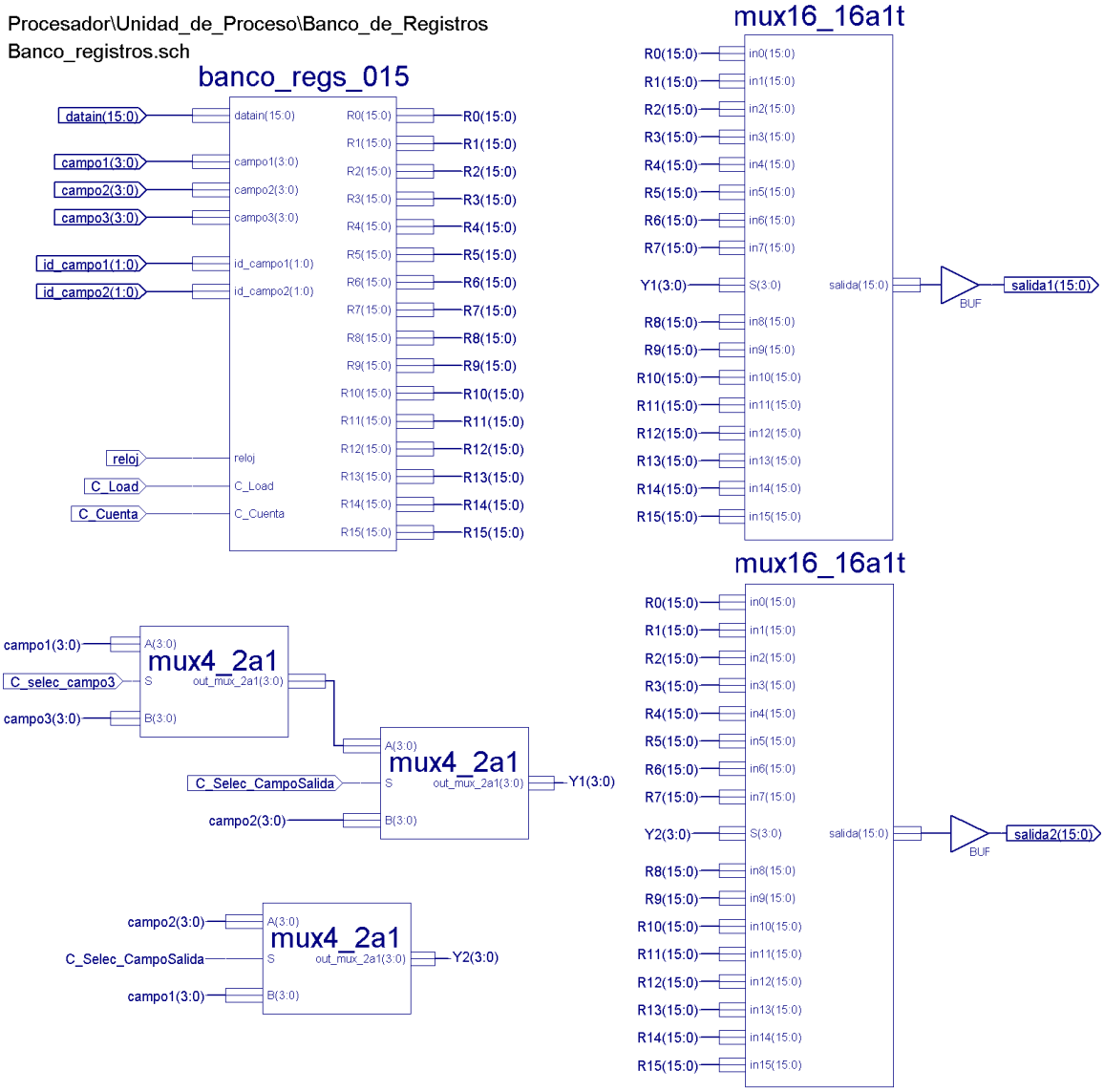
Entrada 9. **C_load.** *Señal de Control* que gobierna la carga de los *Registros*.

Entrada 10. **C_cuenta.** *Señal de Control* que gobierna las operaciones de post-modificación de datos.

Entrada 11. **C_selec_campo3.** *Señal de Control* que controla si la operación se va a realizar con el contenido del *Registro* que apunta *Campo1(3:0)* o con el contenido del *Registro* que apunta *Campo3(3:0)*.



(a) Entradas y Salidas



(b) Esquema interno

Figura 5.3: *Banco de Registros.*

Las salidas del *Banco de Registros* son:

Salida 1. **Salida1(15:0).** Bus de 16 bits de salida de datos para el canal 1.

Salida 2. **Salida2(15:0).** Bus de 16 bits de salida de datos para el canal 2.

Como vemos en la figura 5.3(a), este bloque tiene un bus de entrada de datos de 16 bits y dos buses de salida de datos de 16 bits. Esta arquitectura permite realizar dos lecturas y una escritura en un mismo ciclo de reloj. Además, con cada instrucción de lectura, permite post-incrementar, o post-decrementar, el contenido de cada *Registro*. En las instrucciones de escritura, no tiene sentido realizar esta operación.

En la figura 5.3(b), mostramos el diseño interno del *Banco de Registros*. Este diseño está formado por 4 bloques. El bloque situado en la parte superior izquierda contiene los *Registros* del *Banco*. Lo analizamos en profundidad en el apartado 5.3.2. Debajo de este bloque, situamos un diseño formado por 4 *Multiplexores*. El objetivo de este bloque es seleccionar los *Registros* con los que se van a realizar las operaciones y los canales de salida por los que saldrán esos datos. Por último, los dos bloques situados en la parte derecha son sendos *Multiplexores Tri-estado* idénticos a los presentados en el apartado 4.6.3. Su función es discriminar el dato que saldrá por los *Buses de Salida 1* y *2* respectivamente. Mostramos la estructura interna de este bloque en la figura 4.10(b). La estructura de un *Multiplexor Clásico* y del *Banco de Registros de 0 a 15* los mostramos en los siguientes sub-apartados.

5.3.1. *Multiplexor Clásico.*

En este apartado, presentamos los diseños que hemos realizado para implementar un *Multiplexor* que denominamos *Clásico* en contraposición al *Multiplexor Tri-estado* que presentamos en el apartado 4.6.3. La nomenclatura de estos *Multiplexores* es análoga a la de los *Multiplexores Tri-estado*: *MuxA_Ba1* representa un *Multiplexor* de B entradas de A bits cada una.

Las entradas del *Multiplexor Clásico* son:

Entrada 1. **A(3:0).** Línea para entrada de datos.

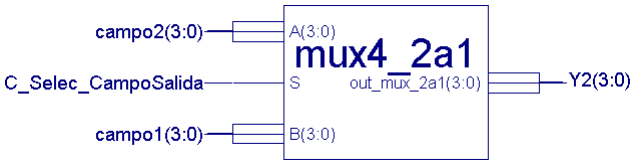
Entrada 2. **B(3:0).** Línea para entrada de datos.

Entrada 3. **S.** Línea para la selección del dato.

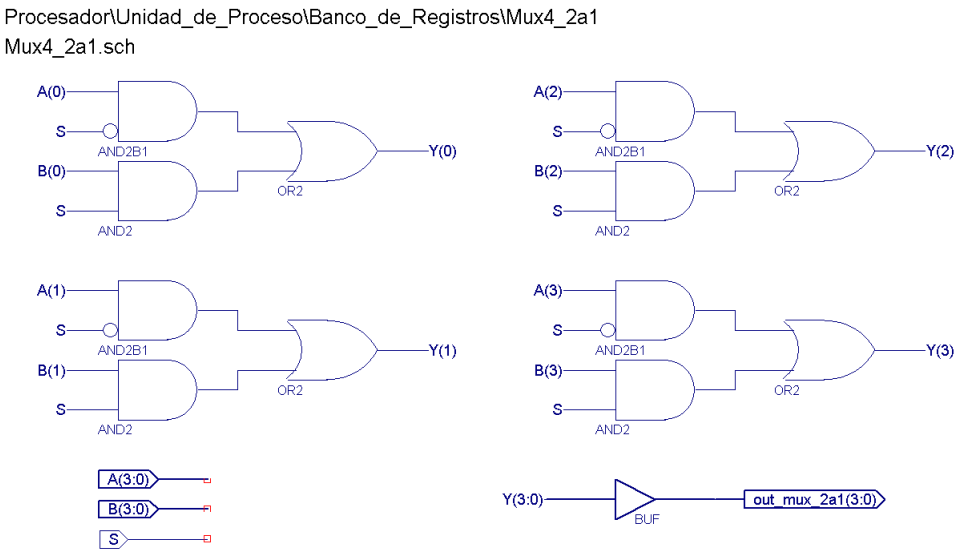
La salida que tiene el *Multiplexor Clásico* es:

Salida 1. **Out_Mux_2a1(3:0).** Bus de salida de datos.

Mostramos la estructura interna de este bloque en la figura 5.4(b). Se basa en un núcleo formado por dos puertas *And* y una puerta *Or* que se repite por cada bit de la línea de entrada. Para *Multiplexores* de otra capacidad, basta con escalar este tipo de arquitectura.



(a) Entradas y Salidas



(b) Esquema interno

Figura 5.4: *Multiplexor Clásico.*

5.3.2. *Banco de Registros de 0 a 15.*

El *Banco de Registros de 0 a 15* contiene los 16 *Registros* de que dispone este procesador.

Las entradas del *Banco de Registros de 0 a 15* son:

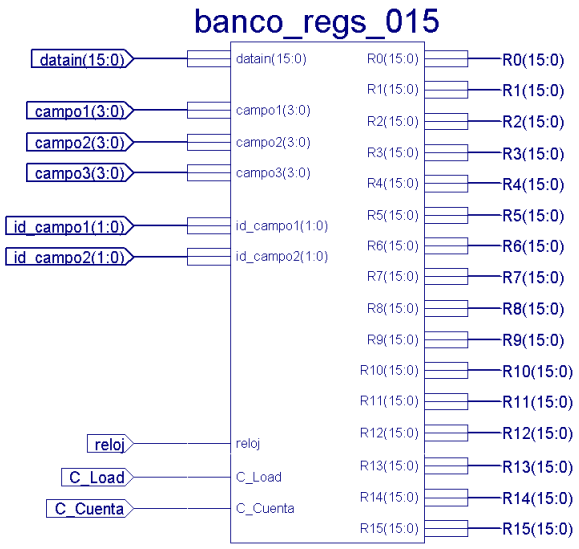
Entrada 1. **Datain(15:0).** Bus de datos de 16 bits utilizado para introducir un dato en el *Banco de Registros.*

Entrada 2. **Campo1(3:0).** Línea de control que selecciona un *Registro* para la realización de operaciones con su contenido.

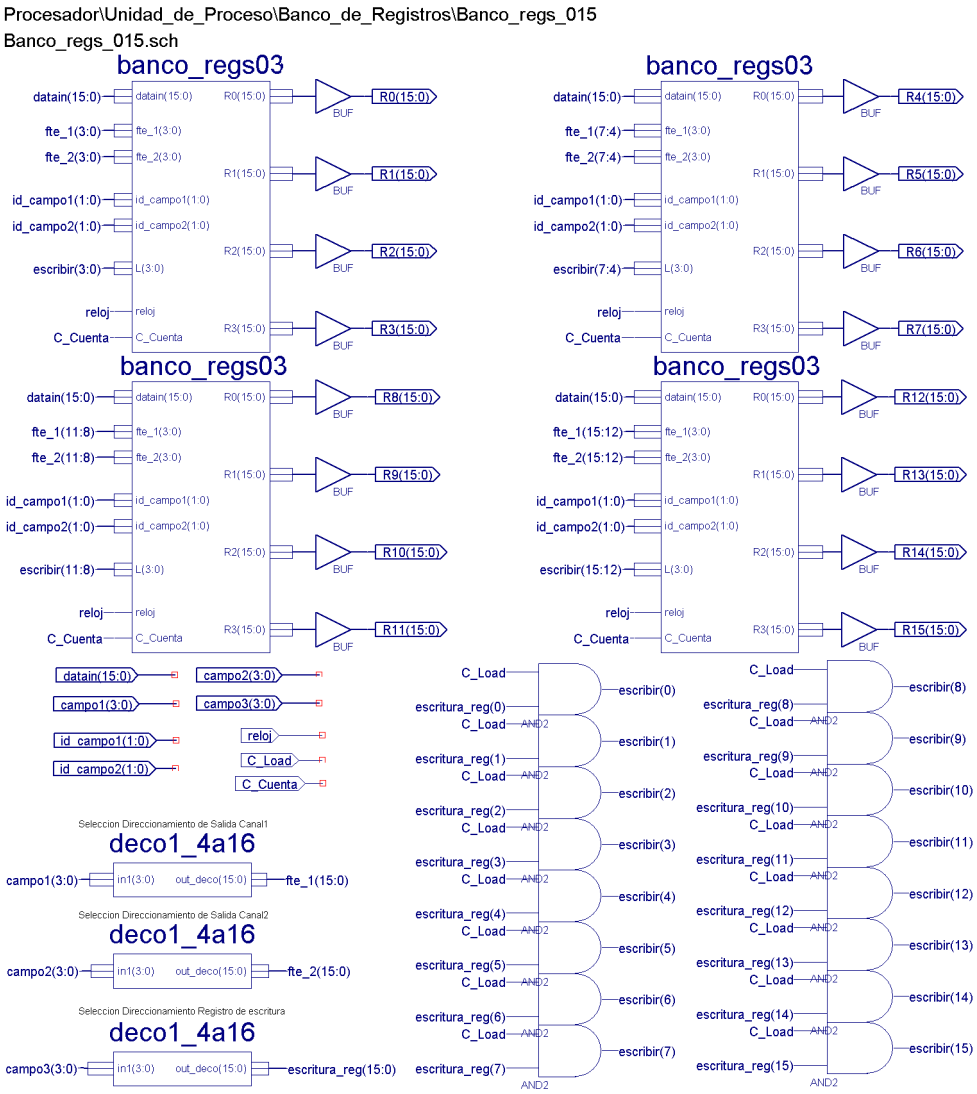
Entrada 3. **Campo2(3:0).** Línea de control que selecciona un *Registro* para la realización de operaciones con su contenido.

Entrada 4. **Campo3(3:0).** Línea de control que selecciona un *Registro* para la realización de operaciones con su contenido.

Entrada 5. **Id_campo1(1:0).** Línea de control para el gobierno de la post-modificación del contenido del dato del *Registro* direccionado por el campo 1.



(a) Entradas y Salidas



(b) Esquema interno

Figura 5.5: *Banco de Registros* de 0 a 15.

Entrada 6. **Id_campo2(1:0).** Línea de control para el gobierno de la post-modificación del contenido del dato del *Registro* direccionado por el campo 2.

Entrada 7. **reloj.** Por esta entrada, llega el reloj global que circula por todo el procesador.

Entrada 8. **C_load.** *Señal de Control* que gobierna la carga de los *Registros*.

Entrada 9. **C_cuenta.** *Señal de Control* que gobierna las operaciones de post-modificación de datos.

Las salidas del *Banco de Registros de 0 a 15* son:

Salida 1. **R0...R15(15:0).** 16 buses de datos que sacan el contenido de los 16 *Registros* de datos.

En la figura 5.5(b), mostramos el diseño interno de este bloque. Como vemos, está dividido en tres sectores. En la parte superior, disponemos de 4 bloques que contienen los *Registros* 0...3, 4...7, 8...11 y 12...15. En la parte inferior izquierda, a partir de los códigos *Campo1(3:0)*, *Campo2(3:0)* y *Campo3(3:0)*, 3 decodificadores seleccionan los *Registros* sobre los que realizamos las operaciones. En la parte inferior derecha, la *Matriz de puertas lógicas And* activa un bit para escribir en cada *Registro*, en caso de realizar una operación de escritura sobre *Registro*.

En los siguientes apartados, presentamos la estructura interna de los decodificadores y de los *Bancos de Registros de 0 a 3*.

5.3.3. Decodificadores.

Los decodificadores *Deco1_AaB* son circuitos que convierten una información binaria (codificada) de A líneas de entrada a un máximo de 2^B salidas (decodoficada).

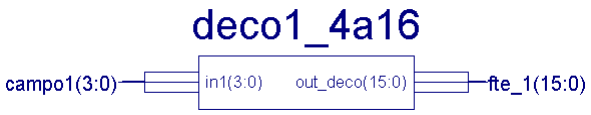
La entrada del *Decodificador* es:

Entrada 1. **in1(3:0).** Línea para entrada de datos codificados.

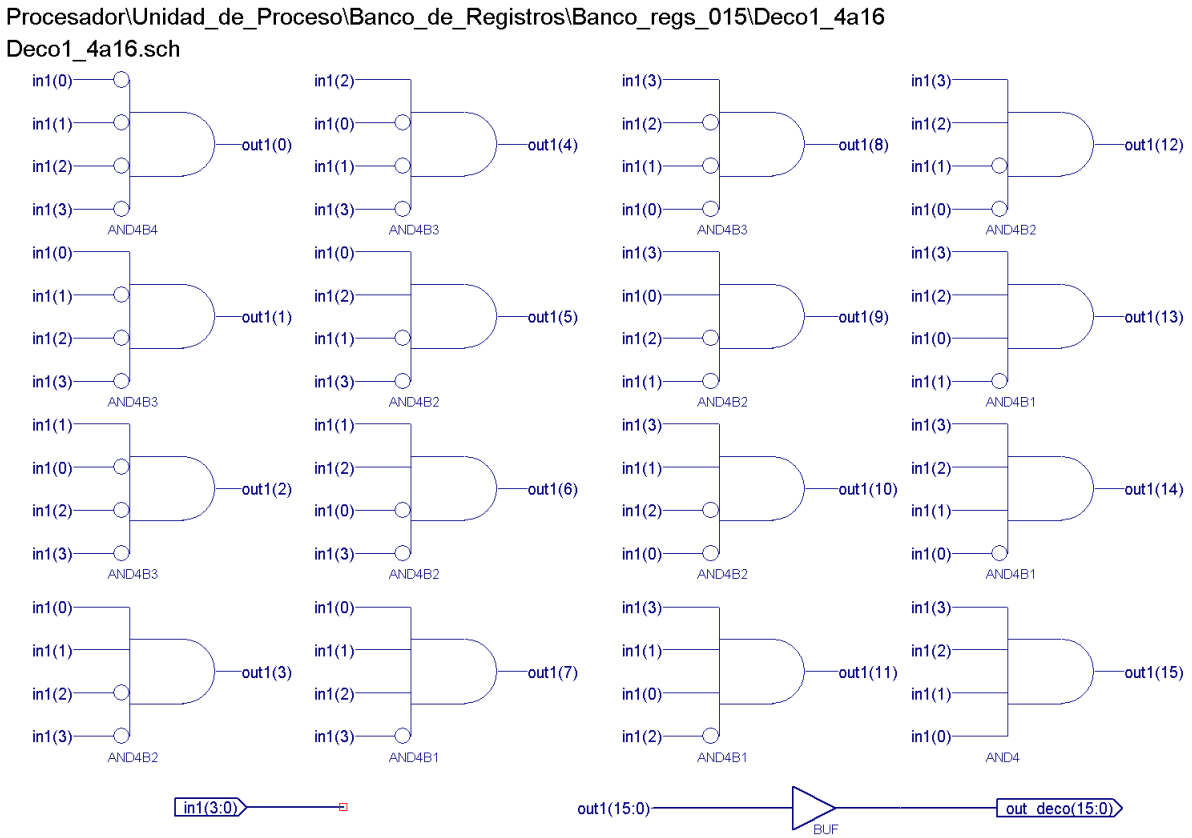
La salida del *Decodificador* es:

Salida 1. **Out_deco(15:0).** Bus de salida para la señal decodificada.

En la figura 5.6(b), mostramos la estructura interna de este bloque. Se basa en 16 puertas lógicas *AND* de 4 entradas que están implementadas de forma que solo una de las 16 puertas active su salida para cada código de las señales de entrada.



(a) Entradas y Salidas



(b) Esquema interno

Figura 5.6: *Decodoficador.*

5.3.4. *Banco de Registros de 0 a 3.*

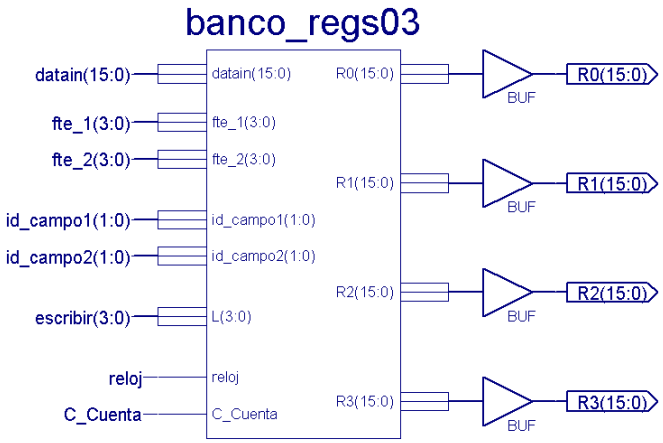
En el diseño, hemos definido 4 bloques *Banco de Registros de 0 a 3* que implementan los 16 *Registros* en grupos de 4.

Las entradas del *Banco de Registros de 0 a 3* son:

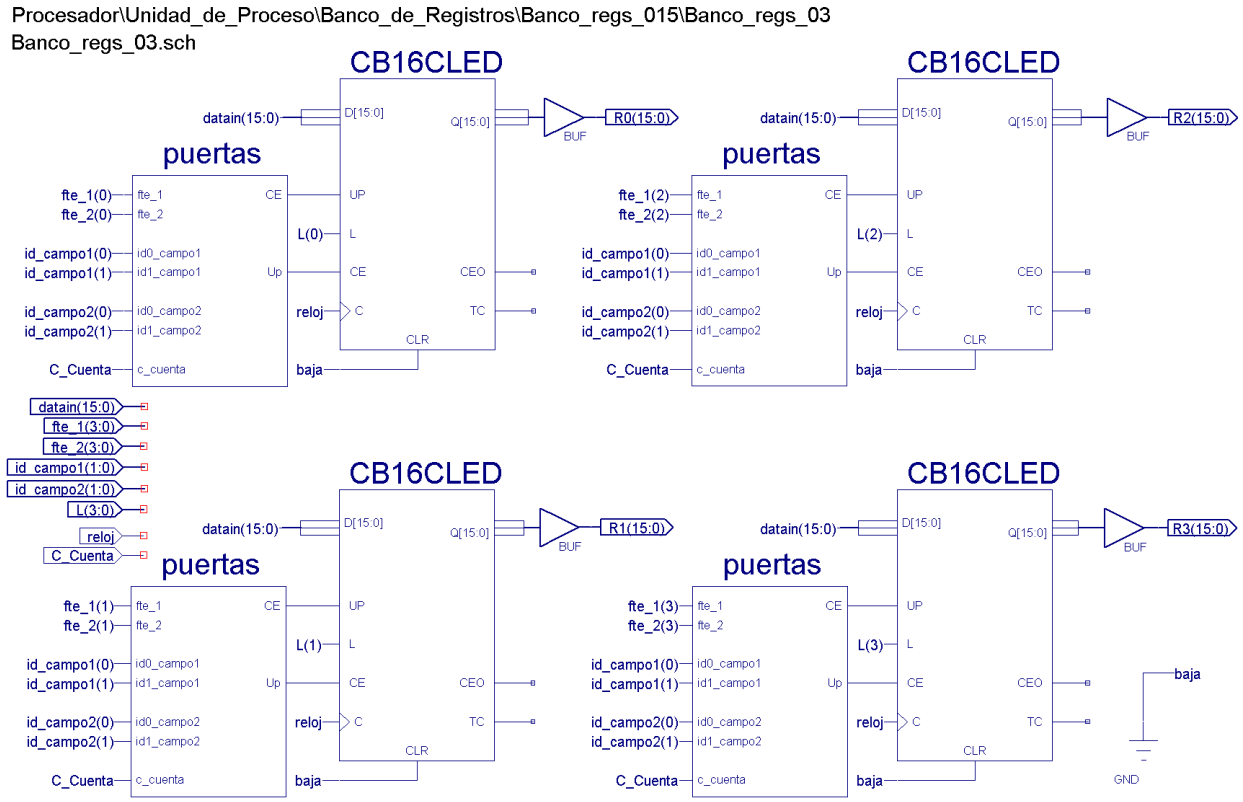
Entrada 1. **Datain(15:0).** Bus de datos de 16 bits utilizado para introducir un dato en el *Banco de Registros.*

Entrada 2. **Fte_1(3:0).** Línea de control que selecciona un *Registro* para la lectura de su contenido.

Entrada 3. **Fte_2(3:0).** Línea de control que selecciona un *Registro* para la lectura de su contenido.



(a) Entradas y Salidas



(b) Esquema interno

Figura 5.7: *Banco de Registros de 0 a 3.*

Entrada 4. **Id_campo1(1:0).** Línea de control para el gobierno de la post-modificación del contenido del dato del *Registro* direccionado por el campo 1.

Entrada 5. **Id_campo2(1:0).** Línea de control para el gobierno de la post-modificación del contenido del dato del *Registro* direccionado por el campo 2.

Entrada 6. **L(3:0).** Línea de control que selecciona un *Registro* para la escritura de un dato.

Entrada 7. **reloj.** Por esta entrada, llega el reloj global que circula por todo el procesador.

Entrada 8. **C_cuenta.** *Señal de Control* que gobierna las operaciones de post-modificación de datos.

La salidas del *Banco de Registros de 0 a 3* es:

Salida 1. **R0...R3(15:0).** 4 buses de datos que sacan el contenido de los 4 *Registros* de datos.

El esquema interno de este bloque está mostrado en la figura 5.7(b). En este figura, vemos cómo hemos implementado los *Registros* con *Registros Contadores con carga en paralelo y posibilidad de incremento y decremento (CB16CLED)*. Estos *Registros* tienen las mismas características que los *Registros CB16CLE* que presentamos en el apartado 4.3. Y añade la posibilidad de controlar el sentido de la cuenta del *Registro Contador*, es decir, permite incremento y decremento de su contenido. Para incrementar, la señal conectada a *U* debe valer 1 y, para decrementar, la señal conectada a *U* debe valer 0. El control de la señal de incremento o decremento se realiza en el bloque *Puertas*. Este bloque lo presentamos en el siguiente apartado.

5.3.5. Puertas.

El bloque *Puertas* generalas señales que permiten realizar las operaciones de post-incremento o post-decremento del dato contenido en el *Registro*.

Las entradas del bloque *Puertas* son:

Entrada 1. **Fte_1.** Línea que se activa si el *Registro* esta seleccionado por el *Campo1*.

Entrada 2. **Fte_2.** Línea que se activa si el *Registro* esta seleccionado por el *Campo2*.

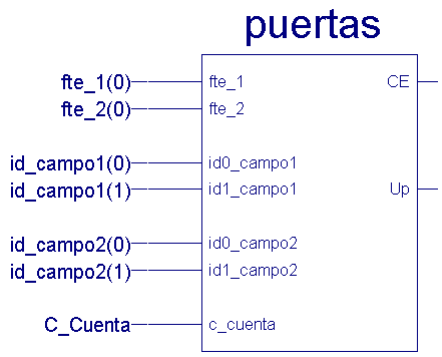
Entrada 3. **Id0_campo1.** Línea de control para el incremento o decremento del contenido del dato del *Registro* direccionado por el campo 1.

Entrada 4. **Id1_campo1.** Línea de control para realizar o no una post-modificación del contenido del dato del *Registro* direccionado por el campo 1.

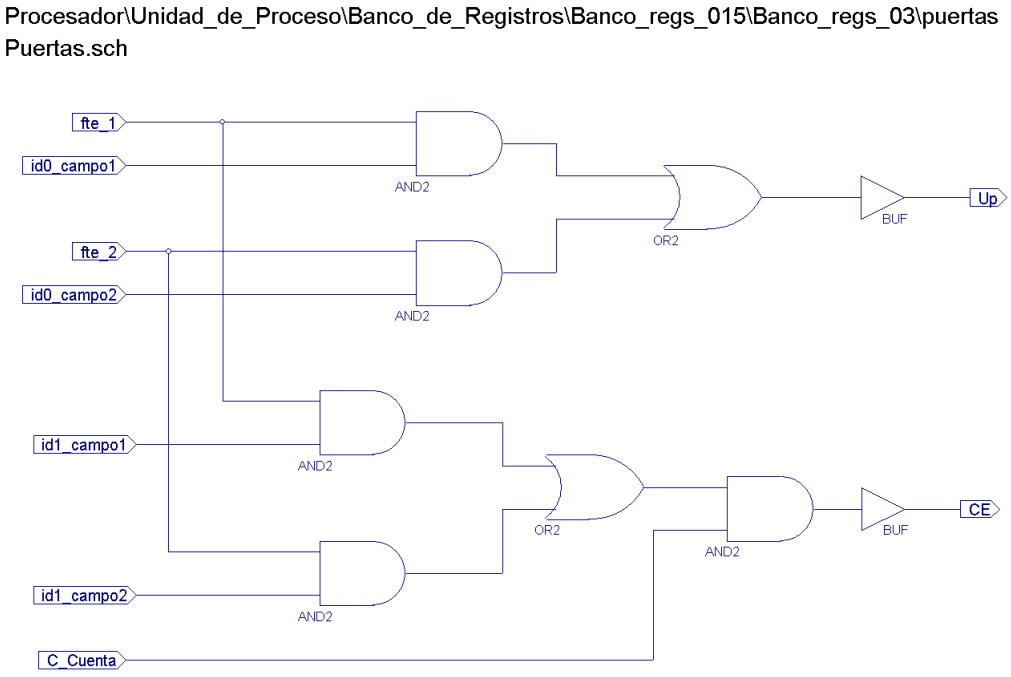
Entrada 5. **Id0_campo2.** Línea de control para el incremento o decremento del contenido del dato del *Registro* direccionado por el campo 2.

Entrada 6. **Id1_campo2.** Línea de control para realizar o no una post-modificación del contenido del dato del *Registro* direccionado por el campo 2

Entrada 7. **C_cuenta.** *Señal de Control* que gobierna las operaciones de post-modificación de datos.



(a) Entradas y Salidas



(b) Esquema interno

Figura 5.8: Generación de Señales de Control para cada *Registro*.

Las salidas del bloque *Puertas* son:

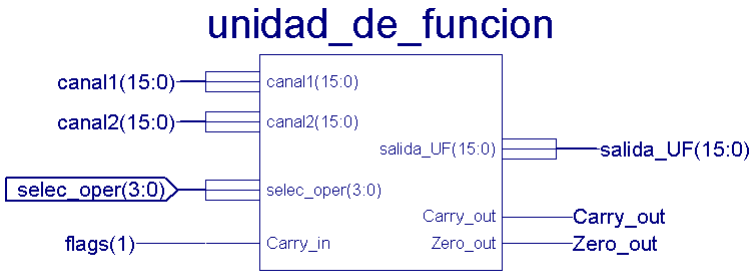
Salida 1. **Up.** Señal que indica la dirección de la post-modificación del dato (incremento o decremento).

Salida 2. **Ce.** Señal que indica si se realiza post-modificación del dato.

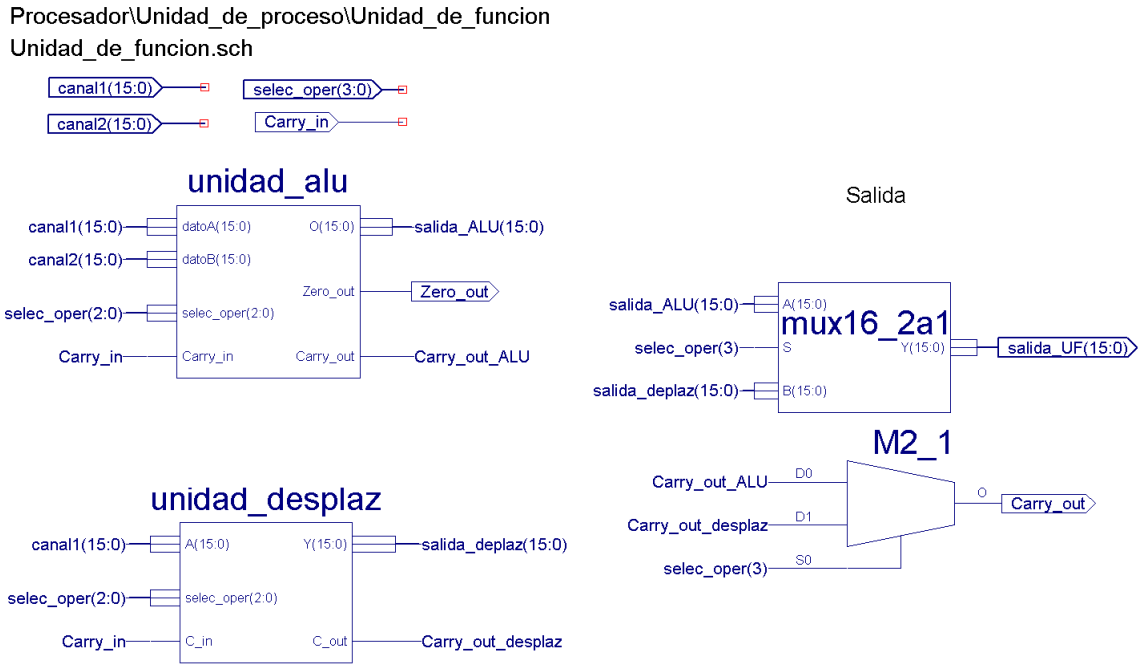
La estructura interna se muestra en la figura 5.8(b). Se basa en puertas lógicas que, según las entradas proporcionadas por los campos *Fte_ X* e *IdY_ campoX*, generan las señales que provocan un incremento o un decremento de una unidad en el *Registro Contador*. Así, implementamos fácilmente las señales que gobiernan las operaciones de post-incremento o post-decremento.

5.4. *Unidad de Función.*

El bloque *Unidad de Función* es el encargado de realizar todas las operaciones de tratamiento de datos en el procesador. Realiza operaciones aritméticas, lógicas, desplazamientos y rotaciones.



(a) Entradas y Salidas



(b) Esquema interno

Figura 5.9: *Unidad de Función.*

Las entradas del bloque de *Unidad de Función* son:

Entrada 1. **Canal1(15:0).** Entrada para el bus de datos (15:0) proveniente del canal 1.

Entrada 2. **Canal2(15:0).** Entrada para el bus de datos (15:0) proveniente del canal 2.

Entrada 3. **Selec_oper(3:0).** *Señales de Control* que deciden la operación que se va a realizar dentro de la *Unidad de Función*. Estas señales vienen definidas por los bits *IR(31), IR(29:27)* de la *Palabra de Instrucción*.

Entrada 4. **Carry_in.** Entrada de acarreo para las operaciones con acarreo. Procede del bit 1 del *Registro Flags*.

Las salidas del bloque de *Unidad de Función* son:

Salida 1. **Salida_UF(15:0).** Bus de salida que lleva el dato tratado hasta el *Multiplexor de Salida*.

Salida 2. **Carry_out.** *Bit de Acarreo* generado por la operación realizada.

Salida 3. **Zero_out.** *Bit de Cero* generado por la operación realizada.

Como hemos visto en anteriores apartados, las operaciones de tratamiento de datos se pueden dividir en dos tipos:

1. *Operaciones aritmético-lógicas.* Se trata de sumas y restas, sin y con acarreo, y operaciones lógicas.
2. *Operaciones de rotación y desplazamiento.* Son movimientos de 1 bit hacia la derecha o izquierda sin y con el *Bit de Acarreo*.

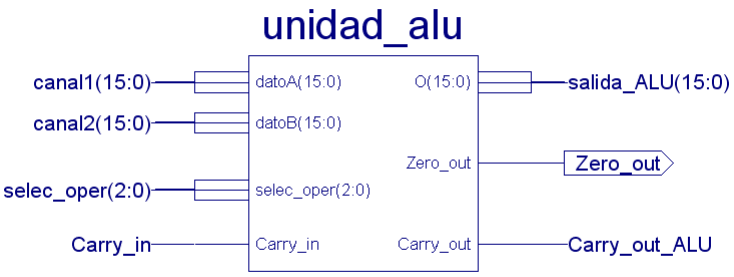
Para la implementación de estos dos tipos de operaciones, hemos diseñado dos bloques, como vemos en la figura 5.9(b). El primero de ellos es la *Unidad Aritmético-Lógica* que realiza sumas, restas y operaciones lógicas. La *Unidad Desplazadora* resuelve el tema de las operaciones de desplazamiento y rotación. Además, un tercer bloque de salida contiene un *Multiplexor* que toma la salida de los dos bloques y saca al exterior de la *Unidad de Función* el dato correcto según la instrucción realizada.

En los siguientes sub-apartados, presentamos estos bloques.

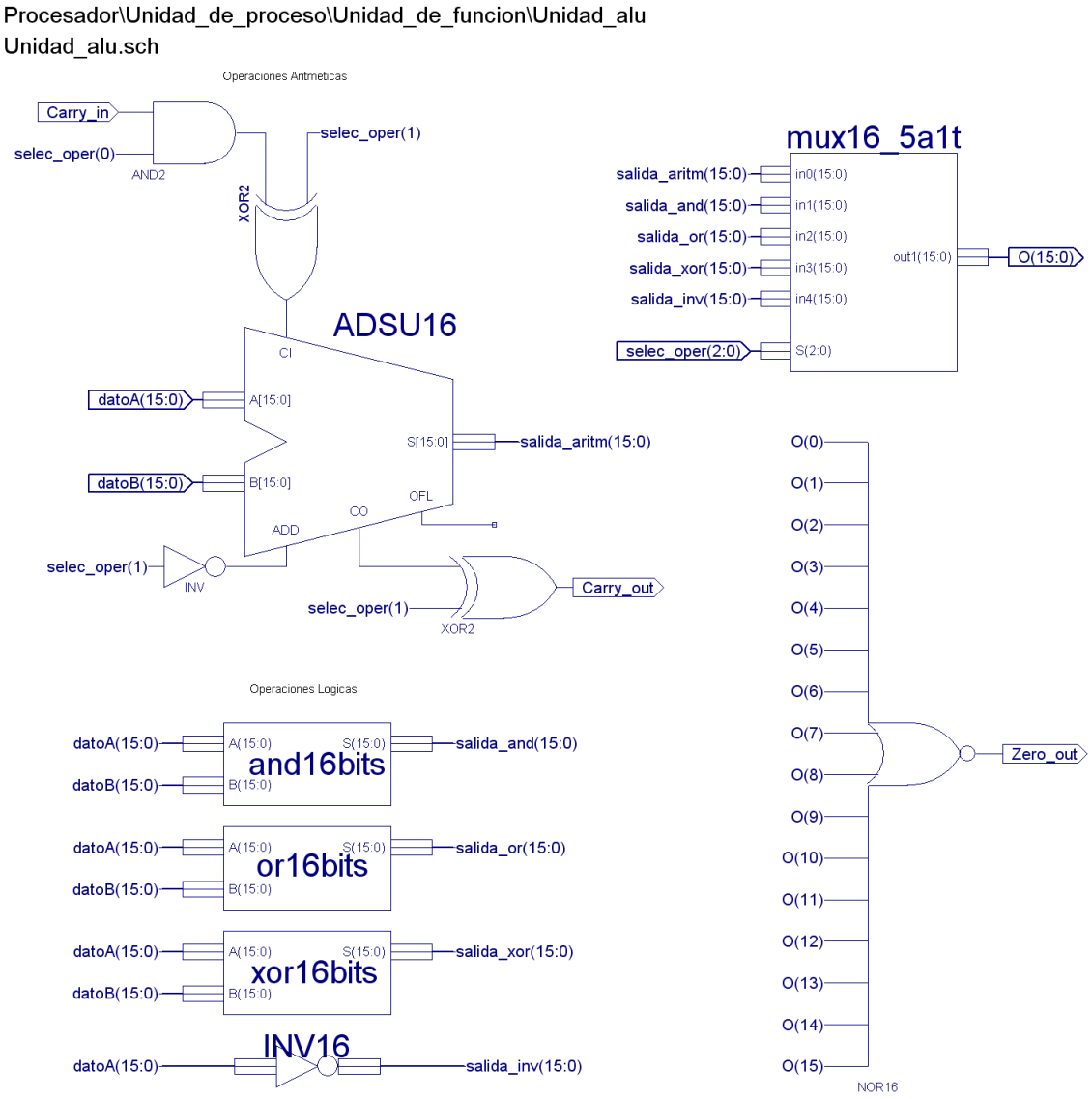
5.4.1. *Unidad Aritmético-Lógica.*

La *Unidad Aritmético-Lógica* realiza las siguientes operaciones aritméticas y lógicas:

- Suma sin acarreo.
- Suma con acarreo.
- Resta sin acarreo.
- Resta con acarreo.
- *And.*
- *Or.*
- *Xor.*
- *Not.*



(a) Entradas y Salidas



(b) Esquema interno

Figura 5.10: *Unidad Aritmético-Lógica.*

Las entradas de la *Unidad Aritmético-Lógica* son:

Entrada 1. **DatoA(15:0).** Entrada para el bus de datos (15:0) proveniente del *Multiplexor Tri-estado del Canal 1.*

Entrada 2. **DatoB(15:0).** Entrada para el bus de datos (15:0) proveniente del *Multiplexor Tri-estado del Canal 2.*

Entrada 3. **Selec_oper(2:0).** Líneas de control que seleccionan la operación que realiza la *Unidad Aritmético-Lógica.*

Entrada 4. **Carry_in.** *Bit de Acarreo* de entrada para las operaciones con acarreo.

Las salidas de la *Unidad Aritmético-Lógica* son:

Salida 1. **O(15:0).** Bus de salida de 16 bits de la *Unidad Aritmético-Lógica.*

Salida 2. **Carry_out.** *Bit de Acarreo* generado al realizar una operación aritmética.

Salida 3. **Zero_out.** *Bit de Cero* generado al realizar una operación aritmética o lógica.

Como vemos en la figura 5.10(b), este bloque está dividido en tres partes: un bloque realiza las operaciones aritméticas, otro bloque realiza las operaciones lógicas y un tercer bloque selecciona la salida. Los dos bloques de operación realizan la operación con los datos simultáneamente, y es este tercer bloque de salida quien selecciona el dato de la operación correcta.

El bloque de la operación aritmética es un sumador/restador *ADSU16* como el presentado en el apartado 4.7. El bit *Selec_ oper(1)* selecciona la operación aritmética: suma o resta. El bit *Selec_ oper(0)* controla si la operación que se va a realizar es sin o con acarreo. En realidad, todas las operaciones son con acarreo, pero el bit *Selec_ oper(0)* pasa por una puerta *And* junto al *Bit de Acarreo*, de forma que si el bit *Selec_ oper(0)* vale 0, el acarreo que entra en el sumador vale 0. De este forma implementamos las operaciones sin acarreo.

El bloque de la operación lógica está formado por cuatro unidades: *Unidad And*, *Unidad Or*, *Unidad Xor* y *Unidad Not*. Cada una realiza su operación sobre el dato simultáneamente.

En el bloque de salida, el *Multiplexor Tri-estado de 5 a 1 de 16 bits* toma todas las entradas: la del bloque aritmético y las cuatro del bloque lógico y, según la *Señal de Control*, selecciona una de ellas como salida de la *Unidad Aritmético-Lógica.* La arquitectura de este multiplexor es análoga a la vista en la figura 4.10(b) del apartado 4.6.3. El *Bit de Acarreo* solo se genera por el sumador, ya que no tiene sentido en una operación lógica. Pero el *Bit de Cero* se activa si todos los bits del dato de salida son 0. La generación de este bit se implementa con una puerta *Or* de 16 entradas, como mostramos en la figura 5.10(b).

(2)	(1)	(0)	Salida
0	X	X	Salida aritmética
1	0	0	Salida And
1	0	1	Salida Or
1	1	0	Salida Xor
1	1	1	Salida Not

Cuadro 5.3: Tabla de Verdad del *Multiplexor de Salida* de la *Unidad Aritmético-Lógica*.

5.4.2. *Unidad de Desplazamiento.*

La *Unidad de Desplazamiento* realiza las operaciones de desplazamiento y rotación. En cada instrucción desplaza, o rota, un solo bit. Por tanto, para hacer, por ejemplo, una rotación de tres bits, es necesario utilizar tres veces la instrucción de rotación.

Las entradas de la *Unidad de Desplazamiento* son:

Entrada 1. **A(15:0).** Entrada para el bus de datos (15:0) proveniente del canal 1.

Entrada 2. **Selec_oper(2:0).** Líneas de control que seleccionan la operación a realizar.

Entrada 3. **Carry_in.** *Bit de Acarreo* de entrada para las operaciones con acarreo.

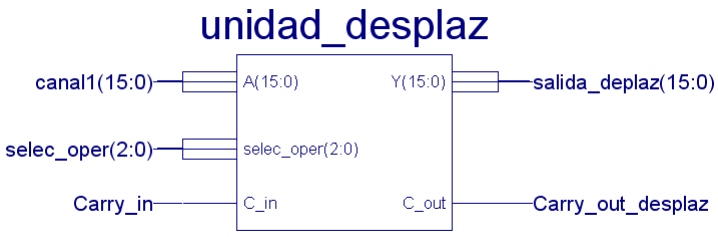
Las salida de la *Unidad de Desplazamiento* son:

Salida 1. **O(15:0).** Bus de salida de 16 bits de la *Unidad de Desplazamiento*.

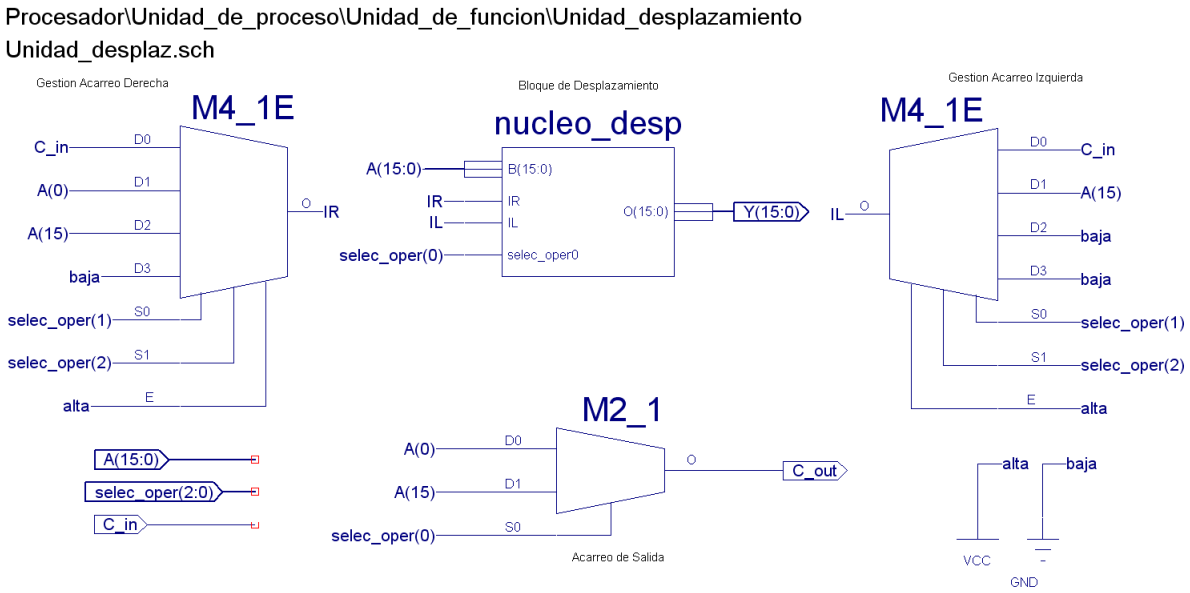
Salida 2. **Carry_out.** *Bit de Acarreo* generado por al realizar una operación.

En la figura 5.11(b), mostramos el esquema interno de este bloque. En él, destaca el bloque central *Núcleo de Desplazamiento* que realiza la propia operación de desplazamiento como tal. Los bloques laterales están diseñados para generar los bits que entran por la derecha o por la izquierda al dato que se va a desplazar o rotar en el *Núcleo de Desplazamiento*. Y el bloque inferior genera el acarreo de salida según la operación que se ha realizado.

Los bloques laterales son los bloques de gestión del *Bit de Acarreo*. Seleccionan las entradas que irán al bit más y al menos significativo del *Núcleo de Desplazamiento* según la instrucción realizada. Simplemente son *Multiplexores de 4 a 1 de 1 bit* que, según la instrucción a ejecutar, proveen del bit adecuado al bloque *Núcleo de Desplazamiento*. El bloque que gestiona el acarreo es similar. También es un *Multiplexor de 2 a 1 de 1 bit* cuyas entradas son el bit más significativo y el menos significativo del dato que se va a desplazar o rotar. Entonces, según sea el sentido del desplazamiento, el acarreo será uno u otro.



(a) Entradas y Salidas



(b) Esquema interno

Figura 5.11: *Unidad de Desplazamiento.*

En el siguiente apartado, mostramos la estructura del *Núcleo de Desplazamiento* y explicamos su funcionamiento para completar las operaciones de desplazamiento y rotación.

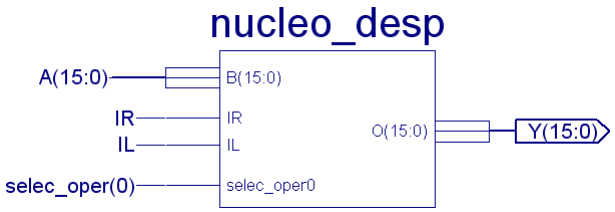
5.4.3. *Núcleo de Desplazamiento.*

El *Núcleo de Desplazamiento* desplaza una posición todos los bits del dato que entra según el sentido seleccionado por la entrada de control correspondiente.

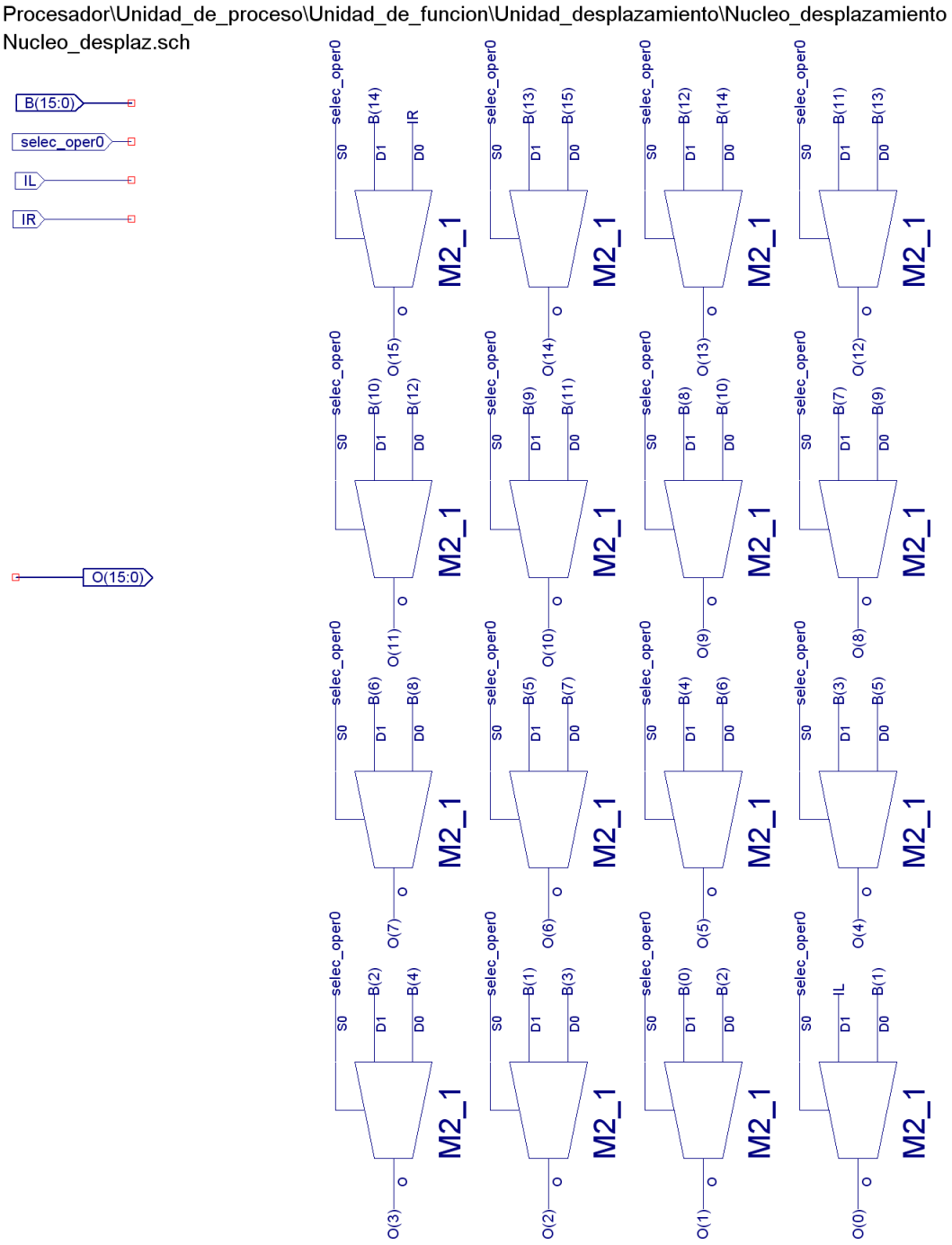
Las entradas del *Núcleo de Desplazamiento* son:

Entrada 1. **B(15:0).** Entrada para el bus de datos (15:0) proveniente del canal 1.

Entrada 2. **IR.** Bit que ocupará la posición del más significativo en un movimiento a la derecha.



(a) Entradas y Salidas



(b) Esquema interno

Figura 5.12: *Núcleo de Desplazamiento.*

Entrada 3. **IL.** Bit que ocupará la posición del menos significativo en un movimiento a la izquierda.

Entrada 4. **Selec_oper(0).** Bit de control que indica el sentido del movimiento de datos.

La salida del *Núcleo de Desplazamiento* es:

Salida 1. **O(15:0).** Bus de salida de 16 bits del *Núcleo de Desplazamiento.*

El diseño interno de este bloque se basa en la técnica de *Multiplexores* para realizar el movimiento de datos. Cada posición tiene asociado un *Multiplexor* a cuyas entradas van conectados los *Multiplexores* de su izquierda y de su derecha. Según el bit de selección de sentido de desplazamiento de datos, el *Multiplexor* saca una de sus dos entradas, implementando, de esta forma, el movimiento de datos. Recordamos que los *Multiplexores* de los bits extremos tienen una de sus entradas conectada a los bits generados por los bloques de acarreo vistos en el apartado anterior.

Para terminar este apartado de la *Unidad de Función*, presentamos el Cuadro B.1 con las operaciones que se realizan según las cuatro *Señales de Control* aplicadas.

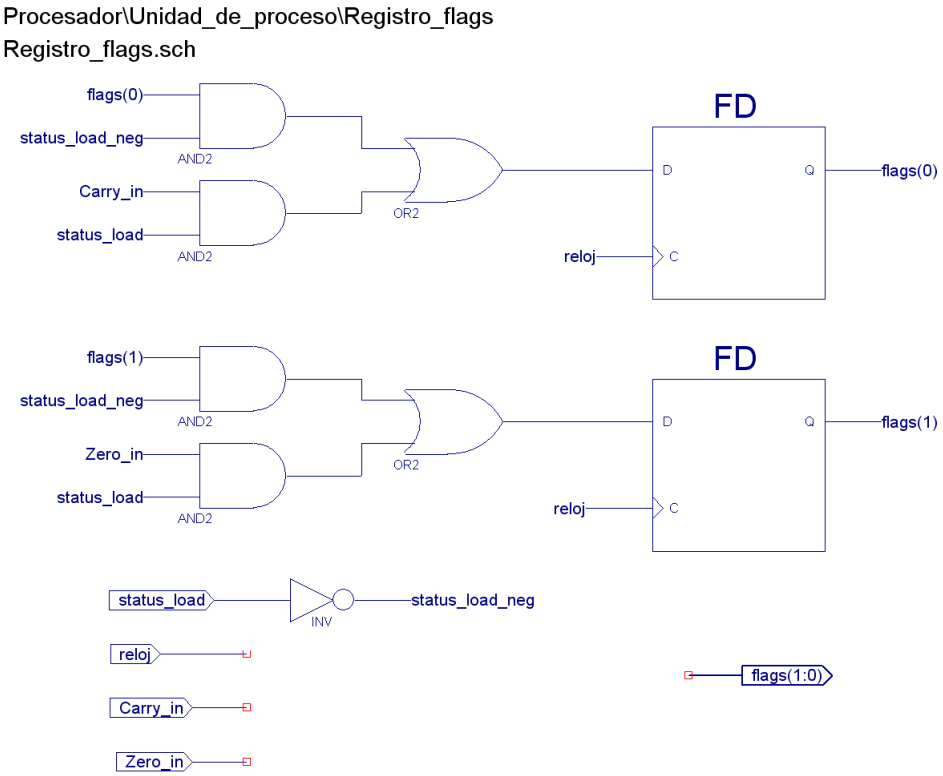
(3)	**(2)**	**(1)**	**(0)**	**Operación**
0	0	0	0	Suma sin acarreo
0	0	0	1	Suma con acarreo
0	0	1	0	Resta sin acarreo
0	0	1	1	Resta con acarreo
0	1	0	0	And
0	1	0	1	Or
0	1	1	0	Xor
0	1	1	1	Not
1	0	0	0	Rotación con acarreo a la derecha
1	0	0	1	Rotación con acarreo a izquierda
1	0	1	0	Rotación sin acarreo a la derecha
1	0	1	1	Rotación sin acarreo a la izquierda
1	1	0	0	Desplazamiento aritmético con acarreo a la derecha
1	1	0	1	Desplazamiento aritmético con acarreo a la izquierda
1	1	1	0	Desplazamiento lógico con acarreo a la derecha
1	1	1	1	Desplazamiento lógico con acarreo a la izquierda

Cuadro 5.4: Código de operaciones de la *Unidad de Función.*

5.5. *Registro de Banderas (Flags).*

Asociado a la *Unidad de Función*, hemos diseñado el *Registro de Banderas (Flags)*. Este *Registro* almacena el estado del procesador después de cada instrucción y es totalmente necesario para realizar saltos condicionales.

(a) Entradas y Salidas



(b) Esquema interno

Figura 5.13: *Registro de Banderas (Flags).*

Las entradas del *Registro de Banderas (Flags)* son:

Entrada 1. **Status_load.** Lines para el control de la actualización del *Registro de Banderas*, ya que no interesa actualizar el *Registro Flags* en todas las instrucciones.

Entrada 2. **Carry_in.** Señal de *Acarreo* procedente de la *Unidad de Función*.

Entrada 3. **Zero_in.** Señal de *Cero* procedente de la *Unidad de Función.*

Entrada 4. **reloj.** Por esta entrada, llega el reloj global que circula por todo el procesador.

La salida del *Registro de Banderas (Flags)* es:

Salida 1. **flags(1:0).** Bus de salida que lleva el contenido del *Registro Flags.*

El *Registro Flags* almacena el bit de *Acarreo* y el bit de *Cero.* Como vemos en la figura, está implementado con dos *Registros tipo D*, como los vistos en el apartado 4.5. En cada ciclo de reloj, escribimos un dato en el *Registro*, pero dependiendo de la *Señal de Control Status_load*, escribimos el dato nuevo o el que ya tenía, es decir, actualizamos o mantenemos el valor anterior.

5.6. *Interface con la Memoria de Datos.*

Este bloque actúa como *Interface entre el Procesador y la Memoria de Datos (SDRAM).* Dispone de un controlador que genera todas las señales que gobiernan el acceso y mantenimiento de la *Memoria SDRAM.*

Las entradas de la *Interface con la Memoria de Datos* son:

Entrada 1. **Dato_in(15:0).** Bus de entrada (15:0) para la línea de datos.

Entrada 2. **Direccion(22:0).** Bus de entrada (22:0) para la línea de direcciones.

Entrada 3. **C_SDRAM_in(2:0).** Entrada de control para las operaciones del controlador de la *Memoria SDRAM.*

Entrada 4. **C_SDRAM_clk.** Entrada de reloj principal del procesador.

Entrada 5. **C_SDRAM_sclkfb.** Entrada de reloj secundario de realimentación de la *Memoria SDRAM.*

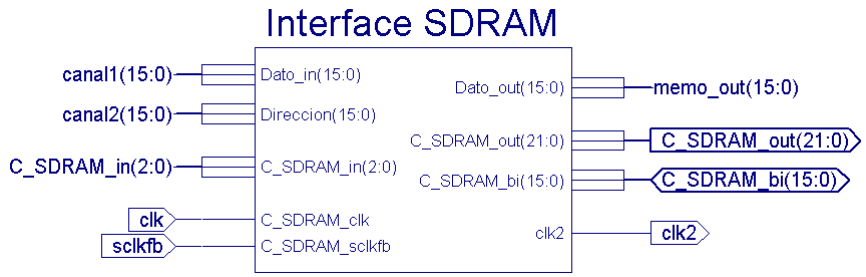
Las salidas de la *Interface con la Memoria de Datos* son:

Salida 1. **Dato_out(15:0).** Bus de salida para los datos almacenados en la *Memoria de Datos.*

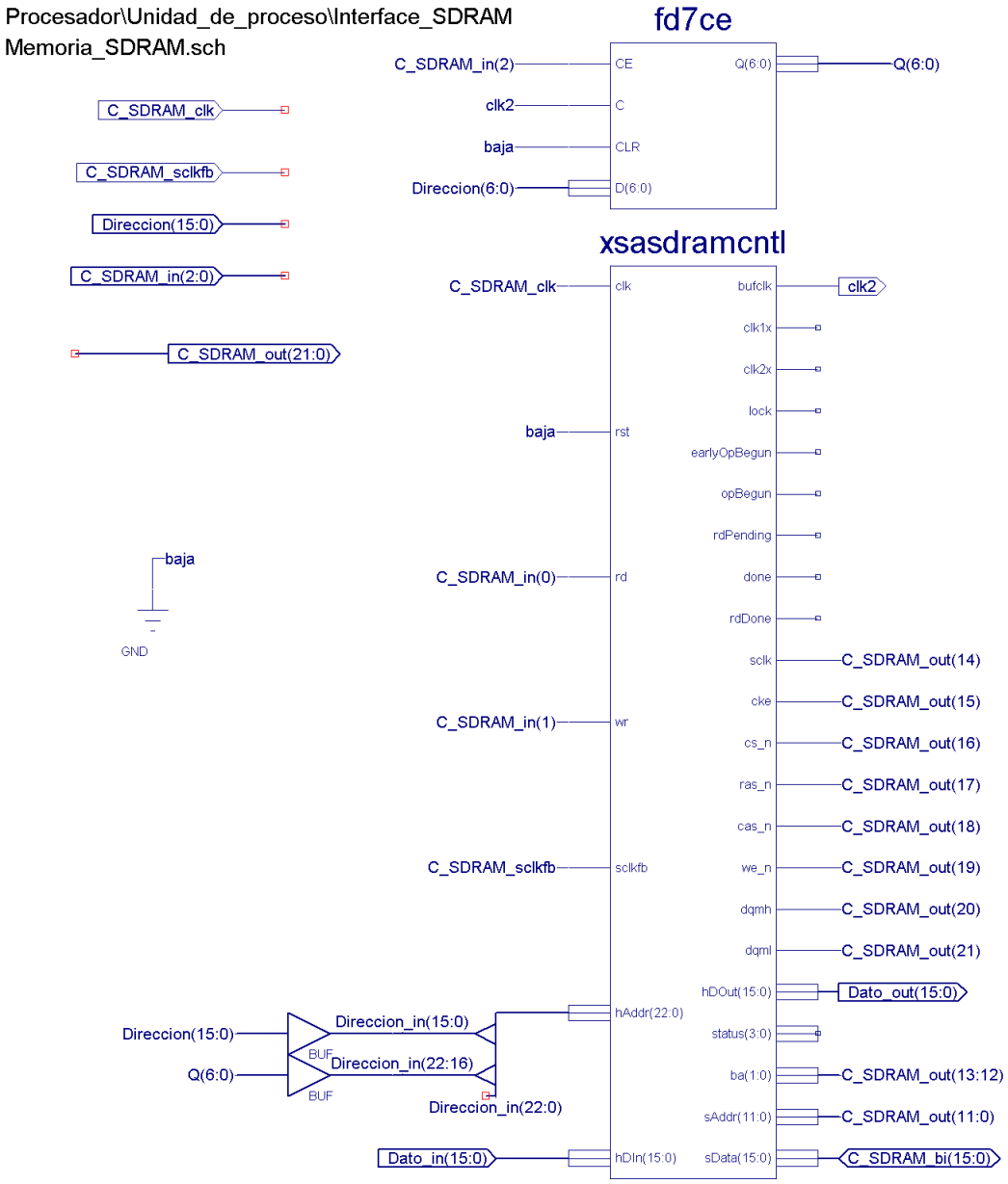
Salida 2. **C_SDRAM_out(21:0).** Bus de salida que indica a la *FPGA* los pines a los que están conectados las *Señales de Control* hacia la *Memoria SDRAM.*

Salida 3. **C_SDRAM_bi(15:0).** Bus bi-direccional que indica a la *FPGA* los pines a los que está conectado el bus de datos de la *Memoria SDRAM.*

Salida 4. **clk2**. Línea de salida de la señal de realimentación procedente de la *Memoria SDRAM.*



(a) Entradas y Salidas



(b) Esquema interno

Figura 5.14: *Interface con la Memoria de Datos.*

Como vemos en la figura 5.14(b), la *Interface con la Memoria de Datos* dispone de un bloque *xsasdramcntl* que es un es un *Controlador* que gestiona el acceso a la *Memoria SDRAM*. Además, este bloque contiene un *Registro Tipo D* de 7 bits, cuya función es mantener la parte alta de la dirección de acceso a la *Memoria SDRAM*.

La *Memoria SDRAM* dispone de una capacidad de 4 Mega-posiciones de 16 bits. Para direccionar esas 4 Mega-posiciones, son necesarios 22 bits[1]. Sin embargo, el bus de entrada de direcciones tiene solamente 16 bits. Para conseguir esos 22 bits, son necesarios 2 ciclos de acceso y un *Registro* que almacene los bits suficientes para obtener la dirección de 22 bits. Este *Registro* es el *Registro tipo D de 7 bits (FD7CE)* que mostramos en la figura.

En los siguientes sub-apartados, analizamos estos dos bloques.

5.6.1. *Registro Tipo D.*

Este *Registro* almacena 7 bits para generar la dirección de 22 bits que entra en el *Controlador de Memoria*.

Las entradas del *Registro Tipo D* son:

Entrada 1. **CE.** Esta entrada se utiliza para realizar la carga de datos en el *Registro tipo D* en el primero de los ciclos de acceso para la obtención de la dirección.

Entrada 2. **C.** Entrada para la señal de reloj de realimentación que controla el funcionamiento del *Registro*.

Entrada 3. **CLR.** Entrada para la realización de un *Clear asíncrono* de los datos contenido en el *Registro*. Nosotros siempre lo tendremos desactivado *(0 lógico)*.

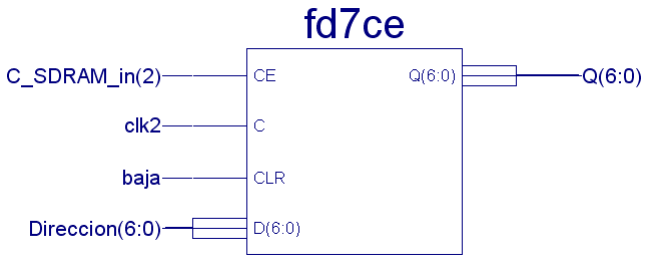
Entrada 4. **D(6:0).** Entrada para los datos que almacenamos en el *Registro* y que proveerán de la parte superior de la dirección al *Controlador*.

La salida del *Registro Tipo D* es:

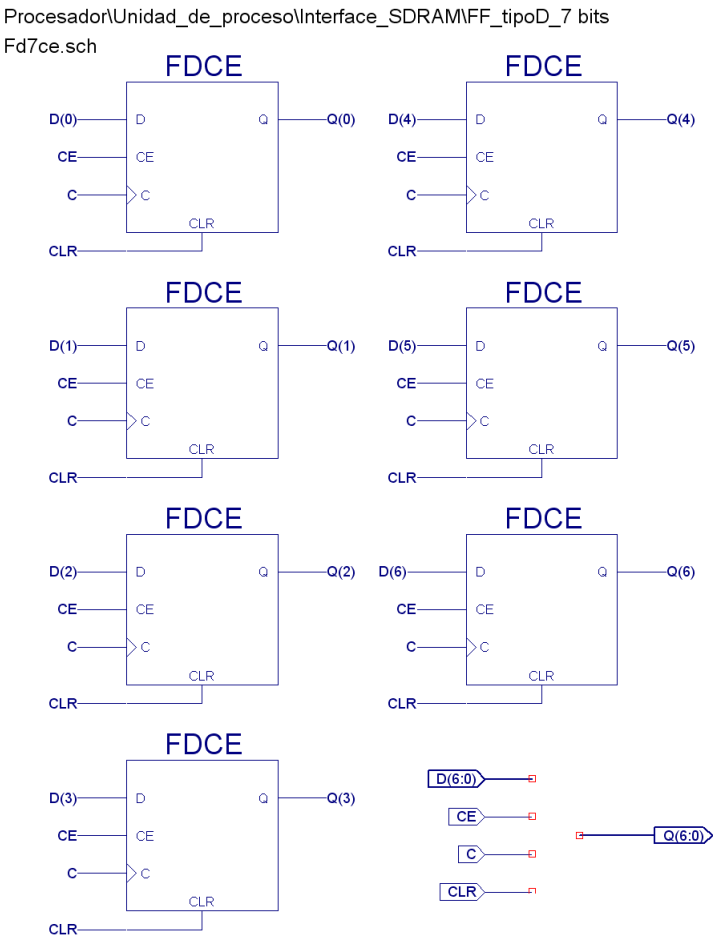
Salida 1. **Q(6:0).** Salida de datos hacia los bits más altos de la entrada de direcciones del *Controlador*.

El *Controlador de la Memoria SDRAM* tiene un bus de entrada de direcciones de 23 bits, aunque solo interprete 22 para la *Memoria SDRAM* de nuestra tarjeta *XSA 50*. Pero, a pesar de que solo interprete estos 22 bits, es necesario introducir los 23 bits en él para completar las instrucciones.

[1]El *Controlador de Memoria* de la figura tiene 23 bits, porque está diseñado para la tarjeta *XSA 100*, cuya capacidad es de 8 Mega-posiciones.



(a) Entradas y Salidas



(b) Esquema interno

Figura 5.15: Registro Tipo D.

La función del *Registro Tipo D* es almacenar los 7 bits más significativos de una dirección de acceso a la *Memoria SDRAM*. En un primer ciclo de acceso, se almacenan esos 7 bits en el *Registro* y, en un segundo ciclo de acceso, se toman esos 7 bits más los 16 procedentes del bus de direcciones para generar los 23 bits del bus de direcciones del *Controlador*. Todo el proceso está automatizado por las *Señales de control*. El programador solo debe utilizar la instrucción de escritura o lectura de memoria, y el procesador secuencia las operaciones.

La estructura de este *Registro* es un diseño propio que hemos basado en el *FD8CE* disponible en las librerías del programa de diseño. Como vemos en la figura 5.15(b), su núcleo son 7 *Registros de 1 bit* conectados en paralelo. Para gobernar la secuencia de operaciones, se utiliza la *Señal de Control CE*. En el primer ciclo de acceso, activamos esta señal para cargar en el *Registro* los 7 bits mencionados. En los siguientes ciclos de reloj, desactivamos esta *Señal de Control*, de forma que siempre estén disponibles a su salida los 7 bits que forman la parte alta del la dirección que proporcionamos al *Controlador*.

5.6.2. *Controlador de Acceso a la Memoria SDRAM.*

El *Controlador de Acceso a la Memoria SDRAM* es el único bloque que no hemos diseñado con la técnica de *Captura esquemática*. Está diseñado mediante lenguaje *VHDL* y está disponible en la página de ejemplos de la web de la empresa *Xess Corp.*, en la sección *Diseños de Xess Corp. (Design Examples from Xess)*. Nosotros hemos utilizado la versión 1.2 de dicho *Controlador*.

(a) Acceso a la página

(b) Selección de descarga

Figura 5.16: Obtención del *Controlador de Acceso a la Memoria SDRAM.*

Desde este enlace, podemos descargar el manual de usuario del *Controlador* en formato *.pdf* y un archivo comprimido con los archivos que implementan el *Controlador* y un diseño, a modo de ejemplo, de su uso. De este archivo comprimido, los archivos necesarios para implementar el controlador son únicamente: *common.vhd, sdramcntl.vhd* y *xsasdramcntl.vhd*.

common.vhd.

En este archivo, están definidas todas las funciones y definiciones que se utilizan en los otros dos archivos. Su código, en lenguaje *VHDL*, se muestra en el *DVD* adjunto.

sdramcntl.vhd.

Este archivo implementa el núcleo del *Controlador* y define su comportamiento para cada caso. Su código, en lenguaje *VHDL*, se muestra en el *DVD* adjunto.

Recordamos que este *Controlador* está diseñado para las tarjetas *XSA 50* y *XSA 100*. De hecho, la versión que descargamos de la web es la versión para la *XSA 100*. Como vemos en la primera página del manual, la versión para la *XSA 100* implementa de 512 columnas mientras que la de la *XSA 50* solo implementa 256. Por tanto, es necesario cambiar este parámetro para que el controlador funcione bien. Para ello, la forma más sencilla es el uso de un editor de texto en el reemplacemos todas[2] las cadenas de caracteres "512"presentes en el documento por la cadena de caracteres "256".

xsasdramcntl.vhd.

Por último, el archivo *xsasdramcntl.vhd* provee de una apariencia externa adecuada a la tarjeta al núcleo del *Controlador*. Su código, en lenguaje *VHDL*, se muestra en el *DVD* adjunto.

Debemos añadir estos tres archivos al proyecto del procesador de la forma que explicamos en el Apéndice B. Después, es necesario crear el símbolo que vemos en la parte inferior de la figura 5.14(b). Este *Controlador* convierte el acceso a una *Memoria SDRAM* en un acceso a una *Memoria RAM* tradicional. Para la correcta secuenciación de operaciones de lectura y escritura, en la figura 5.17, mostramos los cronogramas que deben cumplirse para realizar estas operaciones de lectura y escritura en la *Memoria SDRAM*.

5.7. *Multiplexores Tri-estado.*

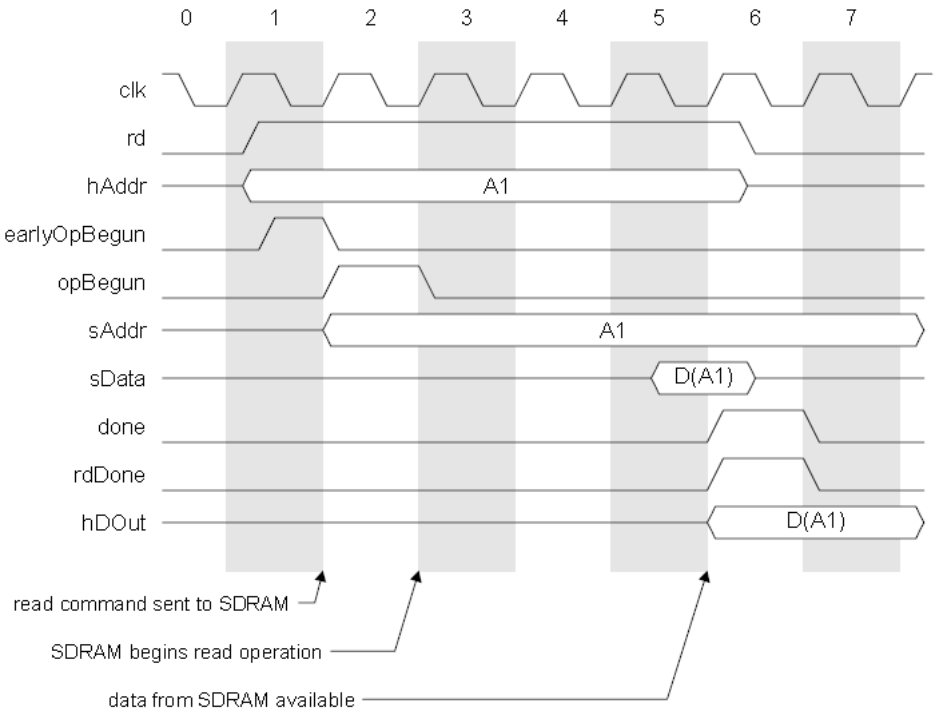
En el bloque principal de la *Unidad de Proceso* (figura 5.1), utilizamos tres *Multiplexores Tri-estado* como los vistos en el apartado 4.6.3.

Dos de ellos son del tipo *Mux16_3a1t*. A estos *Multiplexores*, los llamamos *Multiplexores de Canal*, ya que controlan los datos que circulan por los dos canales. Tienen 3 entradas de datos de 16 bits y una salida de 16 bits, lo que obliga a utilizar 2 *Señales de Control* para gobernarlos.

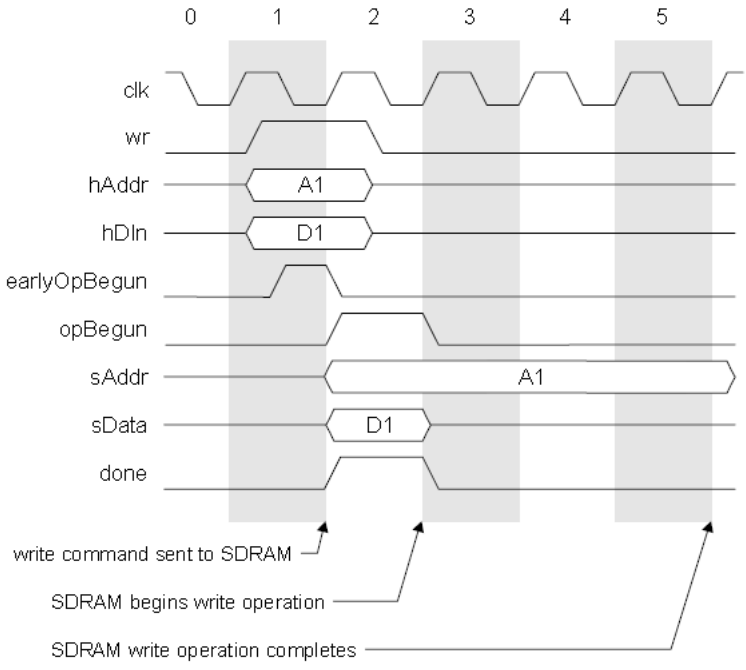
El tercer *Multiplexor* es del tipo *Mux16_4a1t*. Este *Multiplexor* es el *Multiplexores de Salida*, ya que controla el dato que sale de los bloques de *Unidad de Función* y *Memoria de Dato*, *Dato Literal* o Dato procedente del *Puerto de Entrada*. Además de salir fuera del bloque de la *Unidad de Proceso*, este dato se realimenta hacia el *Banco de Registros* para guardar su contenido en uno de estos *Registros*. Para su gobierno, también necesita 2 *Señales de Control*.

Por último, hacemos referencia a la arquitectura interna de estos bloques. Esta arquitectura es análoga a la mostrada para el *Multiplexor Tri-estado de 16 bits de 16 a 1* mostrado en la figura 4.10(b).

[2]Sugerimos comprobar cadena a cadena y no reemplazar todas a la vez.



(a) Cronograma de Lectura



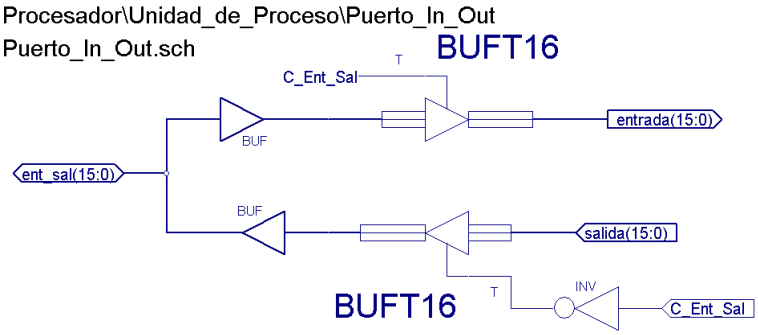
(b) Cronograma de Escritura

Figura 5.17: Cronogramas de gobierno del *Controlador de la Memoria SDRAM.*

5.8. *Puerto de entrada/salida.*

Este bloque implementa el interface de comunicación del procesador con el exterior, esto es, los *Puertos de entrada y de salida.*

(a) Entradas y Salidas



(b) Esquema interno

Figura 5.18: Puerto de entrada/salida.

Las entradas del *Puerto de entrada/salida* son:

Entrada 1. **Salida(15:0).** Entrada para el bus de salida de datos procedente de la salida *Canal 1* del *Banco de Registros.*

Entrada 2. **C_ent_sal.** Señal que controla si el puerto es de entrada o de salida.

Las salidas del *Puerto de entrada/salida* son:

Salida 1. **Entrada(15:0).** Salida de datos que actúa como *Puerto de Entrada* de la *Unidad de Proceso.*

Salida 2. **Ent_Sal(15:0).** Bus bi-direccional que indica a la *FPGA* los pines a los que está conectado el *Puerto de entrada/salida* de la *Unidad de Proceso.*

Como mostramos en la figura 5.18(b), utilizamos dos *Buffers Tri-estado* controlados por una señal y su complementaria para que que no puedan estar los dos activos simultáneamente. En funcionamiento normal, está activado el *Puerto de entrada* que está conectado al *Multiplexor Tri-estado de salida.* Así, su valor no sale de dicho *Multiplexor* y no afecta al resto de los bloques. Para sacar datos, basta con cambiar el valor de la *Señal de Control C_ent_sal.* Los datos que salen del *Puerto de Salida* hacia el exterior del *Procesador* provienen, directamente, de la salida del *Canal 1* del *Banco de Registros.*

Capítulo 6

Implementación y Aplicación práctica.

En los capítulos 3, 4 y 5 hemos ido presentando el *Procesador* que diseñamos. Como hemos visto es un *Procesador* que dispone de un *Set de Instrucciones* muy amplio, un *Banco de 16 Registros de 16 bits ortogonales con post-modificación de datos* en cada acceso, una *Unidad de Función* que realiza todas las operaciones aritméticas, lógicas y de movimiento de bits básicas con los datos, una *Memoria de Datos* externa de elevada capacidad, un *Puerto de entrada/salida* que actúa como *interface* con el exterior, una *Memoria de Instrucciones* que permite el almacenamiento y ejecución de hasta 8 programas diferentes, gracias a los *switches* de que dispone la tarjeta, un *Registro Contador de Programa* que permite la realización de saltos y llamadas y retornos de sub-rutinas. Con todas estas características, consideramos que el *Procesador RISC* que hemos diseñado es muy completo y lo suficientemente potente como para ejecutar programas de cierta complejidad.

Sin embargo, al embeber este diseño en la *FPGA* nos hemos visto muy limitados en el espacio, ya que la *FPGA Xilinx Spartan II* de que disponemos tiene poca capacidad para soportar un diseño del tamaño y complejidad del nuestro. Por esta razón, y para cumplir el segundo objetivo de este proyecto, la implementación del diseño en la *FPGA Xilinx Spartan II*, nos hemos visto obligados a reducir las capacidades del *Procesador*, disminuyendo, de esta forma, su tamaño. Hemos mantenido la estructura y la filosofía de funcionamiento del *Procesador*, pero hemos limitado sus recursos.

Una vez modificado el diseño, generamos un nuevo *Proyecto* con él para compilarlo y generar el *bitstream* que nos permita embeberlo en la *FPGA*. Durante este proceso, *Project Navigator* genera varios informes con el contenido, parámetros, avisos y errores de diseño durante todas las fases de la compilación.

Una vez descargado el diseño sobre la *FPGA*, para demostrar su potencial hemos diseñado una aplicación práctica, esto es, un programa que utiliza algunos recursos del *Procesador*.

En este capítulo, presentamos las reducciones a las que hemos sometido al *Procesador*, presentamos los informes que genera *Project Navigator* al realizar la compilación y generación del *bitstream* y presentamos una aplicación para demostrar el funcionamiento del procesador.

6.1. Implementación en la *XSA 50.*

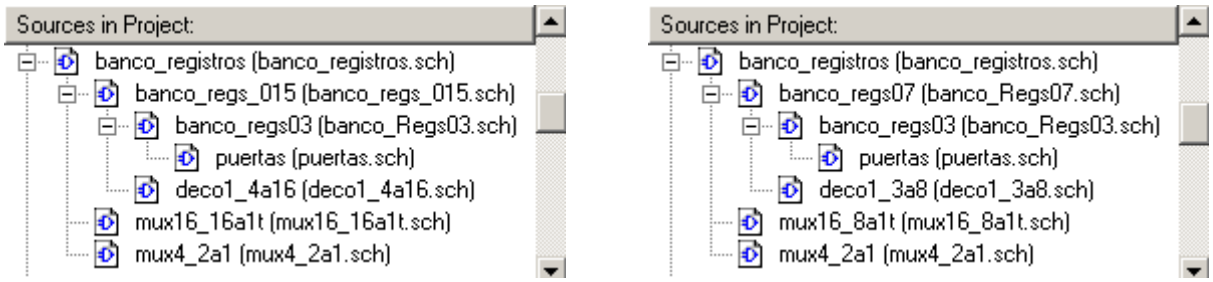
El diseño que hemos ido viendo en los capítulos anteriores puede implementarse sobre la *FPGA Xilinx Spartan II XC2S100* o superiores que es de mayor tamaño que la *FPGA Xilinx Spartan II XC2S50* de la que disponemos. Entonces, para reducir el tamaño del diseño de forma que pudiéramos implementar el diseño sobre nuestra *FPGA* hemos avanzado por tres caminos. Es los siguientes sub-apartados, presentamos estas simplificacioens del diseño.

6.1.1. *Banco de Registros.*

El *Banco de Registros* original disponía de *16 Registros de 16 bits Ortogonales con capacidad de post-modificación de su contenido.* En nuestro diseño, hemos reducido estos 16 *Registros* a un *Banco* de, únicamente, *8 Registros.* Sin embargo, hemos mantenido todas las características del *Banco* original

- Todos los *Registros* son exactamente iguales, y se basan en *Registros Contadores de 16 bits.*
- El acceso para escritura de su contenido es igual en todos, y el acceso para su lectura también.
- La posibilidad de post-modificación del contenido de los datos se mantiene.
- Sigue habiendo un *Canal de entrada de datos* y dos *Canales de salida de datos.*

Estas características tienen que ver con el funcionamiento del *Procesador.* En cuanto su estructura, también hay algunas diferencias. En la figura 6.1, mostramos la estructura de los *Bancos de Registros* original (*16 Registros*) y modificado (*8 Registros*).



(a) *16 Registros* (b) *8 Registros*

Figura 6.1: Esquema del *Banco de Registros.*

En esta figura, vemos las diferencias que existen entre ambos diseños.

En el diseño original, el bloque *Banco_ registros* dispone de 3 bloques: *Banco_ regs_ 015*, *Mux16_ 16a1t* y *Mux4_ 2a1*. Como vimos en el apartado 5.3, el bloque *Banco_ regs_ 015* tiene 16 buses de salida de 16 bits, que proceden, cada una, de uno de los *Registros*. Cada salida, entra en los dos *Multiplexores Tri-estado* que seleccionan el dato que sale por cada canal. Los *Mux4_ 2a1* direccionan los *Registros* sobre los que se harán las operaciones.

Podemos analizar el diseño original de forma paralela. El bloque *Banco_ regs_ 07* tiene 8 buses de salida procedentes de los *Registros*. Estos 8 buses entran en los *Multiplexores Tri-estado* de salida de canal, que, en este caso, son de 8 a 1 (*Mux16_ 8a1t*). Los *Mux4_ 2a1* tienen la misma función que en el caso original. En la figura 6.2, mostramos la estructura interna de este bloque.

La estructura de los dos tipos de *Multiplexores Tri-estado* es igual, y la mostramos en la figura 4.10(b) del apartado 4.6.3. El bloque *Banco_ regs_ 07* del diseño simplificado se corresponde con el bloque *Banco_ regs_ 015* del diseño original.

El bloque *Banco_ regs_ 015* tiene 4 bloques *Banco_ regs_ 03* que implementan los 16 *Registros*. Para direccionar estos 16 *Registros* dispone de varios decodificadores *deco1_ 4a16*. Y, por último, *16 puertas lógicas AND* gobiernan la escritura de los *16 Registros*. Paralelamente, el bloque *Banco_ regs_ 07* tiene solamente 2 bloques *Banco_ regs_ 03* que implementan los 8 *Registros*, y, para direccionar estos 8 *Registros*, dispone de varios decodificadores *deco1_ 3a8*. En este caso, solo son necesarias *16 puertas lógicas AND*. Vemos la estructura interna de este bloque en la figura 6.3.

Los bloques *Banco_ regs_ 03* de *Banco_ regs_ 015* y de *Banco_ regs_ 07* son exactamente iguales. En su interior, contienen los *Registros* en bloques de 4. De esta forma, en el primer caso, con 4 bloques de *4 Registros* implementamos los *16 Registros* del diseño original y, en el segundo caso, con solo 2 bloques implementamos *8 Registros* exactamente iguales a los del diseño anterior.

Con esta reducción, disminuimos el espacio que ocupa el *Banco de Registros* en valores cercanos al *40 %* del diseño original. En concreto, la versión original ocupa *359 slices* de la *FPGA*, mientras que la versión reducida ocupa únicamente *222 slices*. Por otro lado, la versión completa utiliza *512 Tbufs* mientras que la versión más pequeña usa la mitad *256 Tbufs*.

	Versión Completa	**Versión Reducida**
Slices	359	222
Tbufs	512	256

Cuadro 6.1: Comparativa Versión Completa - Versión Reducida.

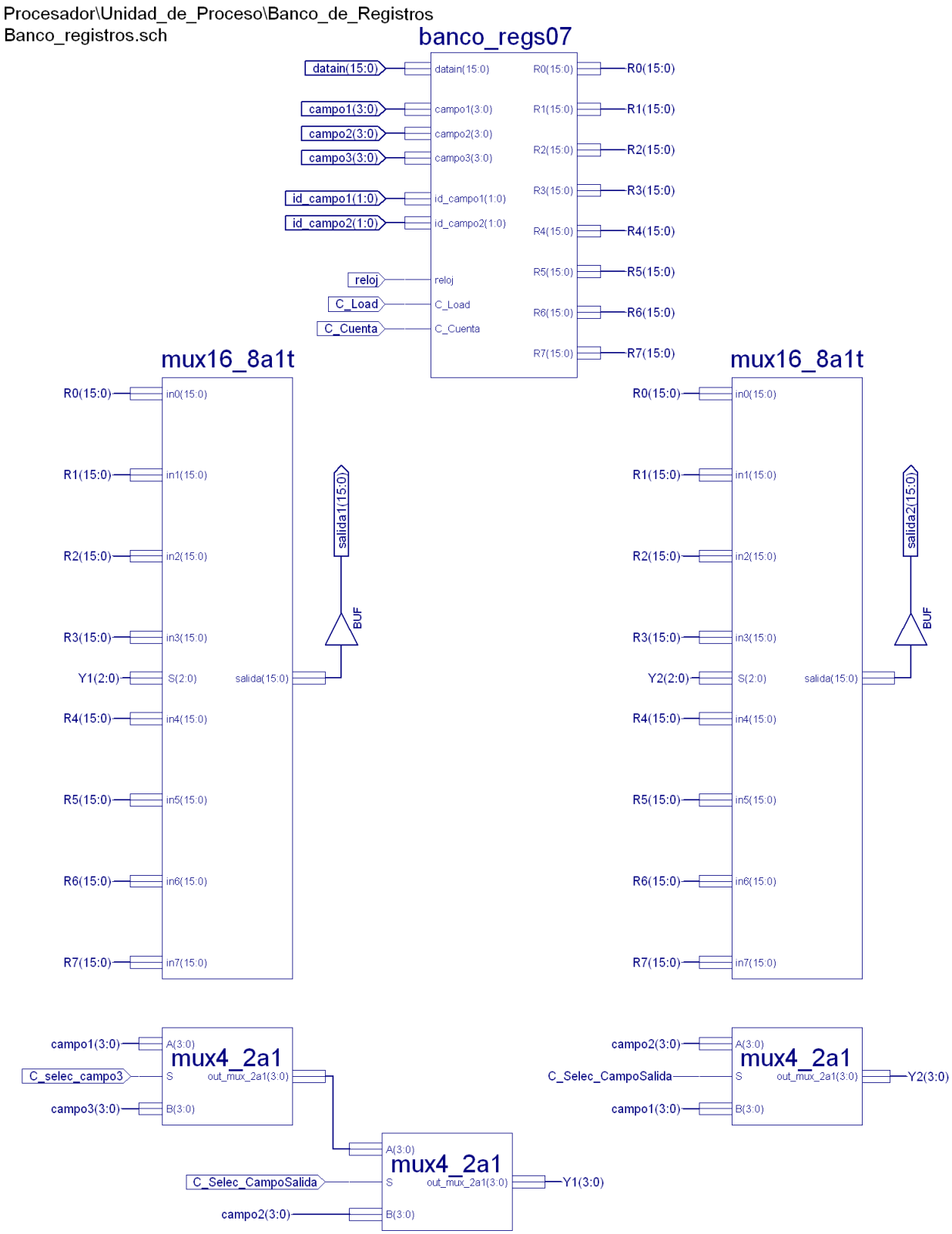


Figura 6.2: Esquema interno de *Banco_ regs*.

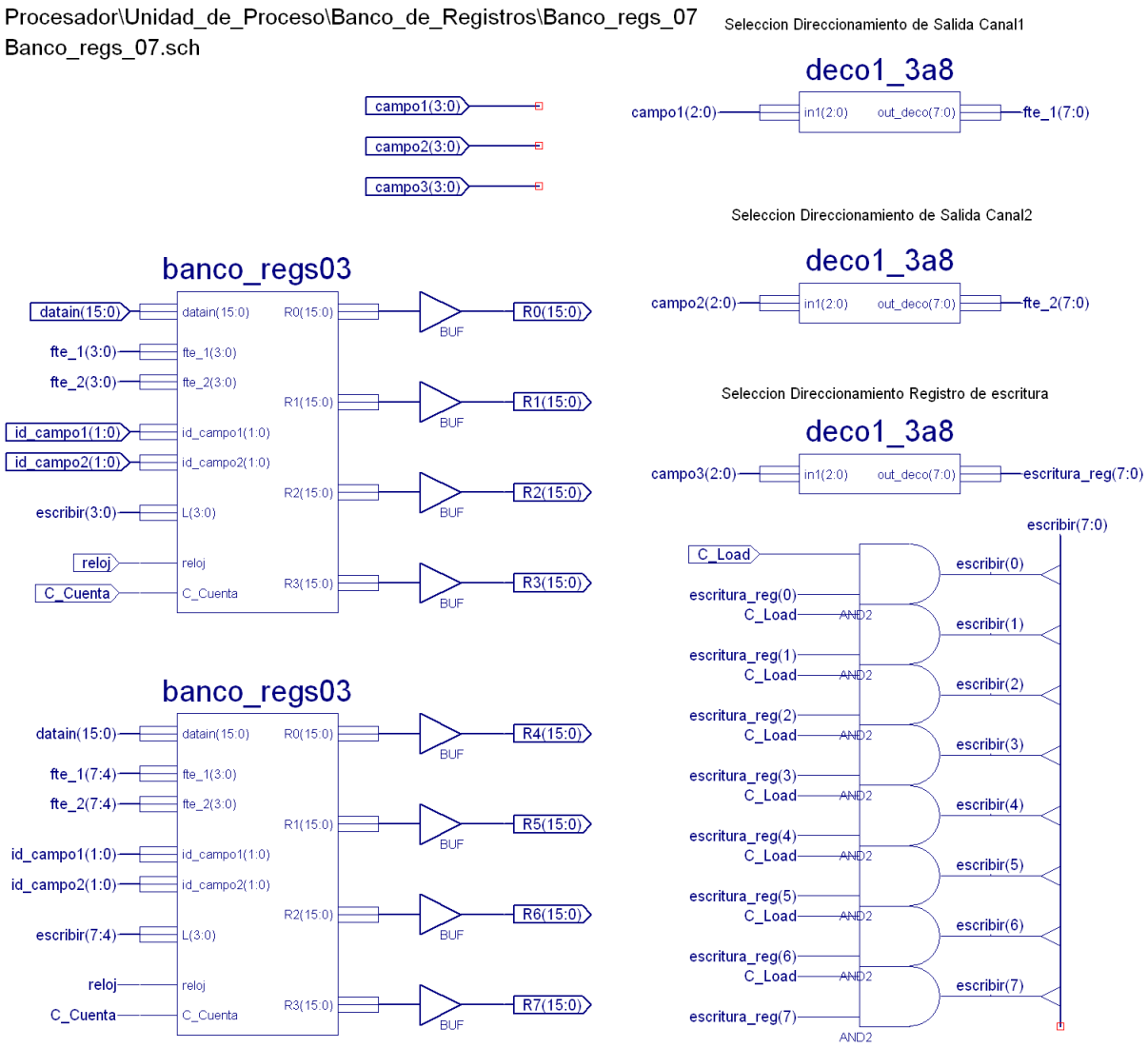
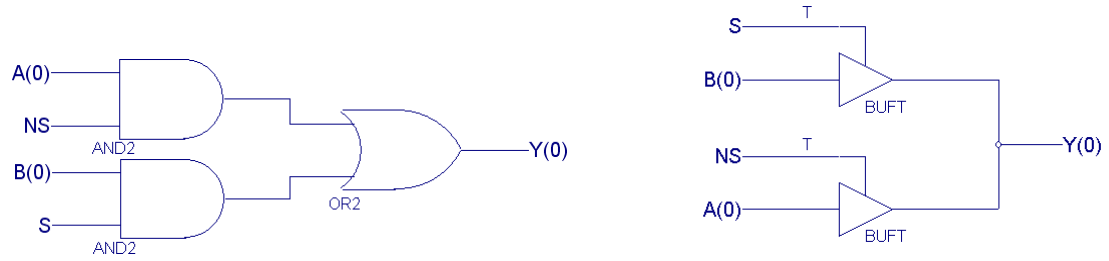


Figura 6.3: Esquema interno de *Banco_regs_07*.

6.1.2. *Multiplexores.*

Como hemos mencionado al comienzo de este capítulo, utilizamos la tecnología de *Buffers Tri-estado* para implementar *Multiplexores*, ya que ocupan mucho menos tamaño en la *FPGA* que los *Multiplexores* creados con la tecnología tradicional de *Puertas Lógicas*.

Por tanto, a la hora de utilizar *Multiplexores* tenemos que llegar a un compromiso entre usar los tradicionales *Multiplexores de Puertas Lógicas* o usar los menos ortodoxos *Multiplexores Tri-estado*. En la figura 6.4, mostramos un ejemplo de *Multiplexor de 1 bit de 2 entradas* diseñado con ambas técnicas.



(a) *Tecnología de Puertas* (b) *Tecnología de Buffers Tri-estado*

Figura 6.4: *Multiplexor de 1 bit de 2 entradas.*

En la *FPGA Xilinx Spartan II*, disponemos de *832 Tbufs*, aunque, por nuestra experiencia, *Project Navigator* no realiza el proceso de mapeado y, por tanto, no genera el *bitstream*, si se utilizan más de 775 *Tbufs* en el diseño.

Para mostrar las ventajas del uso de la tecnología de *Buffers Tri-estado*, mostramos un cuadro comparativo de los recursos de la *FPGA* que se utilizan al implementar un *Multiplexor de 16 bits de 16 entradas y 1 salida* con *Puertas lógicas* y con *Buffers Tri-estado*.

	Multiplexor de Puertas	***Multiplexor Tri-estado***
Slices	130	8
Tbufs	0	256

Cuadro 6.2: Comparativa *Multiplexor de Puertas - Multiplexor Tri-estado.*

La reducción del tamaño ocupado de la *FPGA* es espectacular, aunque conseguimos esta reducción a base de utilizar *Tbufs* en la misma.

Como resumen, dadas las deficiencias de espacio en la *FPGA Xilinx Spartan II* con que nos encontramos al embeber nuestro *Procesador*, hemos utilizado la tecnología de *Buffers Tri-estado* para la implementación de algunos *Multiplexores*.

6.1.3. *Puerto de Entrada/Salida.*

Como vimos en el apartado 5.8, la estructura interna del *Puerto de Entrada/Salida* está formada por *Buffers Tri-estado*. Y en el apartado anterior, hemos visto que tenemos limitado el número de *TBufs*. De hecho, tenemos tan limitados los *TBufs* que no podemos añadir más. Por esta razón, no podemos implementar el *Puerto de Entrada* del *Procesador*. Sin embargo, podemos emular su comportamiento mediante *software*.

El objetivo del *Puerto de Entrada* es introducir un dato en la *Unidad de Proceso* durante la ejecución de un programa. Y podemos llevar a cabo este objetivo introduciendo el dato como un *Dato Literal* desde la *Palabra de Instrucción*. De esta forma, emulamos el comportamiento del *Puerto de Entrada* sin necesidad de utilizar recursos *hardware* de la *FPGA*.

En cualquier caso, el *Puerto de Entrada* del *Procesador* sí está disponible en la versión original del mismo.

6.2. Informes generados durante la compilación.

Durante el proceso de compilación y creación del *bitstream*, *Project Navigator* genera varios informes, como mostramos en la figura 6.5.

El icono ✔ junto a los iconos de *Informe* 📄 significa la correcta generación de todos los informes. Y el icono ❗ junto a cada apartado del *Menú* significa que el informe contiene algún *Aviso* o *Warning*, pero la compilación se ha realizado. Estos *Avisos* no influyen en el funcionamiento del *Procesador*. Aparecen, por ejemplo, al no conectar alguna salida de algún bloque como puede ser las salidas *OF* de los bloques *Sumador* o la salida *TC* de los bloques *Contador*.

6.3. Aplicación práctica.

En este apartado, presentamos la aplicación práctica que hemos diseñado para demostrar el funcionamiento del *Procesador*. Se trata de un programa que, a través de un mínimo *hardware* adicional, genera sonidos. Concretamente, nosotros hemos elegido la *Novena Sinfonía de Beethoven* para probar este *Generador de Música*.

En los siguiente sub-apartados, presentamos el programa, explicamos el *hardware* adicional que debe añadirse para oír la canción y presentamos el *Diagrama de Flujo* y el *Código Fuente* del programa.

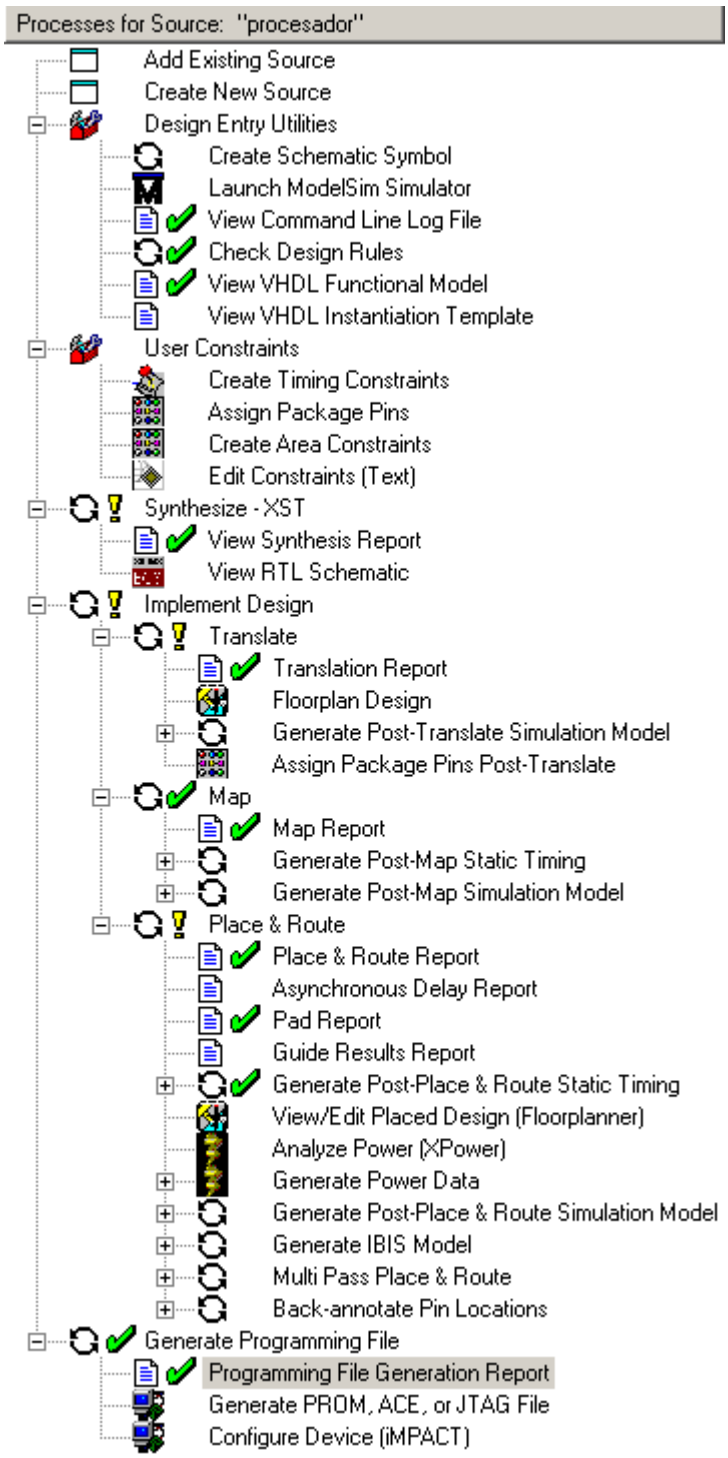


Figura 6.5: Informes de la Compilación.

6.3.1. Presentación del programa.

Este programa se basa en el efecto físico de la generación de sonido y, en concreto, la generación de los sonidos de la escala musical. La música tiene dos componentes principales: sonido y ritmo. Y cualquier *Generador de Música*, como es nuestro programa, debe utilizar estas dos componentes para generar canciones.

Generación de Sonido.

Físicamente, los sonidos se producen cuando un material vibra a una frecuencia determinada. Entonces, para generar estos sonidos, nosotros debemos emular esa vibración a esas frecuencias. Cuando dispongamos de una señal que emule esas vibraciones, ya solo es necesario aplicar dicha señal a un altavoz para oír los sonidos. Por tanto, el objetivo del programa es generar señales de una frecuencia determinada y con una duración determinada.

Los sonidos puros de la escala se basan en señales senoidales de mayor o menos frecuencia. A mayor frecuencia, el sonido generado es más agudo y, a menor frecuencia, obviamente, el sonido es más grave. En la siguiente tabla, presentamos la frecuencia, en *Hertzios (Hertz)*, de los sonidos que implementamos en este programa.

Conocidas estas frecuencias, el siguiente paso es generar señales de este tipo. El *Procesador* genera valores discretos, en ningún caso, valores continuos. Sin embargo, según la velocidad de generación de estos valores, dichos valores pueden interpretarse como una señal continua. La velocidad de la señal del oscilador de la tarjeta, que rige el funcionamiento del *Procesador*, es de *100 MHz*. Aunque la velocidad real del *Procesador* sea 8 veces menor, debido a que una instrucción son 8 ciclos de reloj, la velocidad es lo suficientemente alta como para considerar esos valores discretos como una puntos de una señal continua. Por tanto, basta con generar dichos valores.

Para ello, hemos hecho un muestreo de 1024 posiciones de un periodo de una señal senoidal pura. Además, lo hemos hecho con una resolución de 16 bits para dar la mayor calidad posible al sonido. Posteriormente, y tras normalizar estos valores, los hemos introducido en 1024 posiciones consecutivas de la *Memoria de Datos*, en concreto, en las posiciones *0:1023*.

A continuación, debemos tomar valores para emular una señal de una frecuencia determinada. Para ello, vamos a partir de un ejemplo. Suponemos una señal de frecuencia base X. Supongamos que, al realizar un muestreo sobre esa señal cada Y unidades de tiempo, obtenemos ciertos valores discretos consecutivos. Ahora bien, supongamos que doblamos la frecuencia inicial $(2 \cdot X)$. Al realizar el muestreo cada Y unidades de tiempo, obtenemos los mismos valores que antes pero alternados, es decir, uno sí, uno no. De nuevo aumentamos la frecuencia inicial, y la consideramos 3 veces la original $(3 \cdot X)$. Al muestrear cada Y unidades de tiempo, obtenemos los mismos valores, pero con una alternancia de uno sí, dos no, esto es, uno cada de 3.

Nota	**Frecuencia**
Fa # 3	185 Hertzios.
Sol 3	196 Hertzios.
Sol # 3	208 Hertzios.
La 3	220 Hertzios.
La # 3	233 Hertzios.
Si 3	247 Hertzios.
Do 4	262 Hertzios.
Do # 4	277 Hertzios.
Re 4	294 Hertzios.
Re # 4	311 Hertzios.
Mi 4	330 Hertzios.
Fa 4	349 Hertzios.
Fa # 4	370 Hertzios.
Sol 4	392 Hertzios.
Sol # 4	415 Hertzios.
La 4	440 Hertzios.
La # 4	466 Hertzios.
Si 4	494 Hertzios.
Do 5	523 Hertzios.
Do # 5	554 Hertzios.
Re 5	587 Hertzios.
Re # 5	622 Hertzios.
Mi 5	659 Hertzios.
Fa 5	698 Hertzios.
Fa # 5	740 Hertzios.
Sol 5	784 Hertzios.
Sol # 5	831 Hertzios.
La 5	880 Hertzios.

Cuadro 6.3: Tabla de frecuencias de las notas de la escala musical.

De este ejemplo, podemos sacar una conclusión. Si tomamos una frecuencia pequeña como la frecuencia base de la señal y la muestreamos, al aumentar la frecuencia de la señal y volver a muestrear, basta con tomar los valores del muestreo anterior con cierta alternancia.

Nosotros implementamos este muestreo de la siguiente forma. En la *Memoria de Datos*, hemos introducido 1024 muestras de un periodo de una señal senoidal. Entonces, para emular la frecuencia de un sonido, elegimos muestras con una separación determinada entre ellas. Estas muestras, a través de un *Convertidor Digital-Analógico*, un amplificador y un filtro, se aplican a un altavoz y generan sonidos con cada frecuencia. Esta técnica se conoce como *Técnica DDS (Direct Digital Synthesis)*.

Generación de Ritmo.

En música, el ritmo está definido por las diferentes duraciones de los sonidos.

En nuestro *Generador de Música*, nosotros conseguimos el ritmo de la misma forma: dando diferentes duraciones a los sonidos. Y conseguimos implementar estas diferentes duraciones realizando un mayor o menos número de muestreos.

La idea es similar a la de Generación de Sonido. Tomamos una duración de sonido base que se corresponde con X muestreos. En nuestro caso, la unidad de ritmo base es la corchea. Recordamos que *1 redonda = 2 blancas = 4 negras = 8 corcheas*. Entonces, para generar una negra, basta con tomar el doble de muestras que una corchea, esto es, $2 \cdot X$ muestras. Para generar una blanca, son $4 \cdot X$ muestras y para generar una redonda $8 \cdot X$ muestras. Incluso, se pueden generar notas con *puntillo*, con esta misma filosofía.

6.3.2. *Hardware* adicional necesario.

Nuestro *Procesador* genera señales digitales, pero los sonidos se producen a partir de la aplicación de señales analógicas a un altavoz.

Entonces, una vez que saquemos esas señales digitales por el *Puerto de Salida del Procesador*, debemos hacerlas pasar por un *Convertidor Digital-Analógico* para obtener esa señal analógica. Tras este *Convertidor*, hacemos pasar la señal analógica por un *Amplificador* y un *Circuito Desplazador del nivel de tensión* para ajustar el valor medio de la tensión de la señal analógica a $0Voltios$. Posteriormente, para eliminar el ruido y suavizar la señal en escalera, la siguiente etapa es un filtro analógico. Por último, para generar el sonido, aplicamos esta señal analógica y filtrada a un altavoz.

El esquema de bloques de conexiones para implementar este *hardware* queda como mostramos en la siguiente figura:

Figura 6.6: Esquema de bloques de conexiones *hardware* para el *Generador de Música*.

Mostramos el esquema *Hardware* en la figura 6.7.

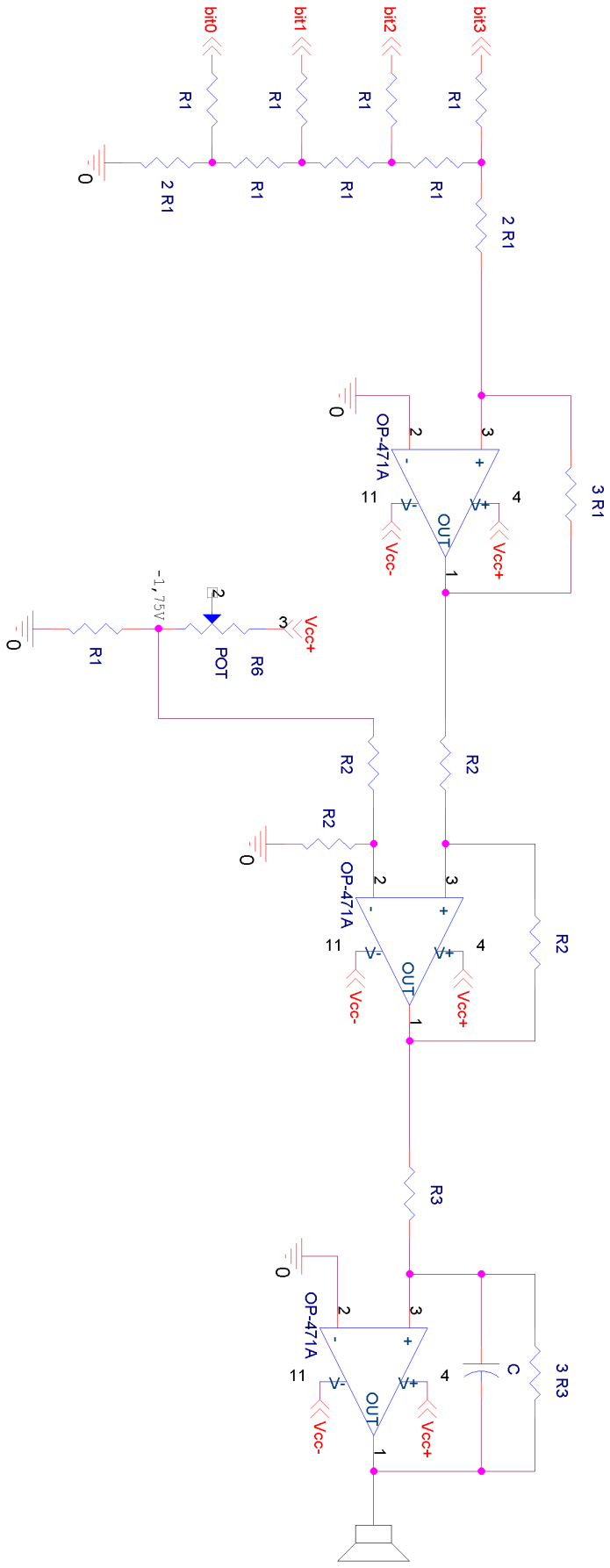


Figura 6.7: Esquema de conexiones *hardware* para el *Generador de Música*.

6.3.3. *Diagrama de Flujo.*

Para terminar todo lo concerniente a la aplicación práctica que hemos diseñado, en este apartado, presentamos tanto el *Diagrama de Flujo* del programa *Generador de Música*.

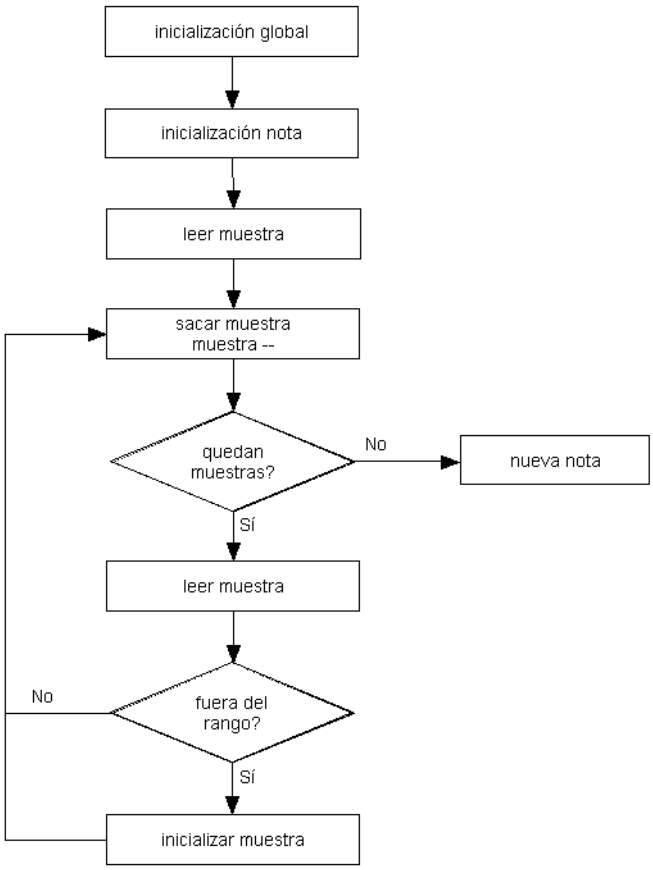


Figura 6.8: *Diagrama de Flujo* del programa *Generador de Música*.

Para entrar más en profundidad, presentamos el *Pseudo-código* del programa que hemos diseñado para *generar música*. El *Pseudo-código* para cada nota es el siguiente:

Nº	**Pnemónico**	**Instrucción**
0	*Rwrite r0,0000*	Inicializacion de *registro r0 (base de la dirección).*
1	*Rwrite r6,0000*	Inicializacion de *registro r6 (contador de muestras).*
2	*Rwrite r4,0000*	Inicializacion de *registro r4 (número de muestras).*
3	*Rwrite r2,0000*	Inicializacion de *registro r2 (registro auxiliar).*
4	*Rwrite r1,0000*	Inicializacion de *registro r1 (desplazamiento de la dirección).*
5	*Mread r3*	Lectura de memoria de un valor del seno y guardado en r3.
6	*Out r3*	Salida por el puerto.
7	*Add r1,1,r2*	Actualización del *registro auxiliar.*
8	*Add r2,0,r1*	Actualización de la posición que leemos de memoria.
9	*Add r6,1,r7*	Actualización del *registro auxiliar.*
10	*Add r7,0,r6*	Actualización del *registro contador.*
11	*Comp r4,r6*	Comprobación de última posición leída.
12	*Jca(si Z) SigNota*	Si ultima posición leída, siguiente nota.
13	*Nop*	No operación.
14	*Btest r2, bit10*	Comprobación de fin de tabla.
15	*Jca(si Z) 5*	Si no fin de tabla, salto a nueva lectura.
16	*Nop*	No operación.
17	*Jia 4*	Si fin de tabla, salto a inicializar tabla.
18	*Nop*	No operación.
19	*Nop*	No operación.

Cuadro 6.4: *Pseudo-código* del programa *Generador de Música.*

Para cada sonido, basta con cambiar dos parámetros:

1. **Desplazamiento:** que implementa la frecuencia del sonido, esto es, hacerlo más grave o más agudo.

2. **Número de muestras:** que implementa la duración de los sonidos.

Capítulo 7

Costes y Presupuesto.

En este capítulo, evaluamos el presupuesto del *Proyecto Diseño de un microprocesador RISC e implementación mediante la FPGA Xilinx Spartan II*. Para ello, en primer lugar, presentamos el coste de los materiales empleados durante todo el desarrollo del *Proyecto* y, posteriormente, estimamos el coste humano durante todo este tiempo.

7.1. Coste de Material.

Hemos dividido el material que empleamos durante el desarrollo de este *Proyecto* en dos grupos. Por un lado, el material estrictamente necesario para el diseño del *Procesador* y su implementación en la *FPGA*, y por otro lado el material adicional que hemos usado tanto para la generación del *Banco de Pruebas* como para la *Aplicación práctica*.

7.1.1. Material básico.

El material básico consta de los dos siguientes dos componentes.

El primero es el *Software* de diseño. Durante este *Proyecto*, hemos utilizado los paquetes de programas *ISE WebPACK 6.3* y *ModelSIM 5.8c*. Como hemos visto, ambos paquetes están disponibles, gratuitamente, en la página web de la empresa *Xilinx Corp.*. Por tanto, suponiendo que disponemos del material mínimo disponible en cualquier laboratorio de la *E.T.S.I.T.*, esto es, un ordenador con el sistema operativo *Windows XP* y una conexión a *internet*, entonces, el coste de este material es nulo.

El segundo componente indispensable es la tarjeta *XSA 50* que incluye la *FPGA Xilinx Spartan II*. Esta tarjeta es un artículo de la empresa *Xess Corp.*, y se obtiene realizando un pedido a través de su página web. El precio de cada unidad es \$59. Además de la propia tarjeta, el lote que incluye este pedido contiene un cable de comunicación entre la tarjeta y el puerto paralelo del ordenador, un cable con su transformador para la alimentación de la tarjeta y un *Software* que permite la descarga de los diseños, en forma de *bitstreams*, a la tarjeta.

En el cuadro 7.1, presentamos un resumen del coste de este material adicional.

7.1.2. Material adicional.

Durante el desarrollo del *Proyecto*, ha habido dos fases en las que hemos utilizado cierto material adicional.

La primera fase fue la fase del *Banco de Pruebas*. En esta fase, probamos todas las instrucciones que diseñamos y comprobamos que su funcionamiento era correcto. Para ello, conectamos los 16 bits del *Puerto de Salida* a a *4 displays de 7 segmentos* a través de *4 Buffers Tri-estado* de 4 bits[1] y *4 Convertidores BCD a 7 segmentos*. Mediante este *hardware* adicional, la labor de chequeo de las instrucciones ha sido mucho más sencilla, ya que, de un vistazo a los *displays*, comprobábamos si las operaciones se habían realizado de forma correcta.

Una vez completada la fase del *Banco de Pruebas*, la *Aplicación Práctica* que diseñamos, exigía bastante *Hardware* adicional, como vimos en el capítulo 6. En concreto, hemos necesitado *24 Resistencias de Película Carbón (hemos utilizado resistencias iguales)*, *1 Condensador Cerámico*, *3 Amplificadores Operacionales*, *1 Potenciómetro* y *1 Altavoz de 8 ohms*.

El cuadro 7.2 es un resumen del coste de este material básico.

7.2. Coste Humano.

El Coste Humano del *Proyecto* viene definido por el tiempo que los dos ingenieros hemos utilizado para completar este *Proyecto*. Consideramos que el salario de un Ingeniero Electrónico en una empresa que se dedica a este tipo de tecnología especializada está en torno a los 15 €/hora.

A continuación, presentamos un desglose con las horas que hemos empleado para cada etapa de este *Proyecto*. Supondremos 40 horas semanales de trabajo.

1. Selección y aprendizaje del *Software*. 10 semanas → 400 horas.
2. Diseño del *Procesador*. 15 semanas → 600 horas.
3. Implementación sobre la *FPGA*. 2 semanas → 80 horas.
4. Simulación y *Banco de Pruebas*. 10 semanas → 400 horas.
5. *Aplicación Práctica*. 10 semanas → 400 horas.

En el cuadro, presentamos un resumen del coste de este material adicional.

[1] Para controlar el acceso del *Puerto de Entrada*, también hemos incluido otros 4 *Buffers Tri-estado*.

Coste del Material básico	
Software de diseño	—
XSA 50	$ 59
Total	$ 59

Cuadro 7.1: Coste del material básico.

Coste del Material adicional	
Buffers Tri-estado 74LS244 (8 udds)	8· $ 0.35
Conversores BCD a 7 segmentos 74LS248 (4 udds)	4· $ 0.95
Displays de 7 segmentos (4 udds)	4· $ 0.65
Resistencias de Película de Carbón (25 udds)	25· $ 0.10
Condensador Cerámico (1 udds)	1· $ 0.05
Altavoz de 8 ohms (1 udds)	1· $ 1.35
Total	$ 13,1

Cuadro 7.2: Coste del material básico.

Coste humano	
Selección y aprendizaje del *Software* (400 horas)	$400 \cdot 15$ €
Diseño del *Procesador* (600 horas)	$600 \cdot 15$ €
Implementación sobre la *FPGA* (80 horas)	$80 \cdot 15$ €
Simulación y *Banco de Pruebas* (400 horas)	$400 \cdot 15$ €
Aplicación Práctica (400 horas)	$400 \cdot 15$ €
Total (2 ingenieros)	56.400 €

Cuadro 7.3: Coste humano.

7.3. Coste Total.

Por último, presentamos el *Presupuesto Total del Proyecto* a partir de los costes que hemos visto en los apartados anteriores.

La empresa *Xess Corp.* tiene su lista de productos en $ americanos. A día de hoy, 12 de abril de 2005, el cambio oficial es $ 1.00 = 0.776150 €. Por tanto, los $ 59 del precio de la tarjeta equivalen a 45,7974 €y los $ 13,1 invertidos para *Material adicional* se corresponden con 10,16756 €.

Coste Total	
Coste del Material básico	45,8 €
Coste del Material adicional	10,17 €
Coste humano	56.400 €
Coste Total (sin I.V.A.)	56.456 €

Cuadro 7.4: Presupuesto total (sin I.V.A.).

En estos costes, no hemos incluido el *Impuesto sobre el Valor Añadido (I.V.A.)*. Este impuesto añade el 16 % al coste original del *Proyecto*. Entonces, el *Presupuesto definitivo del Proyecto* queda como sigue.

Coste Total: 65.489 €

Cuadro 7.5: Presupuesto total.

Apéndice A

Obtención e instalación del software.

Como hemos visto en el apartado 1.2.2, el software utilizado para este proyecto consta de dos paquetes de libre distribución y un tercer paquete de programas obtenido al realizar la compra de la placa. Estos paquetes son los siguientes:

1. ***Xilinx ISE WebPACK***: en nuestro proyecto, hemos utilizado la versión 6.3 del *WebPACK* con el *Service Pack 2*. Sin embargo, ya está disponible la versión 3 en la página web de la empresa *Xilinx*.

2. ***ModelSim XE II***: la versión de este programa utilizado para la simulación de los diseños es la versión 5.8c

3. ***XSTools 4.0***: este paquete de utilidades para el interface entre la tarjeta y el *PC* es la versión 4.0.

A continuación, presentamos un tutorial de instalación de cada paquete de programas. Sugerimos realizar la instalación en el orden en que viene especificado en este tutorial[1].

A.1. Xilinx ISE WebPACK.

A.1.1. Obtención del paquete.

En este apartado, explicamos la forma de obtener los paquetes *Xilinx ISE WebPACK* y *ModelSIM XE*. Dichos paquetes pueden obtenerse, de forma totalmente gratuita, desde la web *www.xilinx.com*.

Figura A.1: Logotipo de la empresa *Xilinx*.

[1]Es estrictamente necesario realizar la instalación sobre uno de estos sistemas operativos: Windows XP o Windows 2000.

Para ello, mediante cualquier navegador accedemos a la portada de esta web, seleccionamos la pestaña *Support* y, a continuación, la opción *Software*.

Figura A.2: Acceso a la página de descarga.

En esta ventana, podemos acceder a las últimas actualizaciones de numerosos recursos software y herramientas *on-line*. Clickeando en la opción *WebPACK ISE / ModelSIM XE II*, accedemos a la ventana del *WebPACK ISE 6.3*.

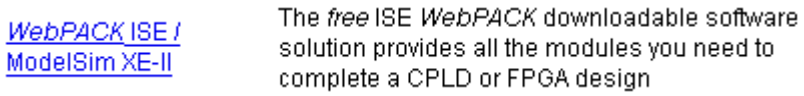


Figura A.3: Acceso a Recursos.

Para bajar los paquetes de programas, es necesario tener una cuenta gratuita en *www.xilinx.com*. Por tanto, si el usuario no tiene una cuenta, es necesario que se registre. Si ya está registrado, el usuario puede ignorar este paso.

El proceso de registro comienza pinchando en el botón *Order & Register*.

Figura A.4: Acceso al Registro.

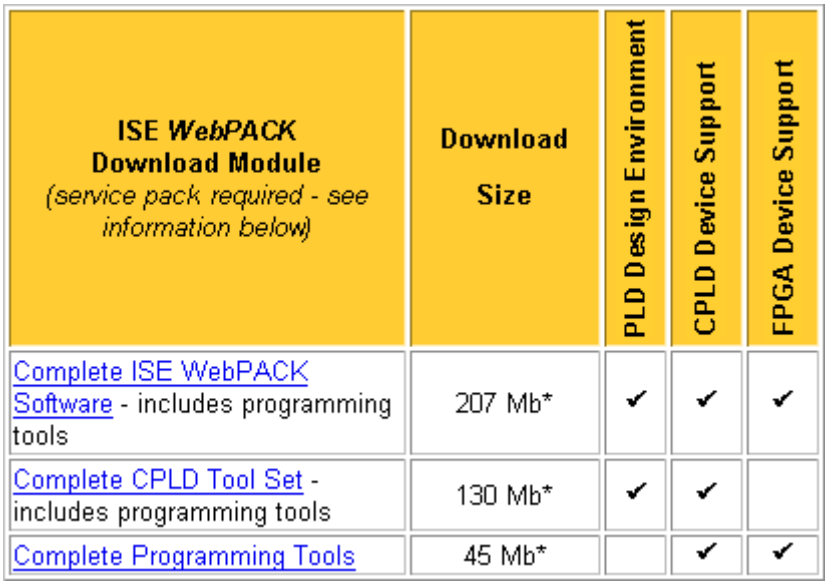
En la ventana de registro, seleccionamos el botón de creación de cuenta (*Create an Account*).

Figura A.5: Creación de la cuenta.

El siguiente paso es completar los formularios de las ventanas de *Create an Account* y *Address Information*. Al llenar este segundo formulario, *Xilinx* envía un correo de confirmación con el *Username* y el *Password* que el usuario eligió. Para terminar el proceso de registro, rellenamos el formulario *ISE WebPACK Survey*. La pantalla resumen informa sobre los datos del registro y permite acceder directamente a la pantalla de descarga del *ISE WebPACK 6.3*.

Figura A.6: Acceso a la Descarga.

Pinchando en este enlace, o accediendo directamente desde la ventana del *WebPACK ISE 6.3*, llegamos a la ventana de descarga. Desde esta ventana, podemos bajar tanto el *Complete ISE WebPACK Software* como el *Complete MXE Simulator*. En este punto, se descargan los archivos que permitirán la instalación de los dos paquetes de programas.



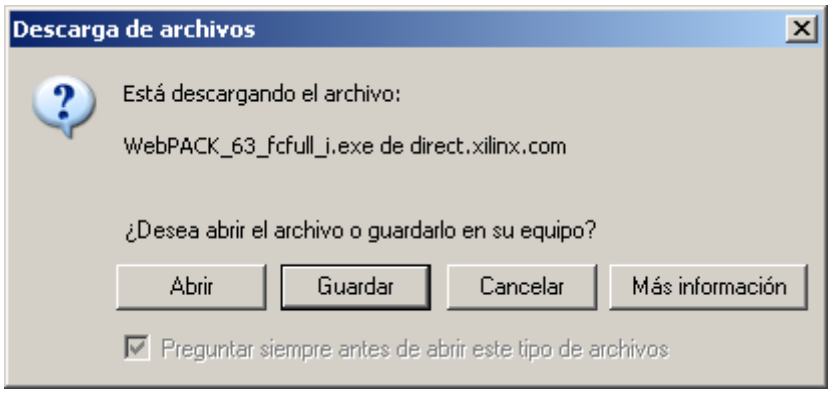
ISE *WebPACK* Download Module *(service pack required - see information below)*	**Download Size**	**PLD Design Environment**	**CPLD Device Support**	**FPGA Device Support**
Complete ISE WebPACK Software - includes programming tools	207 Mb*	✔	✔	✔
Complete CPLD Tool Set - includes programming tools	130 Mb*	✔	✔	
Complete Programming Tools	45 Mb*		✔	✔

Figura A.7: Descarga del *WebPACK*.

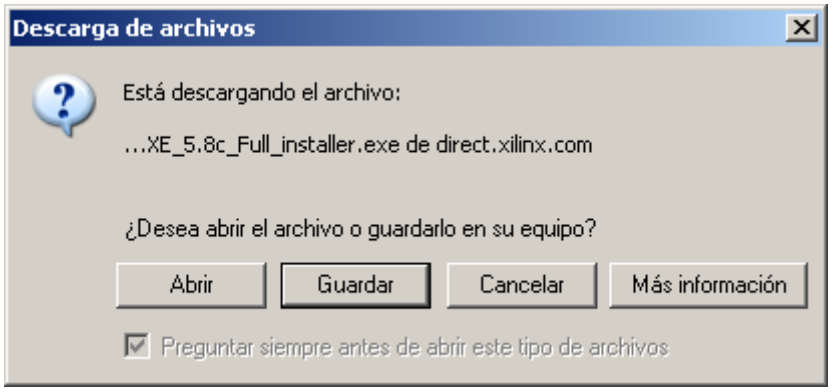
ModelSim XE Download Module	Download Size	MXE Simulator	CPLD VHDL Library	CPLD Verilog Library	FPGA VHDL Library	FPGA Verilog Library
Complete MXE Simulator	97 Mb	✔	✔	✔	✔	✔

Figura A.8: Descarga del *MultiSIM*.

Dado el tamaño en ambos paquetes, aconsejamos guardarlos en el disco duro para ahorrar tiempo en futuras descargas e instalaciones de los mismos.



(a) *WebPACK*



(b) *MultiSIM*

Figura A.9: Guardado de archivos.

A.1.2. Instalación del paquete.

Una vez que se ha completado la descarga del archivo *Complete ISE WebPACK Software*, lo ejecutamos para comenzar el proceso de instalación.

Este archivo es un archivo comprimido y, por tanto, al pinchar en él, el proceso de instalación empieza con la descompresión de los archivos que contiene. Estos archivos se guardans en un directorio temporal y, al finalizar la instalación, serán borrados. La descompresión de archivos dura unos minutos y, como presentamos en la figura, la barra de progreso informa del tiempo restante.

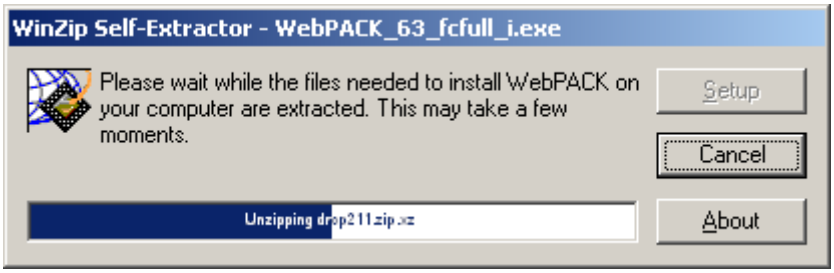


Figura A.10: Descompresión de los archivos de instalación.

Una vez finalizada la descompresión de los archivos, se muestra una pantalla de presentación durante unos segundos, y arranca el programa de instalación.

Figura A.11: Pantalla de Presentación.

En primer lugar, se presenta la licencia del producto, que debemos aceptar para proseguir con la instalación.

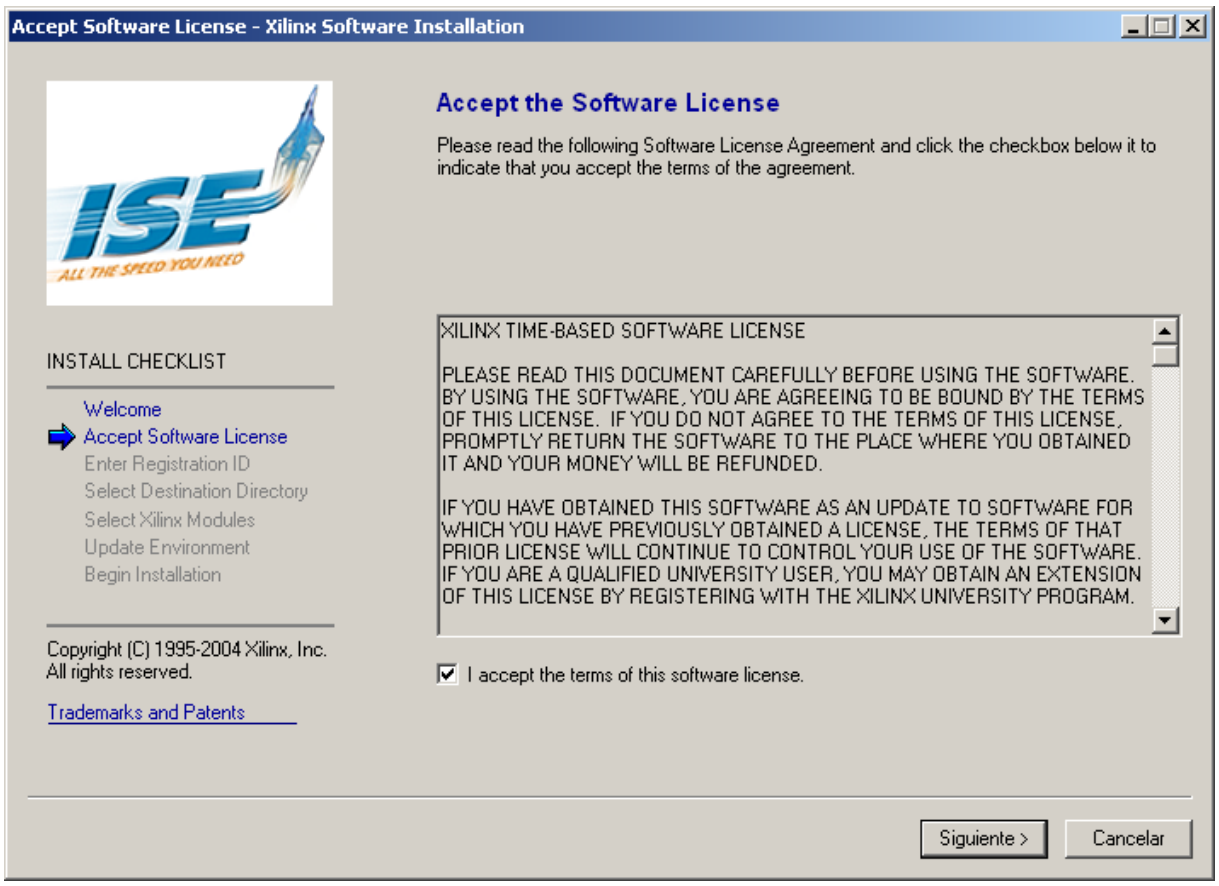


Figura A.12: Aceptación de la licencia.

No debemos olvidar que el *WebPACK* es un programa de libre distribución y que no contiene todas las utilidades que presenta el programa completo. Por esta razón, el programa de instalación saltará alguno de los pasos que sí se dan en la instalación del programa completo y, en concreto, los pasos *Enter Registration ID* y *Select Xilinx Modules*.

En la siguiente pantalla, seleccionamos el directorio de instalación y el directorio del *Menú Inicio* para crear el grupo de archivos que permitan un acceso rápido y cómodo al programa. Por defecto, se presentan los siguientes directorios.

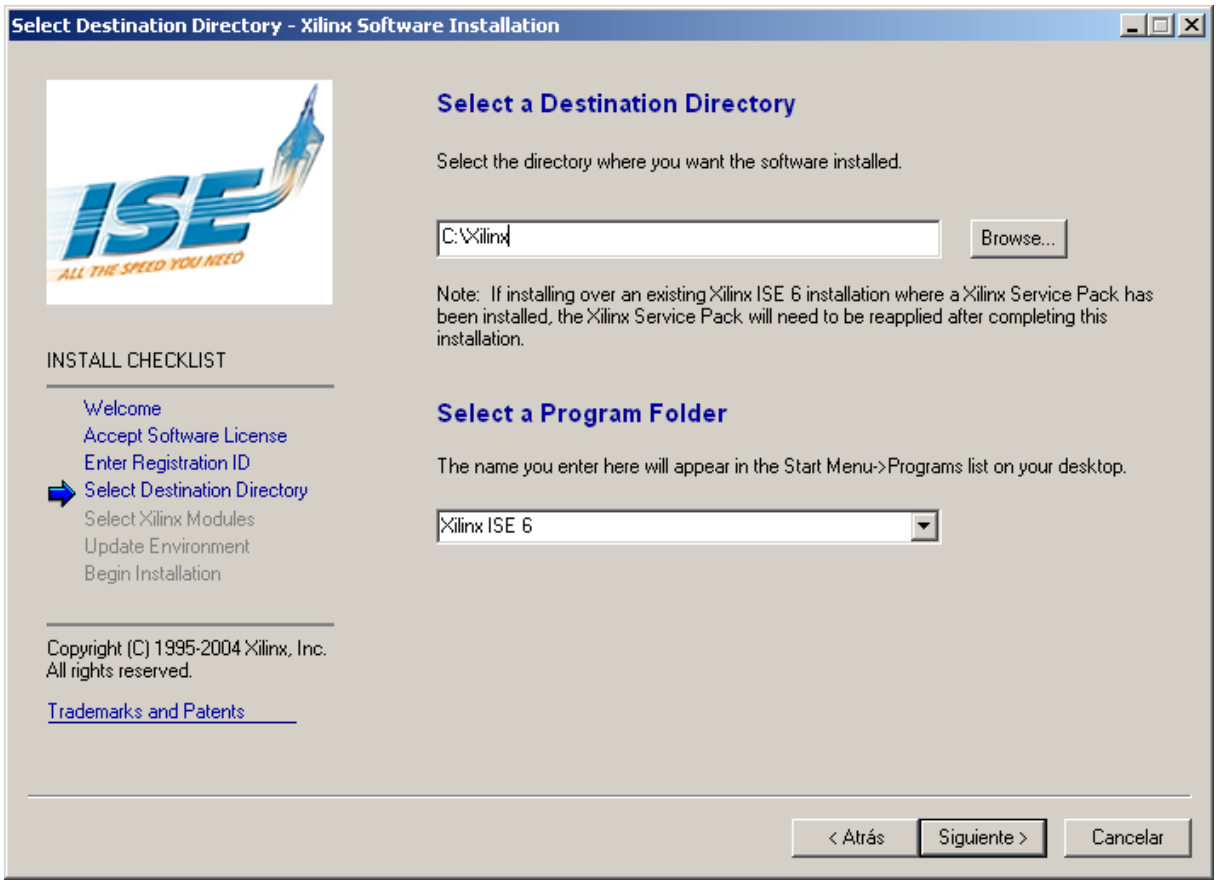


Figura A.13: Directorios de instalación.

Nosotros sugerimos realizar la instalación sobre el directorio *X:/Xilinx*, donde *X* es el nombre la unidad de disco duro donde se quiere hacer la instalación.

En la pantalla de actualización del entorno, deben activarse las dos casillas (*Variable XILINX* y *Variable PATH*) para tener la seguridad de que el software podrá arrancar y funcionar correctamente.

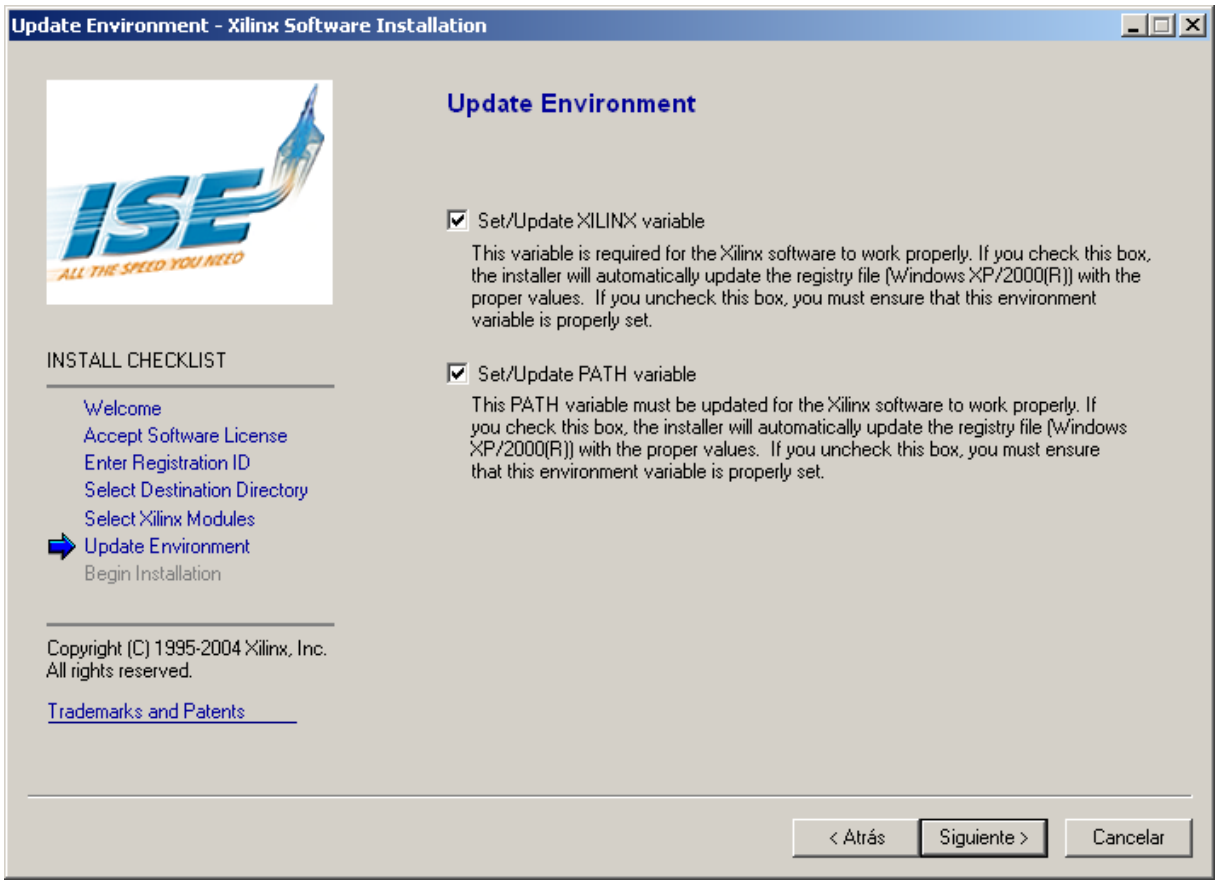


Figura A.14: Actualización de variables.

Si no se activan estas dos casillas, el usuario deberá cambiar el *PATH* manualmente.

La siguiente pantalla es un resumen que informa sobre las características de la instalación que ha seleccionado el usuario. Pinchando en *Install*, la instalación comenzará.

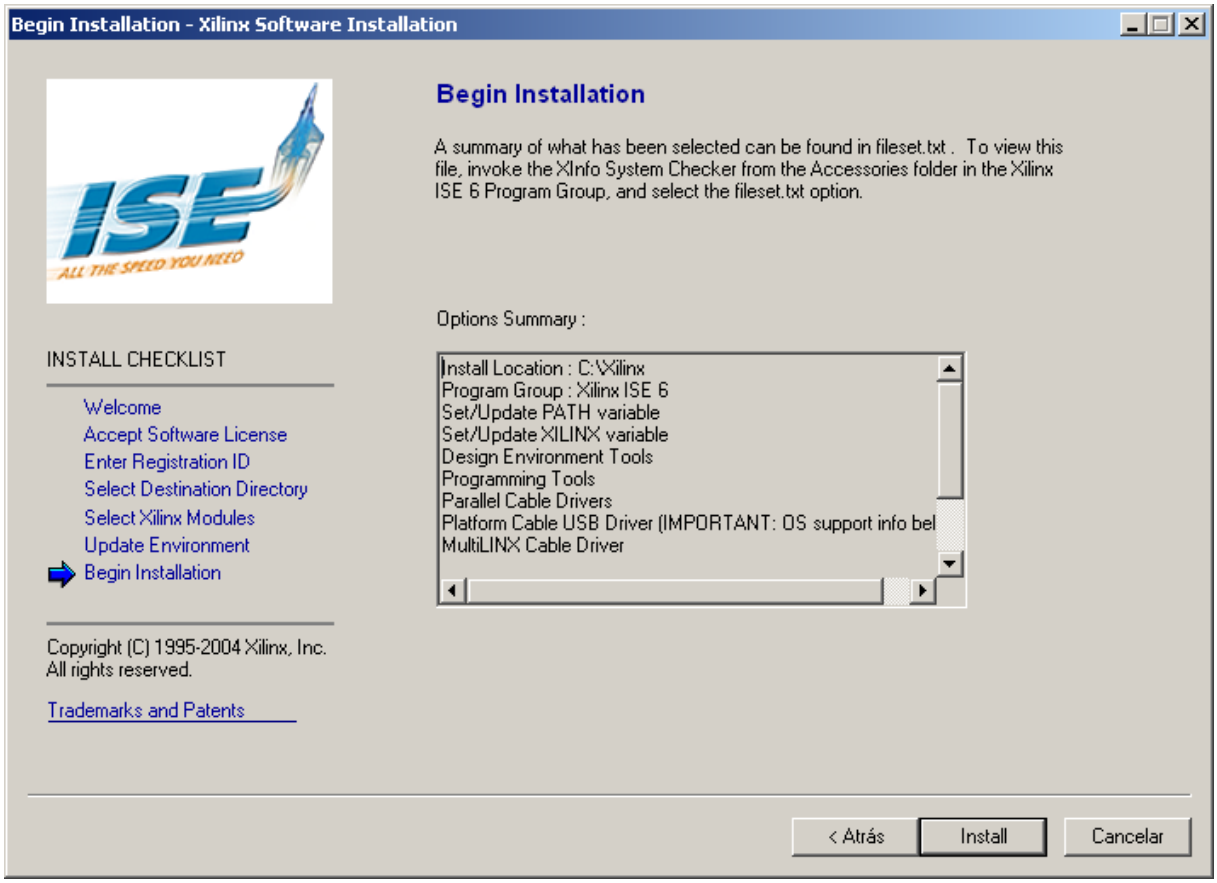


Figura A.15: Resumen de la instalación.

Una vez que se pinche este botón, no se pueden cambiar los parámetros de la instalación.

El proceso de instalación durará unos minutos.

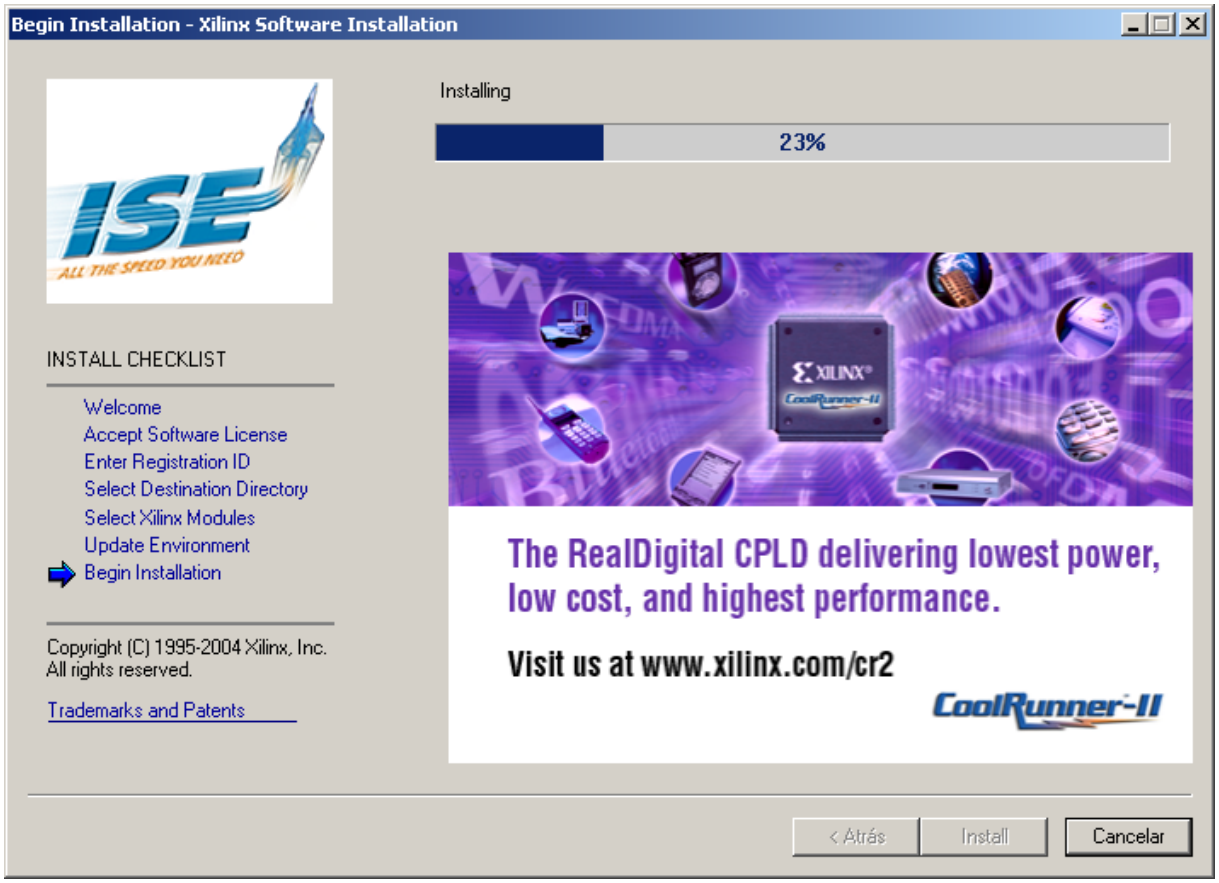


Figura A.16: Proceso de instalación.

La barra de progreso informa sobre la situación de la instalación en cada momento.

Al finalizar, se nos informa sobre la instalación del software para la comunicación del *PC* con la tarjeta a través del cable. Debemos quitar todas las conexiones a puertos *USB* que tengamos conectadas en ese momento, antes de proseguir con la instalación. Una vez esté todo preparado, pinchamos en *Sí*, para finalizar la instalación[2].

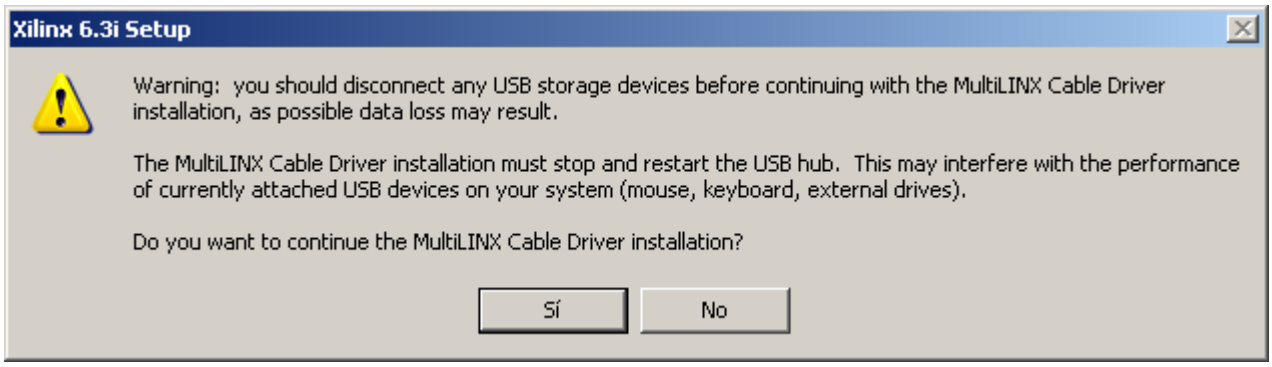


Figura A.17: Instalación de los drivers para la interface entre *PC* y la tarjeta.

Una vez concluida la instalación, el programa da la posibilidad de acceder a la *Ayuda de Xilinx ISE*. En cualquier caso, esta ayuda es accesible en todo momento desde el acceso colocado en el *Menú Inicio*.

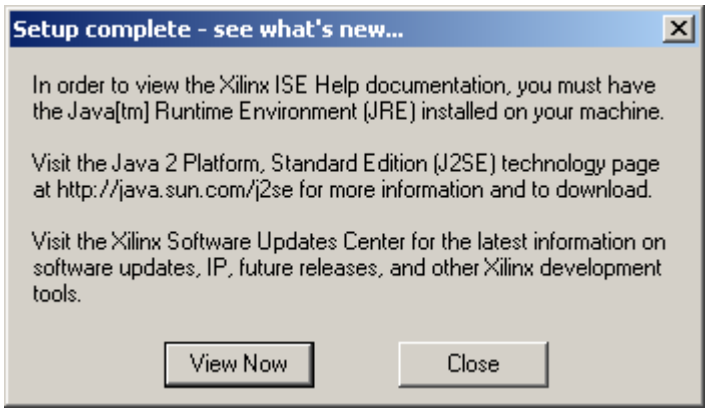


Figura A.18: Acceso a la Ayuda de *Xilinx ISE*.

[2]No es estrictamente necesario instalar este software para la comunicación entre el *PC* y la tarjeta, ya que el software que utilizaremos será el proporcionado por el fabricante de la placa (Ver apartado A.3).

La siguiente pantalla informa de que la instalación ha finalizado.

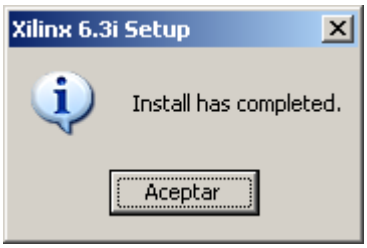


Figura A.19: Finalización de la instalación.

Por último, tras borrar los archivos temporales creados para la instalación, el programa nos pide reiniciar el equipo.

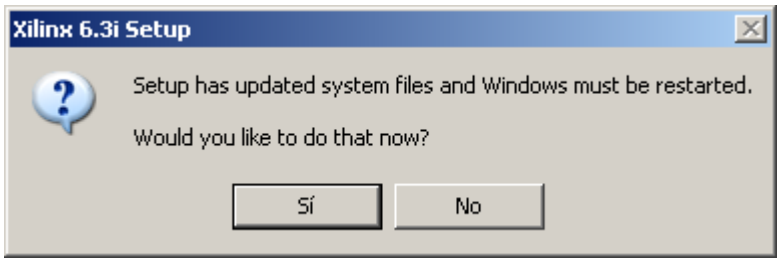


Figura A.20: Reinicio del *PC*.

Después de terminar la instalación y reiniciar el equipo, el usuario puede arrancar *Project Navigator* y trabajar con él sin ningún problema.

A.1.3. *Service Pack.*

Con la instalación del *WebPACK*, *Project Navigator* puede arrancar y funcionar perfectamente. Sin embargo, para estar totalmente actualizado, el paquete necesita la descarga e instalación del último *Service Pack* disponible. Si no está actualizado, al arrancar *Project Navigator*, el usuario será informado acerca del último *Service Pack* disponible en la web.

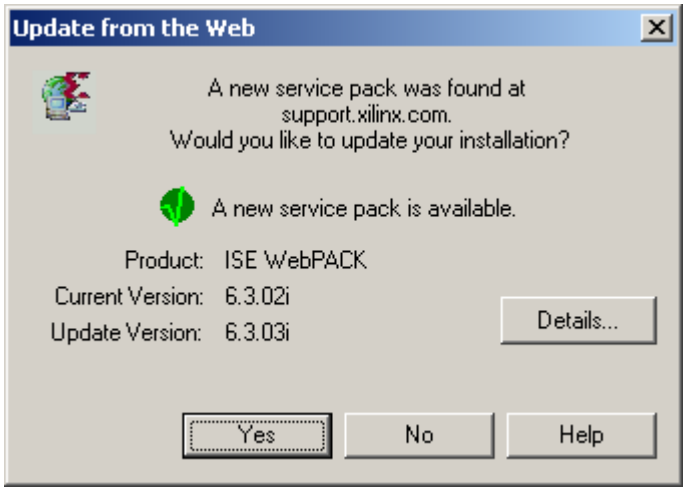


Figura A.21: Aviso de nuevo *Service Pack* disponible.

Obtención del *Service Pack.*

El *Service Pack* puede obtenerse directamente dando al botón *Yes* de la ventana anterior. Para descargarlo, *Project Navigator* debe estar cerrado, como informa el siguiente aviso.

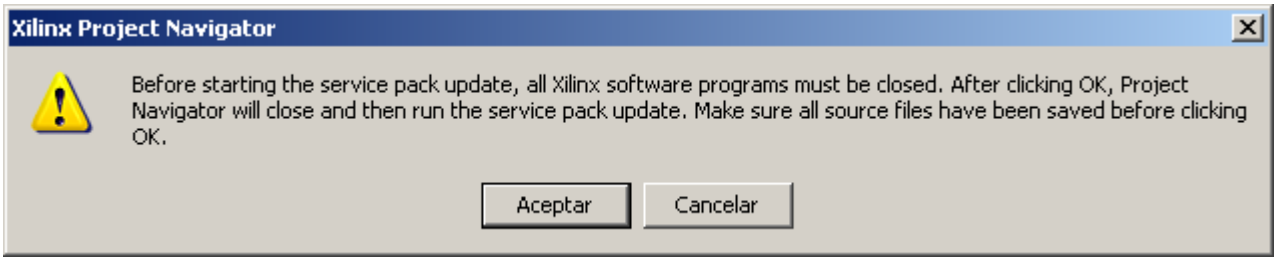


Figura A.22: Cerrado de *Project Navigator.*

Entonces, comienza la descarga desde la web de *Xilinx* y, al terminar, se realiza instalación la actualización automáticamente.

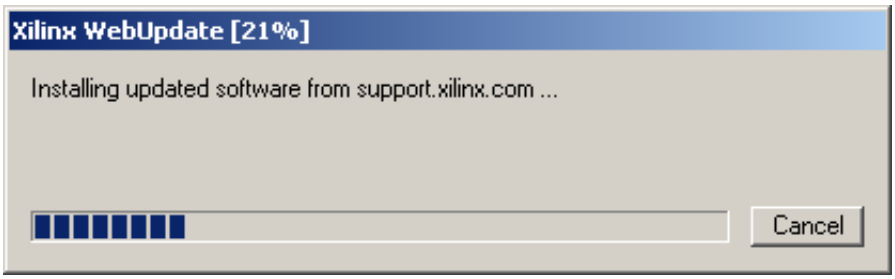


Figura A.23: Descarga en instalación del *Service Pack*.

Tras unos minutos, termina la actualización.

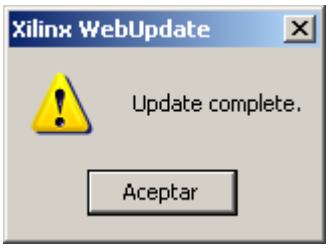


Figura A.24: Actualización completa.

Otra forma de descargar el paquete es desde la web, accediendo a la pantalla de descarga del *WebPACK ISE 6.3* como se vio en el apartado A.1.1. En esa pantalla, bajo el cuadro de descarga del *WebPACK ISE 6.3*, hay un párrafo que nos avisa sobre los *Service Packs*, y tiene un enlace a la pantalla de descarga de los mismos. Pinchando en ese enlace accedemos a la pantalla de descarga de actualizaciones.

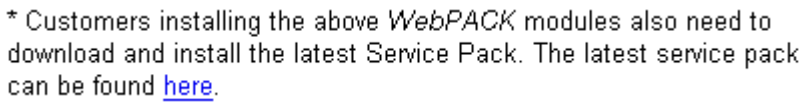
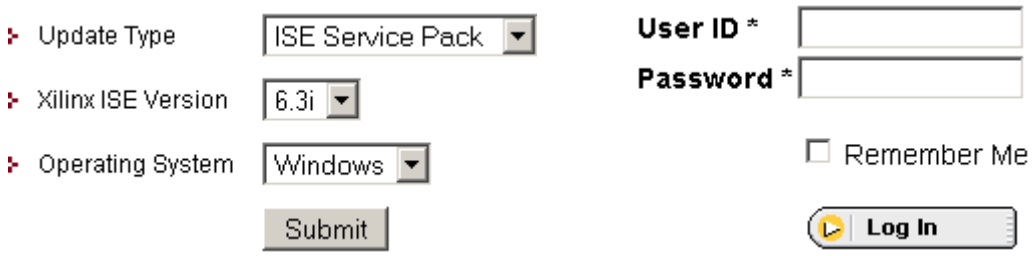


Figura A.25: Acceso a la descarga del *Service Pack* vía web.

Seleccionamos la actualización deseada: *Service Pack*, versión *ISE 6.3* y sistema operativo *Windows*. A continuación, el usuario debe introducir su *User ID* y *Password*[3], y pinchar sobre el botón *Log in* para acceder a la ventana de descarga del *Service Pack*.



(a) Selección (b) Registro

Figura A.26: Actualización.

En la siguiente pantalla, hay muchos recursos y/o actualizaciones que el usuario puede descargarse. De todos, solo nos interesa la última actualización disponible del *Service Pack*.

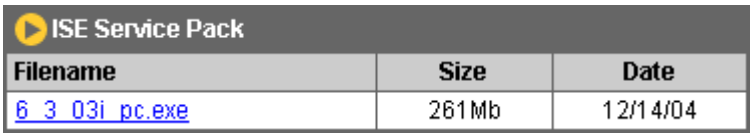


Figura A.27: Selección del *Service Pack*.

Al igual que en las descargas anteriores, aconsejamos guardarlo en el disco duro.

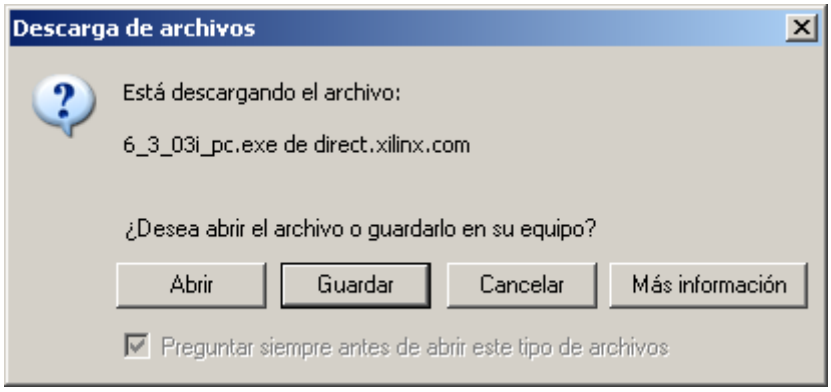


Figura A.28: Guardado de archivo.

[3]En este punto, se supone que el usuario ya está registrado.

Instalación del *Service Pack*.

El proceso de instalación es muy similar al de la instalación del *WebPACK*. El archivo descargado desde la web también es un archivo auto-extraíble que, al ejecutarse, crea un directorio temporal al que lleva todos los archivos que necesita para la instalación, de la misma forma que en la instalación del *WebPACK*.

Al finalizar la descompresión, arranca el programa de instalación del *Service Pack*. La ventana que se presenta permite elegir el directorio sobre el que se realizará la instalación. Por defecto, elige el directorio sobre el que está instalado el *WebPACK*.

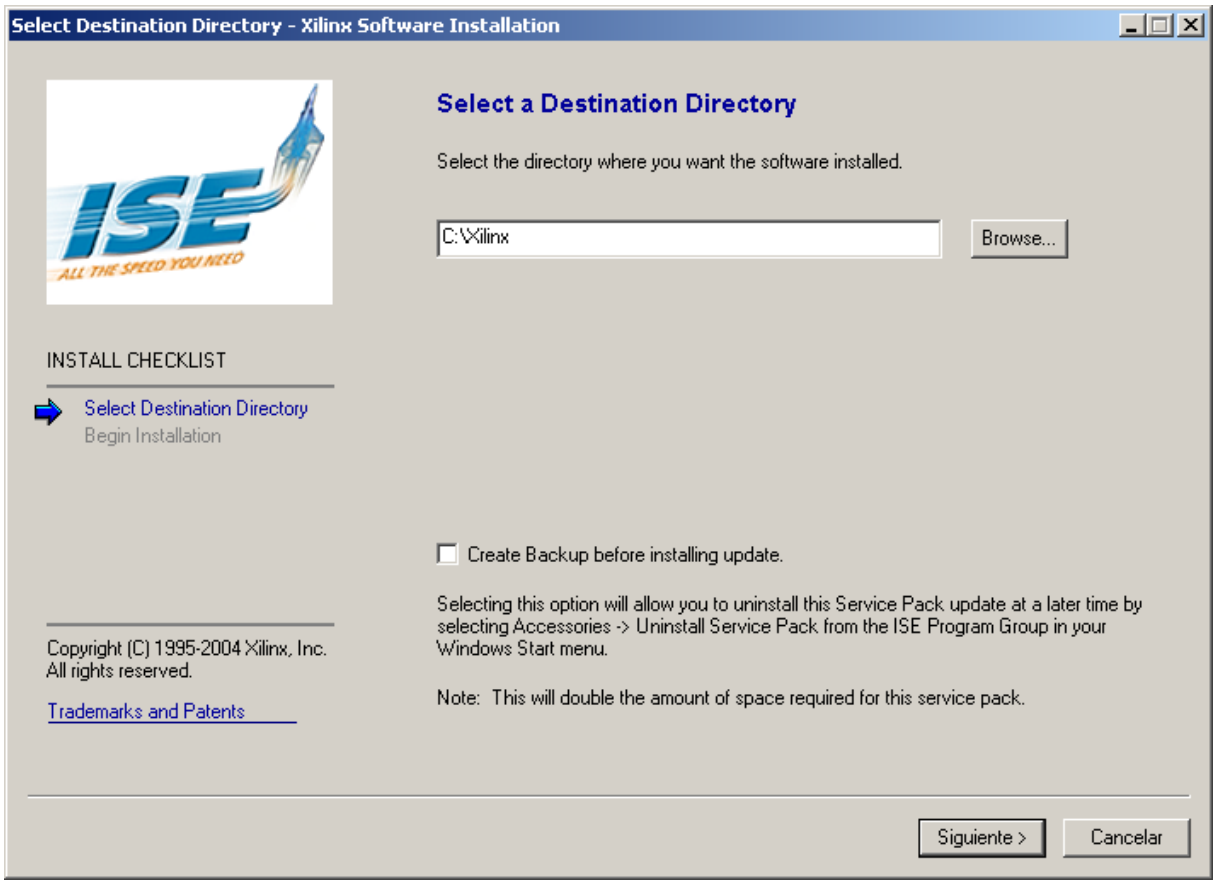


Figura A.29: Selección del directorio y Copia de seguridad.

Además, permite hacer una copia de seguridad de la configuración actual del programa, para poder volver a ella en el caso de querer des-instalar esta actualización.

De la misma forma que en la instalación del *WebPACK*, el programa presenta un resumen antes de comenzar el propio proceso de instalación.

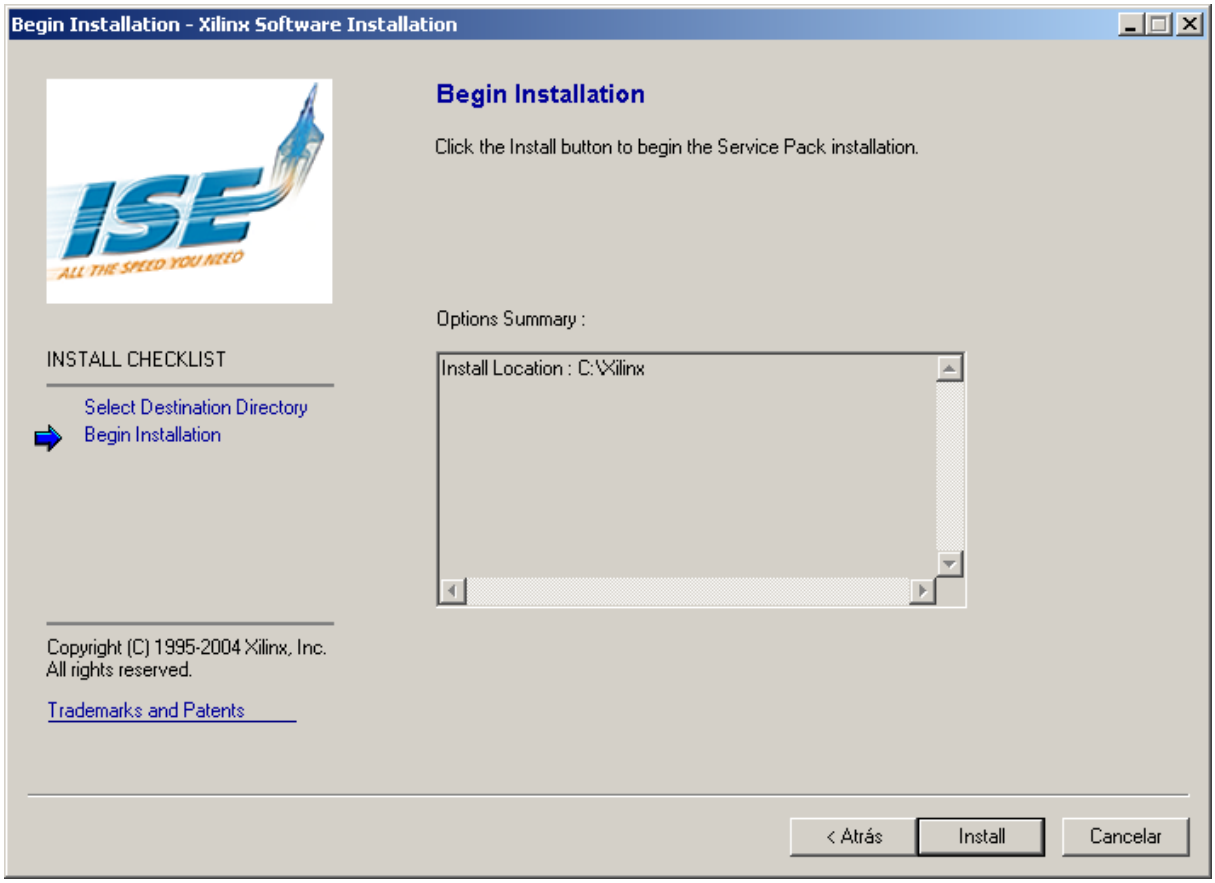


Figura A.30: Resumen de la actualización.

En este caso, la única información que se da al usuario es el directorio de instalación.

Al pinchar en *Install*, comienza la actualización. Si se ha seleccionado la opción de realizar *Copia de seguridad*, el proceso tarda varios minutos más.

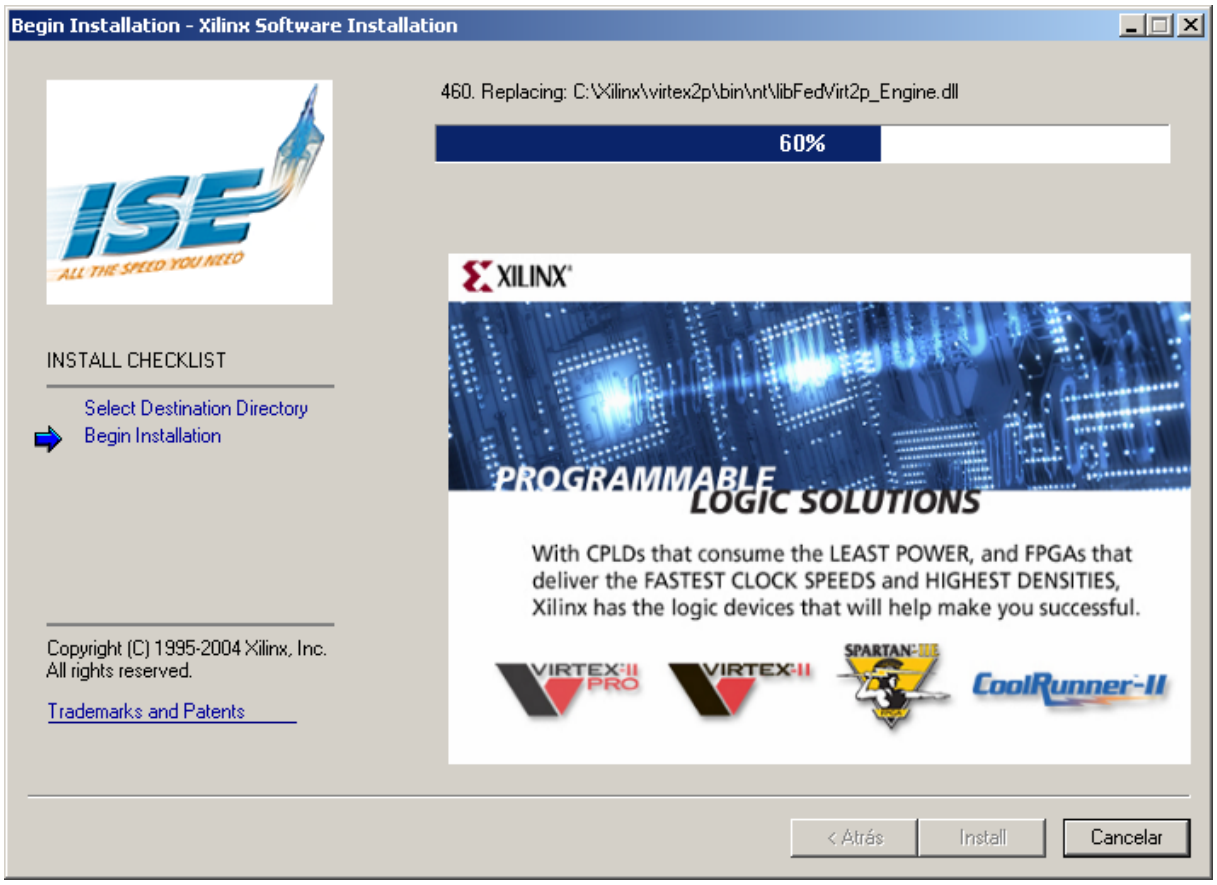


Figura A.31: Proceso de actualización.

Cuando la barra te progreso llega al 100 %, se borran los archivos y el directorio temporal que se crearon para realizar la instalación.

Para terminar este proceso, se nos informa de que la actualización se ha completado.

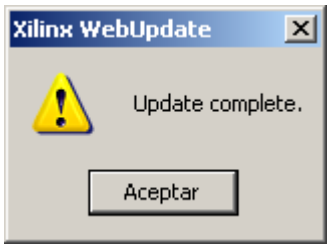


Figura A.32: Actualización completa.

En este punto, el *WebPACK* está completamente instalado y actualizado con la última versión del *Service Pack* disponible.

A partir de aquí, como vemos en el *Apéndice B*, el programa se arranca mediante el acceso a *Project Navigator* disponible en el *menú Inicio*.

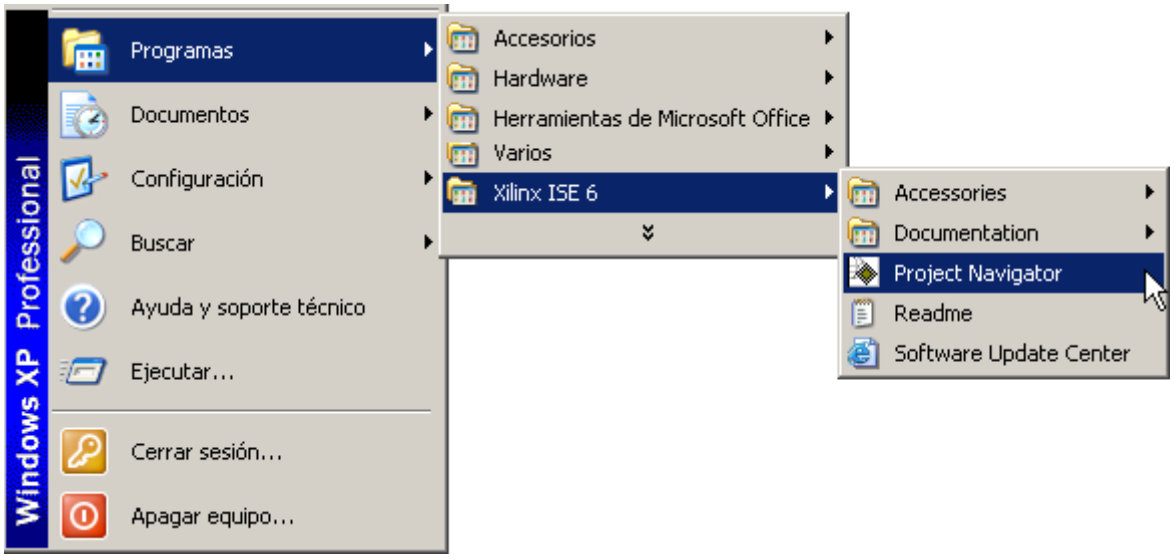


Figura A.33: Arranque del programa.

A.2. ModelSIM XE II.

A.2.1. Obtención del paquete.

Este paquete se descarga de la web de Xilinx, de la misma forma que el paquete *Xilinx ISE Webpack*. Ver A.1.1.

A.2.2. Instalación del paquete.

Al igual que hemos visto en el *WebPACK* y en el *Service Pack*, el archivo de instalación del *ModelSIM XE II* es un archivo comprimido y, por tanto, al pinchar en él, el proceso de instalación comienza con las descompresión de los archivos que contiene. También, los archivos necesarios para la instalación son guardados en un directorio temporal y, al finalizar la instalación, serán borrados. En esta pantalla, debemos seleccionar la opción de que el programa de instalación arranque automáticamente tras finalizar la descompresión de los archivos.

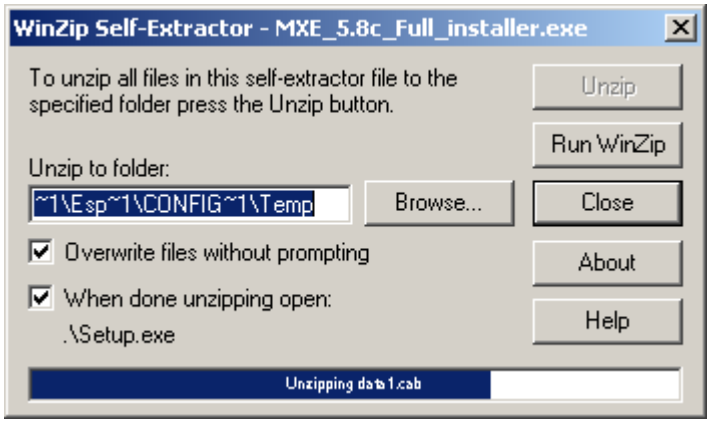


Figura A.34: Descompresión de archivos.

Si hemos seleccionado dicha opción, al terminar la descompresión de archivos, arrancará el programa de instalación. La primera pantalla es la presentación del programa de instalación, con el nombre del programa (*ModelSIM*) y la versión que se instalará (*XE*).

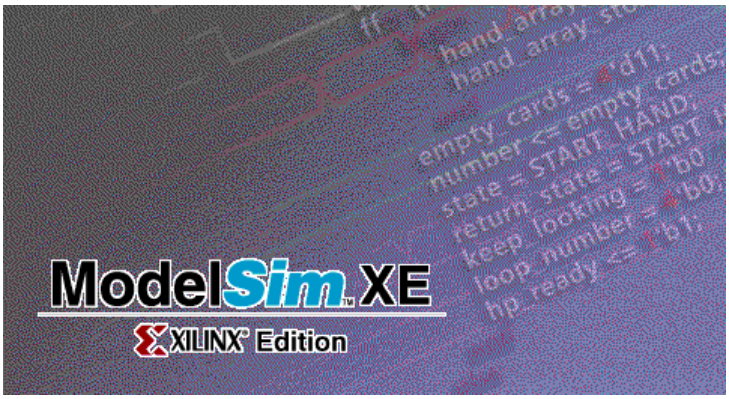


Figura A.35: Pantalla de presentación.

La primera pantalla nos permite elegir entre una instalación libre, o gratuita, del producto y una instalación completa. Nosotros seleccionaremos la versión libre, que ya contiene todos los recursos que necesitamos para simular nuestros circuitos.

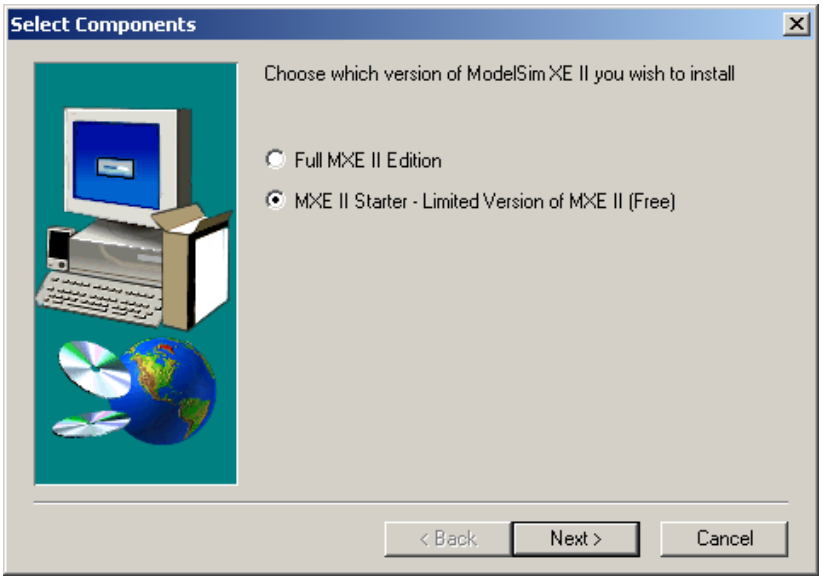


Figura A.36: Tipo de instalación.

La siguiente pantalla que se nos presenta es la pantalla de bienvenida a la instalación del programa *ModelSIM XE II 5.8c*. Con el fin de que la instalación se realice sin ningún problema, se sugiere cerrar cualquier programa que esté corriendo durante el proceso de actualización, especialmente los antivirus. Para comenzar la configuración de los parámetros de instalación, presionamos el botón *Next*.

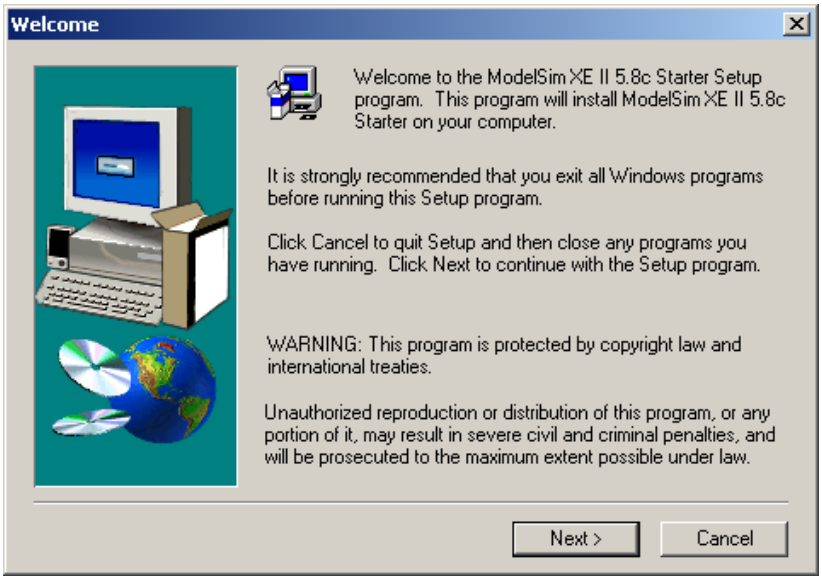


Figura A.37: Pantalla de bienvenida.

Para continuar con la instalación, es necesario aceptar los términos del acuerdo de licencia. Por tanto, debemos presionar el botón *Yes*. De lo contrario, saldremos del programa de instalación.

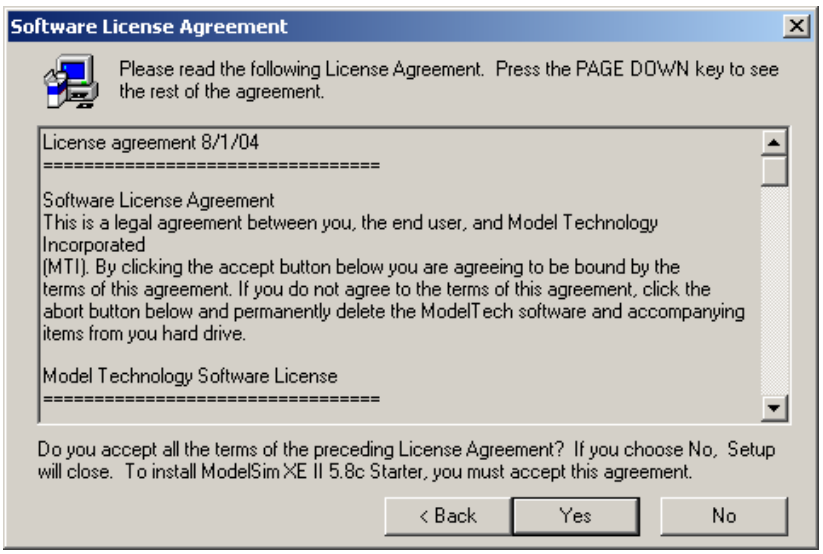


Figura A.38: Aceptación de licencia.

En la siguiente pantalla, podemos elegir el directorio donde queremos realizar la instalación del programa. Con el botón *Browse*, podemos buscar la carpeta donde deseamos realizar la instalación, o bien, escribir una ruta directamente. Si la ruta definida de este segundo modo no existe, el programa nos dará la posibilidad de crearla al realizar la instalación de los archivos. Una vez que hemos escrito la ruta de instalación, presionamos en *Next* para continuar con el siguiente paso de la instalación.

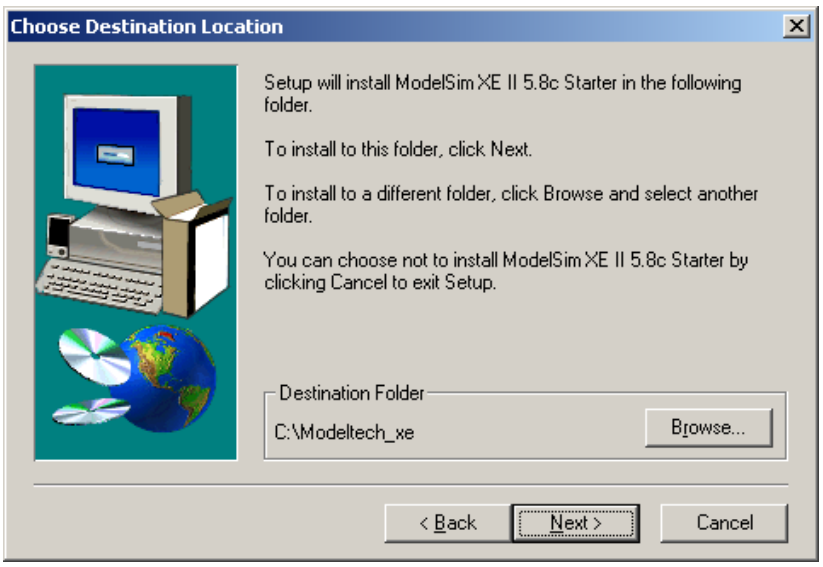


Figura A.39: Selección del directorio de instalación.

Para realizar las simulaciones, el programa *ModelSIM XE II* necesita tener instaladas unas librerías que sean capaces de comprender el lenguaje en el que el *WebPACK* genera los ficheros del diseño. Dadas las características de los diseños que implementaremos en el *WebPACK*, en esta instalación elegiremos la opción *Full VHDL*.

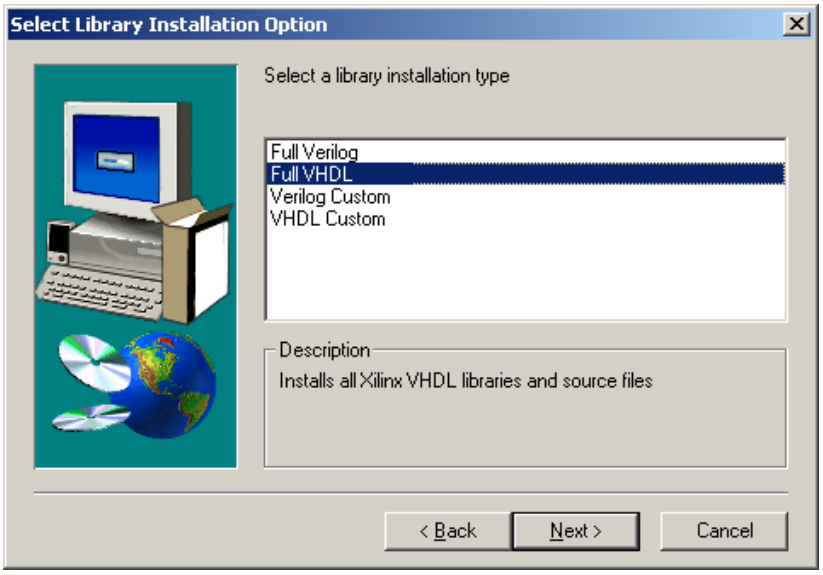


Figura A.40: Selección de la librería de instalación.

Para finalizar la configuración de la instalación, se elige un directorio en el *Menú Inicio* desde el cual el programa será accesible.

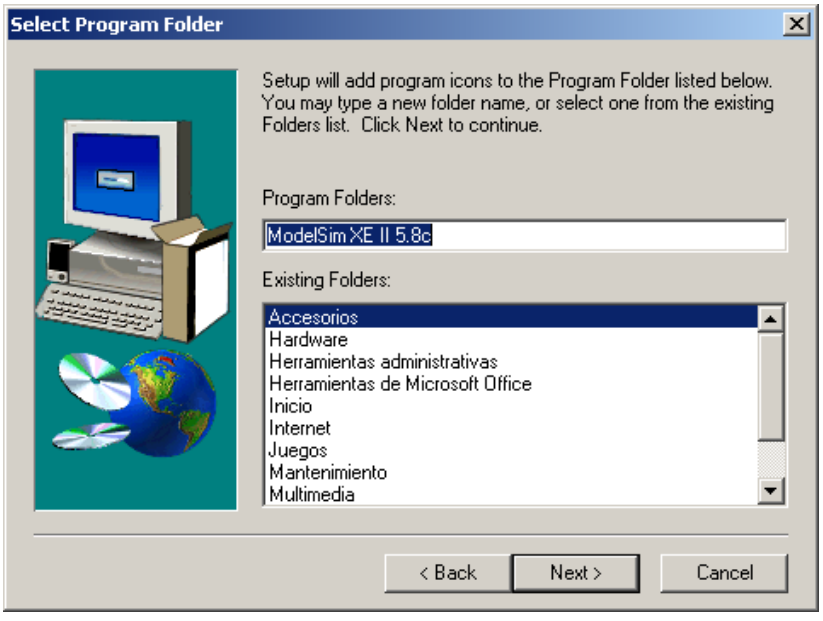


Figura A.41: Selección del directorio del *menú Inicio*.

Presionando en el botón *next*, comienza el proceso de instalación.

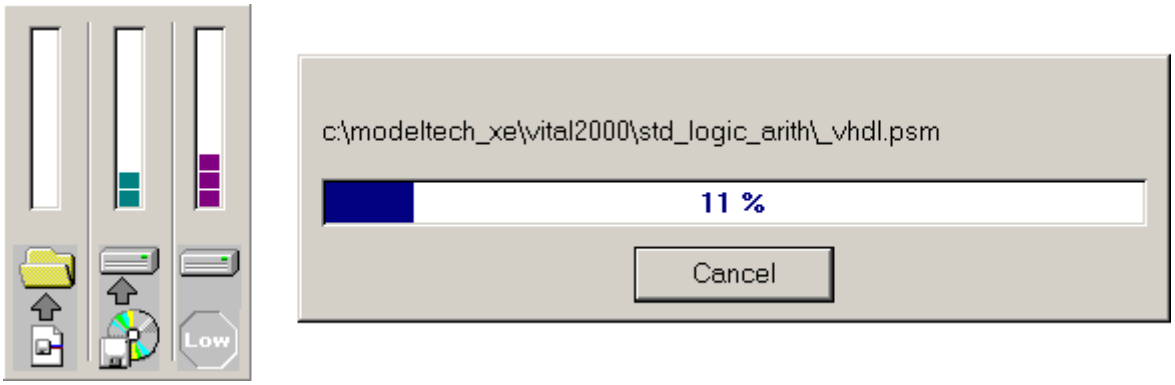


Figura A.42: Proceso de instalación.

Tras la instalación de los archivos, se nos ofrece la posibilidad de crear un acceso directo al programa en el escritorio.

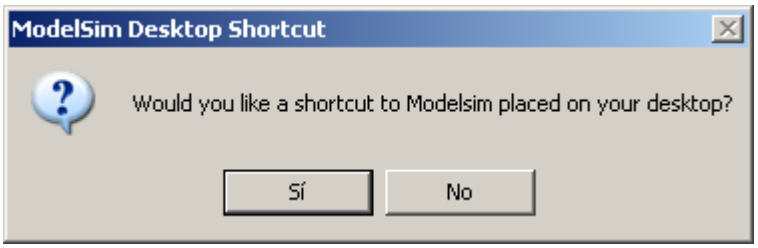


Figura A.43: Creación de acceso directo.

El siguiente paso es añadir *ModeSIM* al path. Aunque nosotros no ejecutaremos el programa desde ventanas de *DOS*, conviene añadirlo.

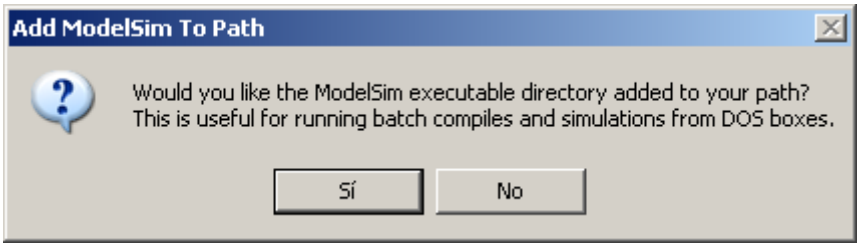


Figura A.44: Añadido al path.

Para completar la licencia, necesitamos registrarnos en la web de *xilinx*. En este momento, no podemos hacerlo pero, en el apartado A.2.3, veremos cómo se realiza.

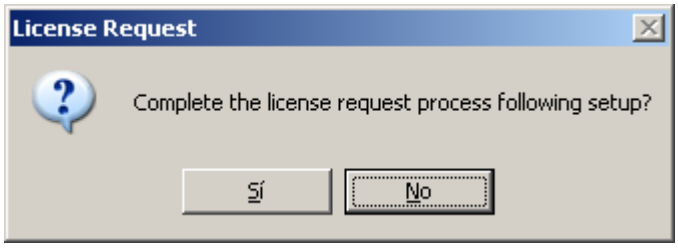


Figura A.45: Comienzo del proceso de licencia.

La última pantalla nos informa de que el programa ha sido correctamente instalado, y de que debemos obtener la licencia para poder utilizarlo.

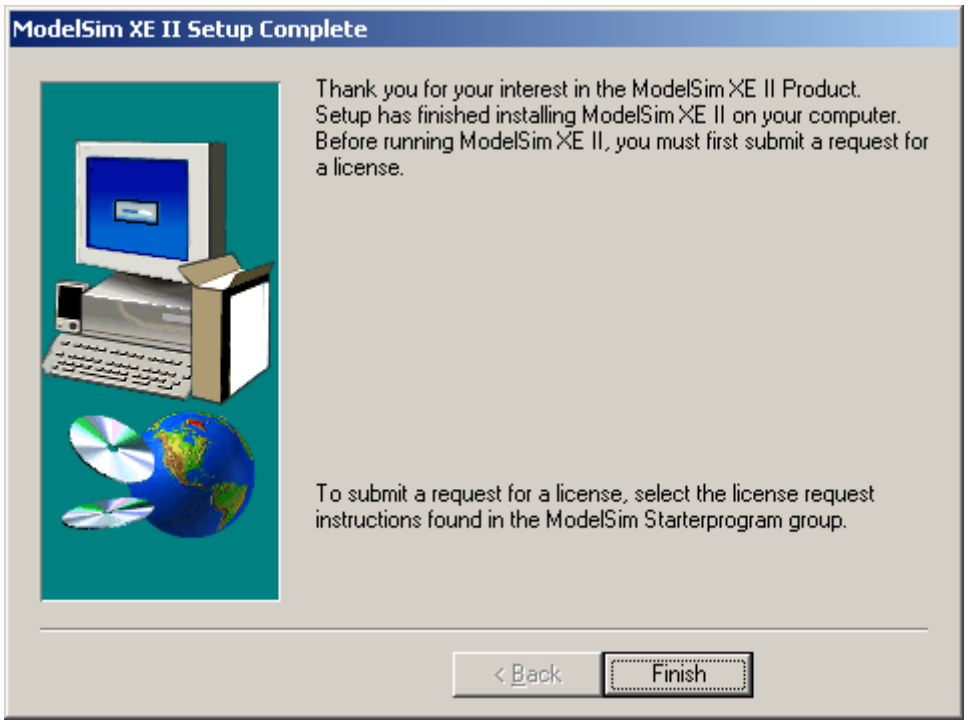


Figura A.46: Finalización de la instalación.

A.2.3. Obtención y validación de la licencia.

Una vez instalado, podemos arrancar el programa desde el acceso directo situado en el *Escritorio*, o bien desde el acceso situado en el *Menú Inicio*. Pinchando en cualquiera de ellos, el programa presenta el siguiente mensaje de aviso y, tras presionar en *Aceptar*, se cierra.

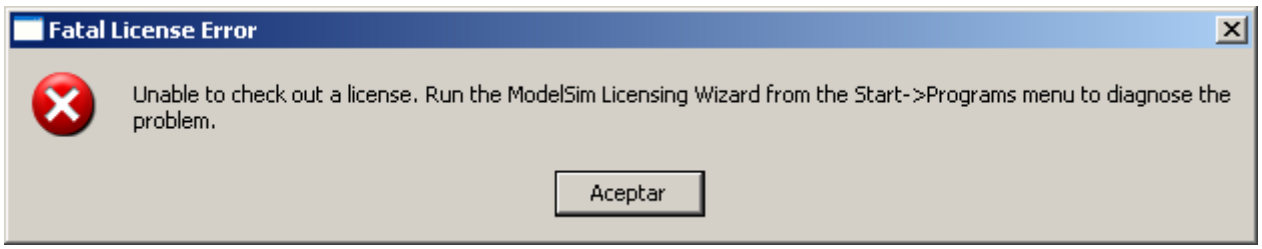


Figura A.47: Error en la comprobación de la licencia.

En efecto, hasta que no validemos la licencia, *ModelSIM XE II* no podrá correr. Para ello, las instrucciones se encuentran en el archivo *lic_request.txt* del directorio donde instalamos el programa. En ese archivo, podemos leer que tenemos varios formas de realizar la validación de la licencia. Por nuestra experiencia, la opción más fácil y cómoda es acceder a la página web que se nos indica y, una vez allí, rellenar la licencia.

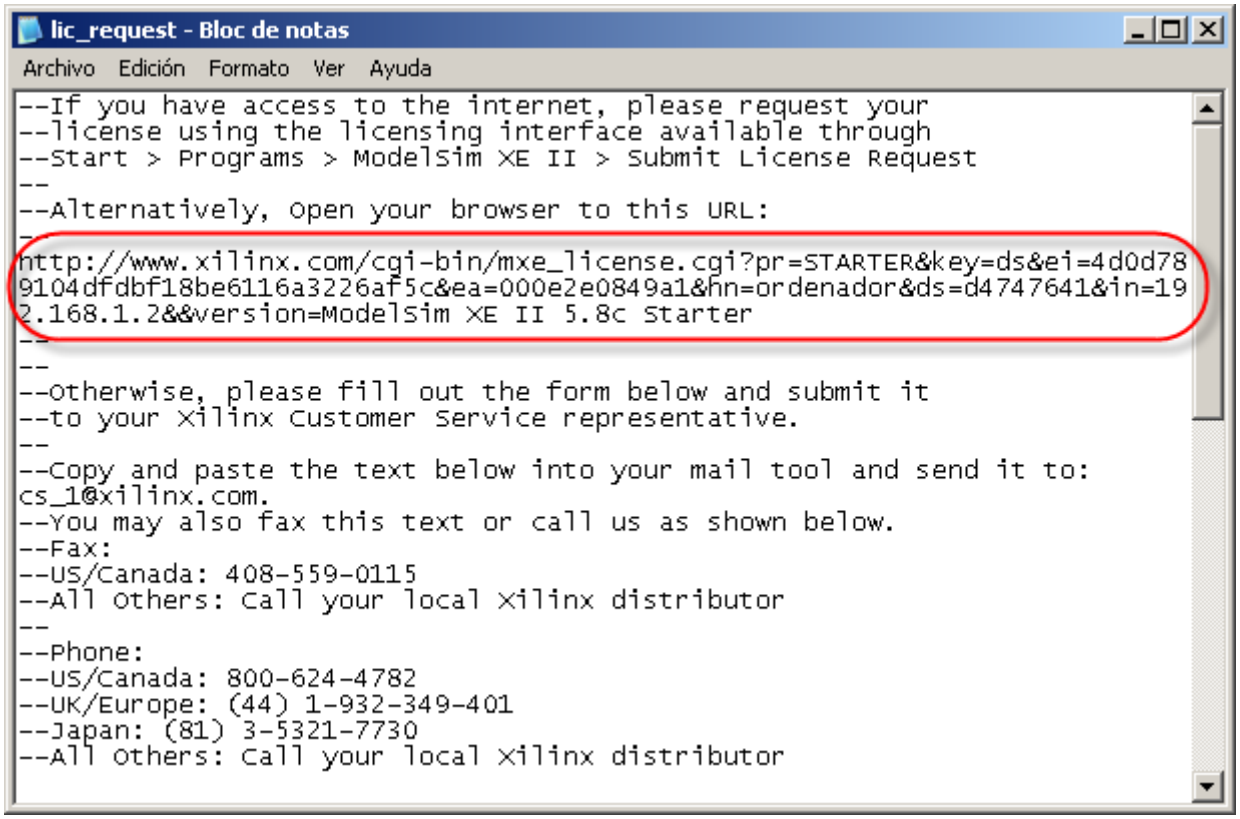


Figura A.48: Archivo de información sobre la licencia.

Por tanto, copiamos la dirección *URL* que aparece en el archivo y la pegamos en un navegador. Entonces, se abre la pantalla de petición de licencia de *Xilinx ModelSIM*. Como suponemos que el usuario ya está registrado, debe pinchar en el enlace *Continue*.

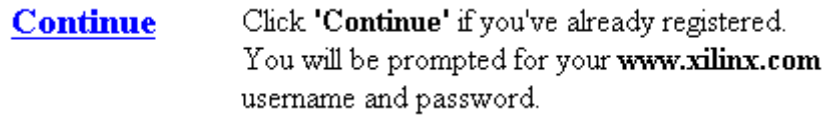


Figura A.49: Petición de licencia.

A continuación, tenemos que rellenar un formulario donde debemos poner una dirección de correo electrónico, a la que *Xilinx* enviará el archivo de licencia.

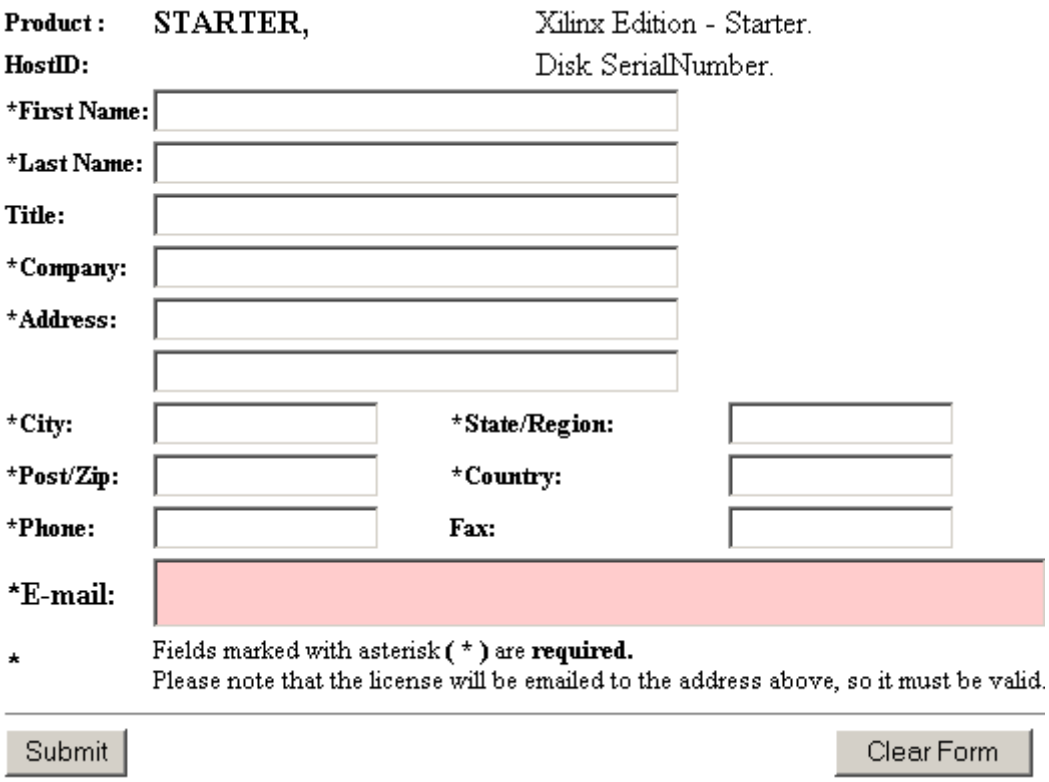


Figura A.50: Formulario de licencia.

Antes de enviarnos la licencia, *Xilinx* nos informa de que tenemos la versión de evaluación, y de que existe una versión comercial mucho más potente. La licencia nos será enviada via correo electrónico.

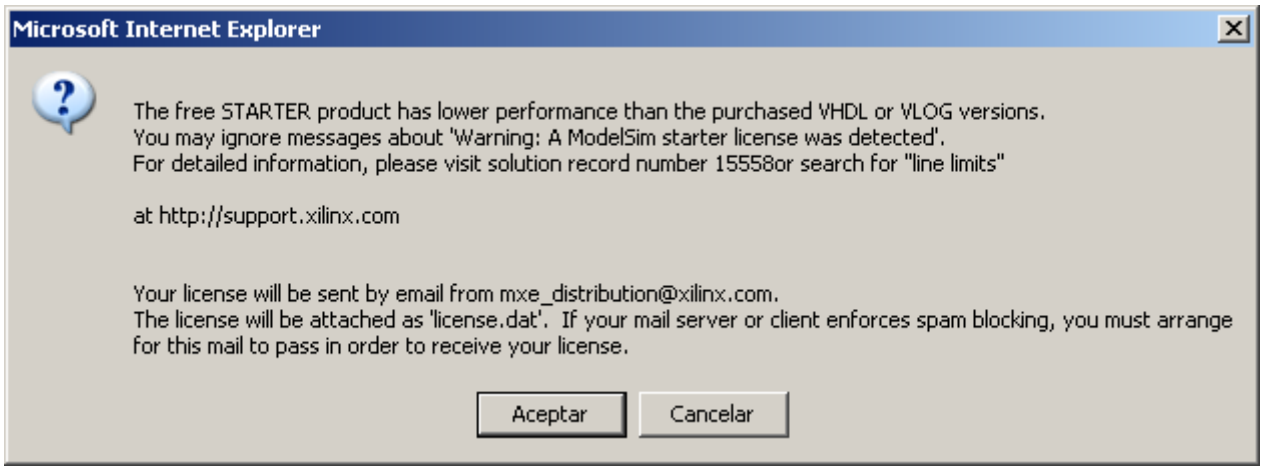


Figura A.51: Aviso de versión de evaluación.

Tras este aviso y para terminar el proceso de petición de licencia, accedemos a una última pantalla donde *Xilinx* nos resume nuestra petición y datos. Ya podemos cerrar el navegador para seguir con la validación de la licencia.

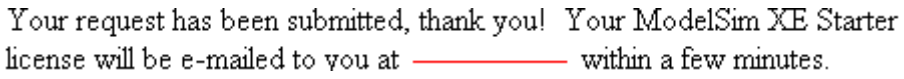


Figura A.52: Resumen de petición de licencia.

Después de unos minutos, recibimos la licencia en la dirección de correo electrónico que hemos suministrado a *Xilinx*. Una vez que dispongamos la licencia en nuestro ordenador, podemos comenzar el proceso de validación. Pero, antes de nada, sugerimos copiar el archivo *license.dat* que acabamos de recibir en el directorio donde instalamos el programa, ya que, cada vez que éste sea arrancado, va a buscar el archivo de licencia, y lo más cómodo y seguro es que esté en dicho directorio.

Para comenzar el proceso de validación, arrancamos el programa *Licensing Wizard* desde el enlace situado en el *menú Inicio*.

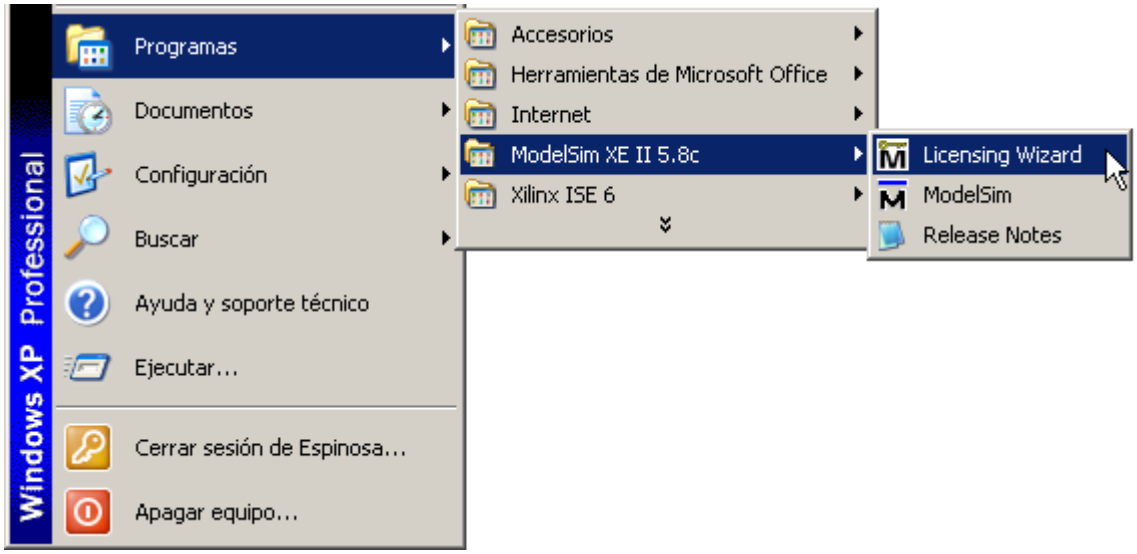


Figura A.53: Enlace del *menú Inicio*.

El programa de validación arranca con una pantalla de presentación. Para seguir, debemos presionar el botón *Continue*.

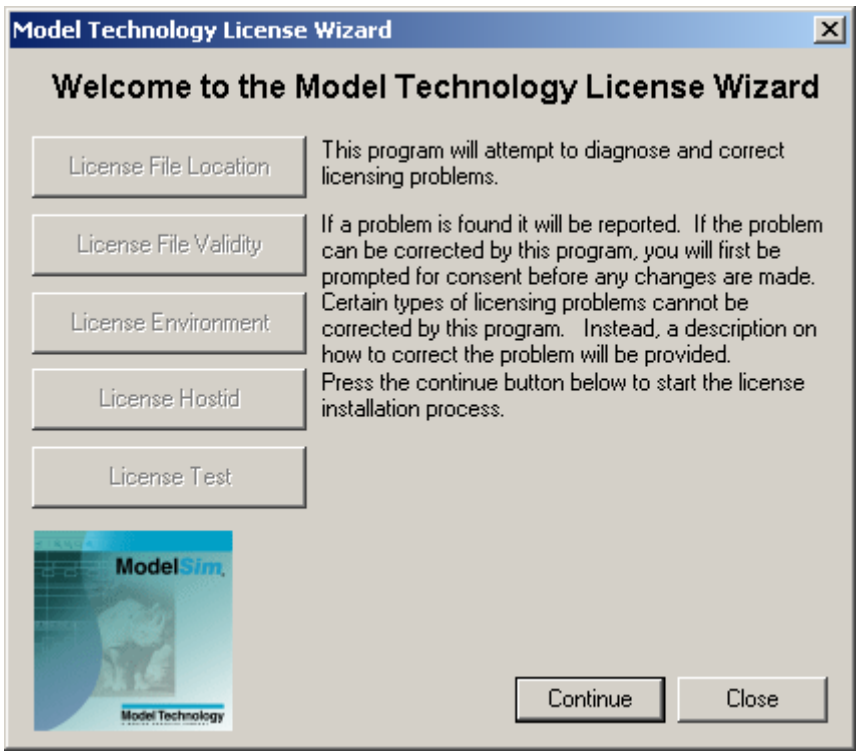


Figura A.54: Programa de validación de la licencia.

El primer paso es dar la ruta donde el programa encontrará el archivo *license.dat* que *Xilinx* nos acaba de enviar.

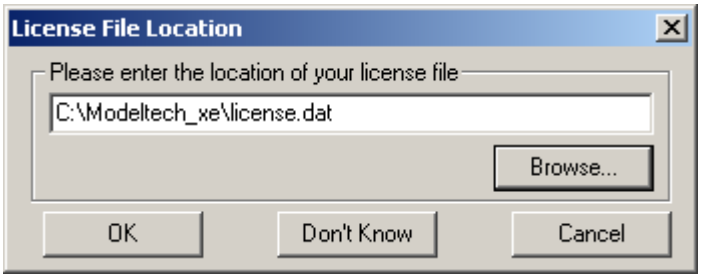


Figura A.55: Ruta del archivo *license.dat*.

Al escribir la ruta donde guardamos el archivo de licencia, el programa nos encontrará una licencia perpetua.

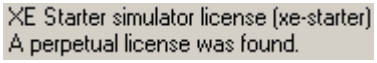


Figura A.56: Licencia perpetua.

El programa de validación de licencia ha concluido con éxito. Presionando el botón *Close*, se cierra el programa, y podemos arrancar el simulador *ModelSIM*.

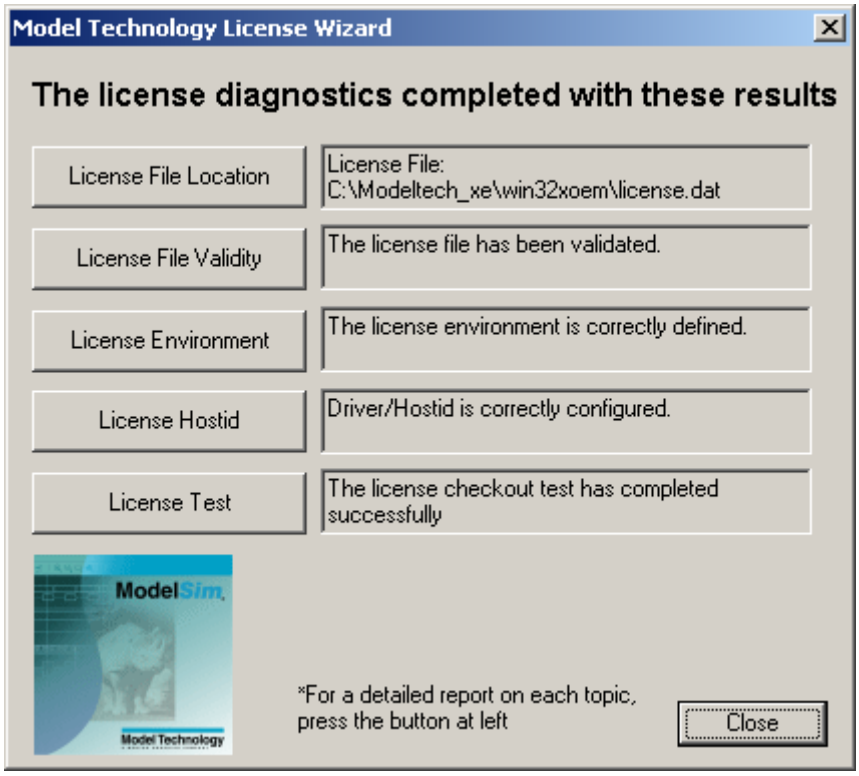


Figura A.57: Finalización del programa de validación de la licencia.

A.3. XSTools.

El último paquete es el formado por las herramientas para la interface con la placa (*XStools*). El *CD* que que contiene estas herramientas forma parte del lote que se obtiene al realizar la compra de la tarjeta.

Para comenzar la instalación, se introduce dicho *CD* en el lector del ordenador. Entonces, el programa de instalación arranca automáticamente.

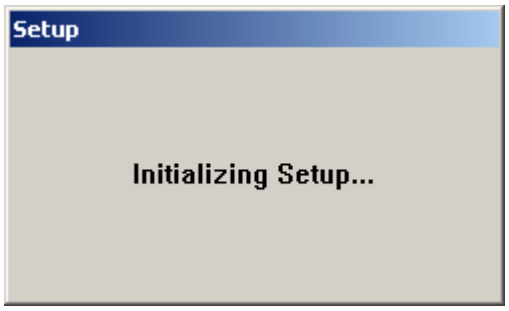


Figura A.58: Arranque automático del programa de instalación.

Lo primero que se nos presenta es la pantalla de bienvenida.

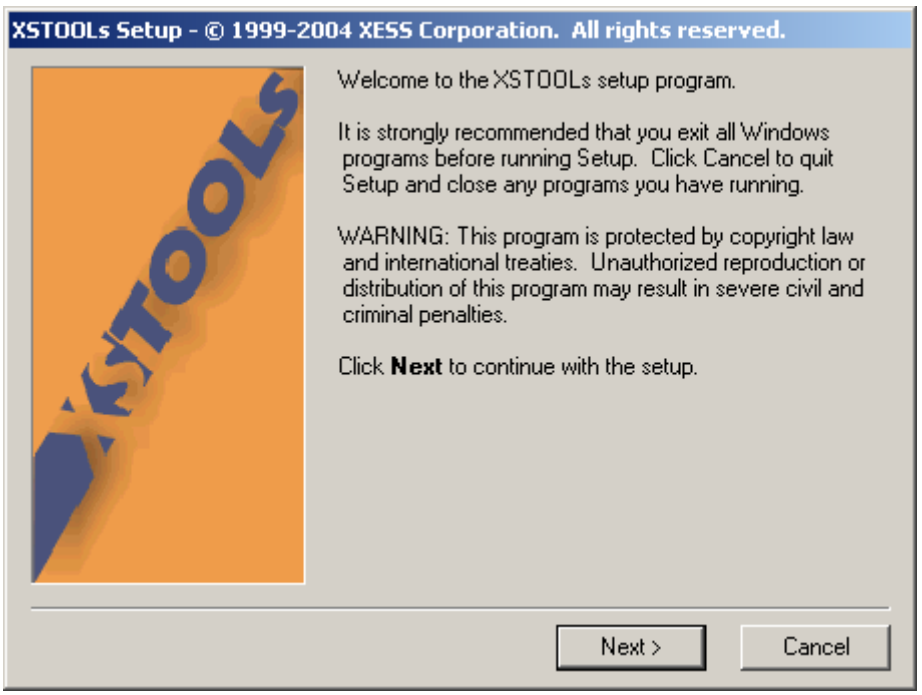


Figura A.59: Pantalla de bienvenida al programa de instalación.

Para realizar la instalación, es necesario entrar en el programa como administrador.

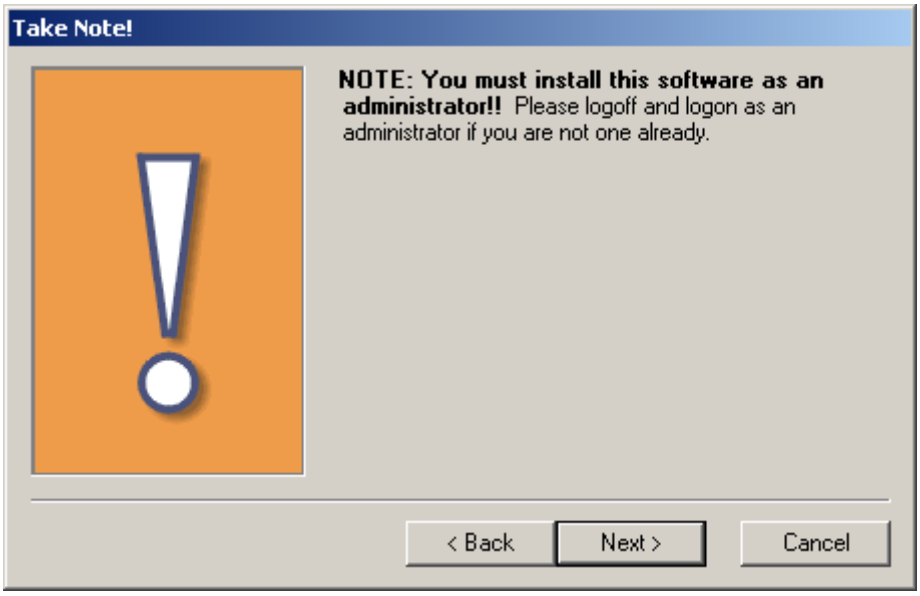


Figura A.60: Acceso como administrador.

En la siguiente pantalla, se presentan las condiciones de la licencia. Para continuar la instalación debemos aceptarlas.

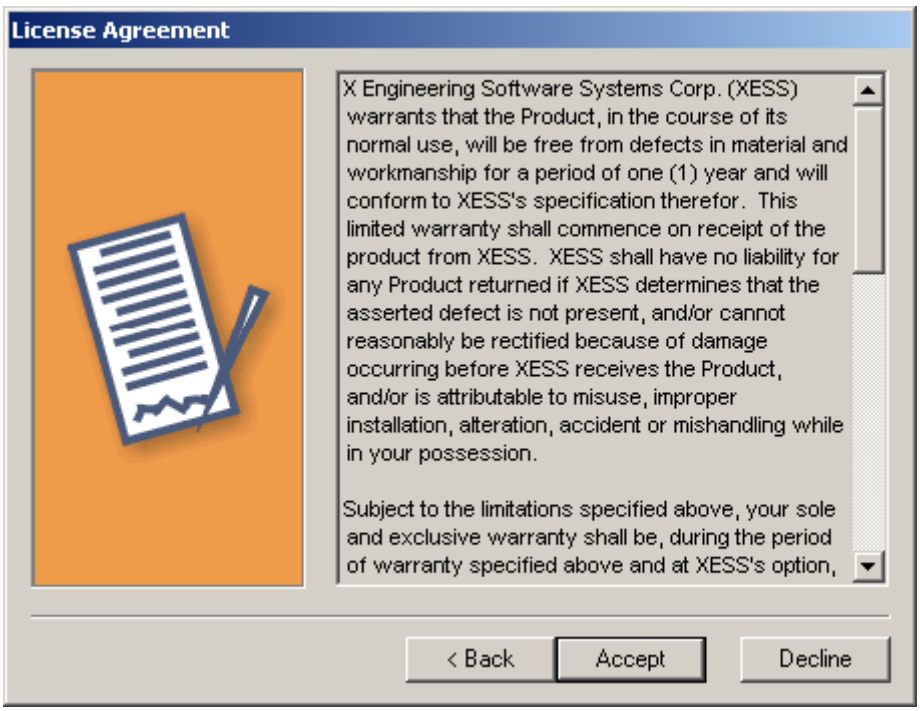


Figura A.61: Aceptación de la licencia.

A continuación, elegimos el directorio de instalación del programa.

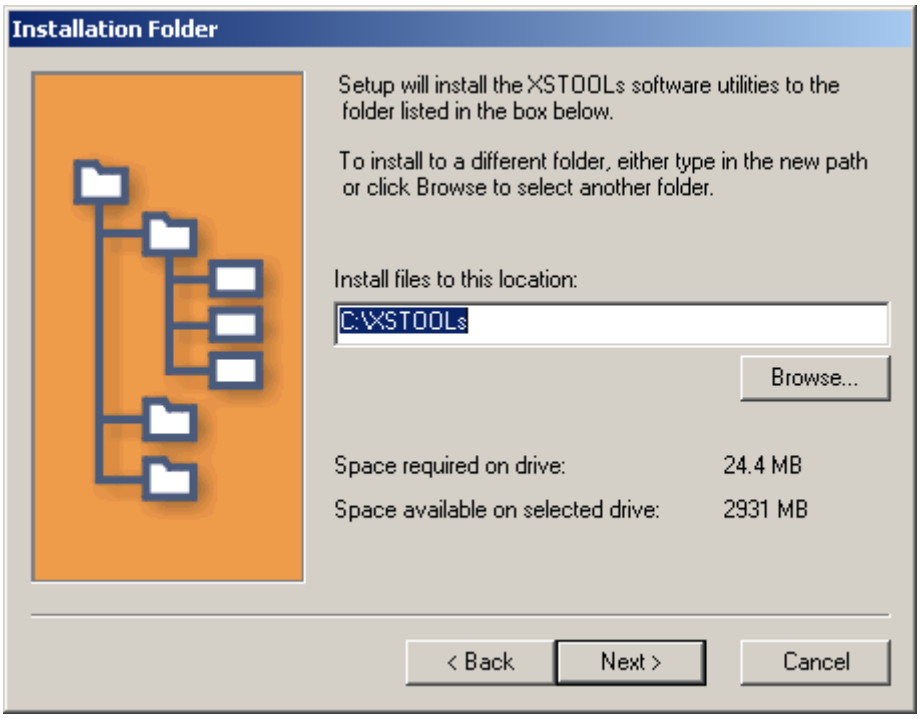


Figura A.62: Directorio de instalación y *menú Inicio*.

El siguiente paso es configurar el acceso desde el *Menú Inicio*.

Figura A.63: Directorio de instalación y *Menú Inicio*.

En este CD, se incluyen también una versión del *WebPACK* de *ISE*, que ya hemos instalado (ver apartado A.1.2) y del *ModelSIM* que también hemos instalado (ver apartado A.2.2). Estas versiones son más antiguas que las que nosotros hemos bajado e instalado anteriormente. Por tanto, no las instalaremos y, de este CD, solo nos centraremos en las herramientas de interface con la tarjeta.

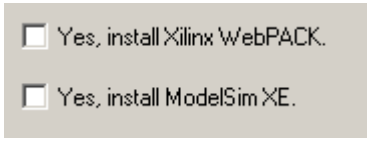


Figura A.64: Instalación del *WebPACK* y del *ModelSIM*.

De la misma forma, tampoco sugerimos la instalación de la versión del *Adobe Acrobat Reader* que se nos presenta, ya que es una versión muy antigua[4].

Figura A.65: Instalación del *Adobe Acrobat Reader 4.0*.

La siguiente pantalla es la última antes de comenzar el proceso de instalación. Pinchando en el botón *Install*, este proceso arranca.

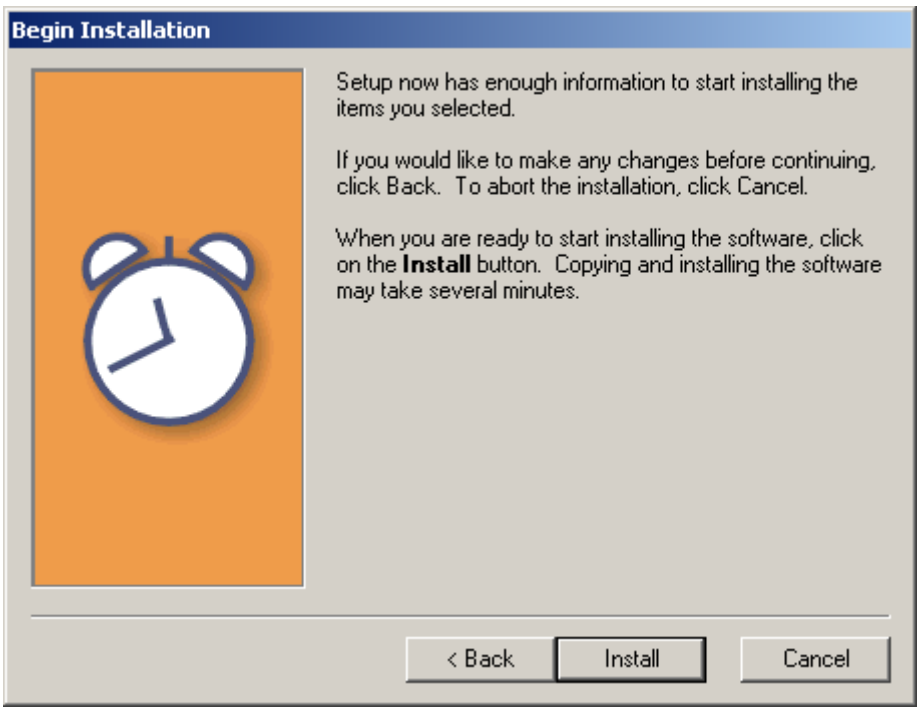


Figura A.66: Comienzo de la instalación.

[4]Generalmente, cualquier PC tiene instalada una versión de este programa mucho más reciente.

La instalación del paquete dura unos minutos, y en la barra de progreso se informa del punto donde ésta se llega.

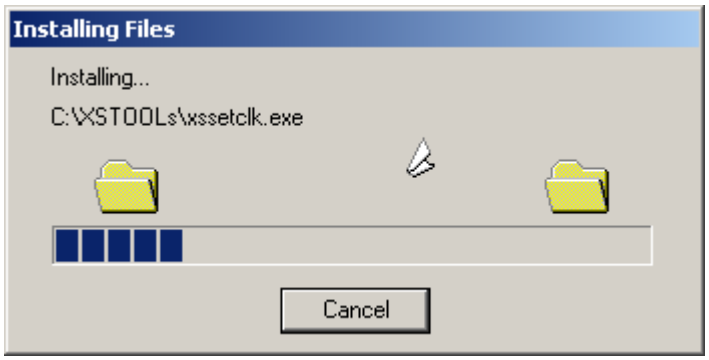


Figura A.67: Progreso de la instalación.

Cuando esta barra de progreso llega al final, se ha completado la instalación. La última pantalla nos informa de ello, y nos sugiere leer el manual de la placa (*XS Board*),para comprender el hardware de la misma.

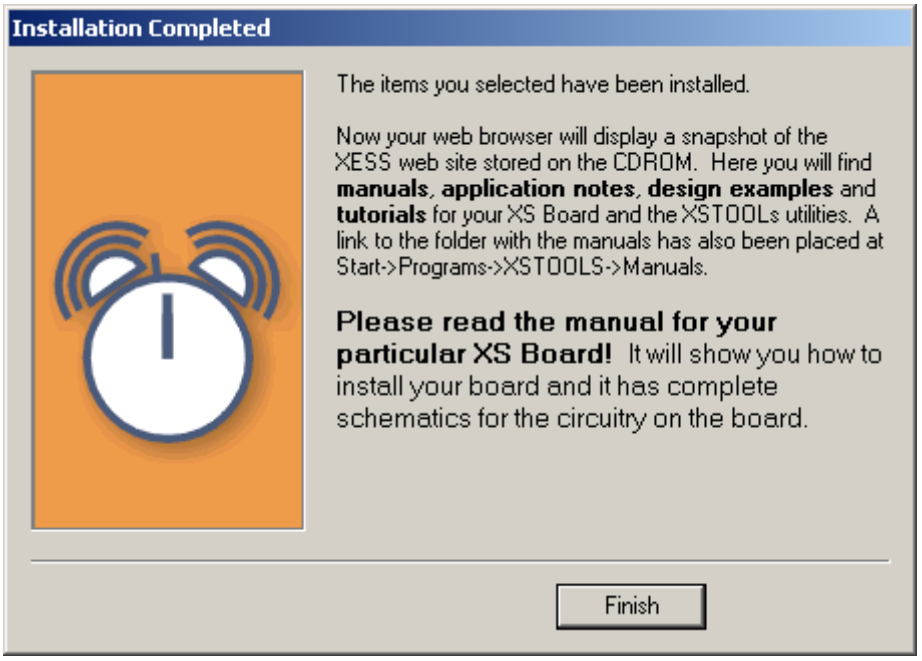


Figura A.68: Fin de la instalación.

Con estos tres paquetes instalados, el usuario puede:

- **realizar cualquier diseño**, con el *WebPACK*,
- **simularlo**, con el *ModelSIM*, y
- **descargarlo a la tarjeta**, con las *XSTools*.

Apéndice B

Tutorial de utilización del software.

En este apéndice, presentamos un pequeño tutorial muy simple para que el usuario se familiarice con el manejo del programa, descubra los recursos de que dispone y aprenda el proceso de diseño, compilación y descarga de los mismos a la *FPGA*. Diseñaremos un sumador/restador de 4 bits cuya salida veremos por el display de 7 segmentos de la tarjeta *XSA 50*. Los 4 bits del primer operando, los 2 bits menos significativos del segundo operando y la operación a realizar se controlarán directamente por el usuario. Los otros dos bits del segundo operando valdrán 0. Para realizar este control, el usuario utilizará utilidad *GXSPort* perteneciente al paquete *XSTools*.

El primer paso de este tutorial es el arranque del programa principal. El software *ISE WEBPACK 6.3* dispone de numerosas utilidades que permiten la creación de diseños, asignación de pines o tiempos de retardo, comprobación de errores, compilado, descarga de diseño a la placa... Todas estas utilidades son accesibles desde el programa principal *Project Navigator*, cuyo *Acceso Directo* se encuentra en la carpeta del *Menú Inicio* que generamos durante la instalación.

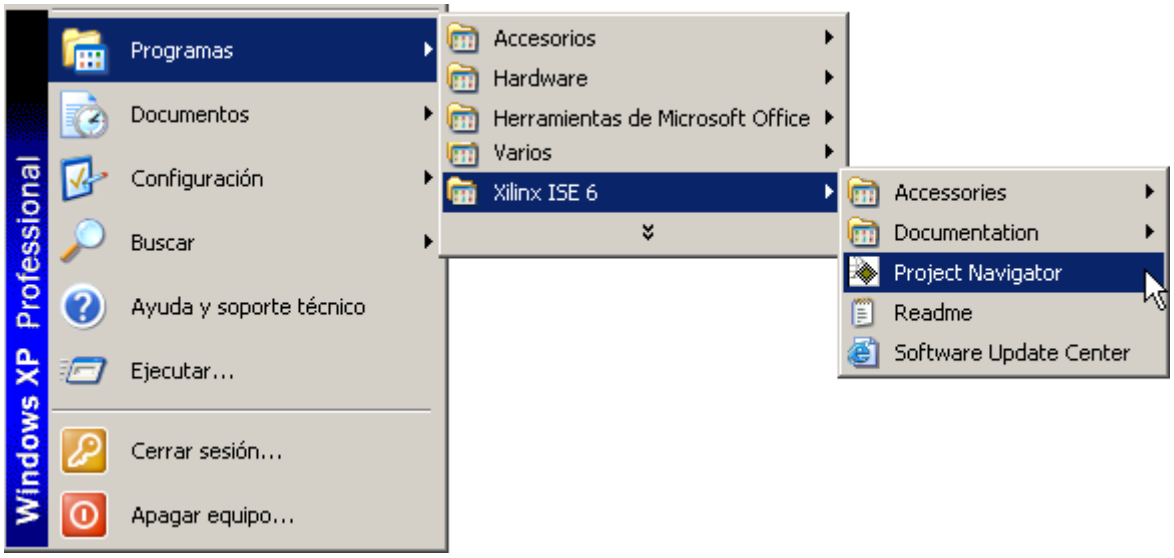


Figura B.1: Acceso a *Project Navigator*.

Tras unos segundos, el programa arranca. La pantalla emergente está dividida en 4 ventanas. La ventana *Sources in Project (Fuentes del Proyecto)* va a contener todos los archivos que formarán parte del *Proyecto*: esquemas, archivos de asignación de pines o cualquier otro tipo de archivos necesarios para la creación del *Proyecto*. La ventana *Processes for Source (Procesos para la Fuente)* contendrá un acceso a todas las utilidades que permiten diseñar, modificar y ejecutar cada una de las *Fuentes del Proyecto*. Además, esta ventana cambia su contenido con cada *Fuente*, porque no se pueden realizar las mismas operaciones sobre todas las *Fuentes*. La ventana inferior informa sobre los eventos que suceden durante las fase de diseño, modificación, compilación y/o descarga del diseño. Tiene varias pestañas donde muestra, por ejemplo, los avisos y los errores que surgen durante la ejecución de cualquiera de los *Procesos* accesibles desde la ventana de *Procesos*. La última ventana, y la más grande, se utiliza para mostrar el contenido de los archivos accesibles desde la ventana de *Fuentes* o desde la de *Procesos*.

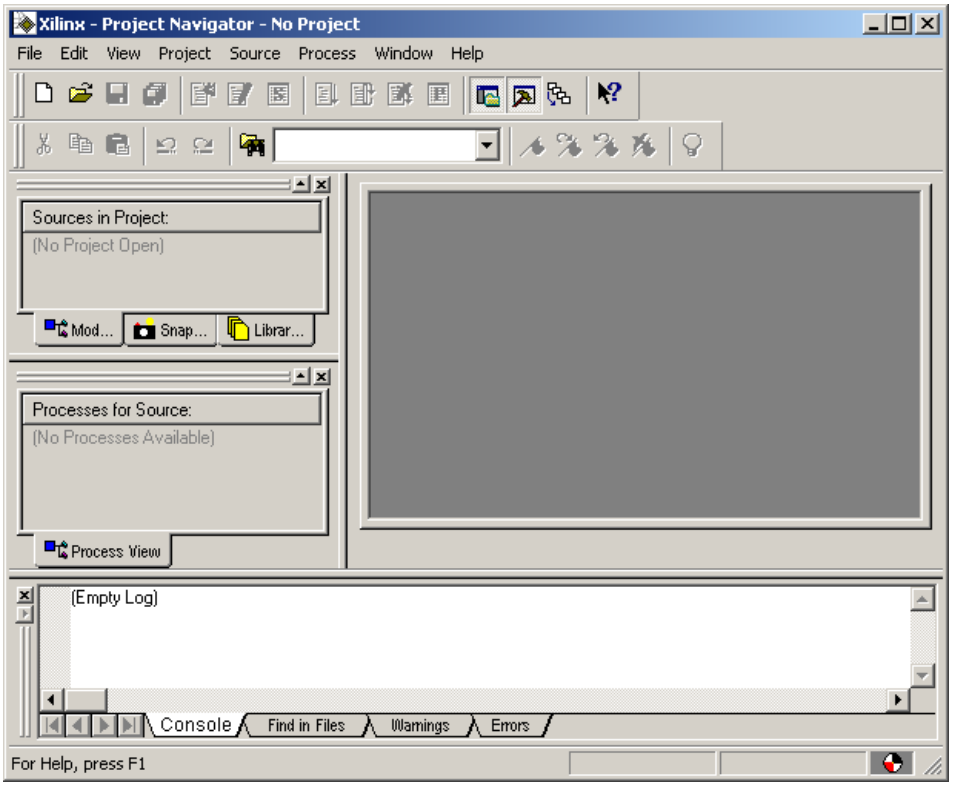


Figura B.2: Pantalla principal de *Project Navigator*.

El proceso de creación de diseños, su modificación, compilación y descarga recibe el nombre de *Proyecto*. En este programa, por tanto, se trabaja con *Proyectos*.

Al arrancar, por defecto, aparece cargado el *Proyecto* abierto la última vez que cerramos el programa. Para abrir uno nuevo, seleccionamos la opción *New Project* del menú *File*.

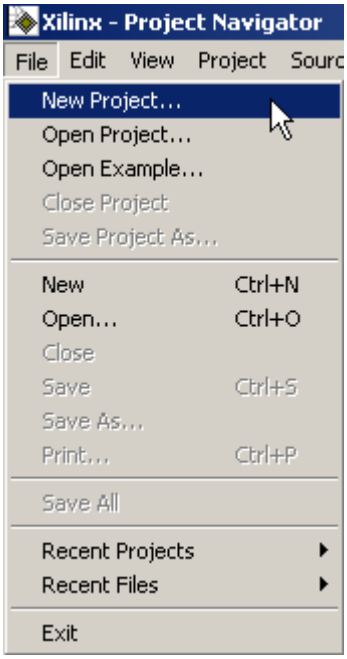


Figura B.3: Nuevo *Proyecto*.

Al crear un nuevo *Proyecto*, la ventana emergente nos permite definir el nombre del *Proyecto* y la carpeta donde lo guardaremos. Debajo, podemos seleccionar el lenguaje con que diseñaremos el bloque principal del *Proyecto*. En nuestro caso, elegimos *Schematics (Diseño mediante Captura Esquemática)*.

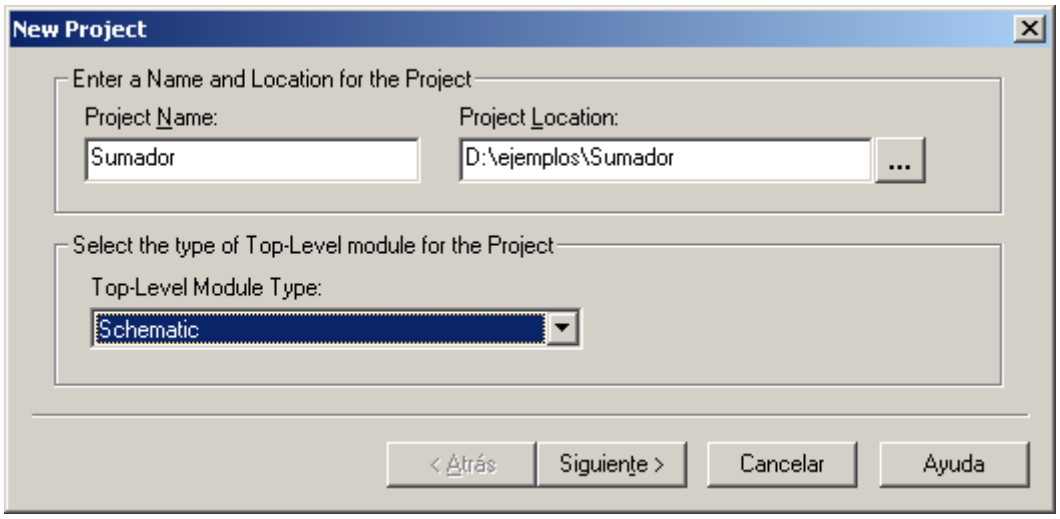


Figura B.4: Propiedades del *Proyecto*.

En la siguiente ventana, seleccionamos el dispositivo sobre el que realizaremos el diseño. En nuestro caso, se trata de la *FPGA Xilinx Spartan II XC2S50 TQ-144 -5*. Al compilar los diseños, éstos se traducen a un lenguaje de programación que se selecciona también en esta ventana. Podemos elegir entre *VHDL* o *Verilog*.

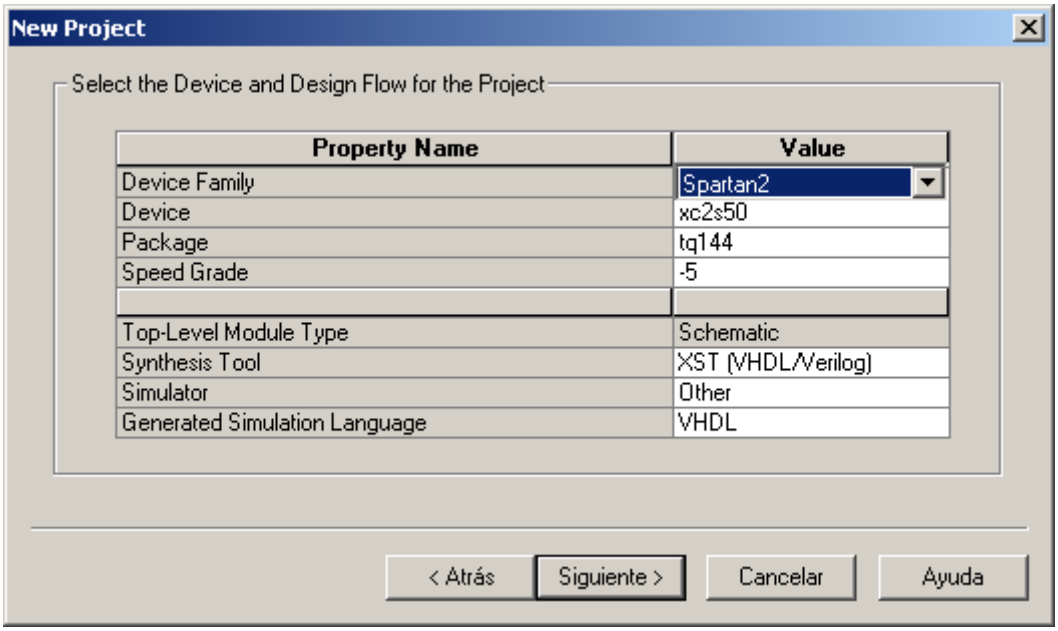


Figura B.5: Selección del Dispositivo.

El siguiente paso nos da la posibilidad de crear nuevas *Fuentes* para el *Proyecto*. Nosotros no las crearemos en este punto.

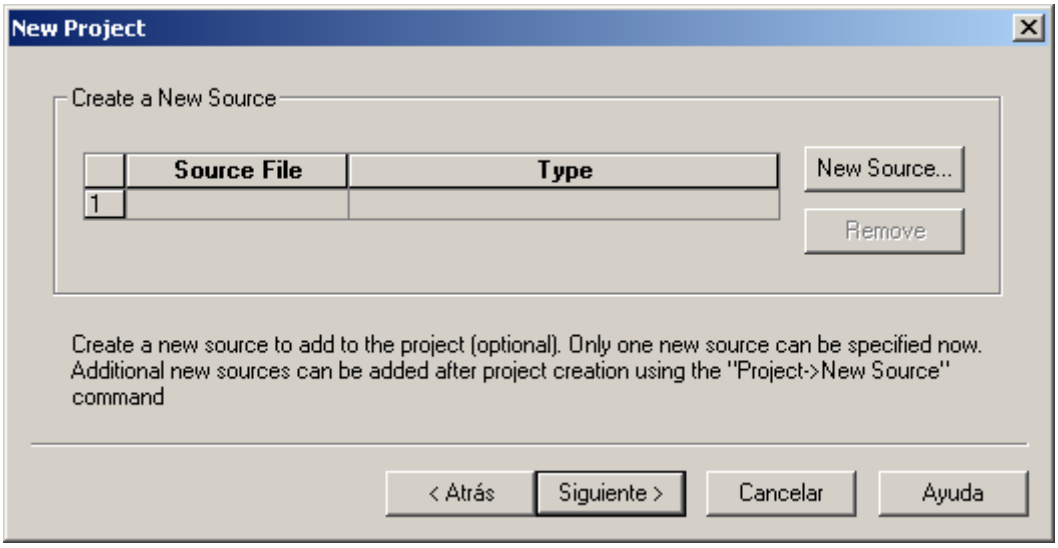


Figura B.6: Nueva *Fuente*.

Además, si ya disponemos de una *Fuente* creada, en este punto es posible añadirla al *Proyecto*. En cualquier caso, siempre es posible añadirla más tarde.

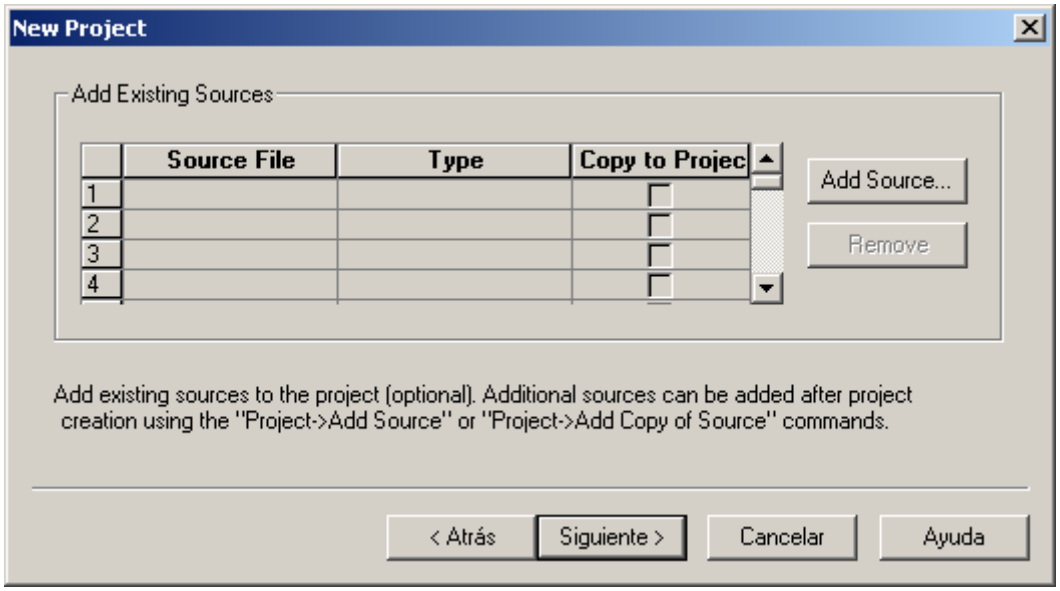


Figura B.7: *Fuente* ya creada.

Antes de finalizar la creación del *Proyecto*, *Project Navigator* presenta una ventana con un resumen de los principales parámetros que utilizará para generar los archivos del *Proyecto*.

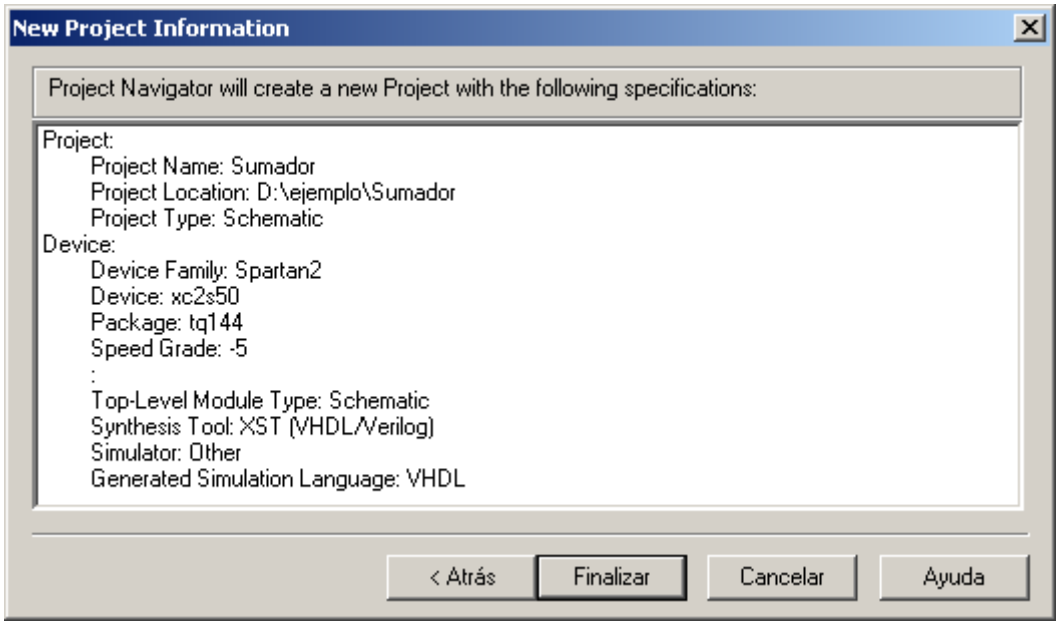
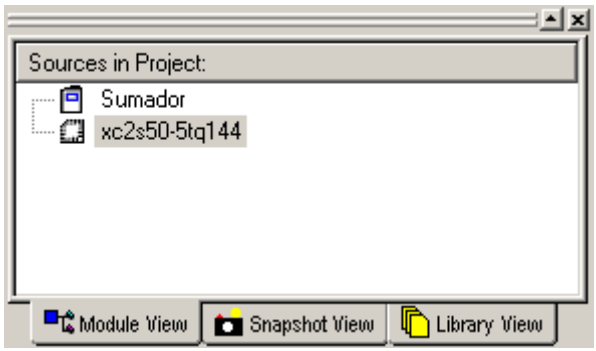
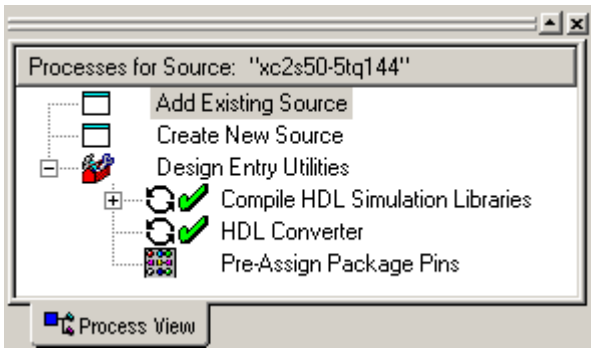


Figura B.8: Resumen de Creación del *Proyecto*.

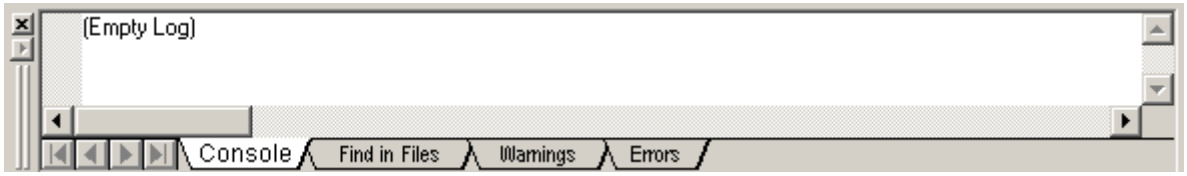
Una vez completada la creación del *Proyecto*, las ventanas de *Fuentes* y *Procesos* se han actualizado con los valores generados durante la creación.



(a) Ventana de *Fuentes*.



(b) Ventana de *Procesos*.



(c) Ventana de *Información*.

Figura B.9: Ventanas en el *Project Navigator*.

Entonces, para crear una nueva *Fuente*, pinchamos con el botón derecho del ratón sobre cualquiera de las *Fuentes* de esta ventana y, del menú emergente, seleccionamos la opción *New Source*.

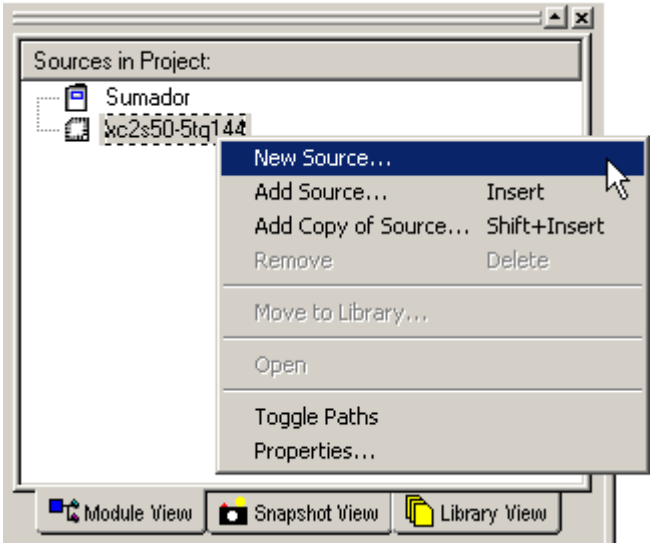


Figura B.10: Nueva *Fuente*.

Para crear una nueva *Fuente*, es necesario seleccionar el tipo de *Fuente* que deseamos generar. Nosotros seleccionaremos *Schematics* para realizar el diseño mediante *Captura Esquemática*. Completamos el nombre y la ruta de localización del archivo, y lo añadimos al *Proyecto*.

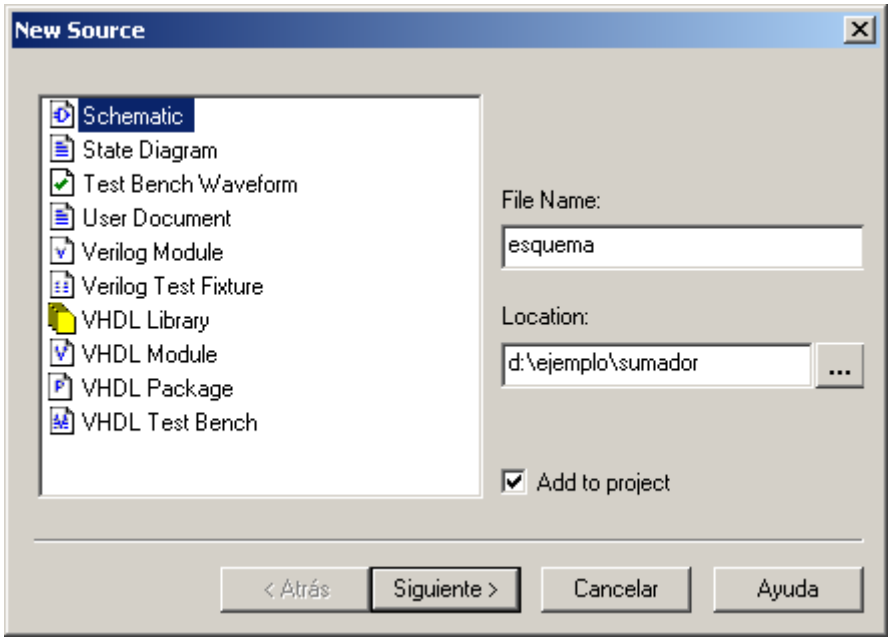


Figura B.11: Tipo de Nueva *Fuente*.

Antes de finalizar la creación de la *Fuente*, se presenta una pantalla resumen con la información disponible para la creación de la fuente.

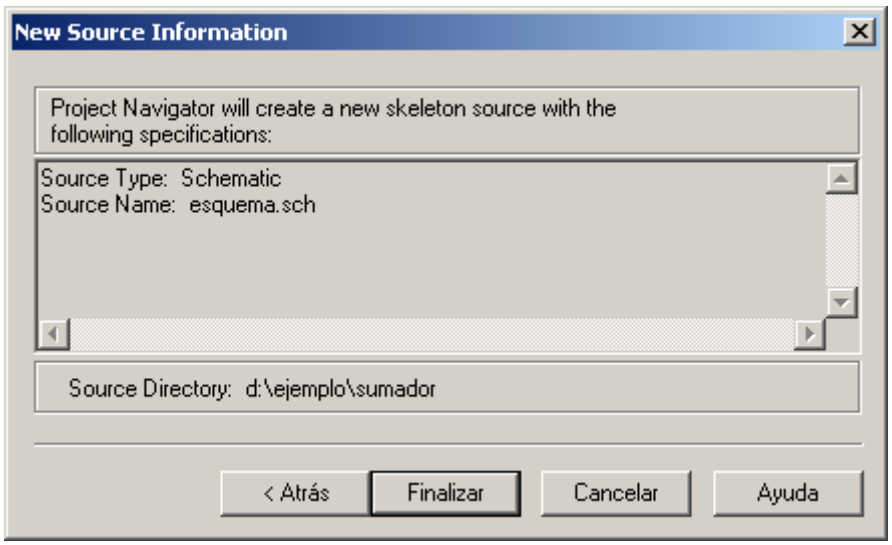


Figura B.12: Resumen de Nueva *Fuente*.

En la ventana de *Fuentes*, aparece la nueva *Fuente* que hemos incluido, con el archivo que la implementa entre paréntesis.

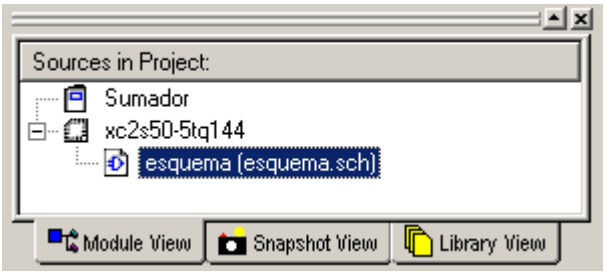
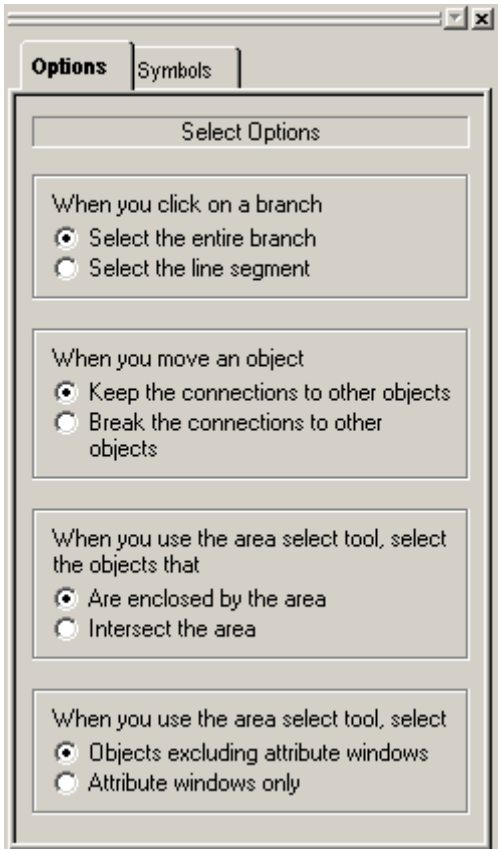
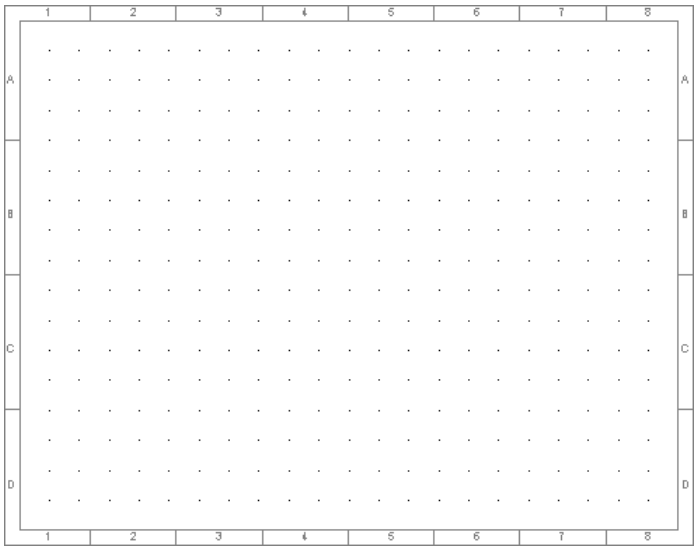


Figura B.13: Ventana de *Fuentes*.

Al pinchar dos veces sobre él, se abre el programa *Xilinx ECS Schematic Editor* que permite realizar los diseños mediante *Captura esquemática*. El entorno de este programa está formado, básicamente, por dos ventanas: venta de *Opciones* y la ventana de *Diseño*.



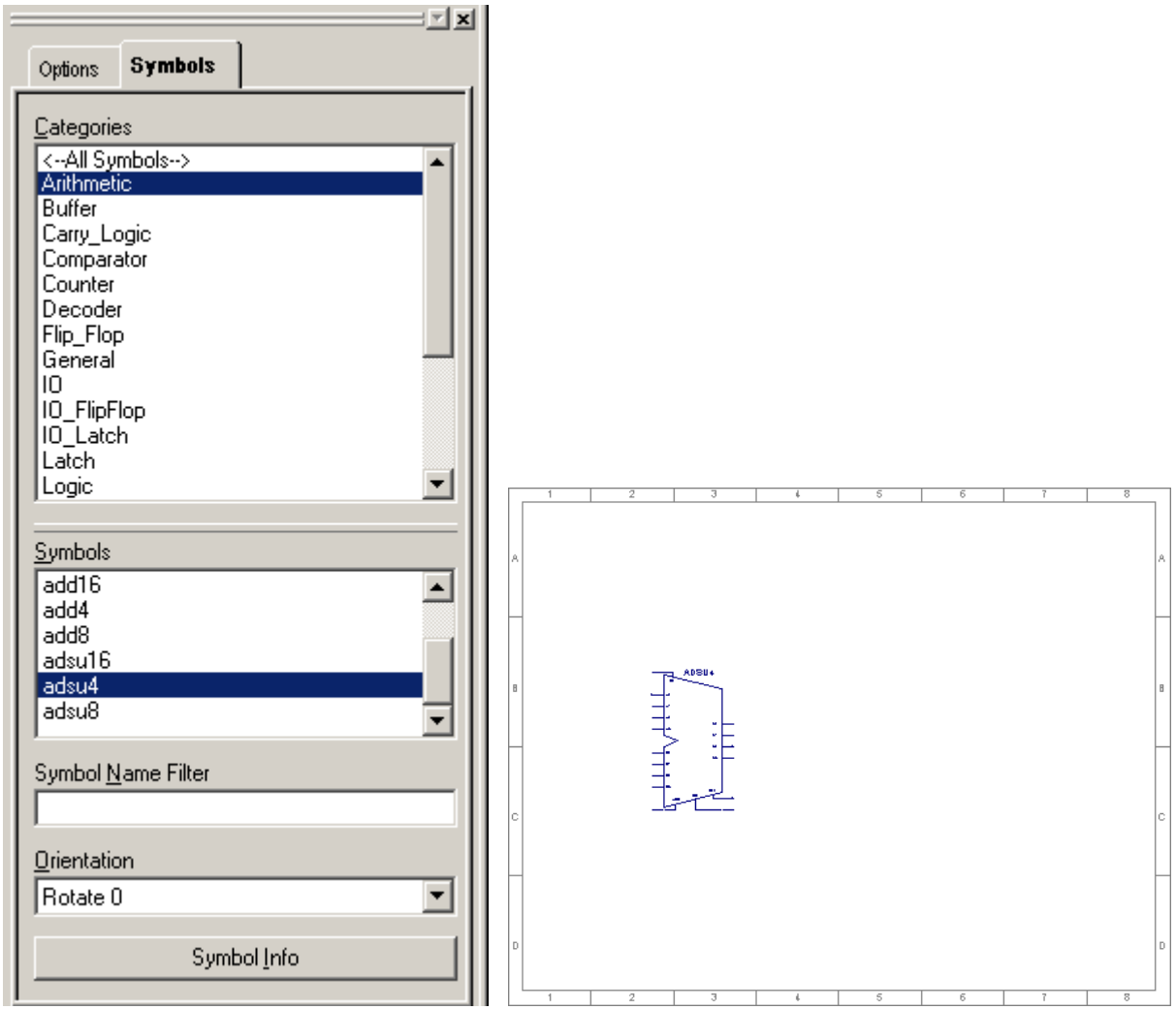
(a) Ventana de *Opciones y Símbolos*.



(b) Ventana de *Diseño*.

Figura B.14: Ventanas en el *ECS*.

La pestaña de *Opciones* se utiliza para seleccionar el tipo de función que se va a realizar sobre la ventana de *Diseño*. La pestaña de *Símbolos* permite seleccionar los elementos para situarlos en la ventana de *Diseño*; además, podemos elegir su orientación y obtener información sobre el elemento seleccionado. Los elementos están agrupados en categorías. Para llevarlo a la ventana de *Diseño*, basta seleccionar el símbolo deseado, y arrastrarlo hasta dicha ventana.



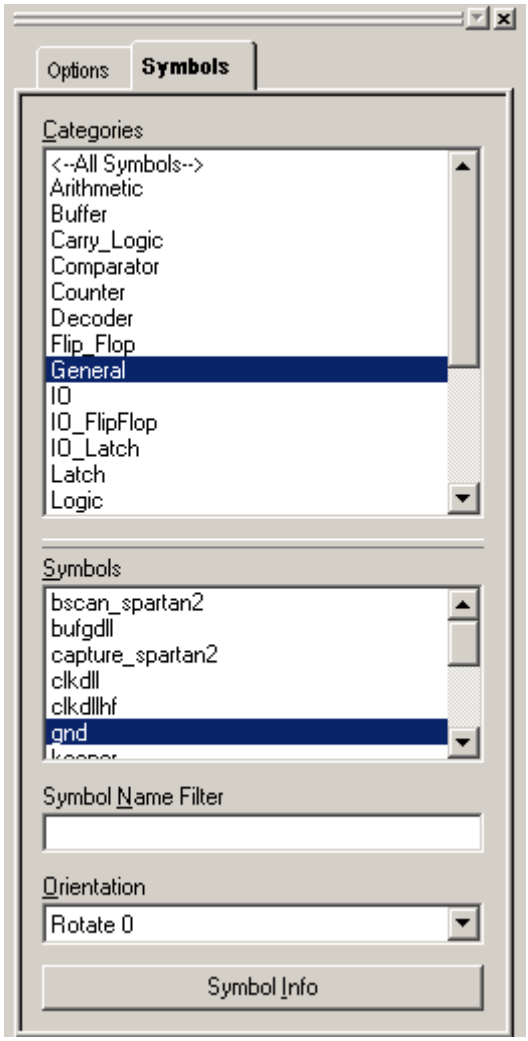
(a) Ventana de *Opciones y Símbolos*.

(b) Ventana de *Diseño*.

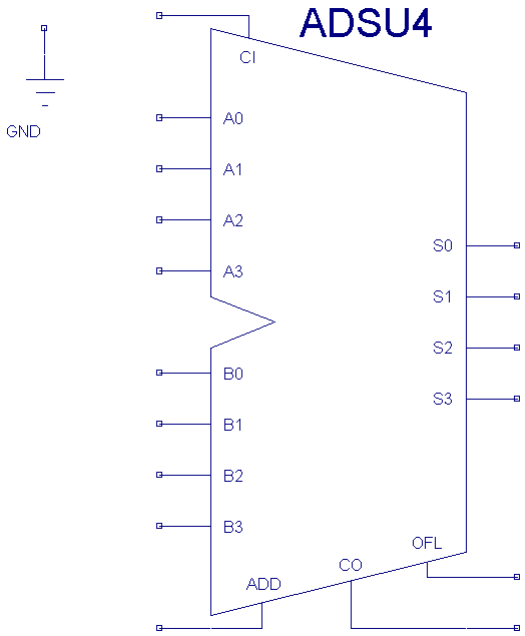
Figura B.15: Ventanas en el *ECS*.

Para comenzar nuestro diseño, seleccionaremos el *Sumador/Restador adsu4* accesible desde la Categoría *Arithmetics*.

Siguiendo con nuestro diseño, de la misma forma que situamos el sumador, vamos a colocar una *Masa* junto a él, para conectarla a la entrada de *acarreo*, de forma que no haya *acarreo* al realizar las operaciones.



(a) Ventana de *Opciones y Símbolos*.



(b) Ventana de *Diseño*.

Figura B.16: Ventanas en el *ECS*.

Tal y como está situada y orientada la *Masa* que acabamos de colocar, es posible que nos moleste a la hora de colocar otros elementos en el diseño. Quizá sea interesante rotarla 180 grados, para facilitar la conexión desde la entrada de *acarreo* del sumador. Pinchando sobre la *Masa* con el segundo botón del ratón, se despliega el menú que permite, entre otras cosas, realizar este tipo de movimientos y obtener información del símbolo.

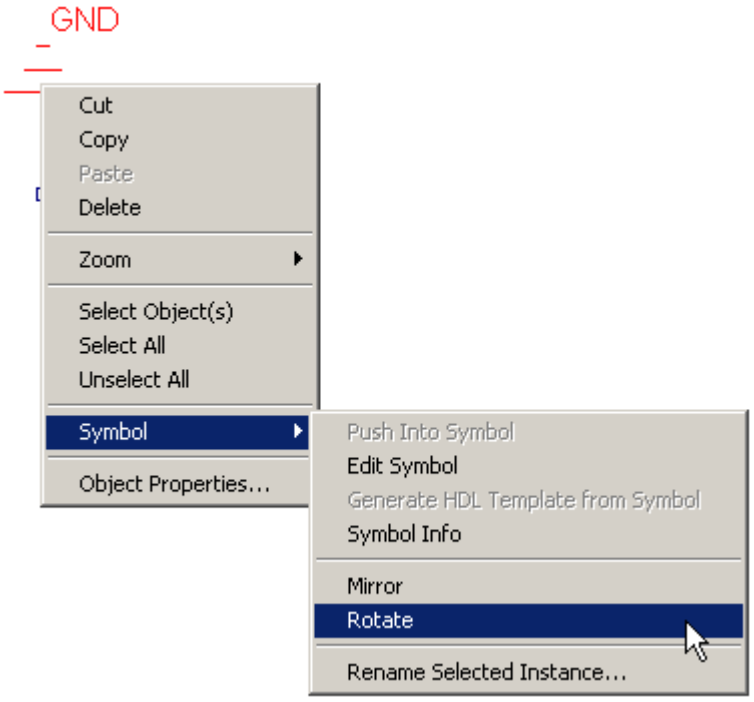


Figura B.17: Menú desplegable del símbolo.

Otra opción muy útil es *Editar el Símbolo*. Como veremos más adelante, podemos utilizar un bloque diseñado por nosotros como si fuese un bloque de los que provee el programa. Para ello, hay que crear un símbolo (ver página 228) y arrastrarlo hasta la ventana de *Diseño* desde la Pestaña de *Símbolos* como hemos visto. Sin embargo, no siempre el aspecto del símbolo es como queremos. Entonces, mediante la opción *Editar el Símbolo* podemos modificar ese aspecto para adecuarlo a nuestro diseño.

Para conectar dos elementos, necesitamos un cable. De la misma forma que otros programas de diseño, *ECS* dispone de una utilidad para conectar elementos con un cable. Como vemos en la siguiente figura, permite la conexión *manual* o mediante *auto-rutado*. La técnica de auto-rutado consiste en que, señalando los puntos origen y final, el propio programa realiza automáticamente el rutado por todo el diseño para conectar los dos extremos.

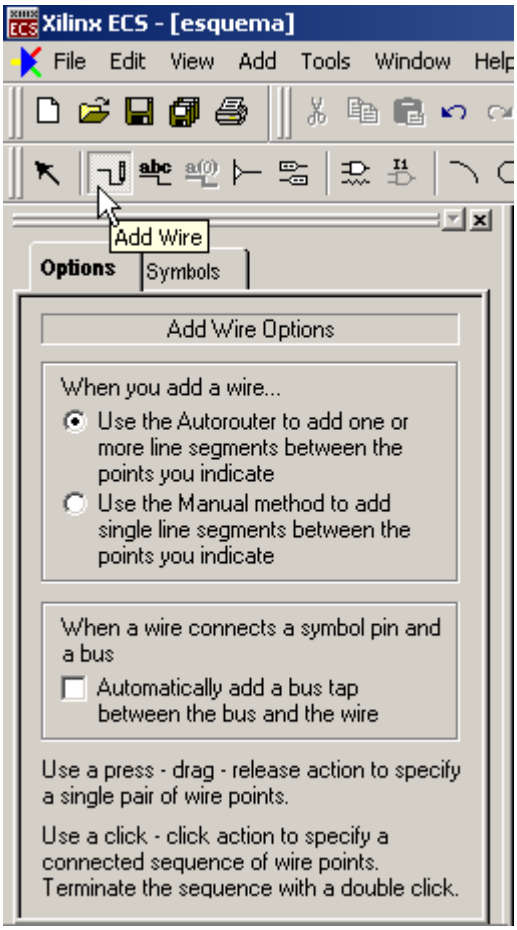


Figura B.18: Menú de cableado.

Por nuestra experiencia, preferimos el auto-rutado, ya que facilita mucho el conexionado. Además, si el rutado no nos gusta, siempre es posible mover los cables y colocarlos a nuestro gusto, sin perder la conexión.

Para hacer la conexión, basta con picar con el botón derecho del ratón sobre el extremo origen y picar, seguidamente, sobre el extremo final. Entonces, el programa de diseño creará un cable para conectar ambos extremos.

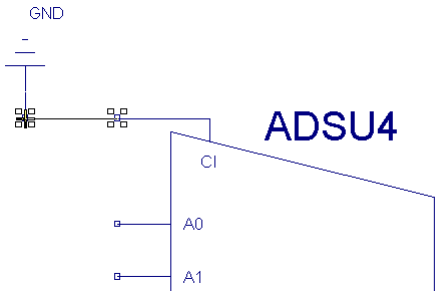
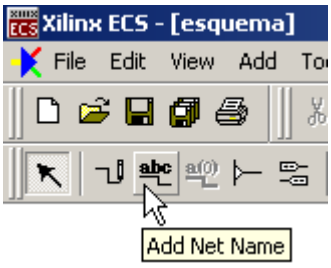
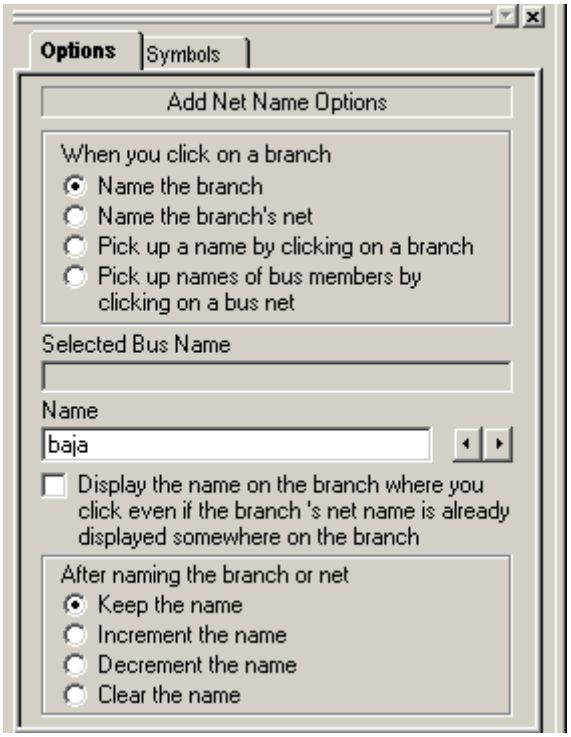


Figura B.19: Conexionado.

Sin embargo, una técnica muy cómoda para hacer conexiones es el uso de *Etiquetas*. Consiste en dar un nombre (*Etiqueta*) a un cable o punto de conexión, de forma que, si nombramos otro punto del circuito u otro cable con esa *Etiqueta*, hacemos un cortocircuito entre ambos, es decir, los hemos conectado. Para poner una etiqueta, utilizamos el botón de la figura. En el menú, disponemos de una ventana donde escribimos el nombre de la etiqueta.



(a) Icono de acceso.



(b) Menú.

Figura B.20: Uso de *Etiquetas*.

Una vez que hemos escrito el nombre, movemos el cursor hacia el cable que queremos *etiquetar*. Cuando el cable se selecciona, pinchamos sobre él, y el cable queda *etiquetado*.

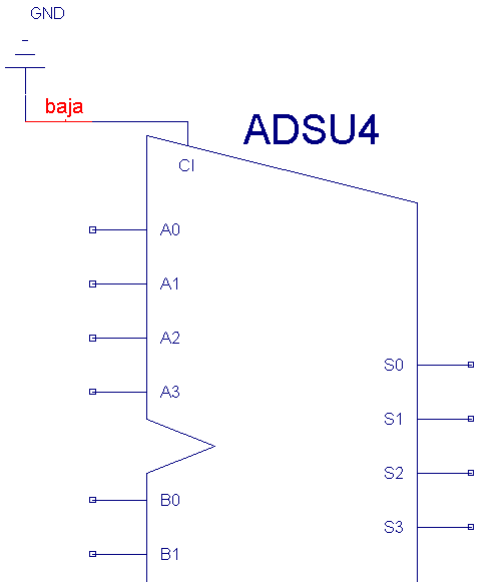


Figura B.21: Etiquetado de cables.

Como vimos en el comienzo de este tutorial, los dos bits más significativos del *Operando B* valdrán 0. Entonces, para conectar esa *Masa* con estos dos bits, basta con *etiquetar* esas dos entradas con el mismo nombre con que hemos *etiquetado* el cable conectado a *Masa*.

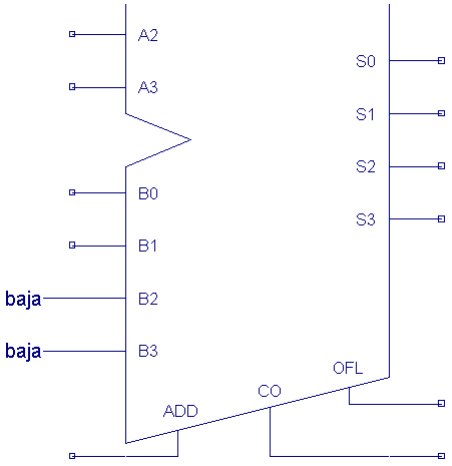


Figura B.22: Etiquetado de cables.

Para conectar un punto del esquema a un pin de entrada o salida de la tarjeta, es necesario colocar un *Buffer*. Para entradas y salidas normales, se utilizan *Ibufs y Obufs* respectivamente. Cuando la entrada es una señal de reloj, como la que entra por el *pin 88* o por el *pin 91*, es necesario utilizar un *(I/O)Bufg*.

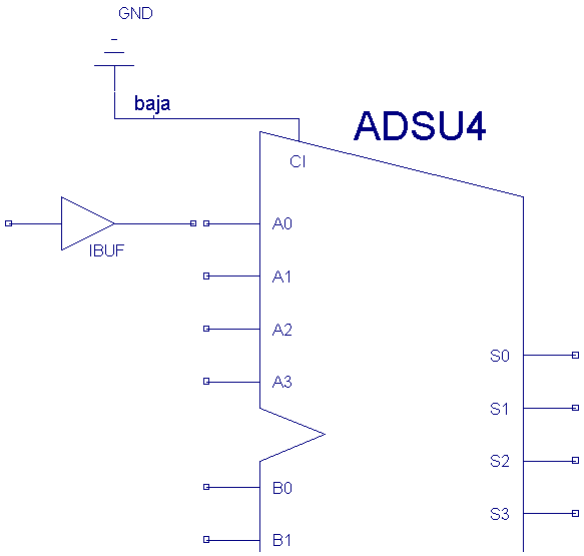
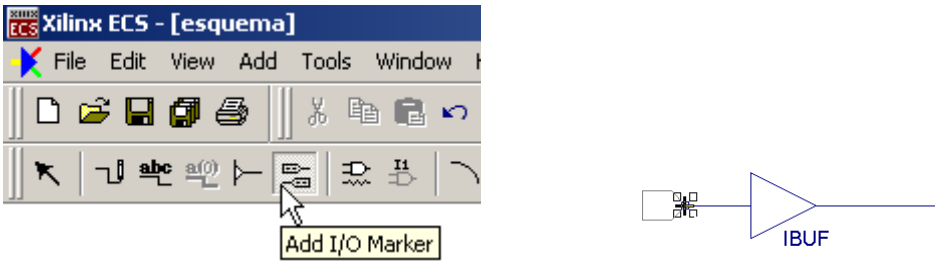


Figura B.23: *Buffer* de entrada.

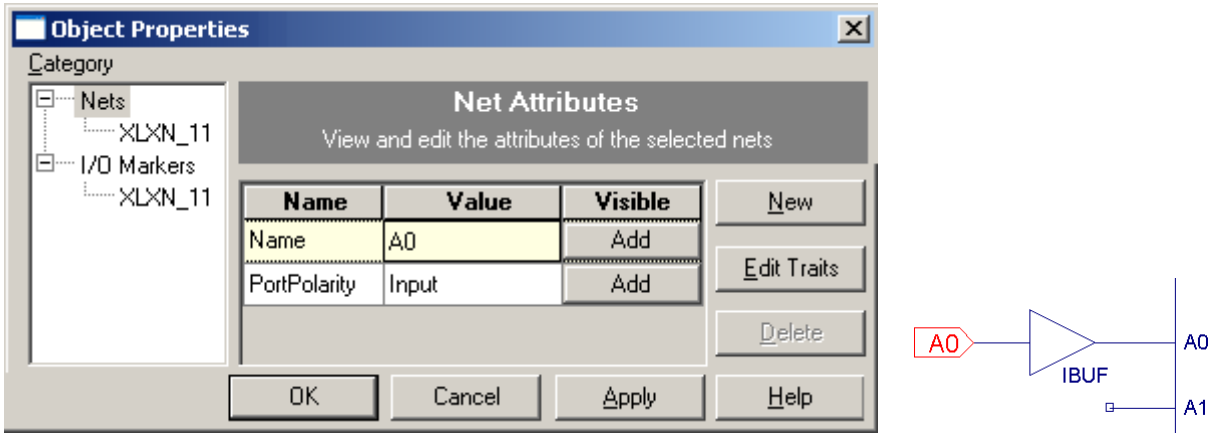
Los *I/O Markers* son los puntos de conexión del bloque actual con bloques de nivel superior o inferior, o con los pines de entrada y salida de la tarjeta. Cuando realizamos el diseño de un bloque y creamos su símbolo, todos los *I/O Markers* serán las entradas y salidas que tendrá el nuevo símbolo generado. Para el bloque de nivel superior, y solo en este bloque, los *I/O Markers* serán los accesos a los pines de la tarjeta, como definiremos en el archivo *.ucf* posteriormente. Para colocar un *I/O Marker*, pinchamos en el icono que se muestra en la figura y, del mismo modo que las *Etiquetas*, lo llevamos al circuito y lo conectamos al cable que deseemos.



(a) Icono de acceso. (b) Conexión.

Figura B.24: *I/O Markers*.

Por defecto, el *I/O Marker* añadido tiene un nombre similar a *XLXN_xx*, donde *xx* es un número. Para cambiar este nombre, pinchamos dos veces sobre el *I/O Marker*. Entonces, se nos abrirá un menú como el de la figura. En él, podemos escribir el nombre que queremos para nuestro *I/O Marker* junto a la propiedad *Name* de la caja. El botón *Edit New Traits* permite cambiar las propiedades del *I/O Marker* y asignarle pines de la tarjeta. Sin embargo, los pines definidos de esta forma no son siempre bien interpretados por el programa de compilación. Así que no utilizaremos esta posibilidad para definir los pines de conexión.



(a) Menú de Opciones. (b) Nuevo nombre.

Figura B.25: *I/O Markers*.

Siguiendo el mismo modo de operación, completamos el esquema como sigue:

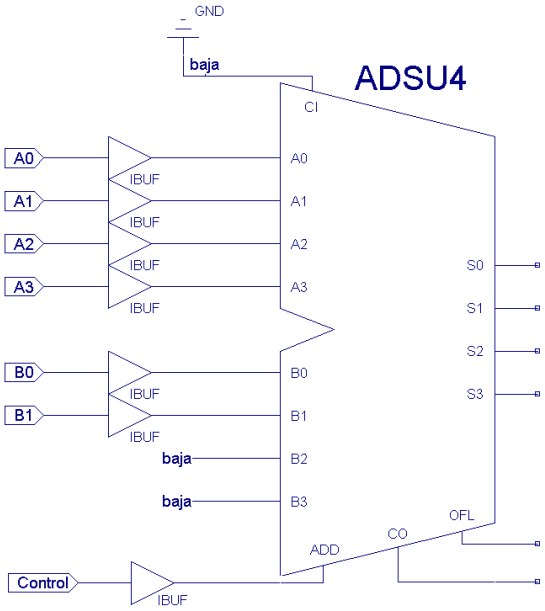


Figura B.26: Esquema general.

Una vez que tenemos este diseño finalizado, vamos a ver como se introduce una nuevo bloque al esquema a partir de un diseño ya hecho. Para ello, volvemos a la pantalla principal de *Project Navigator*. Una vez allí, de la misma forma que antes creamos una *Nueva Fuente*, ahora añadimos una, mediante la opción *Add Source* del menú desplegable.

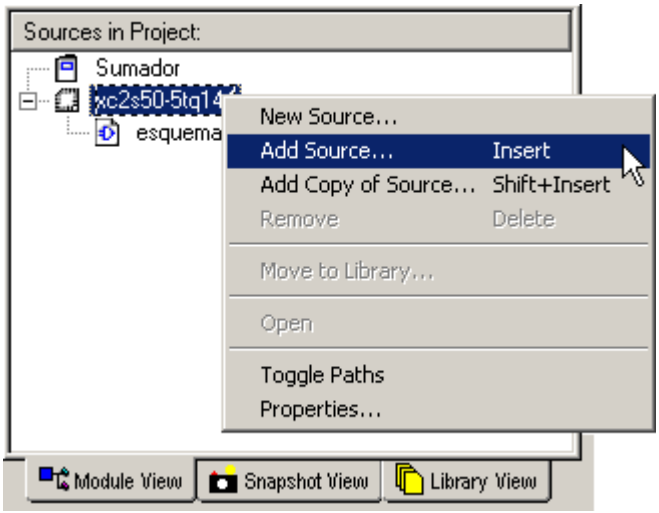


Figura B.27: Añadir *Fuente*.

Como vamos a añadir una *Fuente* ya creada, el primer paso es buscar el archivo que la implementa. Según el tipo de fuente que añadamos, el archivo tendrá una extensión u otra. Por ejemplo, una captura esquemática tendrá extensión *.sch*, mientras que una archivo en escrito en lenguaje *VHDL* tendrá extensión *.vhd*. En nuestro caso, añadiremos el archivo *decode.vhd* que implementa un decodificador de 4 bits de entrada a 8 líneas de salida que conectaremos al display de 7 segmentos del que disponemos en la tarjeta.

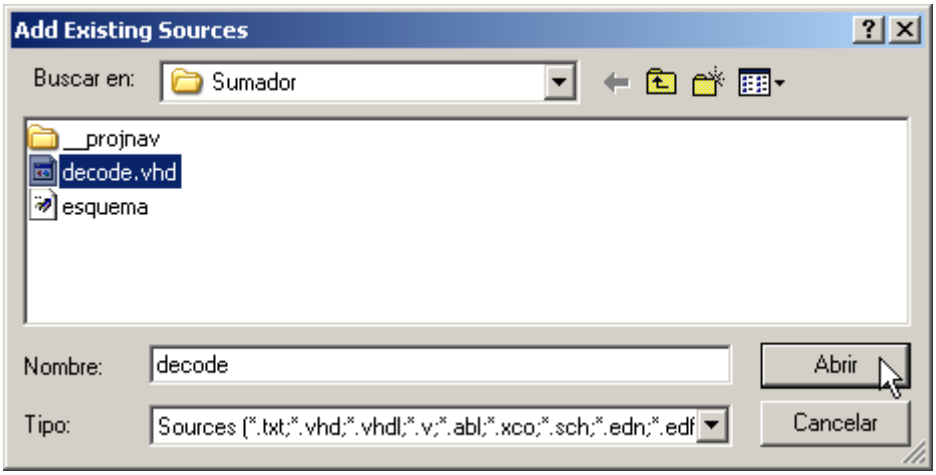


Figura B.28: Selección *Fuente*.

Para hacer simulaciones de los diseños, se utilizan *Bancos de Pruebas* que se añaden de la misma forma. Por eso, en este punto, debemos informar al programa sobre si la *Fuente* que hemos añadido es un *Banco de Pruebas* o un archivo de diseño.

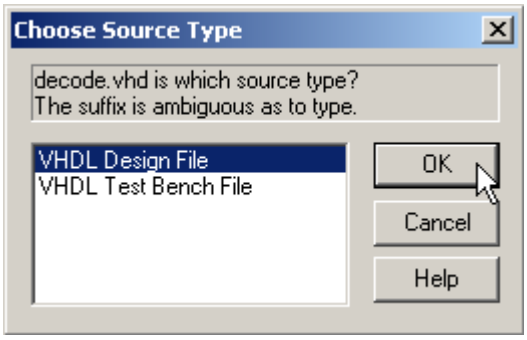
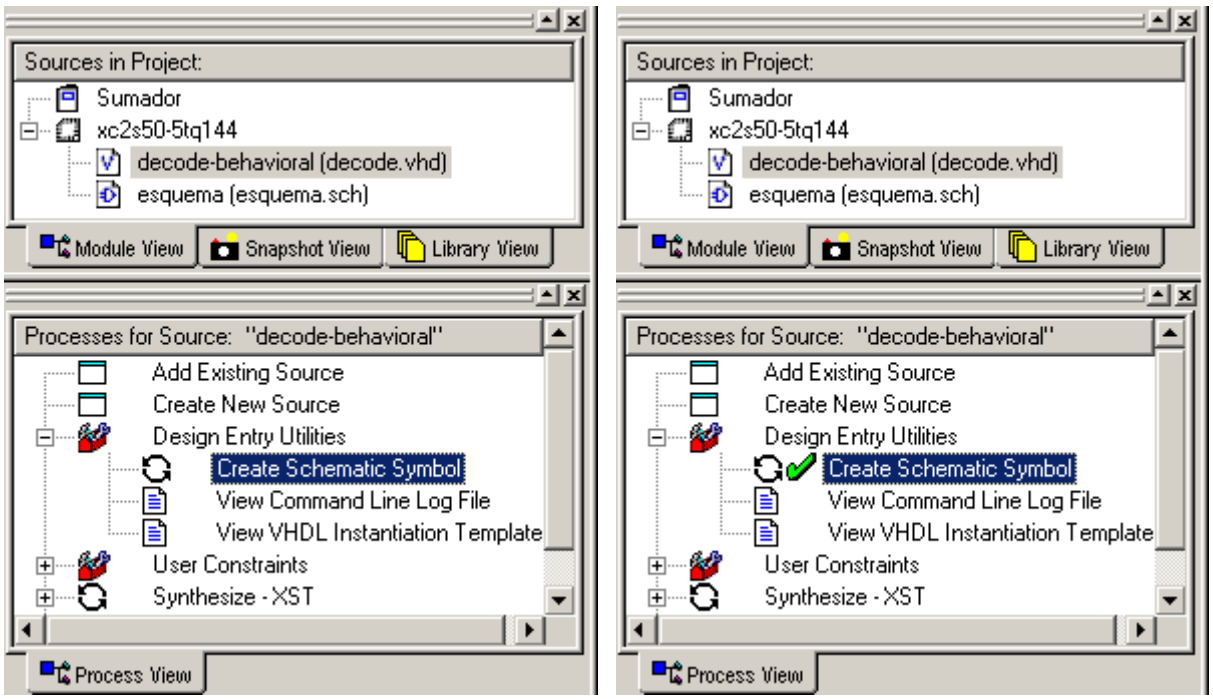


Figura B.29: Tipo de *Fuente*.

Al presionar *OK*, vemos, en la *Ventana de Fuentes*, que esta *Fuente* se ha añadido al *Proyecto*. Para agregar este diseño a nuestro esquema, es necesario crear el símbolo del mismo. Para ello, debemos seleccionar la *Fuente* en la Ventana de *Fuentes*, y en la Ventana de *Procesos*, aparecerán todas las posibilidades de trabajo que podemos realizar con esa*Fuente*. Nosotros vamos a *Crear el Símbolo* para poder incluirlo en el esquema del diseño.

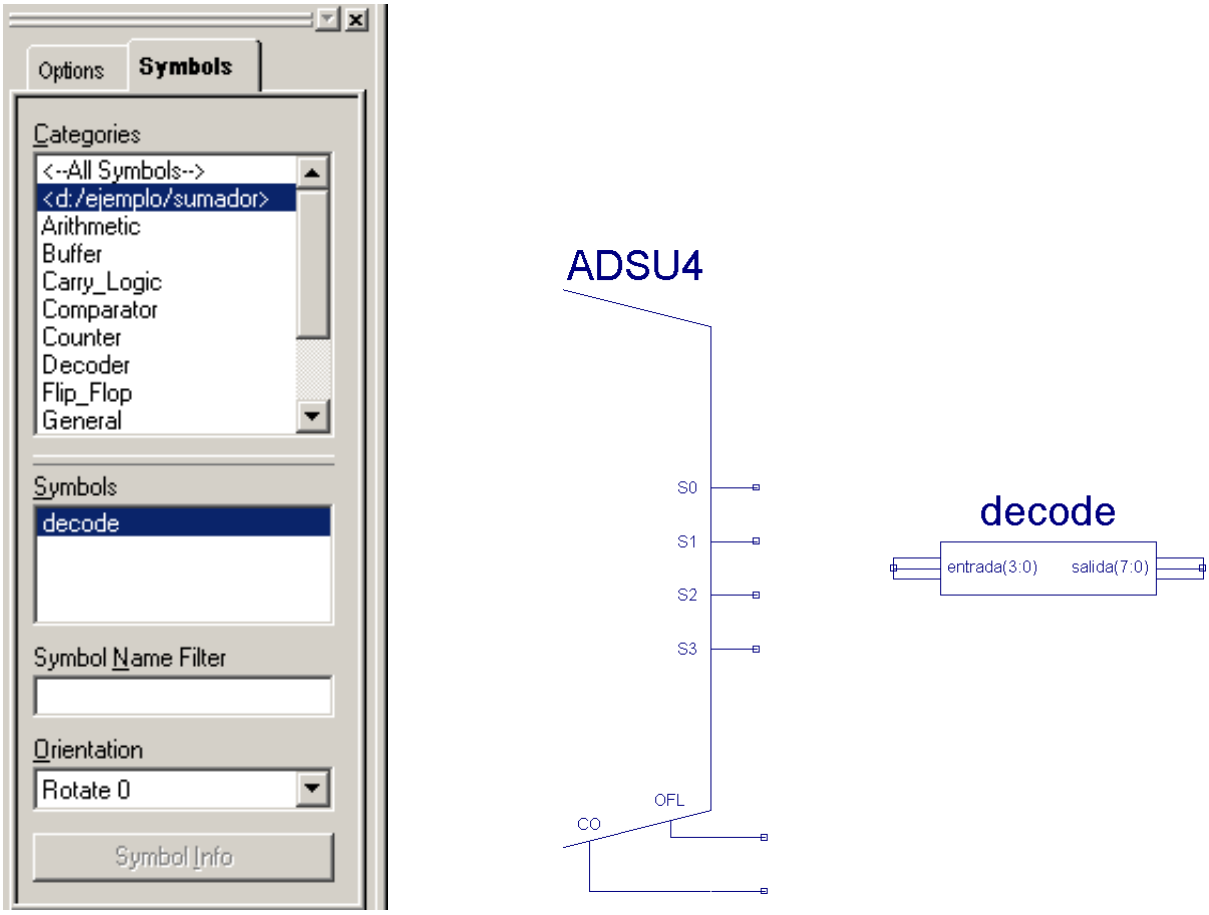


(a) Creación del *Símbolo*.

(b) *Símbolo* creado.

Figura B.30: *Símbolo*.

Una vez que hemos creado el *Símbolo*, para añadirlo al esquema se procede de la misma forma que un bloque normal. En el menú de *Símbolos*, aparecerá una nueva categoría que es el directorio donde se encuentran la *Fuente* y el *Símbolo* creado. En esa categoría, podemos seleccionar el *Símbolo* y arrastrarlo hasta el esquema.



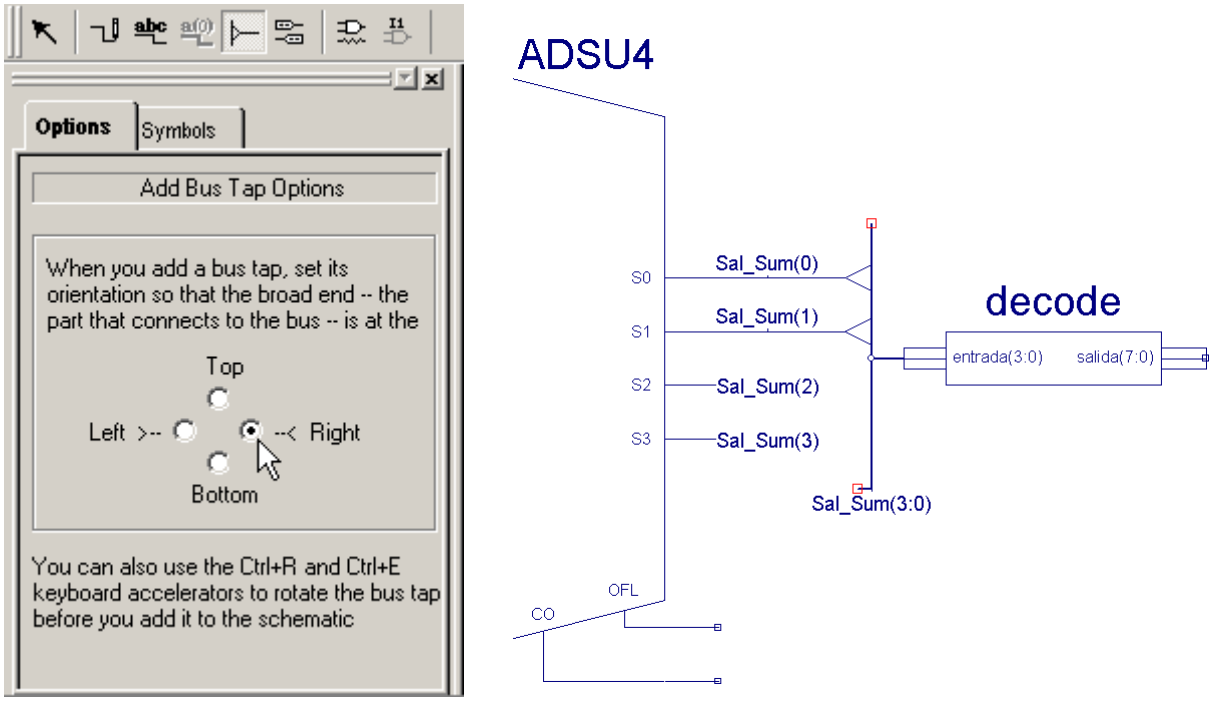
(a) Menú de *Símbolos*.

(b) Esquema de Diseño.

Figura B.31: *Símbolos*.

La situación con la que nos encontramos en este momento es que nosotros disponemos de 4 líneas independientes procedentes del *Sumador* y. sin embargo, tenemos que introducirla en el nuevo bloque *Decode* como bus de 4 bits de anchura.

Para resolver esta situación, se utilizan los *Bus Tap*. Un *Bus Tap* es un elemento que permite acceder a una línea determinada de un bus. Para ello, los colocamos de la forma en que se muestra en la figura, asegurándonos de que quedan conectados tanto a la línea como al bus. Es necesario conocer el nombre del bus para *etiquetar* correctamente la línea. Una costumbre muy aconsejable es *etiquetar* primero el bus y, posteriormente, *etiquetar* las líneas que acceden a él. Sin embargo, la técnica del *etiquetado* que hemos visto facilita mucho el trabajo con buses, ya que la utilización de los *Bus Tap* resulta muy incómoda en diseños más grandes que el que presentamos en este tutorial.



(a) Menú de los *Bus Tap*.

(b) Esquema de Diseño.

Figura B.32: *Bus Tap*.

Los buses se etiquetan con un nombre seguido de los números que se asignan a cada línea entres paréntesis. Lo recomendable es asignar números consecutivos a las líneas del bus. Entonces, para facilitar la tarea de asignación, se utiliza el operando *:* que toma todos los números entre los dos escritos. Así, por ejemplo, el bus *entrada(4:0)* es es un bus de 5 líneas que son accesibles mediante las *Etiquetas entrada(4), entrada(3), entrada(2), entrada(1) y entrada(0).*

Para finalizar el diseño, solamente queda poner el *I/O Marker* de salida. Como vimos, antes de él, es necesario poner un *Buffer* de salida. Sin embargo, vemos que el *Buffer* es de una sola línea, mientras que la salida es un bus de 8 líneas. Pero no es necesario poner 8 *Obufs*, sino que se puede modificar el diseño del *Obuf*, para implementar esos 8 *Obufs* en paralelo. Para ello, basta con renombrar el elemento como mostramos en la figura.

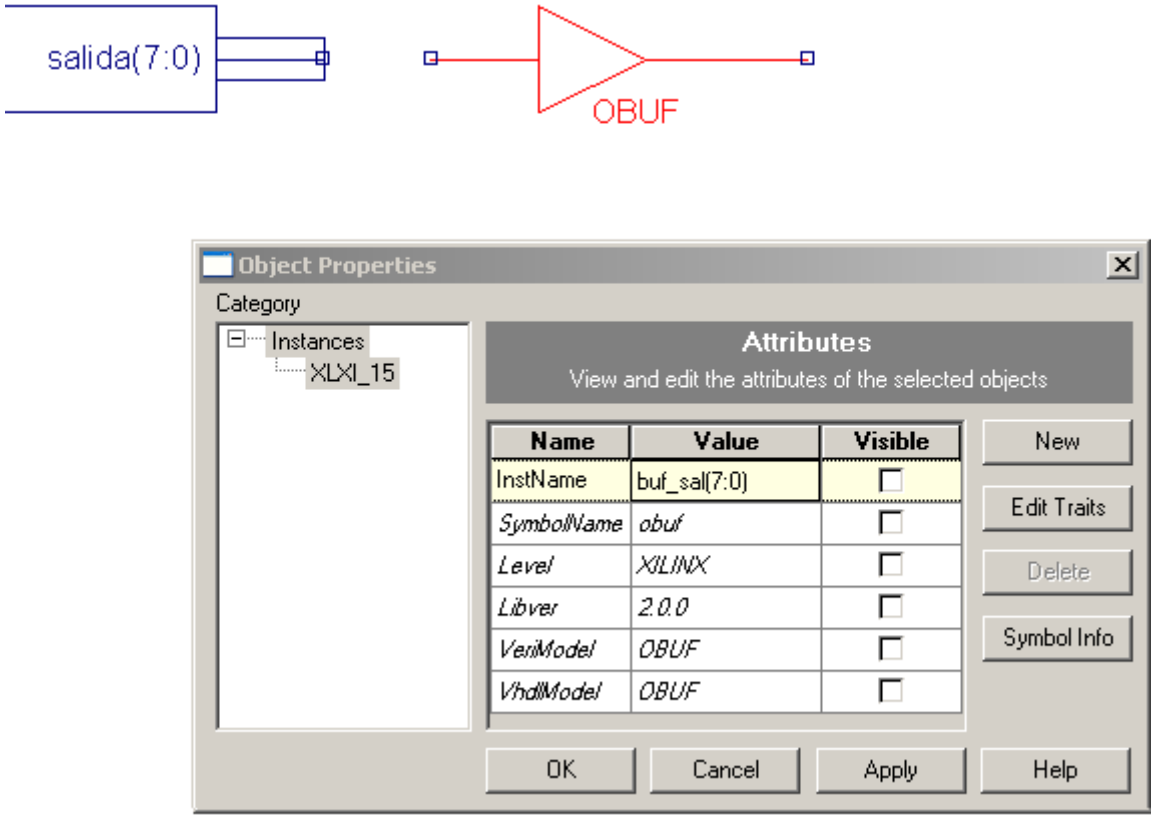


Figura B.33: *Obuf* de 8 bits.

Para completar el diseño, añadimos el *I/O Marker* que, automáticamente, vendrá nombrado como un bus de 8 bits.

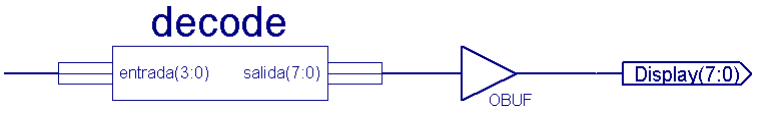


Figura B.34: Esquema de Diseño.

El diseño completo se muestra en la figura:

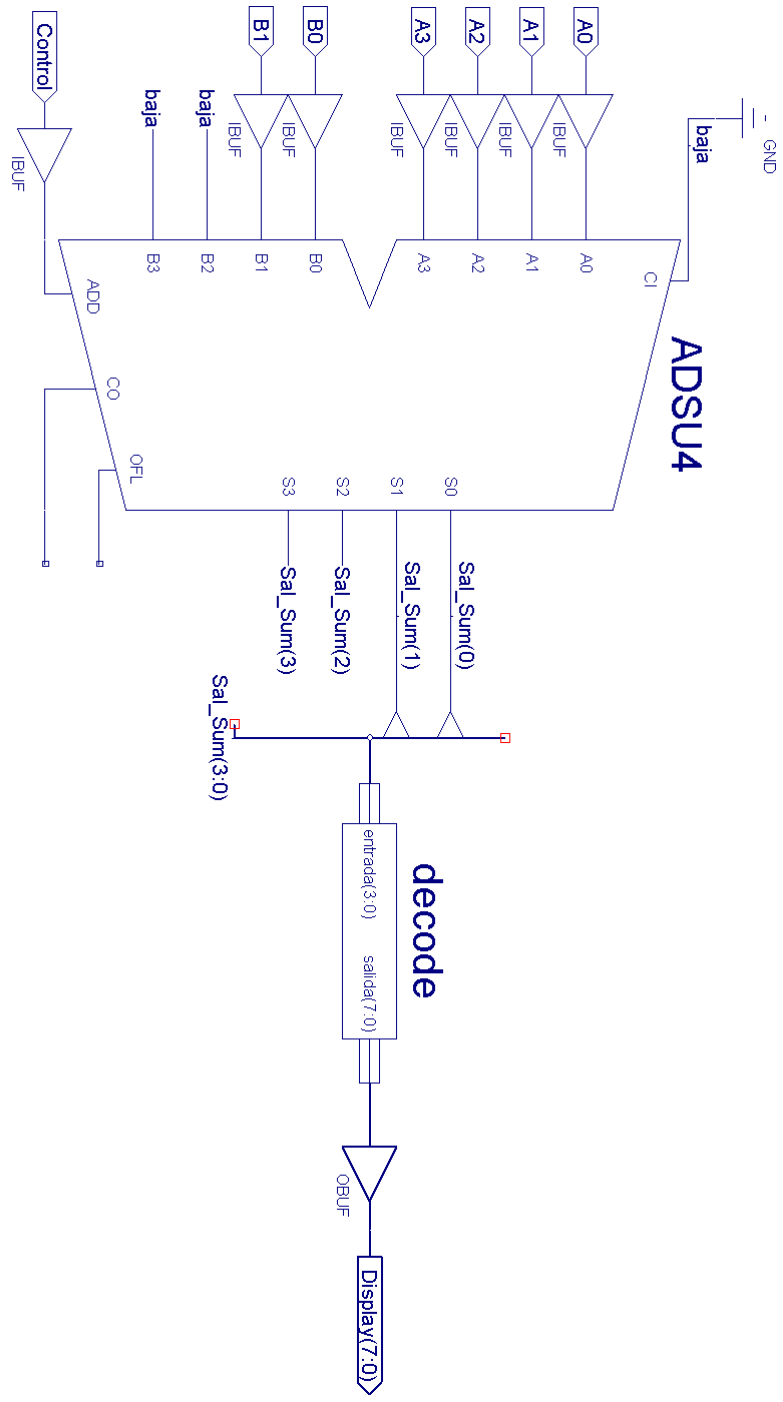
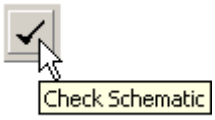
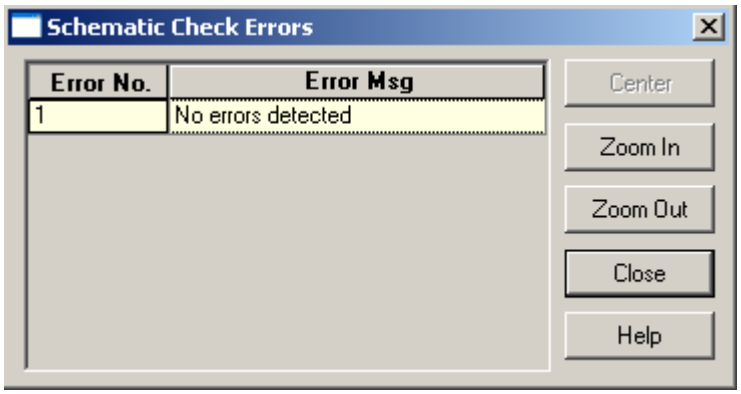


Figura B.35: Esquema de Diseño.

Para comprobar que todo es correcto, *ECS* permite buscar los errores existentes en el diseño. Presionando en el botón, se abre una pantalla que informa sobre los *Avisos (Warnings)* y *Errores (Errors)* que presenta el diseño. En nuestro caso, no nos informan sobre ningún error ni aviso, porque el diseño es correcto.



(a) Botón de Comprobación.



(b) Pantalla de Errores.

Figura B.36: Errores de Diseño

Una vez que hemos comprobado que no existen errores en el diseño, lo guardamos y salimos del programa *ECS*.

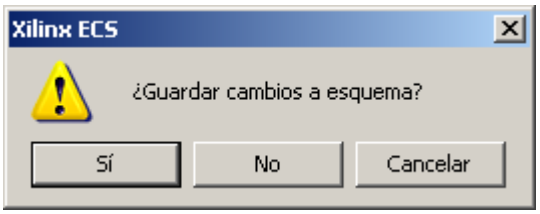


Figura B.37: Guardado del Diseño.

De nuevo en la pantalla de *Project Navigator*, comprobamos que, al incluir el símbolo *Decode* en nuestro diseño, la *Fuente Decode* se ha anidado dentro del *Esquema* en la Ventana de *Fuentes*.

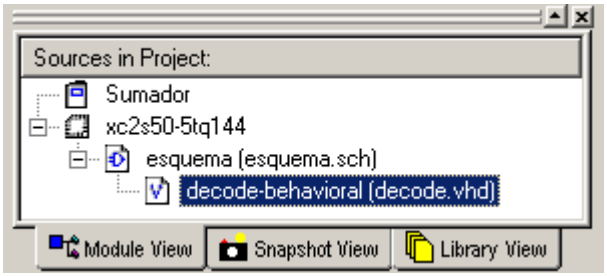


Figura B.38: Ventana de *Fuentes*.

El siguiente paso en el desarrollo del *Proyecto* es asignar los pines de nuestro diseño a los pines de la tarjeta. Para ello, debemos seleccionar el diseño de mayor nivel en la Ventana de *Fuentes* y, en la Ventana de *Procesos*, dentro del menú *User Constraints*, elegir la opción *Assign Package Pins*.

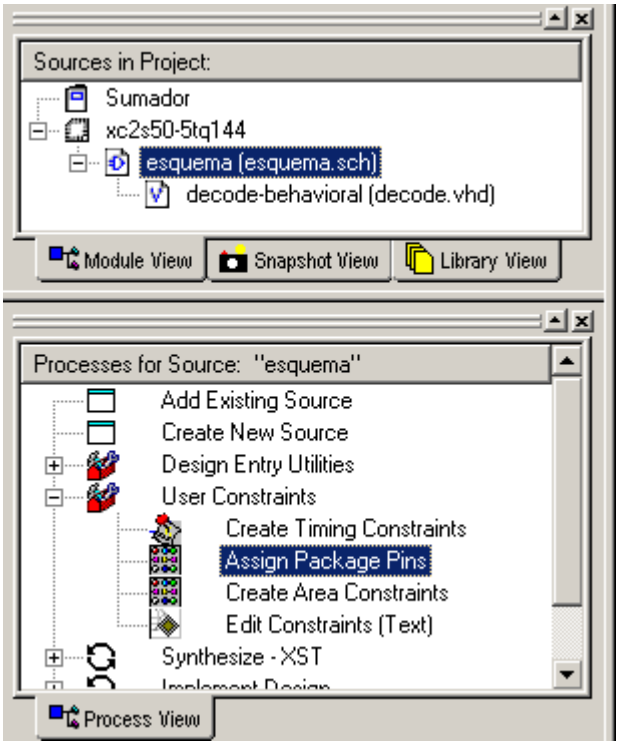


Figura B.39: Asignación de pines.

La asignación de pines se implementa en un archivo de extensión *.ucf*. Para que este archivo sea interpretado durante la compilación, en necesario añadirlo al *Proyecto* como una *Fuente*. Al arrancar el programa de *Asignación de pines* por primera vez, éste nos pregunta si queremos añadir al *Proyecto* actual, la *Fuente* que vamos a generar.

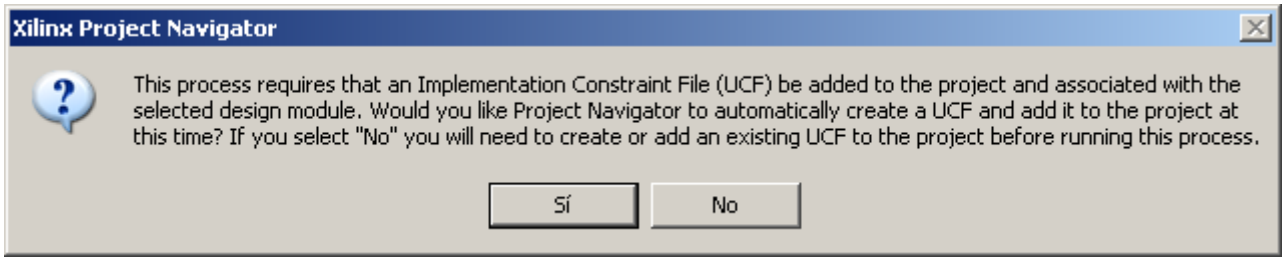


Figura B.40: Añadir *Fuente* al *Proyecto*.

El programa que nos permitirá la asignación de pines se llama *Xilinx PACE*. Su entorno es el que se muestra en la figura.

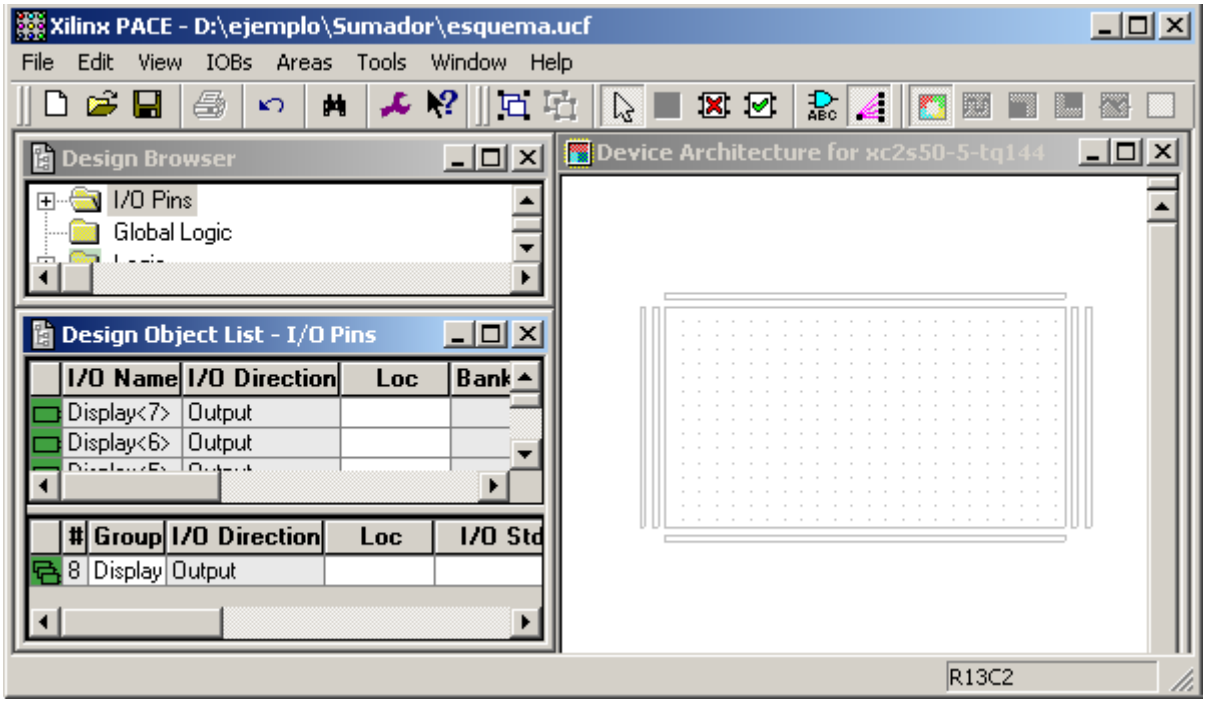


Figura B.41: Entorno de *PACE*.

Como vemos, al igual que *Project Navigator*, el entorno también está organizado en forma de ventanas. Para asignar pines, utilizamos la Ventana de *Design Object List - I/O Pins*, que mostramos en la figura. Los pines se asignan escribiendo en la casilla una letra *P* (o *p*), seguida del número de pin correspondiente de la *FPGA*, por ejemplo *P44* (o *p44*).

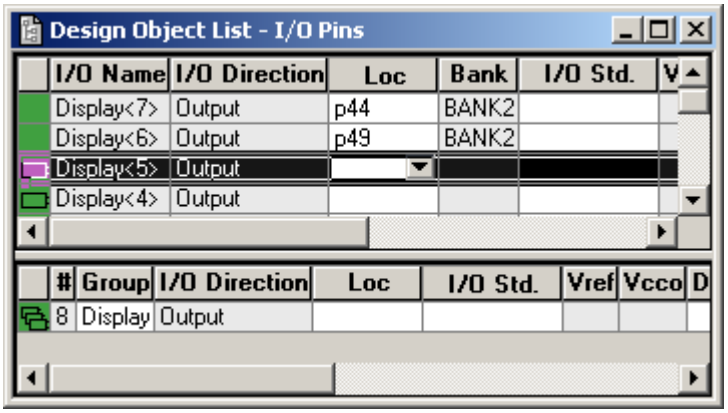


Figura B.42: *Ventana de Asignación de Pines.*

Los pines que debemos asignar se encuentran en el *Manual de Usuario de la tarjeta* XSA 50, *XSA Board V1.1, V1.2 User Manual* en el *Apéndice A: XSA Pin Connections*. Las números que corresponden a los segmentos del *Display* se pueden encontrar en la página *24* del mismo documento, en la sección *Flash RAM*.

Para las conexiones de las entradas, usaremos la utilidad *GXSPort* que permite controlar 8 bits durante la ejecución del programa. Estos 8 bits deben asignarse a los pines *PP-D0...D7* cuyos números también se encuentran en el mismo documento.

Una vez que se ha completado la asignación de todos los pines, cerramos el programa *PACE* guardando todos los cambios realizados para esta asignación. Al volver a *Project Navigator*, comprobamos que el fichero *.ucf* se ha añadido al *Proyecto*.

El último paso es generar el archivo que implementa el diseño (*bitstream*) para descargarlo sobre la *FPGA*. Para ello, teniendo cuidado de seleccionar la *Fuente esquema.sch* en la Ventana de *Fuentes*, pinchamos dos veces sobre *Generate Programming File* de la Ventana de *Procesos*.

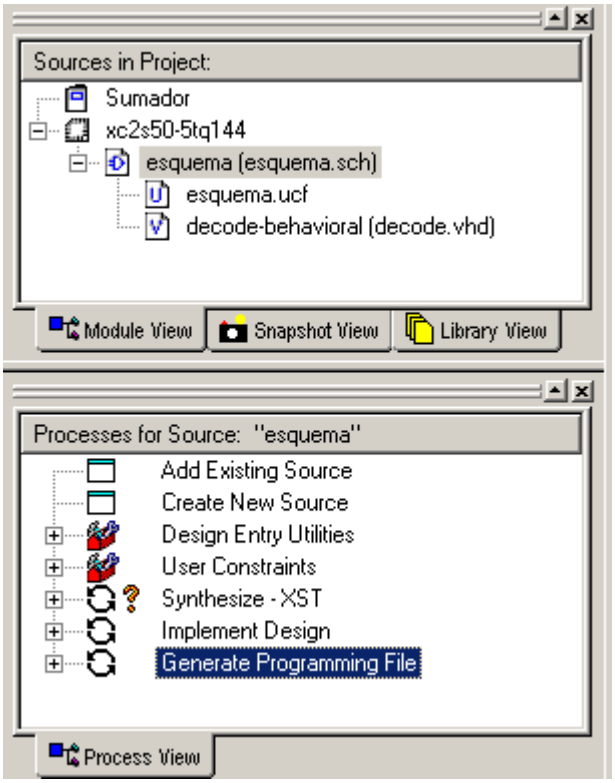


Figura B.43: Generación del *bitstream*.

Durante el proceso de generación del *bitstream*, la *Consola* informa de los *Sucesos*, *Avisos* o *Errores* que vayan teniendo lugar. Los *Avisos* no detienen este proceso, y permiten generar el *bitstream* correctamente, aunque, dependiendo del tipo de aviso, es posible que el diseño no funcione como debe. Generalmente, los *Avisos* no influyen en el comportamiento del diseño. En el caso de los *Errores*, el proceso se detiene y no genera el *bitstream*.

En nuestro caso, vemos que el proceso ha generado varios *Avisos* ( ) pero que ha completado la generación del *bitstream* sin errores ( ).

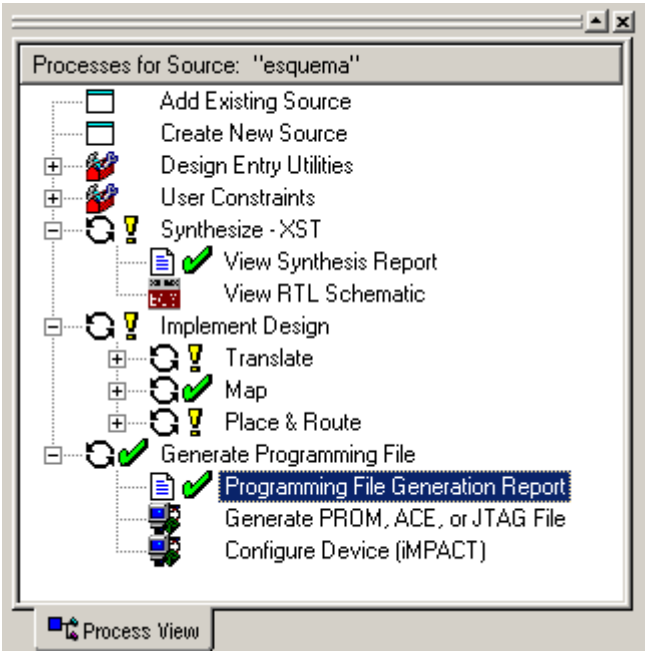


Figura B.44: Generación del *bitstream*.

En este proceso, se han generado varios *Informes (Reports)* ( ) que pueden leerse al pinchar sobre ellos. Si hubiera un error, estos informes no se habrían generado.

Entonces, mediante el *Explorador de Windows*, en la carpeta donde guardamos el *Proyecto* podemos encontrar todos los archivos generados durante este proceso. El archivo que descargaremos sobre la *FPGA* es el archivo de extensión *.bit*. Además, el nombre del archivo es el mismo que el nombre de la *Fuente* que hemos seleccionado para generar el *bitstream*.

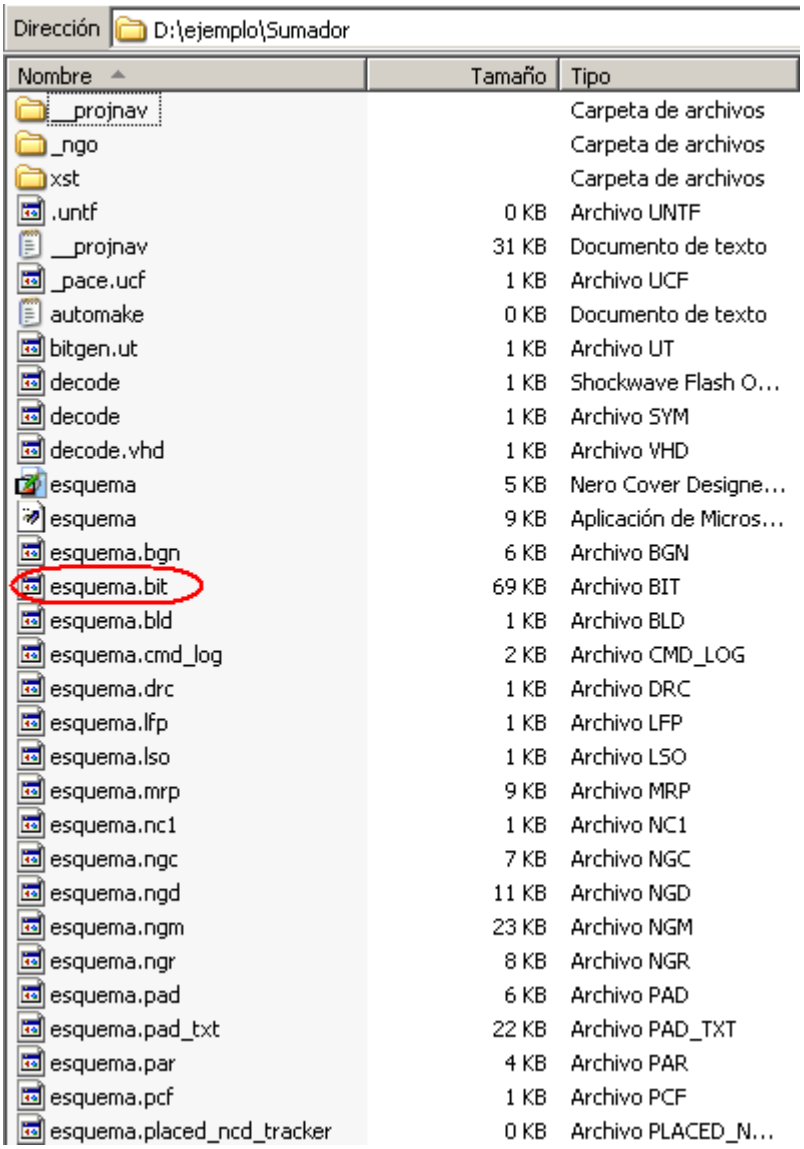


Figura B.45: *Bitstream*.

Una vez que tengamos el *bitstream*, lo podemos descargar sobre la tarjeta. Para ello, es muy importante que todos los *Jumpers* estén configurados de la forma que vimos en el Apartado 2.3.12. El cable paralelo debe estar conectado tanto al *PC* como al *Puerto Paralelo* de la tarjeta y la tarjeta debe estar correctamente alimentada.

Para ello, disponemos de la Utilidad *GXSLoad* del paquete *GXSTools*. El enlace a este programa puede encontrarse en el *Acceso directo* generado en el *Menú Inicio* durante la instalación.

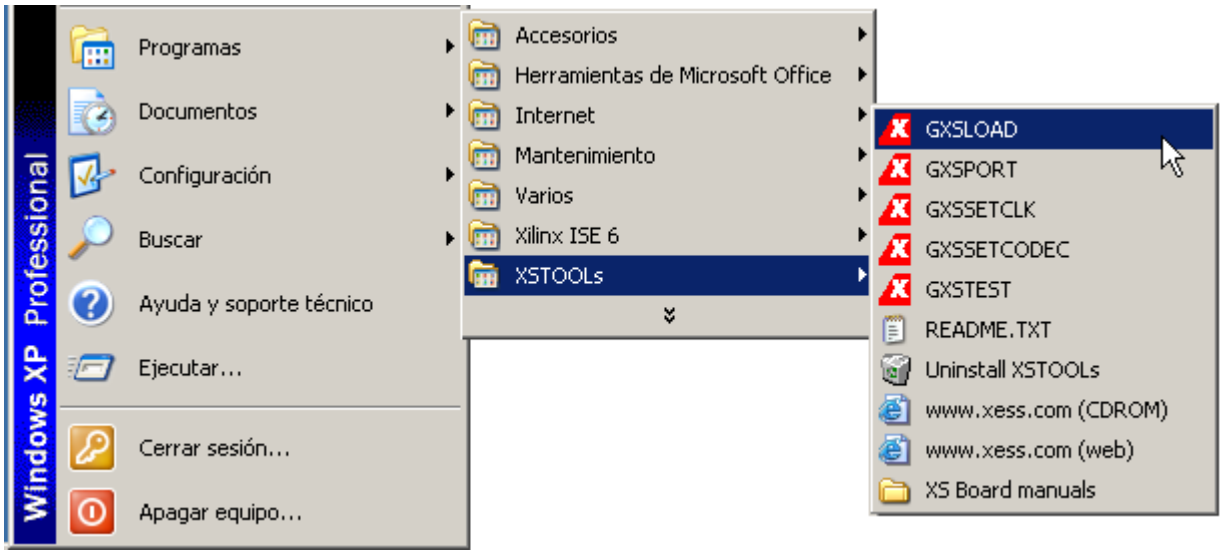


Figura B.46: Acceso a *GXSLoad*.

El programa emergente tiene un entorno como el de la figura. Dispone de 3 ventanas para intercambio de datos con la *FPGA*, la *SDRAM* y la *Flash ROM*. Para descargar el *bitstream* sobre la *FPGA*, basta con arrastrar el archivo *.bit* sobre la ventana *FPGA*, seleccionarlo (fondo azul) y presionar el botón *Load*.

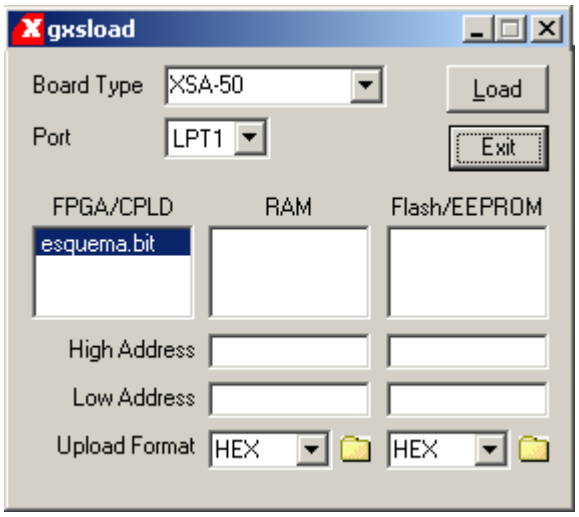


Figura B.47: *GXSLoad*.

Entonces, comienza el proceso de descarga del *bitstream* sobre la *FPGA* de la tarjeta.

Una vez que termina la descarga, el programa está ejecutándose. Por nuestros diseño, el programa tiene funcionamiento asíncrono y no dispone de señales de reloj, es decir, hasta que no modifiquemos una entrada, no veremos ningún cambio en la salida. Entonces, para realizar estas modificaciones, disponemos de la herramienta *GXSPort*. Al iniciarla, tiene un valor para cada bit por defecto. Nosotros podemos modificar todos los valores que queramos, pero hasta que no presionemos el botón *Strobe*, estas modificaciones no tendrán efecto sobre el programa. Tras presionar el botón *Strobe*, hasta que no hagamos alguna modificación de algún bit, este botón queda desactivado. Esta herramienta es muy útil para realizar simulaciones y comprobaciones (a modo de *Bancos de Prueba*) de los diseños realizados.

Como ejemplo, y para comprobar el funcionamiento del diseño, realizamos la siguiente suma binaria.

	0	1	0	1		5
+	0	0	1	0	+	2
	0	1	1	1		7

Cuadro B.1: Ejemplo de Suma.

Entonces, para ello activamos los bits de entrada de la forma que mostramos en la figura B.48. Recordamos que las entradas *A(3:0)* son los bits *D3:D0*, las entradas *B(1:0)* son los bits *D6:D5* y la entrada para el control de la operación es *D4*.

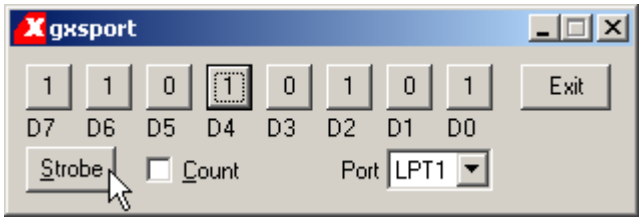


Figura B.48: Modificación de datos desde *GXSPort*.

Al modificar estos datos, y presionar *Strobe*, se actualiza el valor que vemos en el *Display*. En este caso, vemos un *7*. Podemos comprobar que todas las operaciones suma se realizan de forma correcta, y se sacan por el display de forma correcta.

Sin embargo, al poner *D4* en *0* para realizar una resta, debería ocurrir lo siguiente.

	0	1	0	1		5
-	0	0	1	0	+	2
	0	0	1	1		3

Cuadro B.2: Ejemplo de Resta.

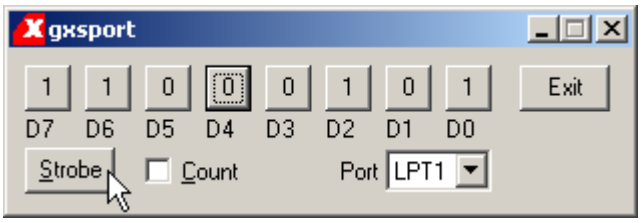


Figura B.49: Modificación de datos desde *GXSPort*.

El resultado que visualizamos en el display es *2*. Es decir, la resta no es correcta.

Cuando suceden errores de este tipo, la forma de proceder que sugerimos es analizar los bloques de forma individual, ya sea simulándolos por separado o interpretando los resultados obtenidos en pruebas anteriores. Así, a la vista de los resultados obtenidos en las operaciones de suma, podemos interpretar que la conversión de datos binarios a *formato Display* es correcta.

Por eliminación, entonces, el error se encuentra en el bloque *Sumador* o en una de sus entradas. Para obtener información sobre el comportamiento de este bloque utilizamos la opción *Symbol Info* que vimos anteriormente. En el documento emergente, podemos ver las características de este bloque. En uno de sus párrafos, nos encontramos con la siguiente información:

> In add mode, CO and CI are active-High. In subtract mode, CO and CI are active-Low.
> OFL is active-High in add and subtract modes.

Figura B.50: Información sobre *ADSU4*.

Es decir, en las restas, el acarreo (o llevada) es activo a nivel bajo. Por tanto, nosotros estamos ejecutando una resta con acarreo, y de ahí ese desfase de una unidad entre el resultado de la resta esperado y el obtenido.

Según el documento, para ejecutar una suma sin acarreo, el *Bit de Acarreo* debe valer 0, y para ejecutar una resta con acarreo, el *Bit de Acarreo* debe valer 1. Estos valores, son los inversos a los que entran por la entrada *ADD* de selección de la operación. Así, si invertimos esta entrada, podemos aplicarla directamente a la entrada de *CI*. A partir de esta experiencia, y de otras muchas, sugerimos leer los archivos de información de los bloques que no sean de diseño propio, antes de utilizarlos, para ver sus características, modos de funcionamiento o niveles de activación.

Por tanto, debemos modificar nuestro diseño. Para ello, pinchamos dos veces en la *Fuente Esquema* de la ventana de *Fuentes*, y se abrirá el programa de Diseño *ECS*. Para conectar un punto a otro, es estrictamente necesario eliminar completamente el cable que está conectado a ese punto. Si no lo eliminamos y realizamos la conexión sobre ese cable, no generará ningún error de diseño, pero sí lo hará durante la *Fase de Síntesis*. Además, es muy complicado localizar este tipo de errores, ya que no son evidentes a ojos del diseñador. Por eso, es muy importante ser estrictos al realizar modificaciones del diseño.

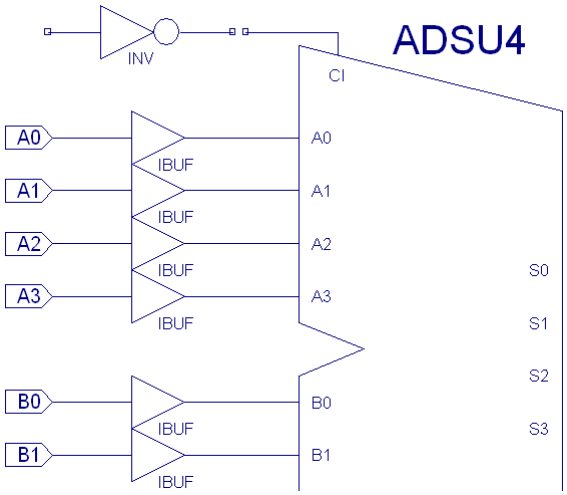


Figura B.51: Inversor para la entrada *CI*.

Tras colocar un inversor, realizamos las conexiones mediante la técnica de *Etiquetado* que hemos visto anteriormente.

El diseño completo se muestra en la figura:

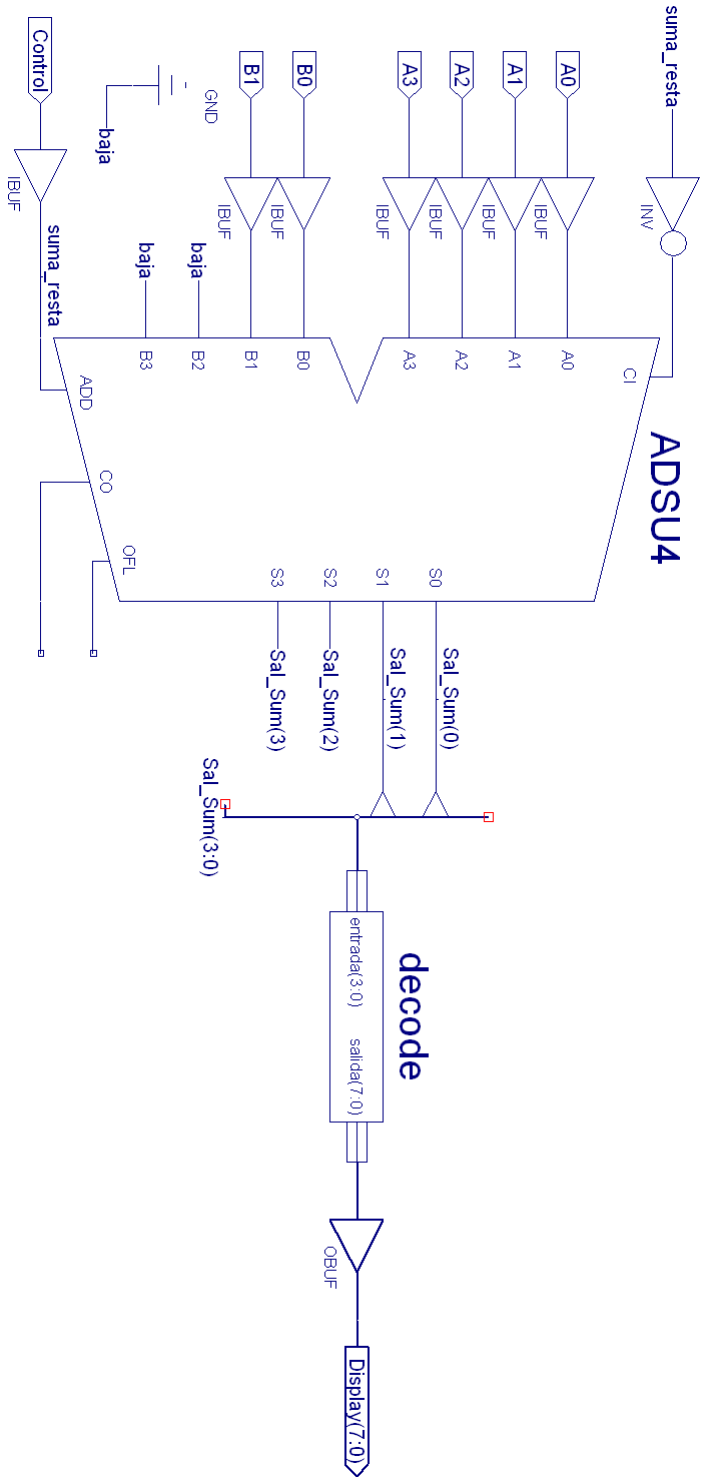


Figura B.52: Esquema de Diseño.

Siguiendo los mismos pasos que vimos anteriormente, guardamos el diseño y, desde *Project Navigator*, generamos el *bitstream*. Descargamos este *bitstream* sobre la *FPGA*, y comprobamos que, tanto las sumas como las restas, se realizan correctamente.

Con este punto damos por finalizado el tutorial. En este apéndice, no hemos pretendido analizar cada característica de cada programa de *ISE WebPACK*, sino presentar una filosofía de manejo de todo este software. Hemos explicado, paso por paso:

- Cómo se crea un *Proyecto*.
- Cómo se crean y añaden nuevas fuentes.
- Cómo se realiza el diseño por *Captura Esquemática*.
- Cómo se asignan los pines de la *FPGA*.
- Cómo se genera el *bitstream*.
- Cómo se descarga sobre la *FPGA*.

Para entrar en más detalle en las peculiaridades de cada programa de este paquete, nos remitimos a la documentación disponible en *www.xilinx.com*, en el documento *ISE 6 In-Depth Tutorial* y, especialmente, en la propia ayuda de cada programa.

Apéndice C

Informes

Project Navigator genera varios informes durante el proceso de compilación de los diseños. En este apéndice, vamos a incluir los informes más significativos que *Project Navigator* genera al compilar el diseño del *Procesador* que hemos embebido en la *FPGA Xilinx Spartan II*.

Estos informes son los siguientes:

1. *Modelo Funcional (VHDL)* (18 páginas). En este informe, *Project Navigator* presenta los *Modelos Funcionales* de los bloques existentes en la hoja de diseño actual. El *Modelo Funcional* de un bloque consiste en la declaración de su entidad y arquitectura.

2. *Map Report* (48 páginas). El informe de *Mapeado* se genera durante este proceso de *Mapeado*. Contiene información muy útil para el diseñador, como los porcentajes de ocupación de la tarjeta, recursos utilizados, organización de los bloques de diseño sobre la tarjeta, informes sobre tiempos, etcétera.

3. *Programming File* (3 páginas). Informa sobre la generación del *bitstream* que podremos descargar sobre la tarjeta.

4. *Fichero .ucf* (13 páginas). Este fichero no es un informe del tipo de los anteriores que genera *Project Navigator*. Sin embargo, este fichero es muy importante, porque contiene información trascendental del diseño. En concreto, son dos puntos muy importantes:

   - La asignación de pines de la tarjeta.
   - El contenido de la memorias implementadas en el interior de la *FPGA*.

   Este fichero es accesible y modificable manualmente, como un fichero de texto, o a partir del programa de asignación de pines *Pace*.

En las siguientes páginas, se presentan todos estos informes.

```
1   ------------------------------------------------------------------------
    ------------
2   -- Copyright (c) 1995-2003 Xilinx, Inc.
3   -- All Right Reserved.
4   ------------------------------------------------------------------------
    ------------
5   --   _____  ____
6   --  /    /\/    /
7   -- /___/  \  /    Vendor: Xilinx
8   -- \    \   \/     Version : 6.3.03i
9   --  \    \         Application :
10  --  /    /         Filename : procesador.vhf
11  -- /___/   /\     Timestamp : 03/08/2005 13:30:00
12  -- \    \  /  \
13  --  \___\/\___\
14  --
15  --Command:
16  --Design Name: FTCE_MXILINX_procesador
17  --
18
19  library ieee;
20  use ieee.std_logic_1164.ALL;
21  use ieee.numeric_std.ALL;
22  -- synopsys translate_off
23  library UNISIM;
24  use UNISIM.Vcomponents.ALL;
25  -- synopsys translate_on
26
27  entity FTCE_MXILINX_procesador is
28     port ( C   : in    std_logic;
29            CE  : in    std_logic;
30            CLR : in    std_logic;
31            T   : in    std_logic;
32            Q   : out   std_logic);
33  end FTCE_MXILINX_procesador;
34
35  architecture BEHAVIORAL of FTCE_MXILINX_procesador is
36     attribute BOX_TYPE   : string ;
37     attribute INIT       : string ;
38     attribute RLOC       : string ;
39     signal TQ      : std_logic;
40     signal Q_DUMMY : std_logic;
41     component XOR2
42        port ( I0 : in    std_logic;
43               I1 : in    std_logic;
44               O  : out   std_logic);
45     end component;
46     attribute BOX_TYPE of XOR2 : component is "BLACK_BOX";
47
48     component FDCE
49        -- synopsys translate_off
50        generic( INIT : bit :=  '0');
51        -- synopsys translate_on
52        port ( C   : in    std_logic;
53               CE  : in    std_logic;
54               CLR : in    std_logic;
55               D   : in    std_logic;
56               Q   : out   std_logic);
```

```
57       end component;
58       attribute INIT of FDCE : component is "0";
59       attribute BOX_TYPE of FDCE : component is "BLACK_BOX";
60
61       attribute RLOC of I_36_35 : label is "R0C0.S0";
62    begin
63       Q <= Q_DUMMY;
64       I_36_32 : XOR2
65          port map (I0=>T,
66                    I1=>Q_DUMMY,
67                    O=>TQ);
68
69       I_36_35 : FDCE
70          port map (C=>C,
71                    CE=>CE,
72                    CLR=>CLR,
73                    D=>TQ,
74                    Q=>Q_DUMMY);
75
76    end BEHAVIORAL;
77
78
79    ------------------------------------------------------------------------
      ------------
80    -- Copyright (c) 1995-2003 Xilinx, Inc.
81    -- All Right Reserved.
82    ------------------------------------------------------------------------
      ------------
83    --   _____  _____
84    --  /    /\/    /
85    -- /___/  \  /    Vendor: Xilinx
86    -- \    \   \/     Version : 6.3.03i
87    --  \    \         Application :
88    --  /    /         Filename : procesador.vhf
89    -- /___/   /\     Timestamp : 03/08/2005 13:30:00
90    -- \    \  /  \
91    --  \___\/\___\
92    --
93    --Command:
94    --Design Name: CB2CE_MXILINX_procesador
95    --
96
97    library ieee;
98    use ieee.std_logic_1164.ALL;
99    use ieee.numeric_std.ALL;
100   -- synopsys translate_off
101   library UNISIM;
102   use UNISIM.Vcomponents.ALL;
103   -- synopsys translate_on
104
105   entity CB2CE_MXILINX_procesador is
106      port ( C   : in    std_logic;
107             CE  : in    std_logic;
108             CLR : in    std_logic;
109             CEO : out   std_logic;
110             Q0  : out   std_logic;
111             Q1  : out   std_logic;
112             TC  : out   std_logic);
```

```
113   end CB2CE_MXILINX_procesador;
114
115   architecture BEHAVIORAL of CB2CE_MXILINX_procesador is
116      attribute HU_SET     : string ;
117      attribute BOX_TYPE   : string ;
118      signal XLXN_1   : std_logic;
119      signal Q0_DUMMY : std_logic;
120      signal Q1_DUMMY : std_logic;
121      signal TC_DUMMY : std_logic;
122      component FTCE_MXILINX_procesador
123         port ( C   : in    std_logic;
124                CE  : in    std_logic;
125                CLR : in    std_logic;
126                T   : in    std_logic;
127                Q   : out   std_logic);
128      end component;
129
130      component AND2
131         port ( I0 : in    std_logic;
132                I1 : in    std_logic;
133                O  : out   std_logic);
134      end component;
135      attribute BOX_TYPE of AND2 : component is "BLACK_BOX";
136
137      component VCC
138         port ( P : out   std_logic);
139      end component;
140      attribute BOX_TYPE of VCC : component is "BLACK_BOX";
141
142      attribute HU_SET of I_Q0 : label is "I_Q0_0";
143      attribute HU_SET of I_Q1 : label is "I_Q1_1";
144   begin
145      Q0 <= Q0_DUMMY;
146      Q1 <= Q1_DUMMY;
147      TC <= TC_DUMMY;
148      I_Q0 : FTCE_MXILINX_procesador
149         port map (C=>C,
150                   CE=>CE,
151                   CLR=>CLR,
152                   T=>XLXN_1,
153                   Q=>Q0_DUMMY);
154
155      I_Q1 : FTCE_MXILINX_procesador
156         port map (C=>C,
157                   CE=>CE,
158                   CLR=>CLR,
159                   T=>Q0_DUMMY,
160                   Q=>Q1_DUMMY);
161
162      I_36_37 : AND2
163         port map (I0=>Q1_DUMMY,
164                   I1=>Q0_DUMMY,
165                   O=>TC_DUMMY);
166
167      I_36_47 : VCC
168         port map (P=>XLXN_1);
169
170      I_36_52 : AND2
```

```
171         port map (I0=>CE,
172                   I1=>TC_DUMMY,
173                   O=>CEO);
174
175   end BEHAVIORAL;
176
177
178   ----------------------------------------------------------------------
      ------------
179   -- Copyright (c) 1995-2003 Xilinx, Inc.
180   -- All Right Reserved.
181   ----------------------------------------------------------------------
      ------------
182   --   _____  ____
183   --  /   /\/   /
184   -- /___/  \  /    Vendor: Xilinx
185   -- \   \   \/     Version : 6.3.03i
186   --  \   \         Application :
187   --  /   /         Filename : procesador.vhf
188   -- /___/   /\     Timestamp : 03/08/2005 13:30:00
189   -- \   \  /  \
190   --  \___\/\___\
191   --
192   --Command:
193   --Design Name: CB16CE_MXILINX_procesador
194   --
195
196   library ieee;
197   use ieee.std_logic_1164.ALL;
198   use ieee.numeric_std.ALL;
199   -- synopsys translate_off
200   library UNISIM;
201   use UNISIM.Vcomponents.ALL;
202   -- synopsys translate_on
203
204   entity CB16CE_MXILINX_procesador is
205      port ( C   : in    std_logic;
206             CE  : in    std_logic;
207             CLR : in    std_logic;
208             CEO : out   std_logic;
209             Q   : out   std_logic_vector (15 downto 0);
210             TC  : out   std_logic);
211   end CB16CE_MXILINX_procesador;
212
213   architecture BEHAVIORAL of CB16CE_MXILINX_procesador is
214      attribute HU_SET     : string ;
215      attribute BOX_TYPE   : string ;
216      signal T2       : std_logic;
217      signal T3       : std_logic;
218      signal T4       : std_logic;
219      signal T5       : std_logic;
220      signal T6       : std_logic;
221      signal T7       : std_logic;
222      signal T8       : std_logic;
223      signal T9       : std_logic;
224      signal T10      : std_logic;
225      signal T11      : std_logic;
226      signal T12      : std_logic;
```

```
227     signal T13      : std_logic;
228     signal T14      : std_logic;
229     signal T15      : std_logic;
230     signal XLXN_1   : std_logic;
231     signal Q_DUMMY  : std_logic_vector (15 downto 0);
232     signal TC_DUMMY : std_logic;
233     component FTCE_MXILINX_procesador
234        port ( C   : in    std_logic;
235               CE  : in    std_logic;
236               CLR : in    std_logic;
237               T   : in    std_logic;
238               Q   : out   std_logic);
239     end component;
240
241     component AND3
242        port ( I0 : in    std_logic;
243               I1 : in    std_logic;
244               I2 : in    std_logic;
245               O  : out   std_logic);
246     end component;
247     attribute BOX_TYPE of AND3 : component is "BLACK_BOX";
248
249     component AND2
250        port ( I0 : in    std_logic;
251               I1 : in    std_logic;
252               O  : out   std_logic);
253     end component;
254     attribute BOX_TYPE of AND2 : component is "BLACK_BOX";
255
256     component VCC
257        port ( P : out   std_logic);
258     end component;
259     attribute BOX_TYPE of VCC : component is "BLACK_BOX";
260
261     component AND4
262        port ( I0 : in    std_logic;
263               I1 : in    std_logic;
264               I2 : in    std_logic;
265               I3 : in    std_logic;
266               O  : out   std_logic);
267     end component;
268     attribute BOX_TYPE of AND4 : component is "BLACK_BOX";
269
270     component AND5
271        port ( I0 : in    std_logic;
272               I1 : in    std_logic;
273               I2 : in    std_logic;
274               I3 : in    std_logic;
275               I4 : in    std_logic;
276               O  : out   std_logic);
277     end component;
278     attribute BOX_TYPE of AND5 : component is "BLACK_BOX";
279
280     attribute HU_SET of I_Q0 : label is "I_Q0_3";
281     attribute HU_SET of I_Q1 : label is "I_Q1_2";
282     attribute HU_SET of I_Q2 : label is "I_Q2_5";
283     attribute HU_SET of I_Q3 : label is "I_Q3_4";
284     attribute HU_SET of I_Q4 : label is "I_Q4_9";
```

```
285      attribute HU_SET of I_Q5 : label is "I_Q5_8";
286      attribute HU_SET of I_Q6 : label is "I_Q6_7";
287      attribute HU_SET of I_Q7 : label is "I_Q7_6";
288      attribute HU_SET of I_Q8 : label is "I_Q8_10";
289      attribute HU_SET of I_Q9 : label is "I_Q9_11";
290      attribute HU_SET of I_Q10 : label is "I_Q10_12";
291      attribute HU_SET of I_Q11 : label is "I_Q11_13";
292      attribute HU_SET of I_Q12 : label is "I_Q12_14";
293      attribute HU_SET of I_Q13 : label is "I_Q13_15";
294      attribute HU_SET of I_Q14 : label is "I_Q14_16";
295      attribute HU_SET of I_Q15 : label is "I_Q15_17";
296   begin
297      Q(15 downto 0) <= Q_DUMMY(15 downto 0);
298      TC <= TC_DUMMY;
299      I_Q0 : FTCE_MXILINX_procesador
300         port map (C=>C,
301                   CE=>CE,
302                   CLR=>CLR,
303                   T=>XLXN_1,
304                   Q=>Q_DUMMY(0));
305
306      I_Q1 : FTCE_MXILINX_procesador
307         port map (C=>C,
308                   CE=>CE,
309                   CLR=>CLR,
310                   T=>Q_DUMMY(0),
311                   Q=>Q_DUMMY(1));
312
313      I_Q2 : FTCE_MXILINX_procesador
314         port map (C=>C,
315                   CE=>CE,
316                   CLR=>CLR,
317                   T=>T2,
318                   Q=>Q_DUMMY(2));
319
320      I_Q3 : FTCE_MXILINX_procesador
321         port map (C=>C,
322                   CE=>CE,
323                   CLR=>CLR,
324                   T=>T3,
325                   Q=>Q_DUMMY(3));
326
327      I_Q4 : FTCE_MXILINX_procesador
328         port map (C=>C,
329                   CE=>CE,
330                   CLR=>CLR,
331                   T=>T4,
332                   Q=>Q_DUMMY(4));
333
334      I_Q5 : FTCE_MXILINX_procesador
335         port map (C=>C,
336                   CE=>CE,
337                   CLR=>CLR,
338                   T=>T5,
339                   Q=>Q_DUMMY(5));
340
341      I_Q6 : FTCE_MXILINX_procesador
342         port map (C=>C,
```

```
343                CE=>CE,
344                CLR=>CLR,
345                T=>T6,
346                Q=>Q_DUMMY(6));
347
348      I_Q7 : FTCE_MXILINX_procesador
349         port map (C=>C,
350                CE=>CE,
351                CLR=>CLR,
352                T=>T7,
353                Q=>Q_DUMMY(7));
354
355      I_Q8 : FTCE_MXILINX_procesador
356         port map (C=>C,
357                CE=>CE,
358                CLR=>CLR,
359                T=>T8,
360                Q=>Q_DUMMY(8));
361
362      I_Q9 : FTCE_MXILINX_procesador
363         port map (C=>C,
364                CE=>CE,
365                CLR=>CLR,
366                T=>T9,
367                Q=>Q_DUMMY(9));
368
369      I_Q10 : FTCE_MXILINX_procesador
370         port map (C=>C,
371                CE=>CE,
372                CLR=>CLR,
373                T=>T10,
374                Q=>Q_DUMMY(10));
375
376      I_Q11 : FTCE_MXILINX_procesador
377         port map (C=>C,
378                CE=>CE,
379                CLR=>CLR,
380                T=>T11,
381                Q=>Q_DUMMY(11));
382
383      I_Q12 : FTCE_MXILINX_procesador
384         port map (C=>C,
385                CE=>CE,
386                CLR=>CLR,
387                T=>T12,
388                Q=>Q_DUMMY(12));
389
390      I_Q13 : FTCE_MXILINX_procesador
391         port map (C=>C,
392                CE=>CE,
393                CLR=>CLR,
394                T=>T13,
395                Q=>Q_DUMMY(13));
396
397      I_Q14 : FTCE_MXILINX_procesador
398         port map (C=>C,
399                CE=>CE,
400                CLR=>CLR,
```

```
401                T=>T14,
402                Q=>Q_DUMMY(14));
403
404      I_Q15 : FTCE_MXILINX_procesador
405         port map (C=>C,
406                   CE=>CE,
407                   CLR=>CLR,
408                   T=>T15,
409                   Q=>Q_DUMMY(15));
410
411      I_36_3 : AND3
412         port map (I0=>Q_DUMMY(2),
413                   I1=>Q_DUMMY(1),
414                   I2=>Q_DUMMY(0),
415                   O=>T3);
416
417      I_36_4 : AND2
418         port map (I0=>Q_DUMMY(1),
419                   I1=>Q_DUMMY(0),
420                   O=>T2);
421
422      I_36_9 : VCC
423         port map (P=>XLXN_1);
424
425      I_36_10 : AND4
426         port map (I0=>Q_DUMMY(3),
427                   I1=>Q_DUMMY(2),
428                   I2=>Q_DUMMY(1),
429                   I3=>Q_DUMMY(0),
430                   O=>T4);
431
432      I_36_14 : AND5
433         port map (I0=>Q_DUMMY(7),
434                   I1=>Q_DUMMY(6),
435                   I2=>Q_DUMMY(5),
436                   I3=>Q_DUMMY(4),
437                   I4=>T4,
438                   O=>T8);
439
440      I_36_15 : AND2
441         port map (I0=>Q_DUMMY(4),
442                   I1=>T4,
443                   O=>T5);
444
445      I_36_19 : AND3
446         port map (I0=>Q_DUMMY(5),
447                   I1=>Q_DUMMY(4),
448                   I2=>T4,
449                   O=>T6);
450
451      I_36_21 : AND4
452         port map (I0=>Q_DUMMY(6),
453                   I1=>Q_DUMMY(5),
454                   I2=>Q_DUMMY(4),
455                   I3=>T4,
456                   O=>T7);
457
458      I_36_22 : AND5
```

```
459         port map (I0=>Q_DUMMY(15),
460                   I1=>Q_DUMMY(14),
461                   I2=>Q_DUMMY(13),
462                   I3=>Q_DUMMY(12),
463                   I4=>T12,
464                   O=>TC_DUMMY);
465
466      I_36_23 : AND2
467         port map (I0=>Q_DUMMY(12),
468                   I1=>T12,
469                   O=>T13);
470
471      I_36_24 : AND3
472         port map (I0=>Q_DUMMY(13),
473                   I1=>Q_DUMMY(12),
474                   I2=>T12,
475                   O=>T14);
476
477      I_36_25 : AND4
478         port map (I0=>Q_DUMMY(14),
479                   I1=>Q_DUMMY(13),
480                   I2=>Q_DUMMY(12),
481                   I3=>T12,
482                   O=>T15);
483
484      I_36_26 : AND4
485         port map (I0=>Q_DUMMY(10),
486                   I1=>Q_DUMMY(9),
487                   I2=>Q_DUMMY(8),
488                   I3=>T8,
489                   O=>T11);
490
491      I_36_27 : AND3
492         port map (I0=>Q_DUMMY(9),
493                   I1=>Q_DUMMY(8),
494                   I2=>T8,
495                   O=>T10);
496
497      I_36_28 : AND2
498         port map (I0=>Q_DUMMY(8),
499                   I1=>T8,
500                   O=>T9);
501
502      I_36_29 : AND5
503         port map (I0=>Q_DUMMY(11),
504                   I1=>Q_DUMMY(10),
505                   I2=>Q_DUMMY(9),
506                   I3=>Q_DUMMY(8),
507                   I4=>T8,
508                   O=>T12);
509
510      I_36_54 : AND2
511         port map (I0=>CE,
512                   I1=>TC_DUMMY,
513                   O=>CEO);
514
515   end BEHAVIORAL;
516
```

```
517
518   ------------------------------------------------------------------------
      ------------
519   -- Copyright (c) 1995-2003 Xilinx, Inc.
520   -- All Right Reserved.
521   ------------------------------------------------------------------------
      ------------
522   --   _____  ____
523   --  /   /\/   /
524   -- /___/  \  /    Vendor: Xilinx
525   -- \   \   \/     Version : 6.3.03i
526   --  \   \         Application :
527   --  /   /         Filename : procesador.vhf
528   -- /___/   /\     Timestamp : 03/08/2005 13:30:00
529   -- \   \  /  \
530   --  \___\/\___\
531   --
532   --Command:
533   --Design Name: CB8CE_MXILINX_procesador
534   --
535
536   library ieee;
537   use ieee.std_logic_1164.ALL;
538   use ieee.numeric_std.ALL;
539   -- synopsys translate_off
540   library UNISIM;
541   use UNISIM.Vcomponents.ALL;
542   -- synopsys translate_on
543
544   entity CB8CE_MXILINX_procesador is
545      port ( C   : in    std_logic;
546             CE  : in    std_logic;
547             CLR : in    std_logic;
548             CEO : out   std_logic;
549             Q   : out   std_logic_vector (7 downto 0);
550             TC  : out   std_logic);
551   end CB8CE_MXILINX_procesador;
552
553   architecture BEHAVIORAL of CB8CE_MXILINX_procesador is
554      attribute HU_SET     : string ;
555      attribute BOX_TYPE   : string ;
556      signal T2       : std_logic;
557      signal T3       : std_logic;
558      signal T4       : std_logic;
559      signal T5       : std_logic;
560      signal T6       : std_logic;
561      signal T7       : std_logic;
562      signal XLXN_1   : std_logic;
563      signal Q_DUMMY  : std_logic_vector (7 downto 0);
564      signal TC_DUMMY : std_logic;
565      component FTCE_MXILINX_procesador
566         port ( C   : in    std_logic;
567                CE  : in    std_logic;
568                CLR : in    std_logic;
569                T   : in    std_logic;
570                Q   : out   std_logic);
571      end component;
572
```

```
573      component AND5
574         port ( I0 : in    std_logic;
575                I1 : in    std_logic;
576                I2 : in    std_logic;
577                I3 : in    std_logic;
578                I4 : in    std_logic;
579                O  : out   std_logic);
580      end component;
581      attribute BOX_TYPE of AND5 : component is "BLACK_BOX";
582
583      component AND2
584         port ( I0 : in    std_logic;
585                I1 : in    std_logic;
586                O  : out   std_logic);
587      end component;
588      attribute BOX_TYPE of AND2 : component is "BLACK_BOX";
589
590      component AND3
591         port ( I0 : in    std_logic;
592                I1 : in    std_logic;
593                I2 : in    std_logic;
594                O  : out   std_logic);
595      end component;
596      attribute BOX_TYPE of AND3 : component is "BLACK_BOX";
597
598      component AND4
599         port ( I0 : in    std_logic;
600                I1 : in    std_logic;
601                I2 : in    std_logic;
602                I3 : in    std_logic;
603                O  : out   std_logic);
604      end component;
605      attribute BOX_TYPE of AND4 : component is "BLACK_BOX";
606
607      component VCC
608         port ( P : out   std_logic);
609      end component;
610      attribute BOX_TYPE of VCC : component is "BLACK_BOX";
611
612      attribute HU_SET of I_Q0 : label is "I_Q0_24";
613      attribute HU_SET of I_Q1 : label is "I_Q1_25";
614      attribute HU_SET of I_Q2 : label is "I_Q2_21";
615      attribute HU_SET of I_Q3 : label is "I_Q3_22";
616      attribute HU_SET of I_Q4 : label is "I_Q4_23";
617      attribute HU_SET of I_Q5 : label is "I_Q5_20";
618      attribute HU_SET of I_Q6 : label is "I_Q6_19";
619      attribute HU_SET of I_Q7 : label is "I_Q7_18";
620   begin
621      Q(7 downto 0) <= Q_DUMMY(7 downto 0);
622      TC <= TC_DUMMY;
623      I_Q0 : FTCE_MXILINX_procesador
624         port map (C=>C,
625                   CE=>CE,
626                   CLR=>CLR,
627                   T=>XLXN_1,
628                   Q=>Q_DUMMY(0));
629
630      I_Q1 : FTCE_MXILINX_procesador
```

```
631         port map (C=>C,
632                   CE=>CE,
633                   CLR=>CLR,
634                   T=>Q_DUMMY(0),
635                   Q=>Q_DUMMY(1));
636
637      I_Q2 : FTCE_MXILINX_procesador
638         port map (C=>C,
639                   CE=>CE,
640                   CLR=>CLR,
641                   T=>T2,
642                   Q=>Q_DUMMY(2));
643
644      I_Q3 : FTCE_MXILINX_procesador
645         port map (C=>C,
646                   CE=>CE,
647                   CLR=>CLR,
648                   T=>T3,
649                   Q=>Q_DUMMY(3));
650
651      I_Q4 : FTCE_MXILINX_procesador
652         port map (C=>C,
653                   CE=>CE,
654                   CLR=>CLR,
655                   T=>T4,
656                   Q=>Q_DUMMY(4));
657
658      I_Q5 : FTCE_MXILINX_procesador
659         port map (C=>C,
660                   CE=>CE,
661                   CLR=>CLR,
662                   T=>T5,
663                   Q=>Q_DUMMY(5));
664
665      I_Q6 : FTCE_MXILINX_procesador
666         port map (C=>C,
667                   CE=>CE,
668                   CLR=>CLR,
669                   T=>T6,
670                   Q=>Q_DUMMY(6));
671
672      I_Q7 : FTCE_MXILINX_procesador
673         port map (C=>C,
674                   CE=>CE,
675                   CLR=>CLR,
676                   T=>T7,
677                   Q=>Q_DUMMY(7));
678
679      I_36_1 : AND5
680         port map (I0=>Q_DUMMY(7),
681                   I1=>Q_DUMMY(6),
682                   I2=>Q_DUMMY(5),
683                   I3=>Q_DUMMY(4),
684                   I4=>T4,
685                   O=>TC_DUMMY);
686
687      I_36_2 : AND2
688         port map (I0=>Q_DUMMY(4),
```

```
689                  I1=>T4,
690                  O=>T5);
691
692      I_36_11 : AND3
693         port map (I0=>Q_DUMMY(5),
694                   I1=>Q_DUMMY(4),
695                   I2=>T4,
696                   O=>T6);
697
698      I_36_15 : AND4
699         port map (I0=>Q_DUMMY(3),
700                   I1=>Q_DUMMY(2),
701                   I2=>Q_DUMMY(1),
702                   I3=>Q_DUMMY(0),
703                   O=>T4);
704
705      I_36_16 : VCC
706         port map (P=>XLXN_1);
707
708      I_36_24 : AND2
709         port map (I0=>Q_DUMMY(1),
710                   I1=>Q_DUMMY(0),
711                   O=>T2);
712
713      I_36_26 : AND3
714         port map (I0=>Q_DUMMY(2),
715                   I1=>Q_DUMMY(1),
716                   I2=>Q_DUMMY(0),
717                   O=>T3);
718
719      I_36_28 : AND4
720         port map (I0=>Q_DUMMY(6),
721                   I1=>Q_DUMMY(5),
722                   I2=>Q_DUMMY(4),
723                   I3=>T4,
724                   O=>T7);
725
726      I_36_31 : AND2
727         port map (I0=>CE,
728                   I1=>TC_DUMMY,
729                   O=>CEO);
730
731   end BEHAVIORAL;
732
733
734   ------------------------------------------------------------------------
      ------------
735   -- Copyright (c) 1995-2003 Xilinx, Inc.
736   -- All Right Reserved.
737   ------------------------------------------------------------------------
      ------------
738   --   ____  ____
739   --  /   /\/   /
740   -- /___/  \  /    Vendor: Xilinx
741   -- \   \   \/     Version : 6.3.03i
742   --  \   \         Application :
743   --  /   /         Filename : procesador.vhf
744   -- /___/   /\     Timestamp : 03/08/2005 13:30:00
```

```
745   -- \   \  /  \
746   --  \___\/\___\
747   --
748   --Command:
749   --Design Name: procesador
750   --
751
752   library ieee;
753   use ieee.std_logic_1164.ALL;
754   use ieee.numeric_std.ALL;
755   -- synopsys translate_off
756   library UNISIM;
757   use UNISIM.Vcomponents.ALL;
758   -- synopsys translate_on
759
760   entity procesador is
761      port ( clk         : in    std_logic;
762             Pin_dato    : in    std_logic_vector (7 downto 0);
763             sclkfb      : in    std_logic;
764             acarreo     : out   std_logic;
765             C_SDRAM_out : out   std_logic_vector (21 downto 0);
766             N_CE        : out   std_logic;
767             N_OE        : out   std_logic;
768             N_RESET     : out   std_logic;
769             N_WE        : out   std_logic;
770             Pin_address : out   std_logic_vector (13 downto 0);
771             reloj       : out   std_logic;
772             sal         : out   std_logic_vector (15 downto 0);
773             zero        : out   std_logic;
774             C_SDRAM_bi  : inout std_logic_vector (15 downto 0));
775   end procesador;
776
777   architecture BEHAVIORAL of procesador is
778      attribute HU_SET     : string ;
779      attribute BOX_TYPE   : string ;
780      signal bc          : std_logic_vector (15 downto 0);
781      signal cc          : std_logic_vector (7 downto 0);
782      signal clk2        : std_logic;
783      signal flags       : std_logic_vector (1 downto 0);
784      signal IR          : std_logic_vector (31 downto 0);
785      signal M           : std_logic_vector (15 downto 0);
786      signal PC_Out      : std_logic_vector (15 downto 0);
787      signal Pila        : std_logic_vector (3 downto 0);
788      signal reloj2      : std_logic;
789      signal salida      : std_logic_vector (15 downto 0);
790      signal XLXN_1      : std_logic;
791      signal XLXN_3      : std_logic;
792      signal XLXN_14     : std_logic;
793      signal XLXN_15     : std_logic;
794      signal XLXN_33     : std_logic;
795      signal XLXN_34     : std_logic;
796      component unidad_de_control
797         port ( reloj       : in    std_logic;
798                Pin_dato    : in    std_logic_vector (7 downto 0);
799                Dato_in     : in    std_logic_vector (15 downto 0);
800                flags       : in    std_logic_vector (1 downto 0);
801                PC_Out      : out   std_logic_vector (15 downto 0);
802                N_WE        : out   std_logic;
```

```
803               N_RESET     : out   std_logic;
804               N_CE        : out   std_logic;
805               N_OE        : out   std_logic;
806               Pin_address : out   std_logic_vector (13 downto 0);
807               IR          : out   std_logic_vector (31 downto 0);
808               Pila        : out   std_logic_vector (3 downto 0);
809               M           : out   std_logic_vector (15 downto 0));
810      end component;
811
812      component unidad_de_proceso
813         port ( literal_in          : in    std_logic_vector (15 downto
       0);
814               C_Selec_Campo3      : in    std_logic;
815               C_Cuenta            : in    std_logic;
816               C_Selec_CampoSalida : in    std_logic;
817               campo3              : in    std_logic_vector (3 downto
      0);
818               campo2              : in    std_logic_vector (3 downto
      0);
819               campo1              : in    std_logic_vector (3 downto
      0);
820               C_Load              : in    std_logic;
821               C_SDRAM_in          : in    std_logic_vector (2 downto
      0);
822               Pila                : in    std_logic_vector (3 downto
      0);
823               C_Status_Load       : in    std_logic;
824               C_Mux_Salida        : in    std_logic_vector (1 downto
      0);
825               clk                 : in    std_logic;
826               sclkfb              : in    std_logic;
827               selec_oper          : in    std_logic_vector (3 downto
      0);
828               C_Ent_Sal           : in    std_logic;
829               id_campo2           : in    std_logic_vector (1 downto
      0);
830               id_campo1           : in    std_logic_vector (1 downto
      0);
831               C_Mux_Canal1        : in    std_logic;
832               C_Mux_Canal2        : in    std_logic;
833               C_subrutina         : in    std_logic;
834               PC_In               : in    std_logic_vector (15 downto
       0);
835               C_SDRAM_bi          : inout std_logic_vector (15 downto
       0);
836               ent_sal             : out   std_logic_vector (15 downto
       0);
837               clk2                : out   std_logic;
838               salida              : out   std_logic_vector (15 downto
       0);
839               C_SDRAM_out         : out   std_logic_vector (21 downto
       0);
840               canal1              : out   std_logic_vector (15 downto
       0);
841               flags               : out   std_logic_vector (1 downto
      0));
842      end component;
843
```

```
844      component CB16CE_MXILINX_procesador
845         port ( C   : in    std_logic;
846                CE  : in    std_logic;
847                CLR : in    std_logic;
848                CEO : out   std_logic;
849                Q   : out   std_logic_vector (15 downto 0);
850                TC  : out   std_logic);
851      end component;
852
853      component GND
854         port ( G : out   std_logic);
855      end component;
856      attribute BOX_TYPE of GND : component is "BLACK_BOX";
857
858      component VCC
859         port ( P : out   std_logic);
860      end component;
861      attribute BOX_TYPE of VCC : component is "BLACK_BOX";
862
863      component CB8CE_MXILINX_procesador
864         port ( C   : in    std_logic;
865                CE  : in    std_logic;
866                CLR : in    std_logic;
867                CEO : out   std_logic;
868                Q   : out   std_logic_vector (7 downto 0);
869                TC  : out   std_logic);
870      end component;
871
872      component OBUF
873         port ( I : in    std_logic;
874                O : out   std_logic);
875      end component;
876      attribute BOX_TYPE of OBUF : component is "BLACK_BOX";
877
878      component CB2CE_MXILINX_procesador
879         port ( C   : in    std_logic;
880                CE  : in    std_logic;
881                CLR : in    std_logic;
882                CEO : out   std_logic;
883                Q0  : out   std_logic;
884                Q1  : out   std_logic;
885                TC  : out   std_logic);
886      end component;
887
888      attribute HU_SET of XLXI_20 : label is "XLXI_20_27";
889      attribute HU_SET of XLXI_34 : label is "XLXI_34_26";
890      attribute HU_SET of XLXI_50 : label is "XLXI_50_28";
891   begin
892      Unidad_de_Control_principal : unidad_de_control
893         port map (Dato_in(15 downto 0)=>salida(15 downto 0),
894                   flags(1 downto 0)=>flags(1 downto 0),
895                   Pin_dato(7 downto 0)=>Pin_dato(7 downto 0),
896                   reloj=>reloj2,
897                   IR(31 downto 0)=>IR(31 downto 0),
898                   M(15 downto 0)=>M(15 downto 0),
899                   N_CE=>N_CE,
900                   N_OE=>N_OE,
901                   N_RESET=>N_RESET,
```

```
902                N_WE=>N_WE,
903                PC_Out(15 downto 0)=>PC_Out(15 downto 0),
904                Pila(3 downto 0)=>Pila(3 downto 0),
905                Pin_address(13 downto 0)=>Pin_address(13 downto 0));
906
907     Unidad_de_proceso_Principal : unidad_de_proceso
908        port map (campo1(3 downto 0)=>IR(21 downto 18),
909                  campo2(3 downto 0)=>IR(15 downto 12),
910                  campo3(3 downto 0)=>IR(25 downto 22),
911                  clk=>clk,
912                  C_Cuenta=>M(2),
913                  C_Ent_Sal=>M(13),
914                  C_Load=>M(1),
915                  C_Mux_Canal1=>M(3),
916                  C_Mux_Canal2=>M(4),
917                  C_Mux_Salida(1 downto 0)=>M(12 downto 11),
918                  C_SDRAM_in(2 downto 0)=>M(10 downto 8),
919                  C_Selec_CampoSalida=>M(0),
920                  C_Selec_Campo3=>M(6),
921                  C_Status_Load=>M(7),
922                  C_subrutina=>M(5),
923                  id_campo1(1 downto 0)=>IR(17 downto 16),
924                  id_campo2(1 downto 0)=>IR(11 downto 10),
925                  literal_in(15 downto 0)=>IR(15 downto 0),
926                  PC_In(15 downto 0)=>PC_Out(15 downto 0),
927                  Pila(3 downto 0)=>Pila(3 downto 0),
928                  sclkfb=>sclkfb,
929                  selec_oper(3)=>IR(31),
930                  selec_oper(2 downto 0)=>IR(29 downto 27),
931                  canal1=>open,
932                  clk2=>clk2,
933                  C_SDRAM_out(21 downto 0)=>C_SDRAM_out(21 downto 0),

934                  ent_sal(15 downto 0)=>sal(15 downto 0),
935                  flags(1 downto 0)=>flags(1 downto 0),
936                  salida(15 downto 0)=>salida(15 downto 0),
937                  C_SDRAM_bi(15 downto 0)=>C_SDRAM_bi(15 downto 0));
938
939     XLXI_20 : CB16CE_MXILINX_procesador
940        port map (C=>clk2,
941                  CE=>XLXN_14,
942                  CLR=>XLXN_1,
943                  CEO=>XLXN_33,
944                  Q(15 downto 0)=>bc(15 downto 0),
945                  TC=>open);
946
947     XLXI_23 : GND
948        port map (G=>XLXN_1);
949
950     XLXI_25 : GND
951        port map (G=>XLXN_3);
952
953     XLXI_31 : VCC
954        port map (P=>XLXN_34);
955
956     XLXI_33 : GND
957        port map (G=>XLXN_15);
958
```

```
959      XLXI_34 : CB8CE_MXILINX_procesador
960         port map (C=>clk2,
961                   CE=>XLXN_34,
962                   CLR=>XLXN_15,
963                   CEO=>XLXN_14,
964                   Q(7 downto 0)=>cc(7 downto 0),
965                   TC=>open);
966
967      XLXI_49 : OBUF
968         port map (I=>reloj2,
969                   O=>reloj);
970
971      XLXI_50 : CB2CE_MXILINX_procesador
972         port map (C=>clk2,
973                   CE=>XLXN_33,
974                   CLR=>XLXN_3,
975                   CEO=>open,
976                   Q0=>reloj2,
977                   Q1=>open,
978                   TC=>open);
979
980      XLXI_52 : OBUF
981         port map (I=>flags(0),
982                   O=>acarreo);
983
984      XLXI_53 : OBUF
985         port map (I=>flags(1),
986                   O=>zero);
987
988   end BEHAVIORAL;
989
990
991
```

```
1     Release 6.3.03i Map G.38
2     Xilinx Mapping Report File for Design 'procesador'
3
4     Design Information
5     ------------------
6     Command Line   : C:/Archivos de programa/Xilinx/bin/nt/map.exe -ints
      tyle ise -p
7     xc2s50-tq144-5 -cm area -pr b -k 4 -c 100 -tx off -o procesador_map.
      ncd
8     procesador.ngd procesador.pcf
9     Target Device  : x2s50
10    Target Package : tq144
11    Target Speed   : -5
12    Mapper Version : spartan2 -- $Revision: 1.16.8.2 $
13    Mapped Date    : Tue Mar 08 13:31:03 2005
14
15    Design Summary
16    --------------
17    Number of errors:      0
18    Number of warnings:    1
19    Logic Utilization:
20      Number of Slice Flip Flops:       318 out of  1,536   20%
21      Number of 4 input LUTs:           907 out of  1,536   59%
22    Logic Distribution:
23        Number of occupied Slices:                          766 out of
       768   99%
24        Number of Slices containing only related logic:     729 out of
       766   95%
25        Number of Slices containing unrelated logic:        37 out of
       766    4%
26           *See NOTES below for an explanation of the effects of unrela
      ted logic
27    Total Number 4 input LUTs:        1,447 out of  1,536   94%
28          Number used as logic:                       907
29          Number used as a route-thru:                 28
30          Number used as 16x1 ROMs:                   512
31       Number of bonded IOBs:            83 out of     92   90%
32          IOB Flip Flops:                              58
33       Number of Tbufs:                 736 out of    832   88%
34       Number of GCLKs:                   2 out of      4   50%
35       Number of GCLKIOBs:                2 out of      4   50%
36       Number of DLLs:                    2 out of      4   50%
37
38       Number of RPM macros:            3
39    Total equivalent gate count for design:  42,497
40    Additional JTAG gate count for IOBs:  4,080
41    Peak Memory Usage:  75 MB
42
43    NOTES:
44
45       Related logic is defined as being logic that shares connectivity
      -
46       e.g. two LUTs are "related" if they share common inputs.
47       When assembling slices, Map gives priority to combine logic that
48       is related.  Doing so results in the best timing performance.
49
50       Unrelated logic shares no connectivity.  Map will only begin
51       packing unrelated logic into a slice once 99% of the slices are
```

```
52       occupied through related logic packing.
53
54       Note that once logic distribution reaches the 99% level through
55       related logic packing, this does not mean the device is completel
     y
56       utilized.  Unrelated logic packing will then begin, continuing un
     til
57       all usable LUTs and FFs are occupied.  Depending on your timing
58       budget, increased levels of unrelated logic packing may adversely
59       affect the overall timing performance of your design.
60
61
62    Table of Contents
63    -----------------
64    Section 1 - Errors
65    Section 2 - Warnings
66    Section 3 - Informational
67    Section 4 - Removed Logic Summary
68    Section 5 - Removed Logic
69    Section 6 - IOB Properties
70    Section 7 - RPMs
71    Section 8 - Guide Report
72    Section 9 - Area Group Summary
73    Section 10 - Modular Design Summary
74    Section 11 - Timing Report
75    Section 12 - Configuration String Information
76    Section 13 - Additional Device Resource Counts
77
78    Section 1 - Errors
79    ------------------
80
81    Section 2 - Warnings
82    --------------------
83    WARNING:DesignRules:372 - Netcheck: Gated clock. Clock net
84       Unidad_de_proceso_Principal_Memoria_SDRAM_total_Controlador_SDRAM
     _clkin_1 is
85       sourced by a combinatorial pin. This is not good design practice.
      Use the CE
86       pin to control the loading of data into the flip-flop.
87
88    Section 3 - Informational
89    -------------------------
90    INFO:LIT:95 - All of the external outputs in this design are using s
     lew rate
91       limited output drivers. The delay on speed critical outputs can b
     e
92       dramatically reduced by designating them as fast outputs in the s
     chematic.
93    INFO:MapLib:562 - No environment variables are currently set.
94
95    Section 4 - Removed Logic Summary
96    ---------------------------------
97     107 block(s) removed
98      37 block(s) optimized away
99     100 signal(s) removed
100
101   Section 5 - Removed Logic
102   -------------------------
```

```
103
104   The trimmed logic report below shows the logic removed from your des
      ign due to
105   sourceless or loadless signals, and VCC or ground connections.  If t
      he removal
106   of a signal or symbol results in the subsequent removal of an additi
      onal signal
107   or symbol, the message explaining that second removal will be indent
      ed.  This
108   indentation will be repeated as a chain of related logic is removed.
109
110   To quickly locate the original cause for the removal of a chain of l
      ogic, look
111   above the place where that logic is listed in the trimming report, t
      hen locate
112   the lines that are least indented (begin at the leftmost edge).
113
114   Loadless block "Unidad_de_Control_principal_Gestion_de_saltos_XLXI_3
      3_XLXI_5"
115   (OR) removed.
116    The signal "Unidad_de_Control_principal_Gestion_de_saltos_XLXI_33_X
      LXN_11" is
117   loadless and has been removed.
118     Loadless block "Unidad_de_Control_principal_Gestion_de_saltos_XLXI
      _33_XLXI_26"
119   (AND) removed.
120    The signal "Unidad_de_Control_principal_Gestion_de_saltos_XLXI_33_X
      LXN_9" is
121   loadless and has been removed.
122     Loadless block "Unidad_de_Control_principal_Gestion_de_saltos_XLXI
      _33_XLXI_24"
123   (AND) removed.
124      The signal "Unidad_de_Control_principal_Gestion_de_saltos_S_relat
      ivo<0>" is
125   loadless and has been removed.
126       Loadless block
127   "Unidad_de_Control_principal_Gestion_de_saltos_Suma_PC_offset/I_36_2
      26" (XOR)
128   removed.
129   Loadless block "Unidad_de_Control_principal_Gestion_de_saltos_buf2_1
      5" (BUF)
130   removed.
131   Loadless block
132   "Unidad_de_proceso_Principal_Memoria_SDRAM_total_Controlador_SDRAM_i
      nt_clk2x_buf
133   " (CKBUF) removed.
134    The signal
135   "Unidad_de_proceso_Principal_Memoria_SDRAM_total_Controlador_SDRAM_i
      nt_clk2x" is
136   loadless and has been removed.
137   Loadless block "Unidad_de_proceso_Principal_Memoria_SDRAM_total_buf_
      dire2_6"
138   (BUF) removed.
139    The signal "Unidad_de_proceso_Principal_Memoria_SDRAM_total_Q<6>" i
      s loadless
140   and has been removed.
141     Loadless block
142   "Unidad_de_proceso_Principal_Memoria_SDRAM_total_ff_extension_dire_I
```

```
_Q6" (FF)
removed.
Loadless block "Unidad_de_proceso_Principal_banco_de_registros_buf1_
0" (BUF)
removed.
Loadless block "Unidad_de_proceso_Principal_banco_de_registros_buf1_
1" (BUF)
removed.
Loadless block "Unidad_de_proceso_Principal_banco_de_registros_buf1_
2" (BUF)
removed.
Loadless block "Unidad_de_proceso_Principal_banco_de_registros_buf1_
3" (BUF)
removed.
 The signal "Unidad_de_proceso_Principal_banco_de_registros_Y2<3>" i
s loadless
and has been removed.
  Loadless block
"Unidad_de_proceso_Principal_banco_de_registros_C_Selec_SalidaC2_buf
0_3" (BUF)
removed.
   The signal
"Unidad_de_proceso_Principal_banco_de_registros_C_Selec_SalidaC2_Y<3
>" is
loadless and has been removed.
    Loadless block
"Unidad_de_proceso_Principal_banco_de_registros_C_Selec_SalidaC2_XLX
I_63" (OR)
removed.
     The signal
"Unidad_de_proceso_Principal_banco_de_registros_C_Selec_SalidaC2_XLX
N_145" is
loadless and has been removed.
      Loadless block
"Unidad_de_proceso_Principal_banco_de_registros_C_Selec_SalidaC2_XLX
I_62" (AND)
removed.
       The signal "IR<21>" is loadless and has been removed.
        Loadless block "Unidad_de_Control_principal_buf2_21" (BUF) r
emoved.
         The signal "Unidad_de_Control_principal_XLXN_259<21>" is lo
adless and has been
removed.
          Loadless block "Unidad_de_Control_principal_Reg_IR_Reg_IRt
otal_XLXI_111/I_Q5"
(FF) removed.
           The signal "Unidad_de_Control_principal_Reg_IR_IR_Bus<21>
" is loadless and has
been removed.
            Loadless block "Unidad_de_Control_principal_Reg_IR_Reg_I
R1/I_Q5" (FF) removed.
     The signal
"Unidad_de_proceso_Principal_banco_de_registros_C_Selec_SalidaC2_XLX
N_143" is
loadless and has been removed.
      Loadless block
"Unidad_de_proceso_Principal_banco_de_registros_C_Selec_SalidaC2_XLX
I_61" (AND)
```

```
183   removed.
184   Loadless block "Unidad_de_proceso_Principal_banco_de_registros_buf2_
      0" (BUF)
185   removed.
186   Loadless block "Unidad_de_proceso_Principal_banco_de_registros_buf2_
      1" (BUF)
187   removed.
188   Loadless block "Unidad_de_proceso_Principal_banco_de_registros_buf2_
      2" (BUF)
189   removed.
190   Loadless block "Unidad_de_proceso_Principal_banco_de_registros_buf2_
      3" (BUF)
191   removed.
192    The signal "Unidad_de_proceso_Principal_banco_de_registros_Y1<3>" i
      s loadless
193   and has been removed.
194     Loadless block
195   "Unidad_de_proceso_Principal_banco_de_registros_C_Selec_SalidaC1_buf
      0_3" (BUF)
196   removed.
197      The signal
198   "Unidad_de_proceso_Principal_banco_de_registros_C_Selec_SalidaC1_Y<3
      >" is
199   loadless and has been removed.
200       Loadless block
201   "Unidad_de_proceso_Principal_banco_de_registros_C_Selec_SalidaC1_XLX
      I_63" (OR)
202   removed.
203        The signal
204   "Unidad_de_proceso_Principal_banco_de_registros_C_Selec_SalidaC1_XLX
      N_145" is
205   loadless and has been removed.
206         Loadless block
207   "Unidad_de_proceso_Principal_banco_de_registros_C_Selec_SalidaC1_XLX
      I_62" (AND)
208   removed.
209        The signal
210   "Unidad_de_proceso_Principal_banco_de_registros_C_Selec_SalidaC1_XLX
      N_143" is
211   loadless and has been removed.
212         Loadless block
213   "Unidad_de_proceso_Principal_banco_de_registros_C_Selec_SalidaC1_XLX
      I_61" (AND)
214   removed.
215          The signal "Unidad_de_proceso_Principal_banco_de_registros_XL
      XN_18<3>" is
216   loadless and has been removed.
217           Loadless block "Unidad_de_proceso_Principal_banco_de_registr
      os_XLXI_3_buf0_3"
218   (BUF) removed.
219            The signal "Unidad_de_proceso_Principal_banco_de_registros_
      XLXI_3_Y<3>" is
220   loadless and has been removed.
221             Loadless block "Unidad_de_proceso_Principal_banco_de_regis
      tros_XLXI_3_XLXI_63"
222   (OR) removed.
223              The signal "Unidad_de_proceso_Principal_banco_de_registro
      s_XLXI_3_XLXN_145" is
```

```
224   loadless and has been removed.
225                Loadless block "Unidad_de_proceso_Principal_banco_de_reg
      istros_XLXI_3_XLXI_62"
226   (AND) removed.
227                 The signal "IR<25>" is loadless and has been removed.
228                  Loadless block "Unidad_de_Control_principal_buf2_25" (
      BUF) removed.
229                   The signal "Unidad_de_Control_principal_XLXN_259<25>"
       is loadless and has been
230   removed.
231                    Loadless block "Unidad_de_Control_principal_Reg_IR_R
      eg_IRtotal_XLXI_111/I_Q9"
232   (FF) removed.
233                     The signal "Unidad_de_Control_principal_Reg_IR_IR_B
      us<25>" is loadless and has
234   been removed.
235                      Loadless block "Unidad_de_Control_principal_Reg_IR
      _Reg_IR0/I_Q1" (FF) removed.
236                The signal "Unidad_de_proceso_Principal_banco_de_registro
      s_XLXI_3_XLXN_143" is
237   loadless and has been removed.
238                Loadless block "Unidad_de_proceso_Principal_banco_de_reg
      istros_XLXI_3_XLXI_61"
239   (AND) removed.
240   The signal "XLXI_50/Q1" is sourceless and has been removed.
241    Sourceless block "XLXI_50/I_36_37" (AND) removed.
242     The signal "XLXI_50/TC" is sourceless and has been removed.
243      Sourceless block "XLXI_50/I_36_52" (AND) removed.
244       The signal "XLXI_50/CEO" is sourceless and has been removed.
245    Sourceless block "XLXI_50/I_Q1/I_36_32" (XOR) removed.
246     The signal "XLXI_50/I_Q1/TQ" is sourceless and has been removed.
247      Sourceless block "XLXI_50/I_Q1/I_36_35" (FF) removed.
248   The signal
249   "Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_Aritm_Lo
      g_Sumador_Re
250   stador/OFL" is sourceless and has been removed.
251   The signal
252   "Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_Aritm_Lo
      g_Sumador_Re
253   stador/dummy" is sourceless and has been removed.
254   The signal "Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/
      TC" is
255   sourceless and has been removed.
256    Sourceless block
257   "Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/I_36_50" (A
      ND) removed.
258     The signal "Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pil
      a/CEO" is
259   sourceless and has been removed.
260   The signal "Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/
      TC_DN" is
261   sourceless and has been removed.
262    Sourceless block
263   "Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/I_TC/I_36_7
      " (AND)
264   removed.
265     The signal "Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pil
      a/I_TC/M0" is
```

```
sourceless and has been removed.
   Sourceless block
"Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/I_TC/I_36_8
" (OR)
removed.
The signal "Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/
TC_UP" is
sourceless and has been removed.
 Sourceless block
"Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/I_TC/I_36_9
" (AND)
removed.
  The signal "Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pil
a/I_TC/M1" is
sourceless and has been removed.
The signal "Unidad_de_Control_principal_Reg_PC_PC/TC" is sourceless
and has been
removed.
 Sourceless block "Unidad_de_Control_principal_Reg_PC_PC/I_36_56" (A
ND) removed.
  The signal "Unidad_de_Control_principal_Reg_PC_PC/CEO" is sourcele
ss and has
been removed.
The signal "Unidad_de_Control_principal_Reg_PC_PC/OR_CE_L" is source
less and has
been removed.
The signal "Unidad_de_Control_principal_Gestion_de_saltos_Suma_PC_of
fset/CO" is
sourceless and has been removed.
 Sourceless block
"Unidad_de_Control_principal_Gestion_de_saltos_Suma_PC_offset/I_36_3
53" (XOR)
removed.
  The signal "Unidad_de_Control_principal_Gestion_de_saltos_Suma_PC_
offset/OFL" is
sourceless and has been removed.
The signal "Unidad_de_Control_principal_Gestion_de_saltos_Suma_PC_of
fset/dummy"
is sourceless and has been removed.
The signal
"Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
Q3" is
sourceless and has been removed.
 Sourceless block
"Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
I_36_87"
(AND) removed.
  The signal
"Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
TC" is
sourceless and has been removed.
   Sourceless block
"Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
I_36_107"
(AND) removed.
    The signal
"Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
CEO" is
```

```
307   sourceless and has been removed.
308    Sourceless block
309   "Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
      I_Q3/I_36_32
310   " (XOR) removed.
311     The signal
312   "Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
      I_Q3/TQ" is
313   sourceless and has been removed.
314      Sourceless block
315   "Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
      I_Q3/I_36_30
316   /I_36_7" (AND) removed.
317       The signal
318   "Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
      I_Q3/I_36_30
319   /M0" is sourceless and has been removed.
320        Sourceless block
321   "Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
      I_Q3/I_36_30
322   /I_36_8" (OR) removed.
323         The signal
324   "Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
      I_Q3/MD" is
325   sourceless and has been removed.
326          Sourceless block
327   "Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
      I_Q3/I_36_35
328   " (FF) removed.
329   The signal
330   "Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
      T3" is
331   sourceless and has been removed.
332   The signal
333   "Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
      OR_CE_L" is
334   sourceless and has been removed.
335   The signal
336   "Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
      I_Q3/I_36_30
337   /M1" is sourceless and has been removed.
338   The signal
339   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
340   _0/TC_DN" is sourceless and has been removed.
341    Sourceless block
342   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
343   _0/I_TC/I_36_7" (AND) removed.
344     The signal
345   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
346   _0/I_TC/M0" is sourceless and has been removed.
347      Sourceless block
348   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
349   _0/I_TC/I_36_8" (OR) removed.
350       The signal
```

```
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
233_Registro
_0/TC" is sourceless and has been removed.
     Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
233_Registro
_0/I_36_120" (AND) removed.
      The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
233_Registro
_0/CEO" is sourceless and has been removed.
The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
233_Registro
_0/TC_UP" is sourceless and has been removed.
 Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
233_Registro
_0/I_TC/I_36_9" (AND) removed.
  The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
233_Registro
_0/I_TC/M1" is sourceless and has been removed.
The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
233_Registro
_1/TC_DN" is sourceless and has been removed.
 Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
233_Registro
_1/I_TC/I_36_7" (AND) removed.
  The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
233_Registro
_1/I_TC/M0" is sourceless and has been removed.
   Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
233_Registro
_1/I_TC/I_36_8" (OR) removed.
    The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
233_Registro
_1/TC" is sourceless and has been removed.
     Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
233_Registro
_1/I_36_120" (AND) removed.
      The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
233_Registro
_1/CEO" is sourceless and has been removed.
The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
233_Registro
_1/TC_UP" is sourceless and has been removed.
 Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
233_Registro
```

```
394   _1/I_TC/I_36_9" (AND) removed.
395     The signal
396   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
397   _1/I_TC/M1" is sourceless and has been removed.
398   The signal
399   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
400   _2/TC_DN" is sourceless and has been removed.
401    Sourceless block
402   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
403   _2/I_TC/I_36_7" (AND) removed.
404     The signal
405   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
406   _2/I_TC/M0" is sourceless and has been removed.
407      Sourceless block
408   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
409   _2/I_TC/I_36_8" (OR) removed.
410       The signal
411   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
412   _2/TC" is sourceless and has been removed.
413        Sourceless block
414   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
415   _2/I_36_120" (AND) removed.
416         The signal
417   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
418   _2/CEO" is sourceless and has been removed.
419   The signal
420   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
421   _2/TC_UP" is sourceless and has been removed.
422    Sourceless block
423   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
424   _2/I_TC/I_36_9" (AND) removed.
425     The signal
426   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
427   _2/I_TC/M1" is sourceless and has been removed.
428   The signal
429   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
430   _3/TC_DN" is sourceless and has been removed.
431    Sourceless block
432   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
433   _3/I_TC/I_36_7" (AND) removed.
434     The signal
435   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
436   _3/I_TC/M0" is sourceless and has been removed.
437      Sourceless block
```

```
438  "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
     233_Registro
439  _3/I_TC/I_36_8" (OR) removed.
440      The signal
441  "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
     233_Registro
442  _3/TC" is sourceless and has been removed.
443       Sourceless block
444  "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
     233_Registro
445  _3/I_36_120" (AND) removed.
446        The signal
447  "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
     233_Registro
448  _3/CEO" is sourceless and has been removed.
449  The signal
450  "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
     233_Registro
451  _3/TC_UP" is sourceless and has been removed.
452   Sourceless block
453  "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
     233_Registro
454  _3/I_TC/I_36_9" (AND) removed.
455    The signal
456  "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
     233_Registro
457  _3/I_TC/M1" is sourceless and has been removed.
458  The signal
459  "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
     0_3_Registro
460  _0/TC_DN" is sourceless and has been removed.
461   Sourceless block
462  "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
     0_3_Registro
463  _0/I_TC/I_36_7" (AND) removed.
464    The signal
465  "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
     0_3_Registro
466  _0/I_TC/M0" is sourceless and has been removed.
467     Sourceless block
468  "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
     0_3_Registro
469  _0/I_TC/I_36_8" (OR) removed.
470      The signal
471  "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
     0_3_Registro
472  _0/TC" is sourceless and has been removed.
473       Sourceless block
474  "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
     0_3_Registro
475  _0/I_36_120" (AND) removed.
476        The signal
477  "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
     0_3_Registro
478  _0/CEO" is sourceless and has been removed.
479  The signal
480  "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
     0_3_Registro
```

```
_0/TC_UP" is sourceless and has been removed.
 Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_0/I_TC/I_36_9" (AND) removed.
  The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_0/I_TC/M1" is sourceless and has been removed.
The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_1/TC_DN" is sourceless and has been removed.
 Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_1/I_TC/I_36_7" (AND) removed.
  The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_1/I_TC/M0" is sourceless and has been removed.
   Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_1/I_TC/I_36_8" (OR) removed.
    The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_1/TC" is sourceless and has been removed.
     Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_1/I_36_120" (AND) removed.
      The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_1/CEO" is sourceless and has been removed.
The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_1/TC_UP" is sourceless and has been removed.
 Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_1/I_TC/I_36_9" (AND) removed.
  The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_1/I_TC/M1" is sourceless and has been removed.
The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_2/TC_DN" is sourceless and has been removed.
 Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_2/I_TC/I_36_7" (AND) removed.
  The signal
```

```
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_2/I_TC/M0" is sourceless and has been removed.
   Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_2/I_TC/I_36_8" (OR) removed.
    The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_2/TC" is sourceless and has been removed.
     Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_2/I_36_120" (AND) removed.
      The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_2/CEO" is sourceless and has been removed.
The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_2/TC_UP" is sourceless and has been removed.
 Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_2/I_TC/I_36_9" (AND) removed.
  The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_2/I_TC/M1" is sourceless and has been removed.
The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_3/TC_DN" is sourceless and has been removed.
 Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_3/I_TC/I_36_7" (AND) removed.
  The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_3/I_TC/M0" is sourceless and has been removed.
   Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_3/I_TC/I_36_8" (OR) removed.
    The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_3/TC" is sourceless and has been removed.
     Sourceless block
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
_3/I_36_120" (AND) removed.
      The signal
"Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
0_3_Registro
```

```
568   _3/CEO" is sourceless and has been removed.
569   The signal
570   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
      0_3_Registro
571   _3/TC_UP" is sourceless and has been removed.
572    Sourceless block
573   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
      0_3_Registro
574   _3/I_TC/I_36_9" (AND) removed.
575     The signal
576   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
      0_3_Registro
577   _3/I_TC/M1" is sourceless and has been removed.
578
579   The trimmed logic reported below is either:
580      1. part of a cycle
581      2. part of disabled logic
582      3. a side-effect of other trimmed logic
583
584   The signal "XLXN_15" is unused and has been removed.
585   The signal "XLXN_3" is unused and has been removed.
586   Unused block
587   "Unidad_de_Control_principal_Gestion_de_saltos_Suma_PC_offset/XST_GN
      D" (ZERO)
588   removed.
589   Unused block
590   "Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
      I_36_99"
591   (AND) removed.
592   Unused block
593   "Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/
      I_Q3/I_36_30
594   /I_36_9" (AND) removed.
595   Unused block "Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pil
      a/I_36_10"
596   (AND) removed.
597   Unused block "Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pil
      a/I_36_11"
598   (AND) removed.
599   Unused block "Unidad_de_Control_principal_Reg_PC_PC/I_36_2" (AND) re
      moved.
600   Unused block
601   "Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_Aritm_Lo
      g_Sumador_Re
602   stador/I_36_353" (XOR) removed.
603   Unused block
604   "Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_Aritm_Lo
      g_Sumador_Re
605   stador/XST_GND" (ZERO) removed.
606   Unused block
607   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
      0_3_Registro
608   _0/I_36_66" (AND) removed.
609   Unused block
610   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
      0_3_Registro
611   _0/I_36_68" (AND) removed.
612   Unused block
```

```
613   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
      0_3_Registro
614   _1/I_36_66" (AND) removed.
615   Unused block
616   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
      0_3_Registro
617   _1/I_36_68" (AND) removed.
618   Unused block
619   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
      0_3_Registro
620   _2/I_36_66" (AND) removed.
621   Unused block
622   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
      0_3_Registro
623   _2/I_36_68" (AND) removed.
624   Unused block
625   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
      0_3_Registro
626   _3/I_36_66" (AND) removed.
627   Unused block
628   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_
      0_3_Registro
629   _3/I_36_68" (AND) removed.
630   Unused block
631   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
632   _0/I_36_66" (AND) removed.
633   Unused block
634   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
635   _0/I_36_68" (AND) removed.
636   Unused block
637   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
638   _1/I_36_66" (AND) removed.
639   Unused block
640   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
641   _1/I_36_68" (AND) removed.
642   Unused block
643   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
644   _2/I_36_66" (AND) removed.
645   Unused block
646   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
647   _2/I_36_68" (AND) removed.
648   Unused block
649   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
650   _3/I_36_66" (AND) removed.
651   Unused block
652   "Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_
      233_Registro
653   _3/I_36_68" (AND) removed.
654   Unused block
655   "Unidad_de_Control_principal_Gestion_de_saltos_Suma_PC_offset/I_36_6
      4" (MUX)
```

```
656   removed.
657
658   Optimized Block(s):
659   TYPE           BLOCK
660   OR2
661              Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_
      micromem/I_36_120
662   VCC        Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_
      micromem/I_36_59
663   AND2
664              Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_
      micromem/I_Q0/I_36_3
665   0/I_36_9
666   AND2
667              Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_
      micromem/I_Q1/I_36_3
668   0/I_36_9
669   AND2
670              Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_
      micromem/I_Q2/I_36_3
671   0/I_36_9
672   VCC        Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/I_
      36_1
673   AND2
674              Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/I_
      Q0/I_36_30/I_36_9
675   AND2
676              Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/I_
      Q1/I_36_30/I_36_9
677   AND2
678              Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/I_
      Q2/I_36_30/I_36_9
679   AND2
680              Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/I_
      Q3/I_36_30/I_36_9
681   VCC        Unidad_de_Control_principal_Reg_PC_PC/I_36_31
682   OR2        Unidad_de_Control_principal_Reg_PC_PC/I_36_68
683   AND2            Unidad_de_Control_principal_Reg_PC_PC/I_Q0/I_36_30/I_
      36_9
684   VCC
685              Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad
      _Aritm_Log_nor16_1bi
686   t/I_36_107
687   GND
688              Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad
      _Aritm_Log_nor16_1bi
689   t/I_36_109
690   AND3
691              Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad
      _de_Desplaz_XLXI_4/I
692   _M23/I_36_30
693   AND3
694              Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad
      _de_Desplaz_XLXI_5/I
695   _M23/I_36_30
696   AND3b1
697              Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad
      _de_Desplaz_XLXI_5/I
```

```
698   _M23/I_36_31
699   OR2
700            Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad
      _de_Desplaz_XLXI_5/I
701   _M23/I_36_38
702   LUT1
703            Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad
      _de_Desplaz_XLXI_5/M
704   23_rt
705   VCC
706            Unidad_de_proceso_Principal_banco_de_registros_banco_regs_
      0_7_Regs_0_3_Registr
707   o_0/I_36_90
708   VCC
709            Unidad_de_proceso_Principal_banco_de_registros_banco_regs_
      0_7_Regs_0_3_Registr
710   o_1/I_36_90
711   VCC
712            Unidad_de_proceso_Principal_banco_de_registros_banco_regs_
      0_7_Regs_0_3_Registr
713   o_2/I_36_90
714   VCC
715            Unidad_de_proceso_Principal_banco_de_registros_banco_regs_
      0_7_Regs_0_3_Registr
716   o_3/I_36_90
717   VCC
718            Unidad_de_proceso_Principal_banco_de_registros_banco_regs_
      0_7_XLXI_233_Registr
719   o_0/I_36_90
720   VCC
721            Unidad_de_proceso_Principal_banco_de_registros_banco_regs_
      0_7_XLXI_233_Registr
722   o_1/I_36_90
723   VCC
724            Unidad_de_proceso_Principal_banco_de_registros_banco_regs_
      0_7_XLXI_233_Registr
725   o_2/I_36_90
726   VCC
727            Unidad_de_proceso_Principal_banco_de_registros_banco_regs_
      0_7_XLXI_233_Registr
728   o_3/I_36_90
729   VCC       XLXI_20/I_36_9
730   GND       XLXI_23
731   GND       XLXI_25
732   VCC       XLXI_31
733   GND       XLXI_33
734   VCC       XLXI_34/I_36_16
735   VCC       XLXI_50/I_36_47
736   GND       XST_GND
737   VCC       XST_VCC
738
739   To enable printing of redundant blocks removed and signals merged, s
      et the
740   detailed map report option and rerun map.
741
742   Section 6 - IOB Properties
743   --------------------------
744
```

IOB Name	Type	Direction	IO Standard	Drive Strength	Slew Rate	Reg (s)	Resistor	IOB Delay
clk	GCLKIOB	INPUT	LVTTL					
sclkfb	GCLKIOB	INPUT	LVTTL					
C_SDRAM_bi<0>	IOB	BIDIR	LVTTL	12	SLOW	INFF		
						OUTFF		
C_SDRAM_bi<1>	IOB	BIDIR	LVTTL	12	SLOW	INFF		
						OUTFF		
C_SDRAM_bi<2>	IOB	BIDIR	LVTTL	12	SLOW	INFF		
						OUTFF		
C_SDRAM_bi<3>	IOB	BIDIR	LVTTL	12	SLOW	INFF		
						OUTFF		
C_SDRAM_bi<4>	IOB	BIDIR	LVTTL	12	SLOW	INFF		
						OUTFF		
C_SDRAM_bi<5>	IOB	BIDIR	LVTTL	12	SLOW	INFF		
						OUTFF		
C_SDRAM_bi<6>	IOB	BIDIR	LVTTL	12	SLOW	INFF		
						OUTFF		
C_SDRAM_bi<7>	IOB	BIDIR	LVTTL	12	SLOW	INFF		
						OUTFF		
C_SDRAM_bi<8>	IOB	BIDIR	LVTTL	12	SLOW	INFF		
						OUTFF		
C_SDRAM_bi<9>	IOB	BIDIR	LVTTL	12	SLOW	INFF		
						OUTFF		
C_SDRAM_bi<10>	IOB	BIDIR	LVTTL	12	SLOW	INFF		
						OUTFF		
C_SDRAM_bi<11>	IOB	BIDIR	LVTTL	12	SLOW	INFF		

774									
				OUTFF					
775	C_SDRAM_bi<12>					IOB		BIDIR	LVTTL
		12	SLOW	INFF					
776									
				OUTFF					
777	C_SDRAM_bi<13>					IOB		BIDIR	LVTTL
		12	SLOW	INFF					
778									
				OUTFF					
779	C_SDRAM_bi<14>					IOB		BIDIR	LVTTL
		12	SLOW	INFF					
780									
				OUTFF					
781	C_SDRAM_bi<15>					IOB		BIDIR	LVTTL
		12	SLOW	INFF					
782									
				OUTFF					
783	C_SDRAM_out<0>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
784	C_SDRAM_out<1>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
785	C_SDRAM_out<2>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
786	C_SDRAM_out<3>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
787	C_SDRAM_out<4>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
788	C_SDRAM_out<5>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
789	C_SDRAM_out<6>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
790	C_SDRAM_out<7>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
791	C_SDRAM_out<8>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
792	C_SDRAM_out<9>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
793	C_SDRAM_out<10>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
794	C_SDRAM_out<11>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
795	C_SDRAM_out<12>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
796	C_SDRAM_out<13>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
797	C_SDRAM_out<14>					IOB		OUTPUT	LVTTL
		12	SLOW						
798	C_SDRAM_out<15>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
799	C_SDRAM_out<16>					IOB		OUTPUT	LVTTL
		12	SLOW						
800	C_SDRAM_out<17>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
801	C_SDRAM_out<18>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					
802	C_SDRAM_out<19>					IOB		OUTPUT	LVTTL
		12	SLOW	OUTFF					

```
803   | C_SDRAM_out<20>                       | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
804   | C_SDRAM_out<21>                       | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
805   | N_CE                                  | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
806   | N_OE                                  | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
807   | N_RESET                               | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
808   | N_WE                                  | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
809   | Pin_address<0>                        | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
810   | Pin_address<1>                        | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
811   | Pin_address<2>                        | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
812   | Pin_address<3>                        | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
813   | Pin_address<4>                        | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
814   | Pin_address<5>                        | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
815   | Pin_address<6>                        | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
816   | Pin_address<7>                        | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
817   | Pin_address<8>                        | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
818   | Pin_address<9>                        | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
819   | Pin_address<10>                       | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
820   | Pin_address<11>                       | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
821   | Pin_address<12>                       | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
822   | Pin_address<13>                       | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
823   | Pin_dato<0>                           | IOB     | INPUT     | LVTTL
          |          |      | INFF     |          | IFD   |
824   | Pin_dato<1>                           | IOB     | INPUT     | LVTTL
          |          |      | INFF     |          | IFD   |
825   | Pin_dato<2>                           | IOB     | INPUT     | LVTTL
          |          |      | INFF     |          | IFD   |
826   | Pin_dato<3>                           | IOB     | INPUT     | LVTTL
          |          |      | INFF     |          | IFD   |
827   | Pin_dato<4>                           | IOB     | INPUT     | LVTTL
          |          |      | INFF     |          | IFD   |
828   | Pin_dato<5>                           | IOB     | INPUT     | LVTTL
          |          |      | INFF     |          | IFD   |
829   | Pin_dato<6>                           | IOB     | INPUT     | LVTTL
          |          |      | INFF     |          | IFD   |
830   | Pin_dato<7>                           | IOB     | INPUT     | LVTTL
          |          |      | INFF     |          | IFD   |
831   | acarreo                               | IOB     | OUTPUT    | LVTTL
          | 12       | SLOW |          |          |       |
```

```
| reloj                                | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| sal<0>                               | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| sal<1>                               | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| sal<2>                               | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| sal<3>                               | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| sal<4>                               | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| sal<5>                               | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| sal<6>                               | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| sal<7>                               | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| sal<8>                               | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| sal<9>                               | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| sal<10>                              | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| sal<11>                              | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| sal<12>                              | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| sal<13>                              | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| sal<14>                              | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| sal<15>                              | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
| zero                                 | IOB     | OUTPUT    | LVTTL
    | 12       | SLOW |          |          |       |
+-------------------------------------------------------------------
-----------------------------------------------------+

Section 7 - RPMs
----------------
XLXI_50_28
XLXI_50/XLXI_50_I_Q1_1
XLXI_50/XLXI_50_I_Q0_0
XLXI_34_26
XLXI_34/XLXI_34_I_Q7_18
XLXI_34/XLXI_34_I_Q6_19
XLXI_34/XLXI_34_I_Q5_20
XLXI_34/XLXI_34_I_Q4_23
XLXI_34/XLXI_34_I_Q3_22
XLXI_34/XLXI_34_I_Q2_21
XLXI_34/XLXI_34_I_Q1_25
XLXI_34/XLXI_34_I_Q0_24
XLXI_20_27
XLXI_20/XLXI_20_I_Q9_11
XLXI_20/XLXI_20_I_Q8_10
XLXI_20/XLXI_20_I_Q7_6
XLXI_20/XLXI_20_I_Q6_7
```

```
871   XLXI_20/XLXI_20_I_Q5_8
872   XLXI_20/XLXI_20_I_Q4_9
873   XLXI_20/XLXI_20_I_Q3_4
874   XLXI_20/XLXI_20_I_Q2_5
875   XLXI_20/XLXI_20_I_Q1_2
876   XLXI_20/XLXI_20_I_Q15_17
877   XLXI_20/XLXI_20_I_Q14_16
878   XLXI_20/XLXI_20_I_Q13_15
879   XLXI_20/XLXI_20_I_Q12_14
880   XLXI_20/XLXI_20_I_Q11_13
881   XLXI_20/XLXI_20_I_Q10_12
882   XLXI_20/XLXI_20_I_Q0_3
883   Unidad_de_proceso_Principal_XLXI_96_XLXI_8_0
884   Unidad_de_proceso_Principal_XLXI_96_XLXI_7_3
885   Unidad_de_proceso_Principal_XLXI_96_buft_2_2
886   Unidad_de_proceso_Principal_XLXI_96_buft_1_1
887   Unidad_de_proceso_Principal_XLXI_77_0
888   Unidad_de_proceso_Principal_XLXI_112_XLXI_174_2
889   Unidad_de_proceso_Principal_XLXI_112_buft_2_1
890   Unidad_de_proceso_Principal_XLXI_112_buft_1_0
891   Unidad_de_proceso_Principal_XLXI_107_XLXI_174_2
892   Unidad_de_proceso_Principal_XLXI_107_buft_2_1
893   Unidad_de_proceso_Principal_XLXI_107_buft_1_0
894   Unidad_de_proceso_Principal_Unidad_de_funcion_total_XLXI_3_0
895   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_XLXI_5_2
896   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_XLXI_5/XLX
897   I_5_I_M23_0
898   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_XLXI_5/XLX
899   I_5_I_M01_1
900   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_XLXI_4_3
901   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_XLXI_4/XLX
902   I_4_I_M23_0
903   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_XLXI_4/XLX
904   I_4_I_M01_1
905   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_XLXI_17_4
906   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_Nucleo_Des
907   plazamiento_XLXI_37_14
908   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_Nucleo_Des
909   plazamiento_XLXI_36_13
910   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_Nucleo_Des
911   plazamiento_XLXI_35_12
912   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_Nucleo_Des
913   plazamiento_XLXI_34_11
914   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_Nucleo_Des
915   plazamiento_XLXI_33_10
916   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
```

```
916   z_Nucleo_Des
917   plazamiento_XLXI_32_9
918   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_Nucleo_Des
919   plazamiento_XLXI_31_8
920   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_Nucleo_Des
921   plazamiento_XLXI_30_7
922   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_Nucleo_Des
923   plazamiento_XLXI_29_2
924   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_Nucleo_Des
925   plazamiento_XLXI_28_6
926   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_Nucleo_Des
927   plazamiento_XLXI_27_5
928   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_Nucleo_Des
929   plazamiento_XLXI_26_4
930   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_Nucleo_Des
931   plazamiento_XLXI_25_3
932   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_Nucleo_Des
933   plazamiento_XLXI_24_1
934   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_Nucleo_Des
935   plazamiento_XLXI_23_0
936   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_de_Despla
      z_Nucleo_Des
937   plazamiento_XLXI_22_15
938   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_Aritm_Log
      _Sumador_Res
939   tador_1
940   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_Aritm_Log
      _nor16_1bit_
941   0
942   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_Aritm_Log
      _mux16_4a1t_
943   1_XLXI_8_0
944   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_Aritm_Log
      _mux16_4a1t_
945   1_XLXI_7_3
946   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_Aritm_Log
      _mux16_4a1t_
947   1_buft_2_2
948   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_Aritm_Log
      _mux16_4a1t_
949   1_buft_1_1
950   Unidad_de_proceso_Principal_Unidad_de_funcion_total_Unidad_Aritm_Log
      _inv16_1_2
951   Unidad_de_proceso_Principal_banco_de_registros_Salida_C2_XLXI_8_4
952   Unidad_de_proceso_Principal_banco_de_registros_Salida_C2_XLXI_7_7
953   Unidad_de_proceso_Principal_banco_de_registros_Salida_C2_XLXI_14_3
954   Unidad_de_proceso_Principal_banco_de_registros_Salida_C2_XLXI_13_2
955   Unidad_de_proceso_Principal_banco_de_registros_Salida_C2_XLXI_12_0
956   Unidad_de_proceso_Principal_banco_de_registros_Salida_C2_XLXI_11_1
```

```
957   Unidad_de_proceso_Principal_banco_de_registros_Salida_C2_buft_2_6
958   Unidad_de_proceso_Principal_banco_de_registros_Salida_C2_buft_1_5
959   Unidad_de_proceso_Principal_banco_de_registros_Salida_C1_XLXI_8_4
960   Unidad_de_proceso_Principal_banco_de_registros_Salida_C1_XLXI_7_7
961   Unidad_de_proceso_Principal_banco_de_registros_Salida_C1_XLXI_14_3
962   Unidad_de_proceso_Principal_banco_de_registros_Salida_C1_XLXI_13_2
963   Unidad_de_proceso_Principal_banco_de_registros_Salida_C1_XLXI_12_0
964   Unidad_de_proceso_Principal_banco_de_registros_Salida_C1_XLXI_11_1
965   Unidad_de_proceso_Principal_banco_de_registros_Salida_C1_buft_2_6
966   Unidad_de_proceso_Principal_banco_de_registros_Salida_C1_buft_1_5
967   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
968   3_36
969   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
970   3/Registro_3_I_TC_31
971   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
972   3/Registro_3_I_T9_18
973   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
974   3/Registro_3_I_T8_17
975   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
976   3/Registro_3_I_T7_29
977   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
978   3/Registro_3_I_T6_30
979   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
980   3/Registro_3_I_T5_28
981   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
982   3/Registro_3_I_T4_25
983   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
984   3/Registro_3_I_T3_26
985   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
986   3/Registro_3_I_T2_27
987   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
988   3/Registro_3_I_T1_32
989   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
990   3/Registro_3_I_T15_24
991   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
992   3/Registro_3_I_T14_23
993   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
994   3/Registro_3_I_T13_22
995   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
996   3/Registro_3_I_T12_21
997   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
998   3/Registro_3_I_T11_20
```

```
999   Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1000  3/Registro_3_I_T10_19
1001  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1002  3/Registro_3_I_Q9_5
1003  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1004  3/Registro_3_I_Q8_6
1005  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1006  3/Registro_3_I_Q7_7
1007  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1008  3/Registro_3_I_Q6_8
1009  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1010  3/Registro_3_I_Q5_9
1011  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1012  3/Registro_3_I_Q4_10
1013  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1014  3/Registro_3_I_Q3_11
1015  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1016  3/Registro_3_I_Q2_12
1017  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1018  3/Registro_3_I_Q1_13
1019  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1020  3/Registro_3_I_Q15_15
1021  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1022  3/Registro_3_I_Q14_16
1023  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1024  3/Registro_3_I_Q13_1
1025  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1026  3/Registro_3_I_Q12_2
1027  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1028  3/Registro_3_I_Q11_3
1029  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1030  3/Registro_3_I_Q10_4
1031  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1032  3/Registro_3_I_Q0_14
1033  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1034  3/I_Q9/I_Q9_I_36_30_0
1035  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1036  3/I_Q8/I_Q8_I_36_30_0
1037  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
```

```
1037  33_Registro_
1038  3/I_Q7/I_Q7_I_36_30_0
1039  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1040  3/I_Q6/I_Q6_I_36_30_0
1041  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1042  3/I_Q5/I_Q5_I_36_30_0
1043  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1044  3/I_Q4/I_Q4_I_36_30_0
1045  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1046  3/I_Q3/I_Q3_I_36_30_0
1047  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1048  3/I_Q2/I_Q2_I_36_30_0
1049  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1050  3/I_Q15/I_Q15_I_36_30_0
1051  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1052  3/I_Q14/I_Q14_I_36_30_0
1053  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1054  3/I_Q13/I_Q13_I_36_30_0
1055  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1056  3/I_Q12/I_Q12_I_36_30_0
1057  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1058  3/I_Q11/I_Q11_I_36_30_0
1059  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1060  3/I_Q10/I_Q10_I_36_30_0
1061  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1062  3/I_Q1/I_Q1_I_36_30_0
1063  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1064  3/I_Q0/I_Q0_I_36_30_0
1065  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1066  2_35
1067  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1068  2/Registro_2_I_TC_31
1069  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1070  2/Registro_2_I_T9_18
1071  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1072  2/Registro_2_I_T8_17
1073  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1074  2/Registro_2_I_T7_29
1075  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
```

```
1076  2/Registro_2_I_T6_30
1077  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1078  2/Registro_2_I_T5_28
1079  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1080  2/Registro_2_I_T4_25
1081  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1082  2/Registro_2_I_T3_26
1083  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1084  2/Registro_2_I_T2_27
1085  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1086  2/Registro_2_I_T1_32
1087  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1088  2/Registro_2_I_T15_24
1089  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1090  2/Registro_2_I_T14_23
1091  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1092  2/Registro_2_I_T13_22
1093  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1094  2/Registro_2_I_T12_21
1095  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1096  2/Registro_2_I_T11_20
1097  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1098  2/Registro_2_I_T10_19
1099  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1100  2/Registro_2_I_Q9_5
1101  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1102  2/Registro_2_I_Q8_6
1103  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1104  2/Registro_2_I_Q7_7
1105  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1106  2/Registro_2_I_Q6_8
1107  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1108  2/Registro_2_I_Q5_9
1109  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1110  2/Registro_2_I_Q4_10
1111  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1112  2/Registro_2_I_Q3_11
1113  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1114  2/Registro_2_I_Q2_12
```

```
1115  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1116  2/Registro_2_I_Q1_13
1117  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1118  2/Registro_2_I_Q15_15
1119  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1120  2/Registro_2_I_Q14_16
1121  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1122  2/Registro_2_I_Q13_1
1123  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1124  2/Registro_2_I_Q12_2
1125  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1126  2/Registro_2_I_Q11_3
1127  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1128  2/Registro_2_I_Q10_4
1129  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1130  2/Registro_2_I_Q0_14
1131  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1132  2/I_Q9/I_Q9_I_36_30_0
1133  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1134  2/I_Q8/I_Q8_I_36_30_0
1135  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1136  2/I_Q7/I_Q7_I_36_30_0
1137  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1138  2/I_Q6/I_Q6_I_36_30_0
1139  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1140  2/I_Q5/I_Q5_I_36_30_0
1141  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1142  2/I_Q4/I_Q4_I_36_30_0
1143  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1144  2/I_Q3/I_Q3_I_36_30_0
1145  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1146  2/I_Q2/I_Q2_I_36_30_0
1147  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1148  2/I_Q15/I_Q15_I_36_30_0
1149  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1150  2/I_Q14/I_Q14_I_36_30_0
1151  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1152  2/I_Q13/I_Q13_I_36_30_0
1153  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
```

```
1153  33_Registro_
1154  2/I_Q12/I_Q12_I_36_30_0
1155  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1156  2/I_Q11/I_Q11_I_36_30_0
1157  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1158  2/I_Q10/I_Q10_I_36_30_0
1159  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1160  2/I_Q1/I_Q1_I_36_30_0
1161  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1162  2/I_Q0/I_Q0_I_36_30_0
1163  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1164  1_34
1165  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1166  1/Registro_1_I_TC_31
1167  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1168  1/Registro_1_I_T9_18
1169  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1170  1/Registro_1_I_T8_17
1171  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1172  1/Registro_1_I_T7_29
1173  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1174  1/Registro_1_I_T6_30
1175  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1176  1/Registro_1_I_T5_28
1177  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1178  1/Registro_1_I_T4_25
1179  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1180  1/Registro_1_I_T3_26
1181  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1182  1/Registro_1_I_T2_27
1183  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1184  1/Registro_1_I_T1_32
1185  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1186  1/Registro_1_I_T15_24
1187  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1188  1/Registro_1_I_T14_23
1189  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1190  1/Registro_1_I_T13_22
1191  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
```

```
1192  1/Registro_1_I_T12_21
1193  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1194  1/Registro_1_I_T11_20
1195  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1196  1/Registro_1_I_T10_19
1197  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1198  1/Registro_1_I_Q9_5
1199  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1200  1/Registro_1_I_Q8_6
1201  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1202  1/Registro_1_I_Q7_7
1203  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1204  1/Registro_1_I_Q6_8
1205  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1206  1/Registro_1_I_Q5_9
1207  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1208  1/Registro_1_I_Q4_10
1209  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1210  1/Registro_1_I_Q3_11
1211  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1212  1/Registro_1_I_Q2_12
1213  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1214  1/Registro_1_I_Q1_13
1215  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1216  1/Registro_1_I_Q15_15
1217  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1218  1/Registro_1_I_Q14_16
1219  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1220  1/Registro_1_I_Q13_1
1221  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1222  1/Registro_1_I_Q12_2
1223  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1224  1/Registro_1_I_Q11_3
1225  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1226  1/Registro_1_I_Q10_4
1227  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1228  1/Registro_1_I_Q0_14
1229  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1230  1/I_Q9/I_Q9_I_36_30_0
```

```
1231  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1232  1/I_Q8/I_Q8_I_36_30_0
1233  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1234  1/I_Q7/I_Q7_I_36_30_0
1235  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1236  1/I_Q6/I_Q6_I_36_30_0
1237  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1238  1/I_Q5/I_Q5_I_36_30_0
1239  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1240  1/I_Q4/I_Q4_I_36_30_0
1241  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1242  1/I_Q3/I_Q3_I_36_30_0
1243  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1244  1/I_Q2/I_Q2_I_36_30_0
1245  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1246  1/I_Q15/I_Q15_I_36_30_0
1247  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1248  1/I_Q14/I_Q14_I_36_30_0
1249  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1250  1/I_Q13/I_Q13_I_36_30_0
1251  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1252  1/I_Q12/I_Q12_I_36_30_0
1253  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1254  1/I_Q11/I_Q11_I_36_30_0
1255  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1256  1/I_Q10/I_Q10_I_36_30_0
1257  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1258  1/I_Q1/I_Q1_I_36_30_0
1259  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1260  1/I_Q0/I_Q0_I_36_30_0
1261  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1262  0_33
1263  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1264  0/Registro_0_I_TC_31
1265  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1266  0/Registro_0_I_T9_18
1267  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1268  0/Registro_0_I_T8_17
1269  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
```

```
1269  33_Registro_
1270  0/Registro_0_I_T7_29
1271  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1272  0/Registro_0_I_T6_30
1273  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1274  0/Registro_0_I_T5_28
1275  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1276  0/Registro_0_I_T4_25
1277  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1278  0/Registro_0_I_T3_26
1279  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1280  0/Registro_0_I_T2_27
1281  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1282  0/Registro_0_I_T1_32
1283  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1284  0/Registro_0_I_T15_24
1285  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1286  0/Registro_0_I_T14_23
1287  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1288  0/Registro_0_I_T13_22
1289  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1290  0/Registro_0_I_T12_21
1291  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1292  0/Registro_0_I_T11_20
1293  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1294  0/Registro_0_I_T10_19
1295  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1296  0/Registro_0_I_Q9_5
1297  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1298  0/Registro_0_I_Q8_6
1299  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1300  0/Registro_0_I_Q7_7
1301  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1302  0/Registro_0_I_Q6_8
1303  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1304  0/Registro_0_I_Q5_9
1305  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1306  0/Registro_0_I_Q4_10
1307  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
```

```
1308  0/Registro_0_I_Q3_11
1309  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1310  0/Registro_0_I_Q2_12
1311  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1312  0/Registro_0_I_Q1_13
1313  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1314  0/Registro_0_I_Q15_15
1315  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1316  0/Registro_0_I_Q14_16
1317  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1318  0/Registro_0_I_Q13_1
1319  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1320  0/Registro_0_I_Q12_2
1321  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1322  0/Registro_0_I_Q11_3
1323  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1324  0/Registro_0_I_Q10_4
1325  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1326  0/Registro_0_I_Q0_14
1327  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1328  0/I_Q9/I_Q9_I_36_30_0
1329  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1330  0/I_Q8/I_Q8_I_36_30_0
1331  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1332  0/I_Q7/I_Q7_I_36_30_0
1333  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1334  0/I_Q6/I_Q6_I_36_30_0
1335  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1336  0/I_Q5/I_Q5_I_36_30_0
1337  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1338  0/I_Q4/I_Q4_I_36_30_0
1339  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1340  0/I_Q3/I_Q3_I_36_30_0
1341  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1342  0/I_Q2/I_Q2_I_36_30_0
1343  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1344  0/I_Q15/I_Q15_I_36_30_0
1345  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1346  0/I_Q14/I_Q14_I_36_30_0
```

```
1347  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1348  0/I_Q13/I_Q13_I_36_30_0
1349  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1350  0/I_Q12/I_Q12_I_36_30_0
1351  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1352  0/I_Q11/I_Q11_I_36_30_0
1353  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1354  0/I_Q10/I_Q10_I_36_30_0
1355  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1356  0/I_Q1/I_Q1_I_36_30_0
1357  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_XLXI_2
      33_Registro_
1358  0/I_Q0/I_Q0_I_36_30_0
1359  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1360  3_36
1361  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1362  3/Registro_3_I_TC_31
1363  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1364  3/Registro_3_I_T9_18
1365  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1366  3/Registro_3_I_T8_17
1367  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1368  3/Registro_3_I_T7_29
1369  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1370  3/Registro_3_I_T6_30
1371  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1372  3/Registro_3_I_T5_28
1373  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1374  3/Registro_3_I_T4_25
1375  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1376  3/Registro_3_I_T3_26
1377  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1378  3/Registro_3_I_T2_27
1379  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1380  3/Registro_3_I_T1_32
1381  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1382  3/Registro_3_I_T15_24
1383  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1384  3/Registro_3_I_T14_23
1385  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
```

```
1385  _3_Registro_
1386  3/Registro_3_I_T13_22
1387  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1388  3/Registro_3_I_T12_21
1389  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1390  3/Registro_3_I_T11_20
1391  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1392  3/Registro_3_I_T10_19
1393  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1394  3/Registro_3_I_Q9_5
1395  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1396  3/Registro_3_I_Q8_6
1397  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1398  3/Registro_3_I_Q7_7
1399  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1400  3/Registro_3_I_Q6_8
1401  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1402  3/Registro_3_I_Q5_9
1403  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1404  3/Registro_3_I_Q4_10
1405  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1406  3/Registro_3_I_Q3_11
1407  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1408  3/Registro_3_I_Q2_12
1409  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1410  3/Registro_3_I_Q1_13
1411  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1412  3/Registro_3_I_Q15_15
1413  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1414  3/Registro_3_I_Q14_16
1415  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1416  3/Registro_3_I_Q13_1
1417  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1418  3/Registro_3_I_Q12_2
1419  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1420  3/Registro_3_I_Q11_3
1421  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1422  3/Registro_3_I_Q10_4
1423  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
```

```
1424  3/Registro_3_I_Q0_14
1425  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1426  3/I_Q9/I_Q9_I_36_30_0
1427  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1428  3/I_Q8/I_Q8_I_36_30_0
1429  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1430  3/I_Q7/I_Q7_I_36_30_0
1431  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1432  3/I_Q6/I_Q6_I_36_30_0
1433  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1434  3/I_Q5/I_Q5_I_36_30_0
1435  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1436  3/I_Q4/I_Q4_I_36_30_0
1437  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1438  3/I_Q3/I_Q3_I_36_30_0
1439  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1440  3/I_Q2/I_Q2_I_36_30_0
1441  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1442  3/I_Q15/I_Q15_I_36_30_0
1443  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1444  3/I_Q14/I_Q14_I_36_30_0
1445  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1446  3/I_Q13/I_Q13_I_36_30_0
1447  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1448  3/I_Q12/I_Q12_I_36_30_0
1449  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1450  3/I_Q11/I_Q11_I_36_30_0
1451  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1452  3/I_Q10/I_Q10_I_36_30_0
1453  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1454  3/I_Q1/I_Q1_I_36_30_0
1455  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1456  3/I_Q0/I_Q0_I_36_30_0
1457  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1458  2_35
1459  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1460  2/Registro_2_I_TC_31
1461  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1462  2/Registro_2_I_T9_18
```

```
1463  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1464  2/Registro_2_I_T8_17
1465  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1466  2/Registro_2_I_T7_29
1467  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1468  2/Registro_2_I_T6_30
1469  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1470  2/Registro_2_I_T5_28
1471  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1472  2/Registro_2_I_T4_25
1473  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1474  2/Registro_2_I_T3_26
1475  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1476  2/Registro_2_I_T2_27
1477  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1478  2/Registro_2_I_T1_32
1479  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1480  2/Registro_2_I_T15_24
1481  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1482  2/Registro_2_I_T14_23
1483  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1484  2/Registro_2_I_T13_22
1485  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1486  2/Registro_2_I_T12_21
1487  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1488  2/Registro_2_I_T11_20
1489  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1490  2/Registro_2_I_T10_19
1491  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1492  2/Registro_2_I_Q9_5
1493  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1494  2/Registro_2_I_Q8_6
1495  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1496  2/Registro_2_I_Q7_7
1497  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1498  2/Registro_2_I_Q6_8
1499  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1500  2/Registro_2_I_Q5_9
1501  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
```

```
1501  _3_Registro_
1502  2/Registro_2_I_Q4_10
1503  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1504  2/Registro_2_I_Q3_11
1505  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1506  2/Registro_2_I_Q2_12
1507  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1508  2/Registro_2_I_Q1_13
1509  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1510  2/Registro_2_I_Q15_15
1511  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1512  2/Registro_2_I_Q14_16
1513  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1514  2/Registro_2_I_Q13_1
1515  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1516  2/Registro_2_I_Q12_2
1517  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1518  2/Registro_2_I_Q11_3
1519  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1520  2/Registro_2_I_Q10_4
1521  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1522  2/Registro_2_I_Q0_14
1523  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1524  2/I_Q9/I_Q9_I_36_30_0
1525  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1526  2/I_Q8/I_Q8_I_36_30_0
1527  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1528  2/I_Q7/I_Q7_I_36_30_0
1529  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1530  2/I_Q6/I_Q6_I_36_30_0
1531  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1532  2/I_Q5/I_Q5_I_36_30_0
1533  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1534  2/I_Q4/I_Q4_I_36_30_0
1535  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1536  2/I_Q3/I_Q3_I_36_30_0
1537  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1538  2/I_Q2/I_Q2_I_36_30_0
1539  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
```

```
1540  2/I_Q15/I_Q15_I_36_30_0
1541  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1542  2/I_Q14/I_Q14_I_36_30_0
1543  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1544  2/I_Q13/I_Q13_I_36_30_0
1545  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1546  2/I_Q12/I_Q12_I_36_30_0
1547  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1548  2/I_Q11/I_Q11_I_36_30_0
1549  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1550  2/I_Q10/I_Q10_I_36_30_0
1551  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1552  2/I_Q1/I_Q1_I_36_30_0
1553  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1554  2/I_Q0/I_Q0_I_36_30_0
1555  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1556  1_34
1557  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1558  1/Registro_1_I_TC_31
1559  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1560  1/Registro_1_I_T9_18
1561  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1562  1/Registro_1_I_T8_17
1563  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1564  1/Registro_1_I_T7_29
1565  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1566  1/Registro_1_I_T6_30
1567  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1568  1/Registro_1_I_T5_28
1569  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1570  1/Registro_1_I_T4_25
1571  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1572  1/Registro_1_I_T3_26
1573  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1574  1/Registro_1_I_T2_27
1575  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1576  1/Registro_1_I_T1_32
1577  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1578  1/Registro_1_I_T15_24
```

```
1579  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1580  1/Registro_1_I_T14_23
1581  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1582  1/Registro_1_I_T13_22
1583  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1584  1/Registro_1_I_T12_21
1585  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1586  1/Registro_1_I_T11_20
1587  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1588  1/Registro_1_I_T10_19
1589  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1590  1/Registro_1_I_Q9_5
1591  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1592  1/Registro_1_I_Q8_6
1593  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1594  1/Registro_1_I_Q7_7
1595  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1596  1/Registro_1_I_Q6_8
1597  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1598  1/Registro_1_I_Q5_9
1599  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1600  1/Registro_1_I_Q4_10
1601  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1602  1/Registro_1_I_Q3_11
1603  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1604  1/Registro_1_I_Q2_12
1605  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1606  1/Registro_1_I_Q1_13
1607  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1608  1/Registro_1_I_Q15_15
1609  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1610  1/Registro_1_I_Q14_16
1611  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1612  1/Registro_1_I_Q13_1
1613  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1614  1/Registro_1_I_Q12_2
1615  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1616  1/Registro_1_I_Q11_3
1617  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
```

```
1617  _3_Registro_
1618  1/Registro_1_I_Q10_4
1619  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1620  1/Registro_1_I_Q0_14
1621  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1622  1/I_Q9/I_Q9_I_36_30_0
1623  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1624  1/I_Q8/I_Q8_I_36_30_0
1625  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1626  1/I_Q7/I_Q7_I_36_30_0
1627  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1628  1/I_Q6/I_Q6_I_36_30_0
1629  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1630  1/I_Q5/I_Q5_I_36_30_0
1631  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1632  1/I_Q4/I_Q4_I_36_30_0
1633  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1634  1/I_Q3/I_Q3_I_36_30_0
1635  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1636  1/I_Q2/I_Q2_I_36_30_0
1637  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1638  1/I_Q15/I_Q15_I_36_30_0
1639  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1640  1/I_Q14/I_Q14_I_36_30_0
1641  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1642  1/I_Q13/I_Q13_I_36_30_0
1643  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1644  1/I_Q12/I_Q12_I_36_30_0
1645  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1646  1/I_Q11/I_Q11_I_36_30_0
1647  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1648  1/I_Q10/I_Q10_I_36_30_0
1649  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1650  1/I_Q1/I_Q1_I_36_30_0
1651  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1652  1/I_Q0/I_Q0_I_36_30_0
1653  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1654  0_33
1655  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
```

```
1656  0/Registro_0_I_TC_31
1657  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1658  0/Registro_0_I_T9_18
1659  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1660  0/Registro_0_I_T8_17
1661  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1662  0/Registro_0_I_T7_29
1663  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1664  0/Registro_0_I_T6_30
1665  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1666  0/Registro_0_I_T5_28
1667  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1668  0/Registro_0_I_T4_25
1669  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1670  0/Registro_0_I_T3_26
1671  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1672  0/Registro_0_I_T2_27
1673  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1674  0/Registro_0_I_T1_32
1675  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1676  0/Registro_0_I_T15_24
1677  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1678  0/Registro_0_I_T14_23
1679  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1680  0/Registro_0_I_T13_22
1681  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1682  0/Registro_0_I_T12_21
1683  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1684  0/Registro_0_I_T11_20
1685  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1686  0/Registro_0_I_T10_19
1687  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1688  0/Registro_0_I_Q9_5
1689  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1690  0/Registro_0_I_Q8_6
1691  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1692  0/Registro_0_I_Q7_7
1693  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1694  0/Registro_0_I_Q6_8
```

```
1695  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1696  0/Registro_0_I_Q5_9
1697  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1698  0/Registro_0_I_Q4_10
1699  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1700  0/Registro_0_I_Q3_11
1701  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1702  0/Registro_0_I_Q2_12
1703  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1704  0/Registro_0_I_Q1_13
1705  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1706  0/Registro_0_I_Q15_15
1707  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1708  0/Registro_0_I_Q14_16
1709  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1710  0/Registro_0_I_Q13_1
1711  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1712  0/Registro_0_I_Q12_2
1713  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1714  0/Registro_0_I_Q11_3
1715  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1716  0/Registro_0_I_Q10_4
1717  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1718  0/Registro_0_I_Q0_14
1719  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1720  0/I_Q9/I_Q9_I_36_30_0
1721  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1722  0/I_Q8/I_Q8_I_36_30_0
1723  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1724  0/I_Q7/I_Q7_I_36_30_0
1725  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1726  0/I_Q6/I_Q6_I_36_30_0
1727  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1728  0/I_Q5/I_Q5_I_36_30_0
1729  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1730  0/I_Q4/I_Q4_I_36_30_0
1731  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1732  0/I_Q3/I_Q3_I_36_30_0
1733  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
```

```
1733  _3_Registro_
1734  0/I_Q2/I_Q2_I_36_30_0
1735  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1736  0/I_Q15/I_Q15_I_36_30_0
1737  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1738  0/I_Q14/I_Q14_I_36_30_0
1739  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1740  0/I_Q13/I_Q13_I_36_30_0
1741  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1742  0/I_Q12/I_Q12_I_36_30_0
1743  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1744  0/I_Q11/I_Q11_I_36_30_0
1745  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1746  0/I_Q10/I_Q10_I_36_30_0
1747  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1748  0/I_Q1/I_Q1_I_36_30_0
1749  Unidad_de_proceso_Principal_banco_de_registros_banco_regs_0_7_Regs_0
      _3_Registro_
1750  0/I_Q0/I_Q0_I_36_30_0
1751  Unidad_de_Control_principal_Reg_PC_PC_17
1752  Unidad_de_Control_principal_Reg_PC_PC/Reg_PC_PC_I_Q9_2
1753  Unidad_de_Control_principal_Reg_PC_PC/Reg_PC_PC_I_Q8_1
1754  Unidad_de_Control_principal_Reg_PC_PC/Reg_PC_PC_I_Q7_9
1755  Unidad_de_Control_principal_Reg_PC_PC/Reg_PC_PC_I_Q6_10
1756  Unidad_de_Control_principal_Reg_PC_PC/Reg_PC_PC_I_Q5_11
1757  Unidad_de_Control_principal_Reg_PC_PC/Reg_PC_PC_I_Q4_12
1758  Unidad_de_Control_principal_Reg_PC_PC/Reg_PC_PC_I_Q3_13
1759  Unidad_de_Control_principal_Reg_PC_PC/Reg_PC_PC_I_Q2_14
1760  Unidad_de_Control_principal_Reg_PC_PC/Reg_PC_PC_I_Q1_15
1761  Unidad_de_Control_principal_Reg_PC_PC/Reg_PC_PC_I_Q15_8
1762  Unidad_de_Control_principal_Reg_PC_PC/Reg_PC_PC_I_Q14_7
1763  Unidad_de_Control_principal_Reg_PC_PC/Reg_PC_PC_I_Q13_6
1764  Unidad_de_Control_principal_Reg_PC_PC/Reg_PC_PC_I_Q12_5
1765  Unidad_de_Control_principal_Reg_PC_PC/Reg_PC_PC_I_Q11_4
1766  Unidad_de_Control_principal_Reg_PC_PC/Reg_PC_PC_I_Q10_3
1767  Unidad_de_Control_principal_Reg_PC_PC/Reg_PC_PC_I_Q0_16
1768  Unidad_de_Control_principal_Reg_PC_PC/I_Q9/I_Q9_I_36_30_0
1769  Unidad_de_Control_principal_Reg_PC_PC/I_Q8/I_Q8_I_36_30_0
1770  Unidad_de_Control_principal_Reg_PC_PC/I_Q7/I_Q7_I_36_30_0
1771  Unidad_de_Control_principal_Reg_PC_PC/I_Q6/I_Q6_I_36_30_0
1772  Unidad_de_Control_principal_Reg_PC_PC/I_Q5/I_Q5_I_36_30_0
1773  Unidad_de_Control_principal_Reg_PC_PC/I_Q4/I_Q4_I_36_30_0
1774  Unidad_de_Control_principal_Reg_PC_PC/I_Q3/I_Q3_I_36_30_0
1775  Unidad_de_Control_principal_Reg_PC_PC/I_Q2/I_Q2_I_36_30_0
1776  Unidad_de_Control_principal_Reg_PC_PC/I_Q15/I_Q15_I_36_30_0
1777  Unidad_de_Control_principal_Reg_PC_PC/I_Q14/I_Q14_I_36_30_0
1778  Unidad_de_Control_principal_Reg_PC_PC/I_Q13/I_Q13_I_36_30_0
1779  Unidad_de_Control_principal_Reg_PC_PC/I_Q12/I_Q12_I_36_30_0
1780  Unidad_de_Control_principal_Reg_PC_PC/I_Q11/I_Q11_I_36_30_0
1781  Unidad_de_Control_principal_Reg_PC_PC/I_Q10/I_Q10_I_36_30_0
1782  Unidad_de_Control_principal_Reg_PC_PC/I_Q1/I_Q1_I_36_30_0
```

```
1783  Unidad_de_Control_principal_Reg_PC_PC/I_Q0/I_Q0_I_36_30_0
1784  Unidad_de_Control_principal_Reg_IR_Reg_IRtotal_XLXI_4_0
1785  Unidad_de_Control_principal_Reg_IR_Reg_IRtotal_XLXI_111_1
1786  Unidad_de_Control_principal_Reg_IR_Reg_IR3_2
1787  Unidad_de_Control_principal_Reg_IR_Reg_IR2_3
1788  Unidad_de_Control_principal_Reg_IR_Reg_IR1_1
1789  Unidad_de_Control_principal_Reg_IR_Reg_IR0_0
1790  Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila_9
1791  Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/Reg_Cont_Pil
      a_Contador_P
1792  ila_I_TC_7
1793  Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/Reg_Cont_Pil
      a_Contador_P
1794  ila_I_T3_6
1795  Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/Reg_Cont_Pil
      a_Contador_P
1796  ila_I_T2_5
1797  Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/Reg_Cont_Pil
      a_Contador_P
1798  ila_I_T1_8
1799  Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/Reg_Cont_Pil
      a_Contador_P
1800  ila_I_Q3_1
1801  Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/Reg_Cont_Pil
      a_Contador_P
1802  ila_I_Q2_2
1803  Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/Reg_Cont_Pil
      a_Contador_P
1804  ila_I_Q1_3
1805  Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/Reg_Cont_Pil
      a_Contador_P
1806  ila_I_Q0_4
1807  Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/I_Q3/I_Q3_I_
      36_30_0
1808  Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/I_Q2/I_Q2_I_
      36_30_0
1809  Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/I_Q1/I_Q1_I_
      36_30_0
1810  Unidad_de_Control_principal_Reg_Cont_Pila_Contador_Pila/I_Q0/I_Q0_I_
      36_30_0
1811  Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem_5
1812  Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/S
      ecuenciador_
1813  micromem_I_Q3_4
1814  Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/S
      ecuenciador_
1815  micromem_I_Q2_3
1816  Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/S
      ecuenciador_
1817  micromem_I_Q1_2
1818  Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/S
      ecuenciador_
1819  micromem_I_Q0_1
1820  Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/I
      _Q3/I_Q3_I_3
1821  6_30_0
1822  Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/I
      _Q2/I_Q2_I_3
```

```
1823  6_30_0
1824  Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/I
      _Q1/I_Q1_I_3
1825  6_30_0
1826  Unidad_de_Control_principal_Micromemoria_ROM_Secuenciador_micromem/I
      _Q0/I_Q0_I_3
1827  6_30_0
1828  Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_mux
      16_16a1t_1_X
1829  LXI_8_12
1830  Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_mux
      16_16a1t_1_X
1831  LXI_7_15
1832  Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_mux
      16_16a1t_1_X
1833  LXI_196_7
1834  Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_mux
      16_16a1t_1_X
1835  LXI_195_6
1836  Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_mux
      16_16a1t_1_X
1837  LXI_194_5
1838  Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_mux
      16_16a1t_1_X
1839  LXI_193_4
1840  Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_mux
      16_16a1t_1_X
1841  LXI_192_3
1842  Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_mux
      16_16a1t_1_X
1843  LXI_191_2
1844  Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_mux
      16_16a1t_1_X
1845  LXI_190_1
1846  Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_mux
      16_16a1t_1_X
1847  LXI_189_0
1848  Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_mux
      16_16a1t_1_X
1849  LXI_14_11
1850  Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_mux
      16_16a1t_1_X
1851  LXI_13_10
1852  Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_mux
      16_16a1t_1_X
1853  LXI_12_8
1854  Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_mux
      16_16a1t_1_X
1855  LXI_11_9
1856  Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_mux
      16_16a1t_1_b
1857  uft_2_14
1858  Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_mux
      16_16a1t_1_b
1859  uft_1_13
1860  Unidad_de_Control_principal_Gestion_de_saltos_Suma_PC_offset_0
1861
1862  Section 8 - Guide Report
```

```
1863  ------------------------
1864  Guide not run on this design.
1865
1866  Section 9 - Area Group Summary
1867  ------------------------------
1868  No area groups were found in this design.
1869
1870  Section 10 - Modular Design Summary
1871  -----------------------------------
1872  Modular Design not used for this design.
1873
1874  Section 11 - Timing Report
1875  --------------------------
1876  This design was not run using timing mode.
1877
1878  Section 12 - Configuration String Details
1879  -----------------------------------------
1880  Use the "-detail" map option to print out Configuration Strings
1881
1882  Section 13 - Additional Device Resource Counts
1883  ----------------------------------------------
1884  Number of JTAG Gates for IOBs = 85
1885  Number of Equivalent Gates for Design = 42,497
1886  Number of RPM Macros = 3
1887  Number of Hard Macros = 0
1888  PCI IOBs = 0
1889  PCI LOGICs = 0
1890  CAPTUREs = 0
1891  BSCANs = 0
1892  STARTUPs = 0
1893  DLLs = 2
1894  GCLKIOBs = 2
1895  GCLKs = 2
1896  Block RAMs = 0
1897  TBUFs = 736
1898  Total Registers (Flops & Latches in Slices & IOBs) not driven by LUT
      s = 123
1899  IOB Latches not driven by LUTs = 0
1900  IOB Latches = 0
1901  IOB Flip Flops not driven by LUTs = 44
1902  IOB Flip Flops = 58
1903  Unbonded IOBs = 0
1904  Bonded IOBs = 83
1905  Shift Registers = 0
1906  Static Shift Registers = 0
1907  Dynamic Shift Registers = 0
1908  16x1 ROMs = 512
1909  16x1 RAMs = 0
1910  32x1 RAMs = 0
1911  Dual Port RAMs = 0
1912  MULTANDs = 0
1913  MUXF5s + MUXF6s = 260
1914  4 input LUTs used as Route-Thrus = 28
1915  4 input LUTs = 907
1916  Slice Latches not driven by LUTs = 0
1917  Slice Latches = 0
1918  Slice Flip Flops not driven by LUTs = 79
1919  Slice Flip Flops = 318
```

```
1920  Slices = 766
1921  Number of LUT signals with 4 loads = 5
1922  Number of LUT signals with 3 loads = 6
1923  Number of LUT signals with 2 loads = 102
1924  Number of LUT signals with 1 load = 658
1925  NGM Average fanout of LUT = 4.07
1926  NGM Maximum fanout of LUT = 100
1927  NGM Average fanin for LUT = 3.2194
1928  Number of LUT symbols = 907
1929  Number of BIPAD symbols = 16
1930
```

```
1     Release 6.3.03i - Bitgen G.38
2     Copyright (c) 1995-2004 Xilinx, Inc.  All rights reserved.
3
4     Loading device database for application Bitgen from file"procesador
      .ncd".
5        "procesador" is an NCD, version 2.38, device xc2s50, package tq14
      4, speed -5
6     Loading device for application Bitgen from file'v50.nph' in environ
      ment
7     C:/Archivos de programa/Xilinx.
8     Opened constraints file procesador.pcf.
9
10    Tue Mar 08 13:31:28 2005
11
12    C:/Archivos de programa/Xilinx/bin/nt/bitgen.exe -intstyle ise -w -g
       DebugBitstream:No -g Binary:no -g Gclkdel0:11111 -g Gclkdel1:11111
      -g Gclkdel2:11111 -g Gclkdel3:11111 -g ConfigRate:4 -g CclkPin:PullU
      p -g M0Pin:PullUp -g M1Pin:PullUp -g M2Pin:PullUp -g ProgPin:PullUp
      -g DonePin:PullUp -g TckPin:PullUp -g TdiPin:PullUp -g TdoPin:PullUp
       -g TmsPin:PullUp -g UnusedPin:PullDown -g UserID:0xFFFFFFFF -g Star
      tUpClk:CClk -g DONE_cycle:4 -g GTS_cycle:5 -g GSR_cycle:6 -g GWE_cyc
      le:6 -g LCK_cycle:NoWait -g Security:None -g DonePipe:No -g DriveDon
      e:No procesador.ncd
13
14    Summary of Bitgen Options:
15    +----------------------+----------------------+
16    | Option Name          | Current Setting      |
17    +----------------------+----------------------+
18    | Compress             | (Not Specified)*     |
19    +----------------------+----------------------+
20    | Readback             | (Not Specified)*     |
21    +----------------------+----------------------+
22    | DebugBitstream       | No**                 |
23    +----------------------+----------------------+
24    | ConfigRate           | 4**                  |
25    +----------------------+----------------------+
26    | StartupClk           | Cclk**               |
27    +----------------------+----------------------+
28    | CclkPin              | Pullup**             |
29    +----------------------+----------------------+
30    | DonePin              | Pullup**             |
31    +----------------------+----------------------+
32    | M0Pin                | Pullup**             |
33    +----------------------+----------------------+
34    | M1Pin                | Pullup**             |
35    +----------------------+----------------------+
36    | M2Pin                | Pullup**             |
37    +----------------------+----------------------+
38    | ProgPin              | Pullup**             |
39    +----------------------+----------------------+
40    | TckPin               | Pullup**             |
41    +----------------------+----------------------+
42    | TdiPin               | Pullup**             |
43    +----------------------+----------------------+
44    | TdoPin               | Pullup               |
45    +----------------------+----------------------+
46    | TmsPin               | Pullup**             |
47    +----------------------+----------------------+
```

UnusedPin	Pulldown**
GSR_cycle	6**
GWE_cycle	6**
GTS_cycle	5**
LCK_cycle	NoWait**
DONE_cycle	4**
Persist	No*
DriveDone	No**
DonePipe	No**
Security	None**
UserID	0xFFFFFFFF**
Gclkdel0	11111**
Gclkdel1	11111**
Gclkdel2	11111**
Gclkdel3	11111**
ActiveReconfig	No*
ActivateGclk	No*
PartialMask0	(Not Specified)*
PartialMask1	(Not Specified)*
PartialGclk	(Not Specified)*
PartialLeft	(Not Specified)*
PartialRight	(Not Specified)*
IEEE1532	No*
Binary	No**

 * Default setting.
 ** The specified setting matches the default setting

Running DRC.
WARNING:DesignRules:372 - Netcheck: Gated clock. Clock net
   Unidad_de_proceso_Principal_Memoria_SDRAM_total_Controlador_SDRAM_clkin_1 is
   sourced by a combinatorial pin. This is not good design practice. Use the CE
   pin to control the loading of data into the flip-flop.

```
104   DRC detected 0 errors and 1 warnings.
105   Creating bit map...
106   Saving bit stream in "procesador.bit".
107   Bitstream generation is complete.
108
```

```
#PACE: Start of Constraints generated by PACE
#PACE: Start of PACE I/O Pin Assignments
NET "C_SDRAM_bi<0>" LOC = "p95";
NET "C_SDRAM_bi<10>" LOC = "p116";
NET "C_SDRAM_bi<11>" LOC = "p114";
NET "C_SDRAM_bi<12>" LOC = "p112";
NET "C_SDRAM_bi<13>" LOC = "p102";
NET "C_SDRAM_bi<14>" LOC = "p100";
NET "C_SDRAM_bi<15>" LOC = "p96";
NET "C_SDRAM_bi<1>" LOC = "p99";
NET "C_SDRAM_bi<2>" LOC = "p101";
NET "C_SDRAM_bi<3>" LOC = "p103";
NET "C_SDRAM_bi<4>" LOC = "p113";
NET "C_SDRAM_bi<5>" LOC = "p115";
NET "C_SDRAM_bi<6>" LOC = "p117";
NET "C_SDRAM_bi<7>" LOC = "p120";
NET "C_SDRAM_bi<8>" LOC = "p121";
NET "C_SDRAM_bi<9>" LOC = "p118";
NET "C_SDRAM_out<0>" LOC = "p141";
NET "C_SDRAM_out<10>" LOC = "p139";
NET "C_SDRAM_out<11>" LOC = "p136";
NET "C_SDRAM_out<12>" LOC = "p134";
NET "C_SDRAM_out<13>" LOC = "p137";
NET "C_SDRAM_out<14>" LOC = "p129";
NET "C_SDRAM_out<15>" LOC = "p131";
NET "C_SDRAM_out<16>" LOC = "p132";
NET "C_SDRAM_out<17>" LOC = "p130";
NET "C_SDRAM_out<18>" LOC = "p126";
NET "C_SDRAM_out<19>" LOC = "p123";
NET "C_SDRAM_out<1>" LOC = "p4";
NET "C_SDRAM_out<20>" LOC = "p124";
NET "C_SDRAM_out<21>" LOC = "p122";
NET "C_SDRAM_out<2>" LOC = "p6";
NET "C_SDRAM_out<3>" LOC = "p10";
NET "C_SDRAM_out<4>" LOC = "p11";
NET "C_SDRAM_out<5>" LOC = "p7";
NET "C_SDRAM_out<6>" LOC = "p5";
NET "C_SDRAM_out<7>" LOC = "p3";
NET "C_SDRAM_out<8>" LOC = "p140";
NET "C_SDRAM_out<9>" LOC = "p138";
NET "clk" LOC = "p88";
NET "N_CE" LOC = "p41";
NET "N_OE" LOC = "p43";
NET "N_RESET" LOC = "p59";
NET "N_WE" LOC = "p58";
NET "pin_address<0>" LOC = "p40";
NET "pin_address<10>" LOC = "p42";
NET "pin_address<11>" LOC = "p47";
NET "pin_address<12>" LOC = "p65";
NET "pin_address<13>" LOC = "p51";
NET "pin_address<1>" LOC = "p29";
NET "pin_address<2>" LOC = "p28";
NET "pin_address<3>" LOC = "p27";
NET "pin_address<4>" LOC = "p74";
NET "pin_address<5>" LOC = "p75";
NET "pin_address<6>" LOC = "p76";
NET "pin_address<7>" LOC = "p66";
NET "pin_address<8>" LOC = "p50";
NET "pin_address<9>" LOC = "p48";
NET "Pin_dato<0>" LOC = "p39";
NET "Pin_dato<1>" LOC = "p44";
NET "Pin_dato<2>" LOC = "p46";
NET "Pin_dato<3>" LOC = "p49";
NET "Pin_dato<4>" LOC = "p57";
NET "Pin_dato<5>" LOC = "p60";
NET "Pin_dato<6>" LOC = "p62";
NET "Pin_dato<7>" LOC = "p67";
```

```
NET "reloj" LOC = "p26";
NET "acarreo" LOC = "p23";
NET "zero" LOC = "p30";
NET "sal<0>" LOC = "p22";
NET "sal<10>" LOC = "p80";
NET "sal<11>" LOC = "p83";
NET "sal<12>" LOC = "p84";
NET "sal<13>" LOC = "p85";
NET "sal<14>" LOC = "p86";
NET "sal<15>" LOC = "p87";
NET "sal<1>" LOC = "p21";
NET "sal<2>" LOC = "p20";
NET "sal<3>" LOC = "p19";
NET "sal<4>" LOC = "p13";
NET "sal<5>" LOC = "p12";
NET "sal<6>" LOC = "p94";
NET "sal<7>" LOC = "p93";
NET "sal<8>" LOC = "p77";
NET "sal<9>" LOC = "p79";
NET "sclkfb" LOC = "p91";
#PACE: Start of PACE Area Constraints
#PACE: Start of PACE Prohibit Constraints
#PACE: End of Constraints generated by PACE
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_0_MROM0"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_0_MROM1"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_0_MROM2"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_0_MROM3"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_0_MROM4"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_0_MROM5"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_0_MROM6"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_0_MROM7"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_0_MROM8"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_0_MROM9"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_0_MROM10
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_0_MROM11
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_0_MROM12
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_0_MROM13
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_0_MROM14
```

```
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_0_MROM15
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_1_MROM0"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_1_MROM1"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_1_MROM2"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_1_MROM3"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_1_MROM4"
INIT =  0000FF00;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_1_MROM5"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_1_MROM6"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_1_MROM7"
INIT =  00000808;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_1_MROM8"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_1_MROM9"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_1_MROM10
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_1_MROM11
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_1_MROM12
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_1_MROM13
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_1_MROM14
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_1_MROM15
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_2_MROM0"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_2_MROM1"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_2_MROM2"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_2_MROM3"
INIT =  FFFF0000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_2_MROM4"
INIT =  FF0000FF;
```

```
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_2_MROM5"
INIT =  FF000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_2_MROM6"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_2_MROM7"
INIT =  00000008;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_2_MROM8"
INIT =  00FF0000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_2_MROM9"
INIT =  80000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_2_MROM10
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_2_MROM11
" INIT = 00FF0000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_2_MROM12
" INIT = FFFF0000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_2_MROM13
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_2_MROM14
" INIT = 80800000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_2_MROM15
" INIT = 80800000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_3_MROM0"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_3_MROM1"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_3_MROM2"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_3_MROM3"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_3_MROM4"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_3_MROM5"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_3_MROM6"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_3_MROM7"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_3_MROM8"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_3_MROM9"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_3_MROM10
" INIT = 00000000;
INST
```

```
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_3_MROM11
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_3_MROM12
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_3_MROM13
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_3_MROM14
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_3_MROM15
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_4_MROM0"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_4_MROM1"
INIT =  80808080;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_4_MROM2"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_4_MROM3"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_4_MROM4"
INIT =  FF00FF00;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_4_MROM5"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_4_MROM6"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_4_MROM7"
INIT =  40404040;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_4_MROM8"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_4_MROM9"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_4_MROM10
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_4_MROM11
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_4_MROM12
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_4_MROM13
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_4_MROM14
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_4_MROM15
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_5_MROM0"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_5_MROM1"
```

```
INIT =  80808080;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_5_MROM2"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_5_MROM3"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_5_MROM4"
INIT =  FF00FF00;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_5_MROM5"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_5_MROM6"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_5_MROM7"
INIT =  40404040;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_5_MROM8"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_5_MROM9"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_5_MROM10
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_5_MROM11
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_5_MROM12
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_5_MROM13
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_5_MROM14
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_5_MROM15
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_6_MROM0"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_6_MROM1"
INIT =  80808080;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_6_MROM2"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_6_MROM3"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_6_MROM4"
INIT =  FF00FF00;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_6_MROM5"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_6_MROM6"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_6_MROM7"
INIT =  00000000;
```

```
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_6_MROM8"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_6_MROM9"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_6_MROM10
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_6_MROM11
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_6_MROM12
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_6_MROM13
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_6_MROM14
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_6_MROM15
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_7_MROM0"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_7_MROM1"
INIT =  00808080;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_7_MROM2"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_7_MROM3"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_7_MROM4"
INIT =  0000FF00;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_7_MROM5"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_7_MROM6"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_7_MROM7"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_7_MROM8"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_7_MROM9"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_7_MROM10
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_7_MROM11
" INIT = 00FF0000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_7_MROM12
" INIT = 00FF0000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_7_MROM13
" INIT = 4E000000;
INST
```

```
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_7_MROM14
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_7_MROM15
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_8_MROM0"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_8_MROM1"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_8_MROM2"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_8_MROM3"
INIT =  FF00FF00;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_8_MROM4"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_8_MROM5"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_8_MROM6"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_8_MROM7"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_8_MROM8"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_8_MROM9"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_8_MROM10
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_8_MROM11
" INIT = FFFFFFFF;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_8_MROM12
" INIT = FFFFFFFF;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_8_MROM13
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_8_MROM14
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_8_MROM15
" INIT = 80808080;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_9_MROM0"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_9_MROM1"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_9_MROM2"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_9_MROM3"
INIT =  FF00FF00;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_9_MROM4"
```

```
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_9_MROM5"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_9_MROM6"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_9_MROM7"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_9_MROM8"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_9_MROM9"
INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_9_MROM10
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_9_MROM11
" INIT = FFFFFFFF;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_9_MROM12
" INIT = FFFFFFFF;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_9_MROM13
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_9_MROM14
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_9_MROM15
" INIT = 80808080;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_10_MROM0
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_10_MROM1
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_10_MROM2
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_10_MROM3
" INIT = 0000FF00;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_10_MROM4
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_10_MROM5
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_10_MROM6
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_10_MROM7
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_10_MROM8
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_10_MROM9
" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_10_MROM1
0" INIT = 00000000;
```

```
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_10_MROM1
1" INIT = 0000FFFF;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_10_MROM1
2" INIT = 0000FFFF;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_10_MROM1
3" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_10_MROM1
4" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_10_MROM1
5" INIT = 00008080;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_11_MROM0
" INIT =  0007FF03;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_11_MROM1
" INIT =  80000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_11_MROM2
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_11_MROM3
" INIT =  0000FF00;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_11_MROM4
" INIT =  00F80000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_11_MROM5
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_11_MROM6
" INIT =  00FF00FF;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_11_MROM7
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_11_MROM8
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_11_MROM9
" INIT =  00F8F0E0;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_11_MROM1
0" INIT = 00030002;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_11_MROM1
1" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_11_MROM1
2" INIT = FF000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_11_MROM1
3" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_11_MROM1
4" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_11_MROM1
5" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_12_MROM0
" INIT =  00000000;
INST
```

```
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_12_MROM1
" INIT =  00800080;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_12_MROM2
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_12_MROM3
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_12_MROM4
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_12_MROM5
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_12_MROM6
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_12_MROM7
" INIT =  00800080;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_12_MROM8
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_12_MROM9
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_12_MROM1
0" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_12_MROM1
1" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_12_MROM1
2" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_12_MROM1
3" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_12_MROM1
4" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_12_MROM1
5" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_13_MROM0
" INIT =  000300F0;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_13_MROM1
" INIT =  80808080;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_13_MROM2
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_13_MROM3
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_13_MROM4
" INIT =  00F80000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_13_MROM5
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_13_MROM6
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_13_MROM7
```

```
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_13_MROM8
" INIT =  00F000E0;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_13_MROM9
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_13_MROM1
0" INIT = 00030003;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_13_MROM1
1" INIT = 00FF00FF;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_13_MROM1
2" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_13_MROM1
3" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_13_MROM1
4" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_13_MROM1
5" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_14_MROM0
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_14_MROM1
" INIT =  00800080;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_14_MROM2
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_14_MROM3
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_14_MROM4
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_14_MROM5
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_14_MROM6
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_14_MROM7
" INIT =  00080008;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_14_MROM8
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_14_MROM9
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_14_MROM1
0" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_14_MROM1
1" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_14_MROM1
2" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_14_MROM1
3" INIT = 00000000;
```

```
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_14_MROM1
4" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_14_MROM1
5" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_15_MROM0
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_15_MROM1
" INIT =  00800080;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_15_MROM2
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_15_MROM3
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_15_MROM4
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_15_MROM5
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_15_MROM6
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_15_MROM7
" INIT =  00080008;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_15_MROM8
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_15_MROM9
" INIT =  00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_15_MROM1
0" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_15_MROM1
1" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_15_MROM1
2" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_15_MROM1
3" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_15_MROM1
4" INIT = 00000000;
INST
"Unidad_de_Control_principal_Micromemoria_ROM_matriz_micromemoria_celda_15_MROM1
5" INIT = 00000000;
```

Bibliografía

[1] Jean P. Deschamps, José M. Angulo, *Diseño de sistemas digitales: metodología moderna*, 1992.

[2] M. Morris Mano, C.R. Kime, *Logic and computer design fundamentals*, 2001.

[3] H. Taub, *Circuitos digitales y microprocesadores*, 1993.

[4] R.J. Tocci, *Sistemas digitales: principios y aplicaciones*, 1996.

[5] V.P. Nelson, H.T. Tagle, B.D. Carroll, J.D. Irwin, *Análisis y diseño de circuitos lógicos digitales*, 1996.

[6] S. Aguilera Navarro, J. M. Meneses Chaus, *Sistemas digitales : arquitecturas de procesadores específicos*, 1990.

[7] A. Lloris Ruiz, A. Prieto Espinosa, L. Parrilla Roure, *Sistemas digitales*, 2003.

[8] E. Mandado, L.J. Álvarez Ruiz de Ojeda, Mª. Dolores Valdés, *Dispositivos Lógicos programables.*, 2000.

[9] L.J. Álvarez Ruiz de Ojeda, *Diseño de Aplicaciones mediante PLDs y FPGAs*, 2001.

[10] S. Brown, J. Rose, *FPGA and CPLD Architectures: A tutorial.*

[11] M. Jacomet, *RISC Processor Design.*

[12] V. Jiménez Luengo, J. Ozalla Calzada, *Desarrollo de Sistemas basados en FPGA*, 2003.

[13] J. Gray, *Designing a Simple FPGA-Optimized RISC CPU and System-on-a-Chip*, 2000.

[14] Xilinx, *ISE 6 In-Depth Tutorial.*

[15] Xilinx, *ISE Quick Start Tutorial.*

[16] Xilinx, *Spartan-II 2.5V FPGA Family: Complete Data Sheet.*

[17] Ediciones Técnicas Rede, *Spartan II: la nueva familia de FPGAs de Xilinx.*

[18] Xess, *XSA Board V1.1, V1.2 User Manual.*

[19] Xess, *XSA Board SDRAM Controller V1.2.*

[20] Samsung electronics, *K4S641632F CMOS SDRAM datasheet.*

[21] Samsung electronics, *SDRAM Device Operations.*

[22] Atmel, *Datasheet Flash RAM AT49F001.*

[23] Xess, *XSA Flash Programming and SpartanII Configuration.*