diseño e implementación de un computador autorreconfigurable

Laura Prada
Rubén del Cura
Cristina Hernández
Eduardo Duque

Facultad de Informática
Universidad Complutense de Madrid
Septiembre de 2002

ÍNDICE

1. INTRODUCCIÓN

1.1 OBJETIVOS INICIALES

El objetivo inicial del proyecto (Diseño e implementación de un computador autorreconfigurable) es dotar a un microprocesador RISC convencional (en estecaso el R2000) de capacidades de autoreconfiguración parcial dinámica, entendiendo por ésta la posibilidad de que el microprocesador pueda cambiar por sí mismo una porción de su diseño hardware sin dejar de ejecutar instrucciones. En particular, se pretende incorporar a su repertorio una colección de instrucciones que permitan reconfigurar dinámicamente el diseño HW de su memoria cache: tamaño de bloque, política de emplazamiento, política de reemplazamiento, etc de manera que se disponga de un sistema que por software pueda adaptar la configuración de su cache a una carga de trabajo dada.

Se parte inicialmente de una implementación completa sobre FPGAs (dispositivo hardware reconfigurable) de un computador formado por un microprocesador RISC, un subsistema jerárquico de memoria de 2 niveles, un subsistema de E/S y un sistema operativo. Por ello las tareas a realizar en principio son:

1. Diseño de un protocolo de configuración de FPGAs, puesto que la virtex es una fpga reconfigurable.

2. Diseño de un protocolo de configuración dinámica de FPGAs, esto es posible debido a que la placa permite cargar sólo una parte del programa a ejecutar, con lo que no es necesario que el computador entero desaparezca de la placa para llevar a cabo la reconfiguración.

3. Diseño de un protocolo de autoreconfiguración dinámica de FPGAs, este punto es básicamente un refinamiento del primero, de manera que la orden de reconfiguración parta del procesador, y no sea independiente del mismo.

4. Rediseño del microprocesador para incorporarle capacidades de autoreconfiguracion, factible puesto que con las instrucciones necesarias, es posible dotar al micro de la potencia necesaria para analizar el código y decidir en cada momento qué tipo de memoria cache va a determinar un mejor rendimiento del código.

5. Comparación de los rendimientos de un computador con cache de diseño fijo y de un computador con cache con diseño autoreconfigurable.

Además de estas tareas de inicio, se incorporan otras debido a ciertas deficiencias que presenta el sistema de memorias de dos niveles del computador de partida. En concreto la memoria cache, debido a que, en el momento de inicio del proyecto, no se dispone de dispositivos de este tipo que funcionen adecuadamente. Por tanto, cabe contar con un punto extra que atañe al diseño de diferentes tipos de cache. En un principio el objetivo fue ese, pero, tras varios estudios, se comprueba la inviabilidad de añadir un dispositivo de esta índole al computador disponible (las características de la ruta de datos y del computador, suponen una barrera para ello), y por tanto, para utilizar memorias cache propiamente dichas, se hace necesario reformar el computador de partida (la ruta de datos y el propio microcomputador). Al final, se adopta un nuevo compromiso que se describe en el siguiente documento.

Los objetivos trazados se justifican por las características de los componentes software y hardware que se utilizan. Así, la placa es reconfigurable, y por tanto se puede aspirar a la reconfiguración; además, es parcialmente reconfigurable, con lo que se puede conseguir teóricamente la reconfiguración parcial y dinámica. En cuanto al software disponible, las

herramientas de Xilinx están dotadas, según el fabricante, de funcionalidades que permiten distinguir la parte de los archivos que se quiere reconfigurar, aislarlos, generar los archivos en el formato adecuado y bajarlos a la placa. De momento, la realidad es otra, y el fabricante no aporta soluciones, por lo que se hace necesario, también en este punto, marcar unos nuevos objetivo acordes con las herramientas que se poseen, y sus funcionalidades.
Todas estas dificultades llevan a elaborar unos nuevos requisitos que se comentan detalladamente en el siguiente documento.

1.2 ARQUITECTURA MIPS

A modo de introducción, se presenta una breve descripción de la arquitectura MIPS, más concretamente del mips disponible. De este modo, el lector podrá comprender con mayor facilidad cada uno de los problemas que se trata de abordar en este proyecto, y, por tanto, también las soluciones que se aportan para resolverlos. La documentación que se aporta está basada en la implementación del MIPS del curso 2000-2001 [MaBP], y se completa con la documentación disponible sobre la arquitectura MIPS [JH94].

A continuación, se muestra una vista panorámica de la organización general del sistema, y se describen los módulos que se implementaron en el citado proyecto de partida.

NOTA: La RAM corresponde a la RAM de la FPGA, y PS2 se corresponde con el puerto PS2 de la placa.

1.2.1 Bus del sistema.

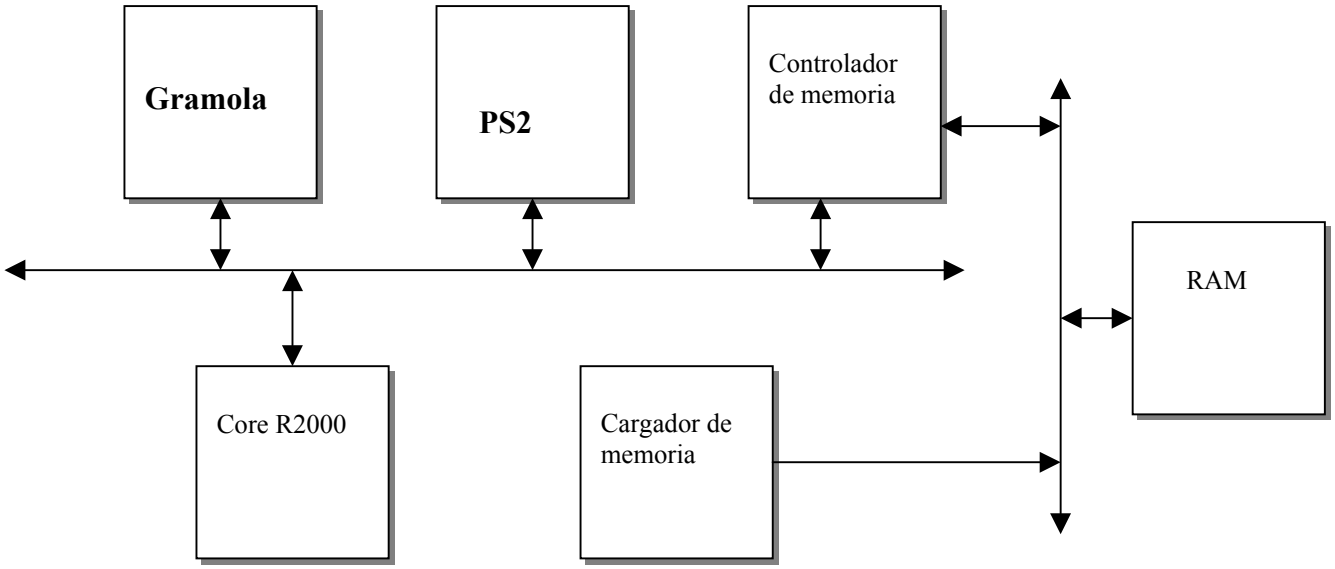
Implementación del protocolo estándar del MIPS

Señales:

- **SysAD :** Bus de datos y direcciones.
- **SysCMD :** Comando de bus.
- Cinco señales para hand-shaking
    - **RdReady :** El periférico puede aceptar una petición de lectura dentro de dos ciclos
    - **WrReady :** El periférico puede aceptar una petición de escritura dentro de dos ciclos
    - **Release :** El bus está libre para aceptar una respuesta
    - **ValidOut :** El micro ha puesto datos correctos en el bus
    - **ValidIn :** El periférico ha puesto datos correctos en el bus

Se han eliminado peticiones por iniciativa de un periférico externo.

1.2.2 Ruta de datos. Esquema general.



1.2.3 Gramola.

Dispositivo de visualización alfanumérico. Características:

- Señal VGA de 256x480 pixels
- Caracteres de 8x8 pixels
- Matriz de 32x60 caracteres
- Juego de 128 caracteres
- Vídeo inverso
- Juego de caracteres redefinible
- Bancos de SelectRAM+ para almacenar el mapa de pantalla y el juego de caracteres
- Lógica del bus del sistema
- E/S mapeada en memoria
- Dirección FFFFC: cambio de color de fondo y carácter
- Dirección FFF8: escribir caracteres en pantalla

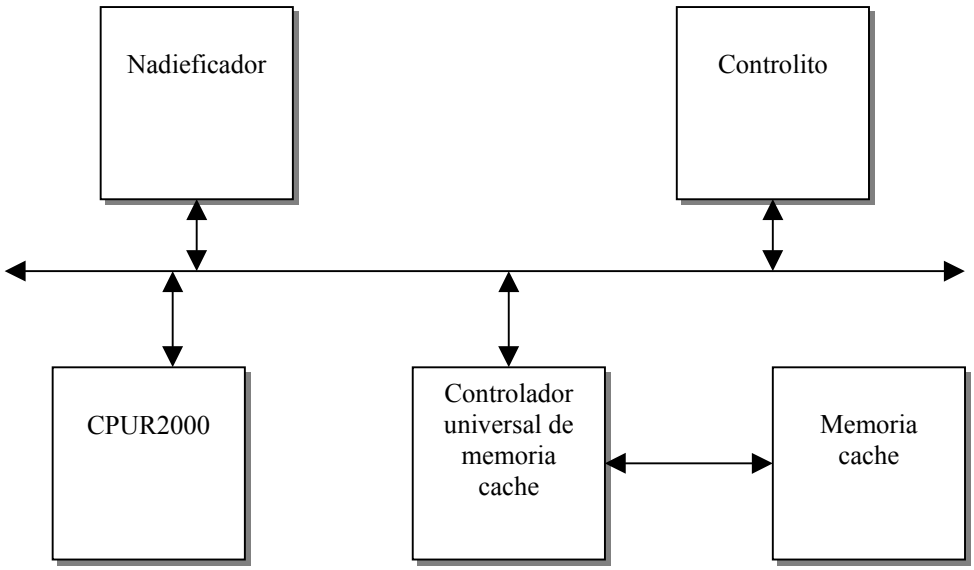
1.2.4 Cargador de memoria.

Carga información procedente del puerto paralelo en la memoria del sistema. Utiliza un protocolo con handshake. Opera con el reset del sistema activo para no interferir con él, esto es, el reset de la FPGA debe estar activado para que se efectúe la carga de los datos en la memoria de la placa.

1.2.5 Core R2000.

El Core R2000 se corresponde con diversos módulos del microcomputador, como son la memoria cache, el módulo nadieficador, la CPU, el controlito, y el controlador universal de memoria cache. Algunos de estos módulos se explican a continuación, los más relevantes, los que no han sido especificados pueden encontrarse en [MaBP].

Esquema general:



Características:

- Procesador RISC clásico, 32 registros de 32 bits modelo lineal de memoria.
- Coprocesadores del R2000 CP0 coprocesador de control, CP1 coprocesador aritmético de coma flotante, inexistente en el microprocesador de partida; CP2, CP3 opcionales.
- No hay gestión de memoria virtual.
- No se permite entrada/salida no alineada.

Componentes principales:

a) CPU. Características generales.

- Dos niveles de microprogramación en la unidad de control microprogramada
- ALU monociclo para operaciones básicas
- Desplazamientos multiciclo
- Banco de registros implementados con SlectRAM+(TM)

b) Controlito

Este componente conecta el bus interno del core del R2000 (BLASA) con el bus externo del sistema, adaptando sus protocolos y regulando el tráfico de información entre ambos buses. Por este motivo es necesario que sea lo más rápido posible.

El controlito permite transferencias de múltiples palabras en modo ráfaga, de manera que se pueden ahorrar ciclos al traer varias palabras seguidas.

Cuando encuentra un fallo de alineamiento o de lectura del bus lo indica a la CPU con una línea de excepción dedicada a tal efecto. Las escrituras no válidas simplemente se ignoran.

Cuando en una lectura se detecta fallo en el bus, se activa la línea de excepción, pero la lectura continua hasta terminar todas las palabras si es que era una lectura en ráfaga.

Controla el ritmo de escrituras de datos hacia el bus BLASA. Estableciendo un ritmo constante configurable durante el diseño.

Esencialmente es una máquina de estados que comunica ambos buses. Está compuesto por tres módulos diferenciados:

- ruta de datos
- registro de estado, el circuito de cálculo del siguiente estado, y los circuitos de generación de las señales de control de la ruta de datos
- la propia máquina de estados.

c) Memoria Cache. Características principales:

- Lógica de control del bus blasa y de control de la memoria cache separadas
- 4 KB además de memoria para tags y bits de estado
- Emplazamiento directo
- 32 líneas de 4 palabras
- Política de Write-back
- Regisros de memoria mapeados, accesibles desde core R2000

d) Controlador de memoria.

El controlador de memoria tiene como misión comunicar al procesador con la memoria externa. Se comunica con el procesador a través del bus del sistema, respondiendo a las direcciones virtuales sobre las que existe memoria física mapeada. Con la memoria externa se comunica a través de líneas dedicadas, aunque posiblemente compartidas con otros dispositivos.

Es labor del controlador de memoria realizar la traducción entre direcciones virtuales lineales y direcciones físicas, produciendo un error de bus cuando se intente acceder a rangos no efinidos. Sin embargo, sólo se notificará el error de bus en las lecturas, ya que, como veremos, no existe forma de indicarlo en las escrituras.

Existe un rango de direcciones virtuales reservado para la entrada y salida mapeada en memoria, ante el cual el controlador de memoria se inhibe, dejando que sea el periférico correspondiente el que responda. Sin embargo, si se realizara una petición de lectura sobre una dirección de E/S

mapeada a la que no atendiera ningún periférico, el sistema se quedaría bloqueado indefinidamente en espera de una respuesta.

El orden en memoria de los bytes que forman una palabra es little-endian, como en un procesador Intel. Por tanto, la palabra 0x11223344 se almacena en memoria de la siguiente forma:

C Dir 0: 0x44
C Dir 1: 0x33
C Dir 2: 0x22
C Dir 3: 0x11

e) Bus B.L.A.S.A.

Es el bus interno del Core R2000 estilo comon data bus, al transferir por él no se indica el receptor del destino sino el origen. Los receptores a los que interese la información la recogerán y procesarán.

Las peticiones se identifican por el que la realiza, el interesado escribe su identificador, el módulo correspondiente responde, 32 bits datos, 32 bits direcciones + líneas de control, permite transferencias de bloque.

Los gráficos correspondientes al ciclo de escritura y de lectura se pueden ver en [MaBP].

El bus BLASA tiene las siguientes líneas:

- Control: indica el origen del dato (CPU, CONTROLADOR DE SISTEMA, CACHE, CONTROL DE RECONFIGURACIÓN o VACIO). También indica el tipo de tipo transferencia: PETICIÓN DE LECTURA, RESPUESTA A LECTURA, PETICIÓN DE ESCRITURA.

- 32 bits de datos

- 32 bits de direcciones

- Tamaño de la palabra a transferir : uno, dos o cuatro bytes.

Esto permite realizar todas las transferencias en un solo ciclo.

Los protocolos de escritura y lectura se detallan en los documentos adjuntos. No se incluye la comunicación con el Control de Reconfiguración aunque en principio el protocolo está preparado para ello.

Tal y como están definidos es posible implementar cualquier política de caché tanto de lectura como de escritura. Así mismo, es posible hacer funcionar el sistema con la cache desactivada.

De la memoria cache al controlador de sistema van dos líneas:

- **Línea de fallo:** indica al controlador de sistema que la cache no posee la página pedida y éste debe ser el que la procese.

- **Tamaño de línea:** indica al controlador el tamaño de la línea. Esto es porque cuando se produce un fallo de cache el controlador trae todas las palabras de la línea que ha fallado, así se aprovecha la capacidad de ráfaga del bus del computador. Cuando la caché está

desactivada (p.ej, porque se está reconfigurando), este valor siempre vale 1, de modo que el controlador trae palabras.

Si el procesador pide unapalabramenor de 32 bits se le devuelve en el bus la palabra del tamaño pedido en la dirección solicitada (el resto de la palabra de 32 bits contendrá basura). Si la cache es la que la responde, esto no es un problema porque ella devuelve el dato correcto. Si es el controlador del sistema quien responde directamente a la CPU, porque la cache está desactivada, se encargará de dar la palabra pedida en la parte correspondiente de la palabra mayor de 32 bits.

La línea de Interlock permite que tanto el controlador de sistema como la cache bloqueen al procesador, se usa para la resolución de los fallos en el acceso a memoria. El procesador no avanza hasta que Interlock se libere.

1.2.6 Microprograma.

De 35 entradas, proporciona el siguiente estado y el valor de las señales de control de la CPU Las señales de control se generan mediante lógica combinacional, no están en el microprograma. La tabla del microprograma se puede visualizar en [MaBP].

1.3. NUEVA VERSIÓN. 2002

1.3.1. Problemas con la reconfiguración parcial.

Los objetivos iniciales a nivel de reconfiguración parcial no se han podido llevar a cabo por una serie de problemas que a continuación presentamos.

El principal problema que se tuvo en la reconfiguración fue la generación de un .bit parcial. Al no disponer de ninguna herramienta que generara automáticamente los archivos fue necesaria su generación manual. Xilinx no dispone de unos manuales centrados en el tema de generación de .bits parciales, sólo dispone de pequeños fragmentos de ***Aplications Notes*** donde se menciona estos tipos de archivos. El no disponer de un ejemplo preciso de un .bit parcial válido para nuestra placa nos llevó a tener que probar multitud de combinaciones en los mapas de bits de configuración, no llegando a dar con la adecuada.

Como se menciona en las conclusiones de la parte del presente documento referente a la reconfiguración, la herramienta que hubiera facilitado el trabajo es ***Modular Desing.*** Dicha herramienta hubiera hecho posible la generación de mapas de bits parciales partiendo de un diseño modular del circuito a implementar. De esta manera hubiéramos partido de archivos que con seguridad eran válidos a la hora de ponernos a reconfigurar.

Otro problema importante fue el hecho de que pasaran por la columna reconfigurable señales del circuito. Este problema aparecía en las implementaciones a pesar de que se prohibía expresamente en las restricciones de usuario. Esto significó que no se pudieran realizar estudios sobre dos circuitos que solo se distinguieran en el módulo reconfigurable, los cambios en ambos circuitos eran demasiado grandes, ya que la herramienta enrutaba las señales de forma distinta en ambos circuitos, cuando se eliminaban manualmente de esta columna. Hubiera significado un gran avance el poder hacer esto, ya que hubiéramos sabido a ciencia cierta entre que dos direcciones de memoria estaba albergado el módulo reconfigurable, pudiendo comparar los cálculos con dichas direcciones. Esto se hizo pero sobre un archivo que solo difería en dos puertas lógicas, por lo tanto nos daba una pequeña información sobre las direcciones que ocupaba la columna reconfigurable.

El principal problema que se ha encontrado es el hecho de que no haya documentación adecuada en Xilinx para realizar la reconfiguración parcial. No hay ejemplos adecuados en los que se pueda confiar como ciertos, ya que normalmente aparecen incompletos. Una muestra de la falta de precisión de la documentación de la FPGA, es que el CRC se mencionara siempre como algo no necesario para una reconfiguración parcial. Sin embargo, nosotros hemos llegado a la conclusión de que el CRC es algo muy relacionado con el hecho de que no nos funcionara adecuadamente la placa una vez cargado un .bit parcial. Este algoritmo sólo venía desarrollado en una de las aplications notes [XILI00b].

Por lo tanto la falta de la documentación precisa por parte de los fabricantes y el no disponer de las herramientas adecuadas fueron nuestro principal obstáculo a la hora de alcanzar el éxito en la reconfiguración parcial.

1.3.2 Cache. Nueva instrucción.

A pesar de las condiciones que inicialmente reunía la arquitectura del MIPS para la incorporación de una o varias memorias cach ede cara a la reconfiguración se han presentado una serie de problemas que han hecho cambiar el rumbo de los objetivos iniciales.

Los factores más significativos han sido los siguientes:

1. **Arquitectura del bus BLASA:** la ruta de datos del core del R2000 trabaja sobre un bus bidireccional que comunica la CPU, la memoria cache y el controlito (módulo que se comunica con el resto del MIPS: memoria principal y subsistema de entrada/salida). El protocolo de comunicación del bus provocaba el bloqueo del micro por una falta de robustez en su estructura acompañada del tráfico de un alto número de señales de control, algunas de ellas irrelevantes (en el **APÉNDICE B** puede encontrarse documentación relativa al bus BLASA). Este bloqueo aparecía en cuanto el micro no trabajaba directamente con el controlito, o lo que es lo mismo, en cuanto se introducía otro elemento en la ruta de datos como en este caso era la memoria cache.

2. **Flujo de datos con memoria principal:** el intercambio de datos por parte del controlito con memoria principal no respondía al protocolo de E/S en operaciones de escritura en memoria ni de lectura en ráfagas continuas. Esto quiere decir que la lectura de palabras funcionaba correctamente así se limitase la operación a la petición o respuesta en demanda de una única palabra de memoria. En cuanto las transferencias eran de tamaño bloque (2 o 4 palabras) o bien se realizasen dos o más peticiones consecutivas de palabras distintas las transferencias no daban como resultado la lectura de los datos deseados.

3. **Reconfiguración parcial dinámica:** la reconfiguración parcial dinámica no ha sido posible por los motivos citados en la sección **1.3.1**.

Teniendo en cuenta estos factores y tras barajar otras alternativas se han tomado las siguientes:

1. **Arquitectura del bus BLASA:** Se ha eliminado el bus BLASA e implementado otro nuevo, más sencillo y basado en la comunicación mediante buses unidireccionales, sin la existencia de un árbitro y un protocolo de comunicación tipo strobe.

2. **Flujo de datos con memoria principal:** en vez de implementar varias memorias cache y ante la imposibilidad de darles una funcionalidad completa se han implemetado dos versiones distintas (de cara a la reconfiguración) de buffers intermedios entre memoria principal y el micro.

3. **Reconfiguración parcial dinámica:** se ha añadido al repertorio de instrucciones una nueva con el objetivo de seleccionar la memoria cache con la que trabajará el MIPS y se ha implementado un simulador de reconfiguración que realiza la labor de dejar al R2000 sin cache durante un periodo de tiempo y tras el mismo volver a suministrar la cache que haya sido seleccionada mediante la instrucción. La diferencia radica en que ambas memorias están presentes en la arquitectura en todo momento.

No obstante se llevaron a cabo las siguientes pruebas antes de tomar estas decisiones:

1.3.2.1 Consecuencias del Bus BLASA: cambios.

Partiendo de la configuración de la cache presentada en el proyecto del curso académico 2000-2001, una configuración que inicialmente no simulaba, la cache fue dotada de los siguientes componentes para intentar incrementar su robustez y fiabilidad:

1. Salidas triestado para todos los buses bidireccionales. El objetivo de esta mejora era evitar posibles colisiones entre datos de entrada y salida que se transmitieran a través del mismo bus.

2. La cache original dividía su funcionalidad en dos partes fundamentales: una dedicada a la comunicación con la cache física que actuaba como unidad de control encargada de

implementar la política de lectura-escritura y otra encargada de la comunicación con el controlito para la transmisión y recepción de bloques con memoria principal. Surgieron problemas entre ambas máquinas de estado puesto que había señales compartidas por ambas. La solución adoptada se basó en el añadido de estados y señales para disponer las dos máquinas de estado de un modo totalmente independiente.

3. Reducción del número de dependencias entre estados aumentado en algunos casos la secuencia de instrucciones a ejecutar y el número de estados de la máquina.

4. Para descartar posibles errores de colisión de datos en memoria principal se llevaron a cabo una serie de pruebas con distintos rangos de memoria principal sobre los que direccionar los programas (datos e instrucciones). Estas colisiones tendrían su exponente más negativo con la E/S mapeada.

5. Se llevó a cabo una reestructuración de las dos máquinas de estados, elementos de almacenamiento y ruta de datos para dejar restringidas las unidades de control al cometido de la activación de las señales que regían el protocolo sin interferir con la funcionalidad de la ruta.

Los resultados de todas estas mejoras no llegaron más allá de las simulaciones sin llegar a funcionar sobre la placa.

Este es un punto de inflexión en que se decide cambiar el protocolo de comunicación entre el micro, la cache y el controlito reduciéndolo a uno lo más sencillo posible. Se intenta evitar elementos estructurales conflictivos del bus BLASA como el uso de buses bidireccionales, el bus de control o un árbitro en permanente espera que vertía basura sobre los buses cuando ningún otro elemento lo hacía con datos válidos.

1.3.2.2 Pruebas y versiones.

A pesar de las mejoras en la cache y el buen funcionamiento de la nueva ruta, siguen existiendo los problemas y se decide implementar una nueva cache partiendo desde cero y haciendo de ella un módulo flexible para probar cuantas configuraciones fuera posible. Estas son algunas de las configuraciones que han sido probadas:

1. Lectura de palabras.

2. Lectura de bloques.

3. L/E de palabras.

4. L/E de bloques.

5. Combinaciones de las anteriores.

6. Lectura repetida de la misma palabra.

7. Lectura repetida de 2 palabras distintas.

Los únicos resultados positivos se han obtenido con las versiones de lectura de palabra y lectura repetida de la misma palabra. Estas son las dos versiones sobre las cuales hemos trabajado en los últimos meses de cara a la reconfiguración.

1.3.2.3 Nueva instrucción.

Hemos de mencionar que a consecuencia de lo anterior la funcionaldidad de la nueva instrucción del repertorio radica en el intercambio de líneas de direccionamiento de una cache a otra en vez de reconfigurar el contenido de la placa. Los detalles en torno a la nueva instrucción están detallados en el apartado 3 del presente documento.

2. ENTORNO DE DESARROLLO

2.1 HARDWARE XSV800

2.1.1 Descripción de la FPGA.

La Virtex posee una arquitectura flexible y regular compuesta por un array de bloques lógicos reconfigurables (CLBs) rodeados de bloques de entradas y salidas programables, todos ellos interconectados por una rica jerarquía de rápidos y versátiles recursos de routing. La abundancia de estos recursos permite que se pueda diseñar sobre esta FPGA diseños muy complejos.

Las FPGAs de la familia Virtex están basadas en tecnología SRAM, y se pueden adaptar a los circuitos de los usuarios cargando datos de configuración en las celdas internas de memoria. En algunos modelos, la FPGA puede leer sus propios datos de configuración de la PROM externa.

El array de puertas programables de la Virtex está formado por dos tipos de elementos: bloques lógicos reconfigurables (CLBs) y bloques de entrada/salida programables:

1. CLBs nos proporcionan los elementos funcionales para construir la lógica del circuito

2. IOBs nos proporcionan el interfaz necesario entre el conjunto de pines y la CLBs

Las CLBs estan interconectadas a través de una matriz de routing (GRM). Así cada CLB estará conectada a la matriz GRM, que conecta líneas de conexión verticales y horizontales. Además dicha matriz también está conectada con los IOBs.

La arquitectura de la Virtex también conecta los siguientes circuitos con la GRM:

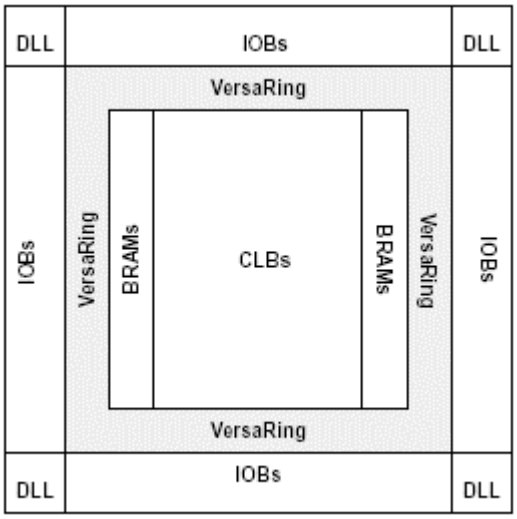
1. Bloques de memoria de 4096 bits cada uno

2. Clock DLLs

3. Buffers triestado (BUFTS) asociados a CLBs y unidos a líneas horizontales de routing

Todos lo valores de configuración pueden ser cambiados cuando sea preciso cambiar el diseño del circuito.

A continuación mostramos un dibujo con el diseño de la arquitectura de la Virtex.



Los principales elementos que componen se detallan en los siguientes apartados. [XESS01] [XILI00c]

2.1.2 Bloque de entrada/salida.

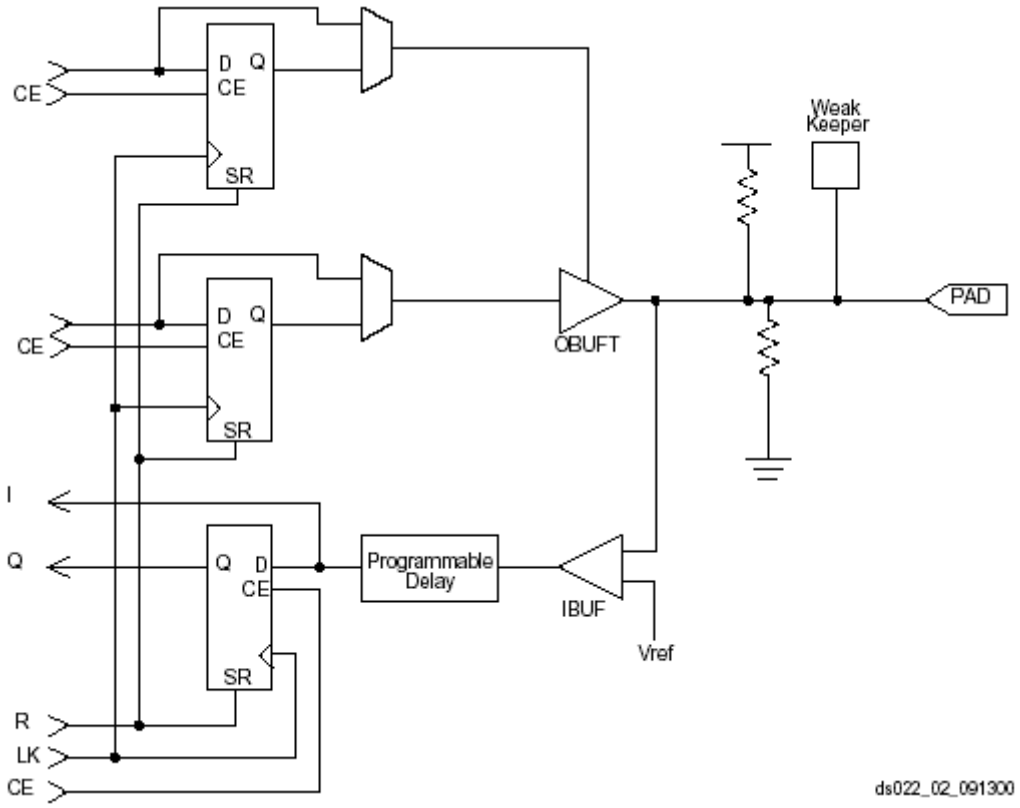
En cada IOB hay tres elementos de almacenamiento. Estos elementos son flip-flops de tipo D o latches sensibles a nivel. Cada IOB posee una señal de reloj (CLK) unida a cada uno de los flip-flops.

Además de las señales de control CLK y de la CE, todos los flip-flops poseen una señal de set/reset (SR).

El buffer de salida y todas las señales de control del IOB tienen controles de polaridad independientes.

Todos los pads están protegidos contra daños producidos por descargas electrostáticas y por subidas de tensión.

A continuación mostramos un diseño de un bloque de entrada/salida

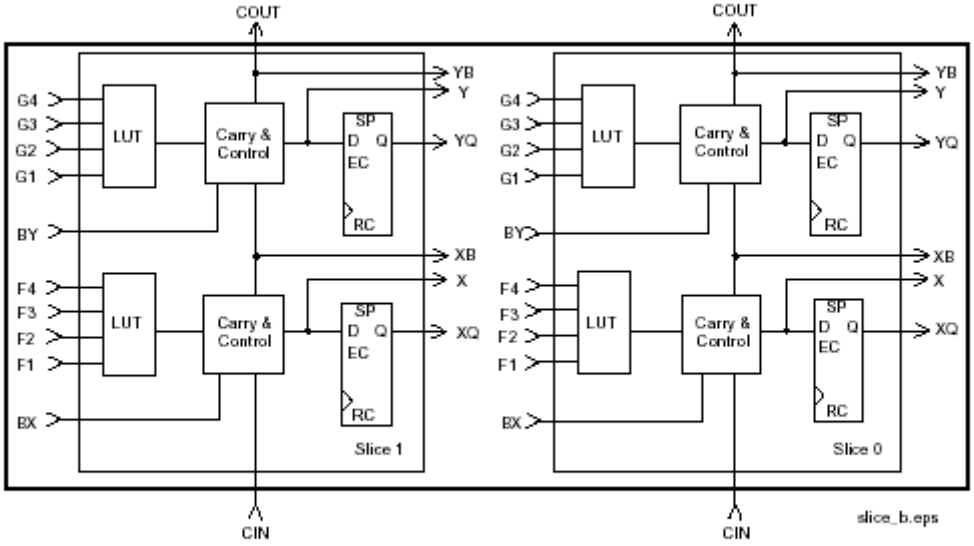


2.1.3 Bloque lógico configurable.

El bloque básico dentro del CLB de la Virtex es la celda lógica (LC). Cada una de estas celdas incluye una función generadora de cuatro entradas, un carry y un elemento de almacenamiento. La salida de la función generadora es dirigida tanto a la salida del CLB como a la entrada del flip-flop de tipo D. Cada CLB de la Virtex posee cuatro LC organizadas en dos slices similares.

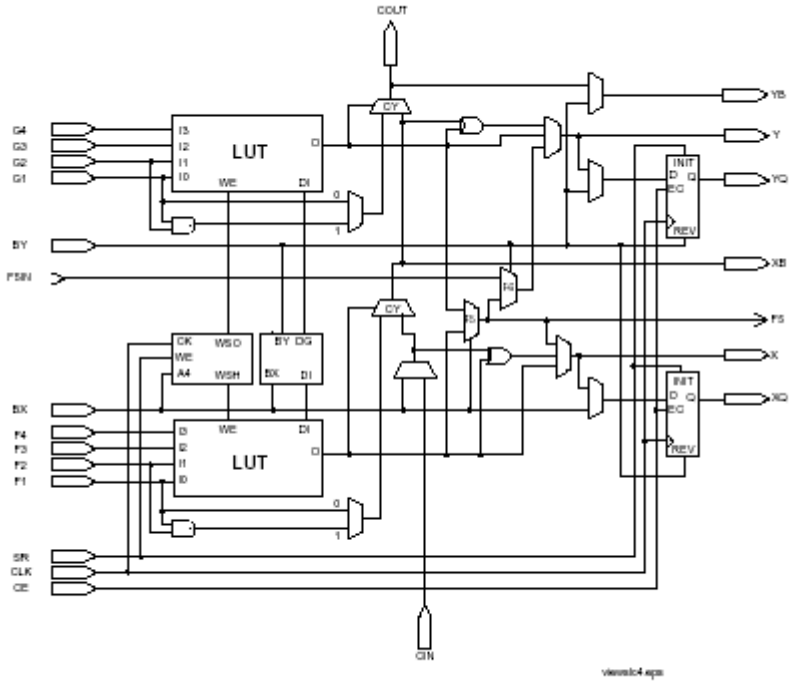
Además de las LCs, cada CLB contiene lógica que combina funciones generadoras para obtener funciones de cinco o seis entradas. Por lo tanto, cada vez que nos enfrentamos a estimar el número de puertas lógicas de un diseño, debemos de tener en cuenta que un CLB cuenta por 4'5 LCs.

A continuación mostramos un diseño completo de una CLB.



2.1.4 Look-up tables.

Las funciones generadoras de la Virtex están implementadas como look-up tables de cuatro entradas (LUTs). Además, para funcionar como función generadora, cada LUT puede proveer de una RAM asíncrona de 16 x 1 bit. A parte de esto, cada LUT de un mismo slice puede ser combinada para crear RAM de 16 x 2 bits o 32 x 1 bit. o una RAM de 16 x 1 bit de puerto dual.



2.1.5 Elementos de almacenamiento.

Los elementos de almacenamiento dentro de los slices pueden ser configurados como flip-flops de tipo D o bien como latches sensibles a nivel. Los biestables D pueden tener como entradas las salidas de las funciones generadoras o bien las entradas del slice directamente, pasando a través de las funciones generadoras.

Además del reloj y de la señal de enable, cada slice tiene una señal de sincronización y otra de set/reset, respectivamente BY y SR. La señal de set/reset fuerza a elemento de almacenamiento al estado de inicialización especificado en la configuración. La señal BY lo fuerza al estado opuesto. Estas señales pueden ser configuradas para poder trabajar de forma asíncrona.

2.1.6 Lógica adicional.

Existe un multiplexor F5 presente en cada slice. Dicho multiplexor combina las salidas de las funciones generadoras. Así se puede implementar cualquier función de cinco entradas.

De igual manera, el multiplexor F6 combina las salidas de cuatro funciones generadoras seleccionando una de las salidas de los multiplexores F5.

2.1.7 Lógica aritmética.

Los carry logic dedicados proveen a la Virtex de la capacidad de realizar funciones aritméticas a velocidade muy rápida.

2.1.8 BUFTs.

Cada CLB contiene dos buffer triestado (BUFTs) que van conectados a buses dentro del chip.

Cada BUFT tiene un pin de control independiente al igual que un pin de entrada independiente.

2.1.9. Bloques de SelectRAM.

Los bloques de SelectRAM están organizados en columnas. Todos los modelos de Virtex contienen dos de estas columnas, una a lo largo del límite vertical. Estas columnas se extienden a lo largo de todo el chip.

2.1.10 Matriz programable de routing.

Todos los CLBs están conectados a esta matriz que se denomina GRM. El GRM está compuesto por un array de switches de routing localizado en las interconexiones de los cruces de las líneas verticales y horizontales de las líneas de cruce.

Routing Local

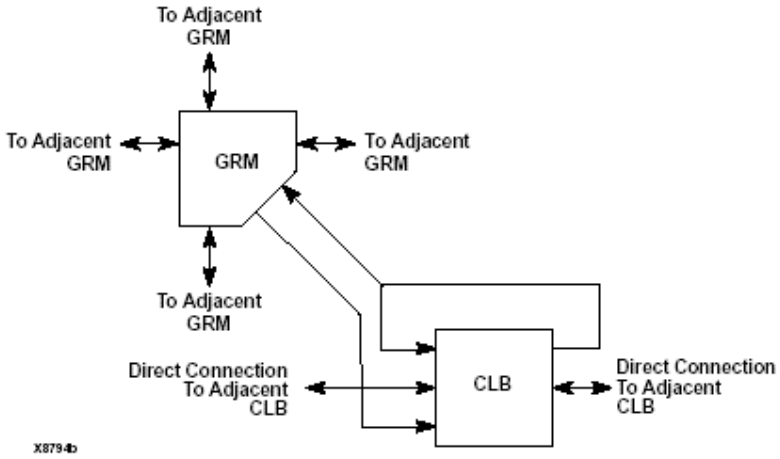
Los VersaBlock proporcionan los recursos para el routing local. Proporcionan tres tipos de conexiones:

1. Interconexiones entre las LUTs, biestables y el GRM.

2. Caminos internos en los CLBs que proporcionan velocidades muy rápidas en conexiones entre las LUTs y el mismo CLB.

3. Caminos directos entre los CLBs situados en la misma línea horizontal y que son adyacentes proporcionando así conexiones muy rápidas y eliminando el retardo del GRM.

El diseño que mostramos en la página siguiente se corresponde con un routing local de la Virtex.



Routing de proposito general

La mayoría de las señales de la Virtex son enrutadas a través de routing de proposito general, y por lo tanto la mayoría de los recursos de interconexión son de este tipo. Los recursos de routing de propósito general están localizados en las líneas horizontales y verticales asociadas con las columnas horizontales y verticales de CLBs.

- Adyacente a cada CLB existe una matriz de routing general (GRM).
- Existen 24 líneas de longitud simple que une señales de una GRM con las GRMs adyacentes en las cuatro direcciones.
- 72 buffers que enrutan las señales de una GRM con las GRMs situadas a seis bloques de distancia en las cuatro direcciones.
- 12 líneas largas bidireccionales que distribuyen las señales de forma rápida y eficiente a lo largo de toda la FPGA.

Routing de entrada/salida

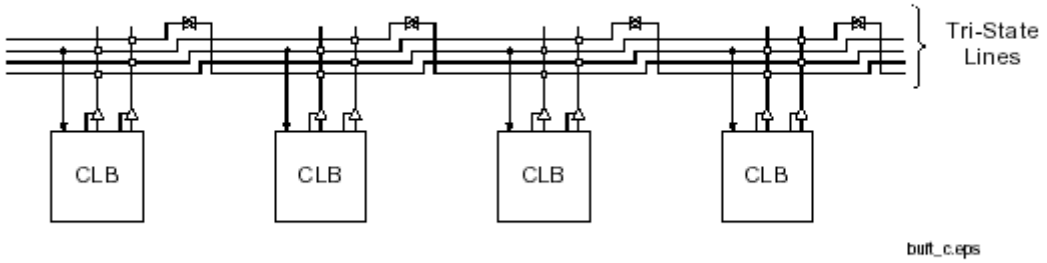
La Virtex posee recursos de routing adicionales alrededor de todo su perímetro de tal manera que se proporciona un interfaz entre las CLBs y las IOBs. Estos recursos adicionales son llamados el VersaRing.

Routing dedicado

Algunas clases de señal requieren recursos de routing que estén únicamente dedicados a ellas para conseguir el máximo rendimiento. En la arquitectura de la Virtex se provee a dos tipos de señal de estos recursos dedicados:

- Recursos de routing horizontales son dedicados a los BUFTs para funcionar como buses.
- Dos líneas por cada CLB para conducir los carry logic a los CLBs adyacentes verticalmente.

Un ejemplo de líneas dedicadas son las de los BUFTs, diseño que mostramos a continuación.



Routing global

Este tipo de routing distribuye la señal de reloj y otras señales a lo largo de toda la FPGA.

2.1.11 CPLD.

El CPLD es uno de los dos chips programables que contiene la placa XSV. El modelo utilizado (XC95108) consta de 24000 puertas.

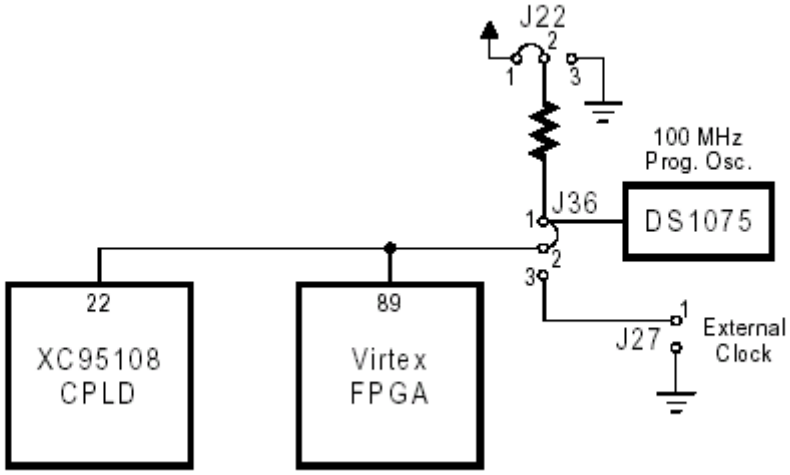
El CPLD se usa para controlar la configuración de la Virtex FPGA a través del puerto paralelo, puerto serie o la Flash RAM. También controla la configuración del chip Ethernet PHY.

El CPLD puede servir de interfaz de comunicación entre el puerto paralelo y la Flash RAM, entre el puerto paralelo y la FPGA o entre la Flash RAM y la FPGA. Para ello debe ser configurado de la forma adecuada. Estas configuraciones son almacenadas en una memoria no volátil interna al CPLD para ser reestablecidas tras el encendido de la placa, a menos que el interfaz haya sido borrado.

Para poder realizar la reconfiguración del CPLD debemos fijar un derivador en el jumper J23 de la placa.

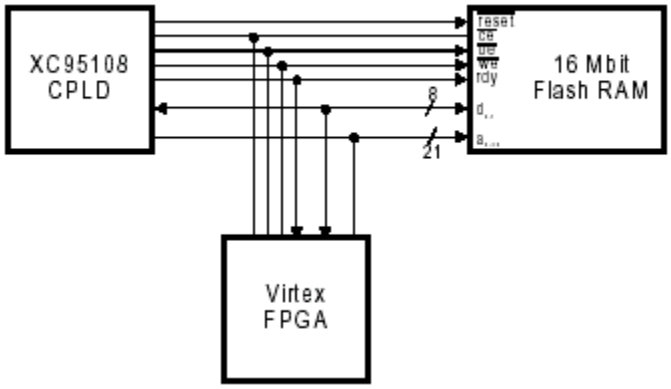
2.1.12 Oscilador.

Tanto el CPLD como la FPGA reciben la señal de reloj de un oscilador programable que tiene una frecuencia máxima de 100MHz (chip DS1075). Se puede fijar un valor para dividir la frecuencia del oscilador. Se puede también insertar una señal de reloj externa que sustituya la salida del oscilador.



2.1.13 Flash RAM.

Tanto el CPLD como la FPGA tienen acceso a la Flash RAM (dos módulos de 8 Mbit cada uno). Típicamente, el CPLD programa la Flash con datos que le llegan a través del puerto serie o paralelo. Si los datos almacenados en la Flash son un bitstream de configuración de la FPGA, entonces el CPLD puede ser configurado para programar la FPGA con el bitstream de la Flash.



Después del encendido la FPGA puede leer y/o escribir en la Flash. (Por supuesto el CPLD y la Flash tienen que estar programadas para no entrar en conflicto por el acceso a la memoria Flash) La Flash puede ser deshabilitada poniendo el pin **/CE** a Vcc, en cuyo caso las líneas de entrada / salida conectadas a la Flash pueden ser usadas como líneas de comunicación de propósito general entre la FPGA y el CPLD

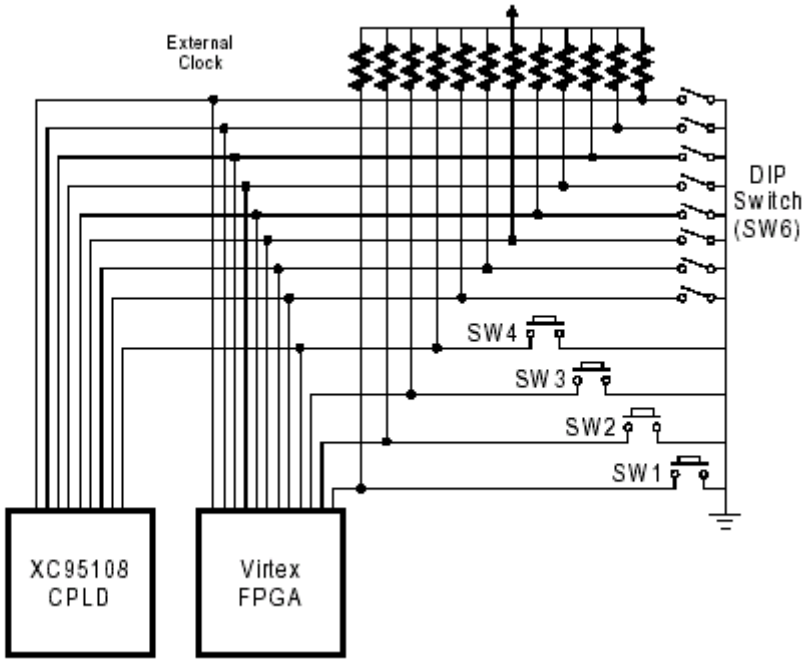
2.1.14 Botones y switches.

El CPLD esta conectada a los switches y a uno de los pushbuttons.

Cuando se pulsa un pushbutton, el pin conectado al CPLD y a la FPGA pasa a tomar valor de tierra. En otro caso está en alta.

Cada switch toma valor de tierra cuando esta cerrado (en posición ON) y cuando el switch está abierto toma valor de alta a través de un resistor.

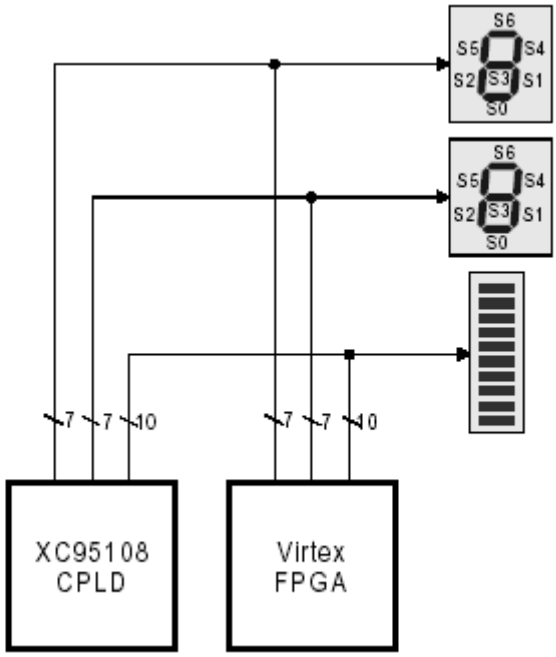
Cuando no se usan los switches deben mantenerse abiertos (posición OFF)



La figura muestra las conexiones de la FPGA y el CPLD a los switches. Los DIP switches comparten los mismos pines que las ocho líneas más significativas del bus de direcciones de la Flash RAM. Si las Flash RAM esta programada con varios bitstream, los switches pueden usarse para seleccionar uno de ellos para ser cargado en la FPGA mediante el CPLD

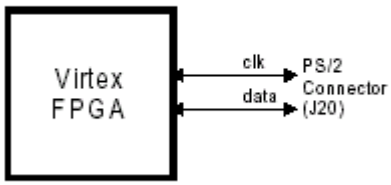
2.1.15 Leds.

La placa XSV cuenta con una barra de 10 leds y dos bloques de leds de 7 segmentos para dígitos, usados por la FPGA y el CPLD. Todos los leds están activos en alta, por lo que se les aplica lógica positiva. Los leds también comparten líneas con el bus de direcciones de la Flash RAM.



2.1.16 Puerto serie.

El CPLD controla el interfaz con el puerto serie. Las cuatro líneas activas se conectan a pines de entrada / salida de propósito general del CPLD.



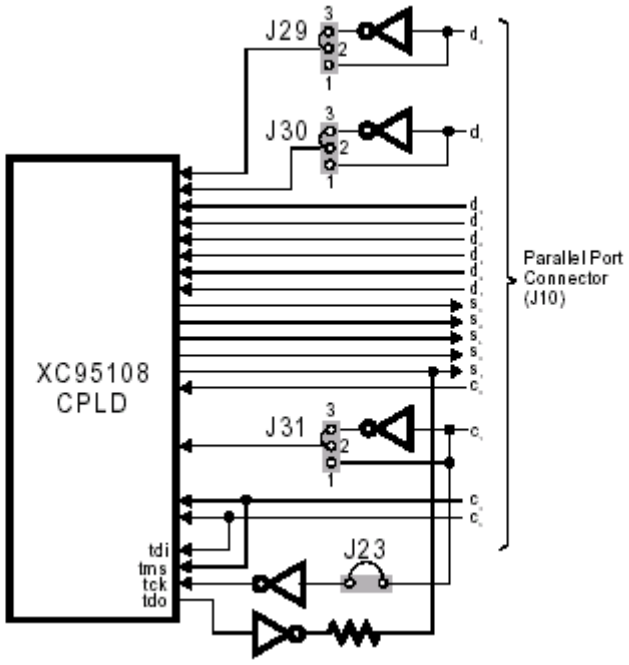
PS/2 Port Pin	Virtex FPGA Pin
CLK	13
DATA	17

2.1.17 Puerto paralelo.

El CPLD controla el interfaz con el puerto paralelo. Las 17 líneas activas se conectan a pines de entrada / salida de propósito general del CPLD.

Cuatro de las líneas del puerto paralelo están también conectadas al JTAG, a través de las cuales el CPLD puede ser programado.

El CPLD puede ser programado para actuar como un interfaz entre la FPGA y el puerto paralelo (el fichero **dwnldpar.svf** es un ejemplo de este tipo de interfaces) Se pueden insertar inversores Schmitt-trigger en las líneas de señal d0, d1 y c1 poniendo derivadores en los pines 2 y 3 de los jumpers J29, J30 y J31 respectivamente. Estos inversores hacen que la placa XSV sea compatible con las aplicaciones GSXPORT y GXSLOAD. Si la aplicación requiere acceso directo a estas líneas, se pueden mover estos derivadores de uno o más de estos jumpers a los pines 1 y 2; pero GXSLOAD no funcionara si se deshabilita el inversor de la línea de señal **c1**.



2.1.18 Proceso de arranque de la placa.

[XILI00]

Tras el encendido, la configuración de la FPGA es automáticamente borrada. Un 0 lógico en la señal **/PROG** limpia la configuración lógica y mantiene a la FPGA en el estado de "memoria de configuración borrada"; estado en el que se mantiene mientras **/PROG** continue a 0 manteniendo el valor de la señal **/INIT** a cero para indicar que la memoria esta siendo limpiada.

Cuando el valor de **/PROG** cambia, la FPGA continúa manteniendo el valor de **/INIT** a cero hasta que finalice el borrado de toda la memoria. El pulso mínimo para **/PROG** es de 300 ns, no habiendo un valor máximo. El valor de **/INIT** puede ser mantenido a cero desde el exterior de la placa para evitar su configuración.

Una vez que el valor de **/INIT** pasa a ser 1, la configuración puede comenzar. Ya no son necesarias más esperas, pero no es necesario que la configuración comience nada más

producirse la transición en el valor de **/INIT**.

Durante la configuración se produce una comprobación del valor del CRC (palabra de paridad) del bitstream que se esta cargando. Si el valor del CRC no es correcto, la señal **/INIT** pasa a valer 0, indicando así que se ha producido un error en el CRC. La secuencia de comienzo, si se produce esta situación, es abortada y la FPGA no pasa a estado activo.

Para reconfigurar el circuito, el valor de la señal **/PROG** debe ser puesto a 0 lógico para borrar la configuración. Apagando y encendiendo la placa también conseguimos que la FPGA sea borrada para una nueva configuración.

Cuando se ha verificado el valor del CRC, la FPGA entra en una secuencia de encendido, en la cual se activa la señal **DONE** para indicar que la configuración se ha realizado correctamente y se activan las entradas / salidas.

El diagrama de la siguiente página nos muestra el flujo de control de una la secuencia de encendido de la placa:

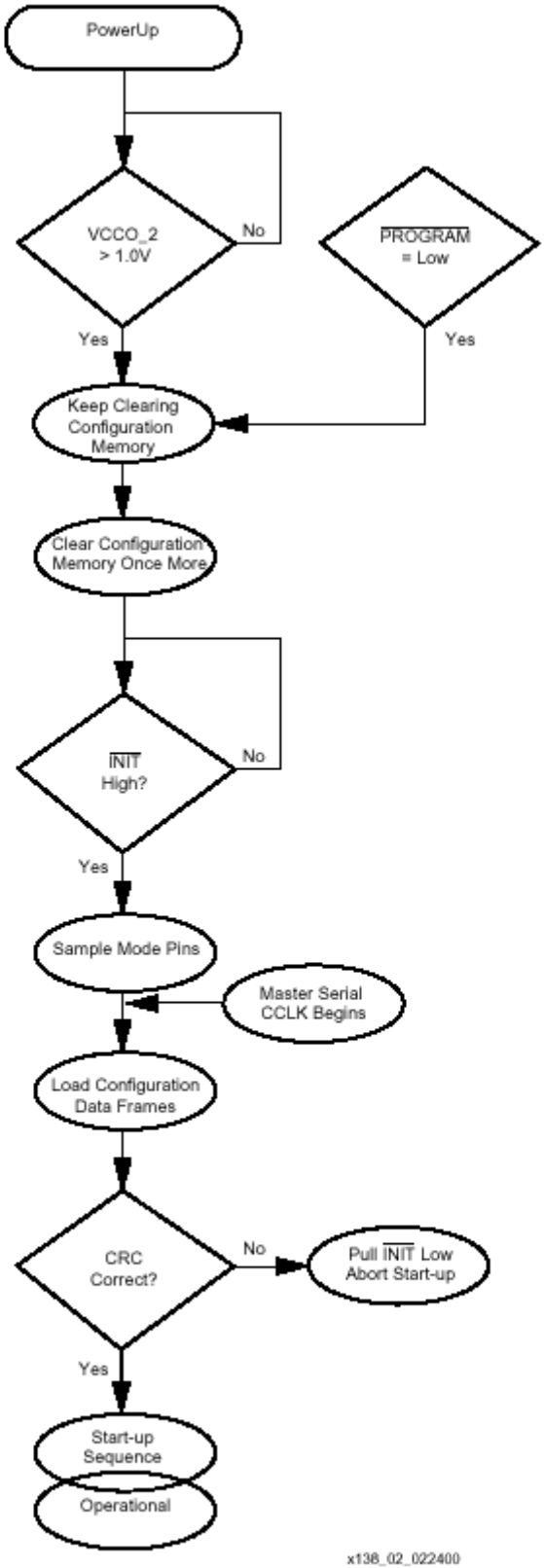


Figure 2: **Configuration Flow Diagram**

2.2. SOFTWARE

2.2.1 Entorno Xilinx Foundation 3.1

2.2.1.1 PROJECT MANAGER.

El Project Manager es una aplicación que maneja y supervisa todas las herramientas de Foundation Series en el proceso de diseño, y las integra en un único entorno.

Este entorno incluye herramientas como el Schematic Editor, HDL Editor, y otros productos de terceros. El Project Manager asume la responsabilidad de numerosas operaciones. Así Project Manager desarrolla las siguientes funciones:

- Automáticamente carga todos los recursos del diseño cuando un proyecto es abierto.
- Chequea si todos los recursos del proyecto están disponibles y los refresca.
- Muestra el flujo de diseño del proyecto.
- Provee de accesos para las distintas herramientas involucradas en el diseño.
- Provee de un interfaz para las herramientas externas de terceros.
- Muestra todos los errores y mensajes en la ventana de mensajes.
- Provee de transferencia automática de archivos entre las herramientas implicadas en el proceso de diseño.
- Almacena información de estado del diseño.

El Project Manager está diseñado para trabajar con un único proceso al mismo tiempo.

A continuación explicamos las herramientas que nosotros utilizamos en el proyecto. Estas pueden ser invocadas tanto desde el Project Manager como independientemente.

2.2.1.2 FLOORPLANNER.

El FloorPlanner es una herramienta gráfica de "placement" que permite manual o automáticamente situar la lógica desde un archivo de mapeo (NCD file) en un floorplan de la FPGA. El usuario del interfaz gráfico puede utilizar los elementos de dicho interfaz para cambiar la jerarquía del diseño, modificar la situación de los componentes y señales, proporcionar restricciones de usuario (UCF file),….

2.2.1.3 FPGA EDITOR.

La fpga editor es una aplicación gráfica para mostrar y configurar FPGAs. La FPGA Editor necesita un archivo de descripción nativa del circuito (.ncd file). Este archivo contiene la lógica de mapeo del circuito sobre los componentes de la FPGA (CLBs e IOBs). Además, la FPGA Editor lee y escribe en un archivo de restricciones físicas (PCF).

La siguiente es una lista de alguna de las funcionalidades que se pueden desarrollar con la FPGA Editor.

- Realiza un place y un route de las componentes más críticas antes que ejecutar las herramientas automáticas de place y route.

- Finaliza el placement y el routing si los programas de routing no finalizan el enrutado del diseño.

- Realiza pruebas sobre el diseño para examinar los estados de las señales dentro de la placa donde será volcado el diseño.

- Nos permite poder modificar el routing de las señales para colocarlas en los lugares más adecuados.

- Permite eliminar del diseño cualquier pad, señal o componente que no nos convenga.

2.2.1.4 CONSTRAINTS EDITOR .

El Constraints Editor es una GUI (Graphical User Interface) que se puede utilizar para crear y modificar restricciones.

Los archivos de entrada pueden ser alguno de los siguientes:

- UCF (User Constraint File). Las restricciones son creadas por el usario y escritas sobre este archivo.

- NGD (Native Generis Database). Este archivo sirve de entrada a la herramienta llamada mapper, que genera la base de datos física del diseño.

Una vez que las restricciones son realizadas correctamente y no tienen ningún error, el Constraints Editor devuelve como salida un UCF. El NGD es leído por el programa MAP, que genera un NCD y un PCF. Las herramientas de implementación usan tanto el NCD como el PCF para generar un bitstream.

2.2.1.5 FLOW ENGINE.

El Flow Engine es la parte del Sistema de Desarrollo de Xilinx que permite implementar el diseño. El Flow Engine primero traduce el archivo de diseño al formato de la base de datos interna de Xilinx (NGD). Entonces implementa el diseño, incluyendo un mapping, placing y routing para las FPGAs y fitting para las CPLDs. Finalmente genera un archivo bitstream. El Floor Engine permite controlar el proceso de diseño, implementación y proporciona una guía para las distintas revisiones que puede sufrir el circuito.

2.2.1.6 HDL EDITOR .

El HDL Editor es un editor de texto para los archivos HDL. Además de las opciones que proporciona un editor de texto convencional, este editor proporciona de un interfaz para conectar con las herramientas externas de síntesis, comandos para modificar la jerarquía de los circuitos, y coloreado sintáctico. Dicho reconocimiento de la sintaxis es válido para tres lenguajes: VHDL, ABEL y Verilog.

2.2.1.7 PROJECT MANAGER para desarrollo de configuraciones del CPLD.

El uso de la aplicación en este caso es análogo al desarrollo de configuraciones de la FPGA. En los proyectos se incluirán los archivos fuente y tras una fase de síntesis y análisis, en la que se selecciona el tipo de CPLD a utilizar, obtenemos un archivo JEDEC de configuración. Para conseguir un archivo descargable mediante la aplicación GXSLOAD se deberá realizar un paso adicional, usando la herramienta JTAG Programer que se explica a continuación.

NOTA: El uso del Project Manager de la versión 3.1 del Xilinx Foundation con CPLDs añade una restricción a la fase de síntesis: la codificación de estados debe ser binaria.

2.2.1.8 JTAG PROGRAMER.

Esta herramienta ha sido utilizada para la generación de los archivos SVF, necesarios para cargar una configuración del CPLD.

Para poder generar estos archivos necesitamos un archivo JEDEC (.JED) Como ya se ha visto, este tipo de archivos se obtienen como resultado de la fase de implementación del Project Manager.

Esta aplicación puede ser lanzada directamente desde el Project Manager tanto desde el menú de herramientas de implementación como desde el botón de programación (Programming) de la ventana principal.

2.2.2 GXSTOOLs.

GXSTOOLs es un paquete de herramientas para el control y configuración de la placa de prototipado a través de puerto paralelo.

Las herramientas de las que consta el paquete son:

1. GXSTEST
2. GXSSETCLK
3. GXSLOAD
4. GXSPORT

Para poder utilizar estas aplicaciones deberemos conectar la placa (por el conector J10) al puerto paralelo.

2.2.2.1 GXSTEST.

Esta herramienta permite realizar un chequeo del estado de la placa. Para que este chequeo sea posible debemos tener fijados:

- Un derivador en el jumper J23.
- Un derivador en los pines 2 y 3 del jumper J31.
- Un derivador en los pines 2 y 3 del J22.

- Un derivador en los pines 1 y 2 del J36.

Tras elegir el tipo de placa y el puerto al que está conectada. GXSTEST configurará la FPGA para ejecutar un procedimiento de prueba en la placa XSV. Si el chequeo se realiza correctamente se mostrara una "O" en el display de 7 segmentos, en otro case se mostrara una "E". También se mostrara una ventana en el PC dando información sobre el resultado de la operación, mostrando una serie de errores típicos en caso de que ésta haya fallado.

NOTA: Tras esta operación, el CPLD queda programado con un interfaz especial con el puerto paralelo (contenido en el archivo **dwnldtst.svf**) Para usar la placa se deberá cargar el programa con el funcionamiento habitual (**dwnldpar.svf**) como se explica en el apartado que habla sobre la aplicación GXSLOAD (Punto 2.2.2.3)

2.2.2.2 GXSSETCLK.

Como ya se explicó en la descripción de elementos de la placa XSV800, en el oscilador se puede fijar un valor para dividir su frecuencia inicial de 100MHz. Los valores permitidos para la división de frecuencia son los pertenecientes al rango [1..2052]. La señal obtenida en la división de frecuencia es la que se envía al resto de elementos de la placa como señal de reloj. El valor del divisor se almacena en una memoria no volátil del propio chip oscilador. Para fijar este valor no tenemos más que ejecutar la aplicación GXSSETCLK e indicar el valor del divisor de frecuencia y el puerto al que está conectado la palca.

Se puede también insertar una señal de reloj externa que sustituya la salida del oscilador. Para ello sólo hay que elegir la opción **"External Clock"** en la aplicación. En este caso hay que proporcionar una señal de reloj a través de la conexión J27 de la placa.

GXSSETCLK carga una configuración del CPLD que permite programar el oscilador, por lo que después de su uso necesitaremos reprogramar el CPLD con el circuito que actúa de interfaz entre el puerto paralelo y la FPGA usando la aplicación GXSLOAD y el archivo **dwnldpar.svf**.

2.2.2.3 GXSLOAD.

La herramienta GXSLOAD permite tanto programar la FPGA, el CPLD, como cargar datos en la Flash y conseguir volcados de memoria RAM de la placa.

a) Programación del interfaz:

La configuración del interfaz que se guardará en el CPLD puede ser realizada usando la aplicación GXSLOAD seleccionando el tipo de placa a configurar, arrastrando un archivo .SVF en la zona de **FPGA/CPLD** y pulsando el botón **LOAD**. Para configurar el CPLD con el interfaz del puerto paralelo debemos usar el archivo **dwnldpar.svf** de la carpeta XSTOOLS\XSV.

Para poder realizar la reconfiguración del CPLD debemos fijar un derivador en el jumper J23 de la placa. Una vez que la configuración se ha completado, se puede quitar el derivador de J23 para evitar una reconfiguración accidental.

b) Configuración de la Virtex por medio de Bitstreams:

Por medio del GSXLOAD podemos descargar archivos de configuración (Bitstreams) en la FPGA. Para ello debemos asegurarnos que esta cargado en el CPLD el interfaz entre el puerto

paralelo y la FPGA y que hay situado un derivador entre los pines 2 y 3 del jumper J31 de la placa.

Una vez seleccionada el tipo de placa podemos arrastrar un archivo .BIT a la zona de **FPGA/CPLD** de la aplicación e iniciar la descarga. Solo se puede seleccionar un archivo para descargar. El archivo de configuración pasara a través del puerto paralelo y del CPLD llegando finalmente a la FPGA.

c) Almacenamiento de diseños no volátiles en la placa.

Los diseños son almacenados en una memoria SRAM del chip de la FPGA, que se borra al apagar la fuente de alimentación. Para solucionarlo podemos utilizar los chips de memoria Flash RAM que posee la placa. Cargando un interfaz adecuado en el CPLD podemos conseguir que la FPGA se reconfigure con los archivos almacenados en la Flash RAM nada más encender la placa.

Los archivos descargados en la Flash son del tipo .EXO o .MCS, por lo que deberemos en primer lugar transformar los archivos .BIT en uno de formato EXO o MCS con los siguientes comandos:

promgen –u 0 file.bit –p exo –s 2048
promgen –u 0 file.bit –p mcs –s 2048

Con estos comandos, los archivos de configuración son transformados en archivos de tipo EXO o MCS, comenzando en la dirección cero hasta el limite superior de 2048 KBytes.

NOTA: Antes de programar la Flash, los ocho switches de la placa deben ser colocados en posición OFF.

Los archivos EXO o MCS son cargados en la Flash por medio del programa GXSLOAD, arrastrándolos en la zona **Flash/EEPROM**. Los pasos provocados son:

1. Borrado completo de la Flash RAM

2. Reconfiguración del CPLD con un interfaz entre la Flash y el puerto paralelo. (Archivo **fintfc.svf** de la carpeta XSTOOLS\XSV)

3. El contenido del archivo EXO o MCS es volcado a la Flash a través del puerto paralelo.

4. El CPLD es reprogramado con el circuito que configura la FPGA con el contenido de la Flash al encender la fuente de alimentación de la placa (Archivo **fconfig.svf** de la carpeta XSTOOLS\XSV)

También es posible volcar el contenido de la Flash a un archivo mediante la herramienta GXSLOAD.

d) Descargando y cargando datos en la RAM de la placa XSV:

La aplicación GXSLOAD permite cargar datos a la RAM de la placa por medio de archivos EXO, MCS, HEX o XES. Para ello se configura el CPLD para que actúe de interfaz entre el puerto paralelo y la RAM. Una vez cargados los datos en la RAM se configurará la FPGA con el archivo marcado en la zona **FPGA/CPLD**. En caso de no haber ninguno seleccionado la FPGA se quedara a la espera de ser configurada.

De forma análoga la aplicación puede volcar el contenido de la RAM en un archivo.

2.2.2.4 GXSPORT.

Esta herramienta permite enviar datos a través del puerto paralelo. Para ello solo hay que seleccionar el valor a enviar en cada bit de datos. Existe una opción para que los datos a enviar se vayan actualizando de la misma forma que un contador.

2.2.3 LoadMem.

[MaBP]

LoadMem es un programa implementado por Ivan Magán Barroso, Javier Basilio Pérez Ramas y Miguel Péon Quirós (integrantes del equipo que desarrolló la versión 2000-2001 del presente proyecto) . El objetivo: enviar datos y programas a la memoria de la placa a partir de ficheros en código binario para el R2000. En su aspecto y utilización es exactamente igual que la utilidad de descarga de los ficheros *.bit (**GXSLOAD**). De hecho la programación está basada en el código y las librerías sobre las cuales se implementaron las herramientas **GXSTOOLS**. Los archivos sobre los que trabaja esta utilidad (*.paca) son generados con la herramienta **ConSpim**.

Para el correcto funcionamiento del software se han de cumplir los siguientes requisitos**:**

1. Es necesario que la máquina sobre la que se va a utilizar tenga instalada la librería DriverLINX Port I/O para poder realizar las descargas por el puerto paralelo (Win2000 puso algunas dificultades a la hora de conceder permisos para acceder a este puerto).

2. El sistema descargado en la placa debe contar con el "cargador de memoria" integrado y activado en el momento de realizar la descarga.

3. El fichero *.paca debe tener el formato adecuado.

4. El CPLD debe estar programado adecuadamente con un circuito que permita conectar las líneas del puerto paralelo. Esto requiere varias aclaraciones:

   Originalmente el CPLD de la placa viene cargado con un circuito que permite las descargas hacia la FPGA, pero que tras la misma, desconecta las líneas del puerto paralelo.

   Para poder utilizar el LoadMem ha de cambiarse el circuito del CPLD por uno que permita las descargas y además deje las líneas del puerto paralelo conectadas a la FPGA. Desde un punto de vista práctico, el CPLD debe haber sido programado con las GXSTOOLS (concretamente con la herramienta GXSLOAD) y el archivo "dwnldpar.svf" antes de utilizar este software.

2.2.4 PCSpim.

[La97]

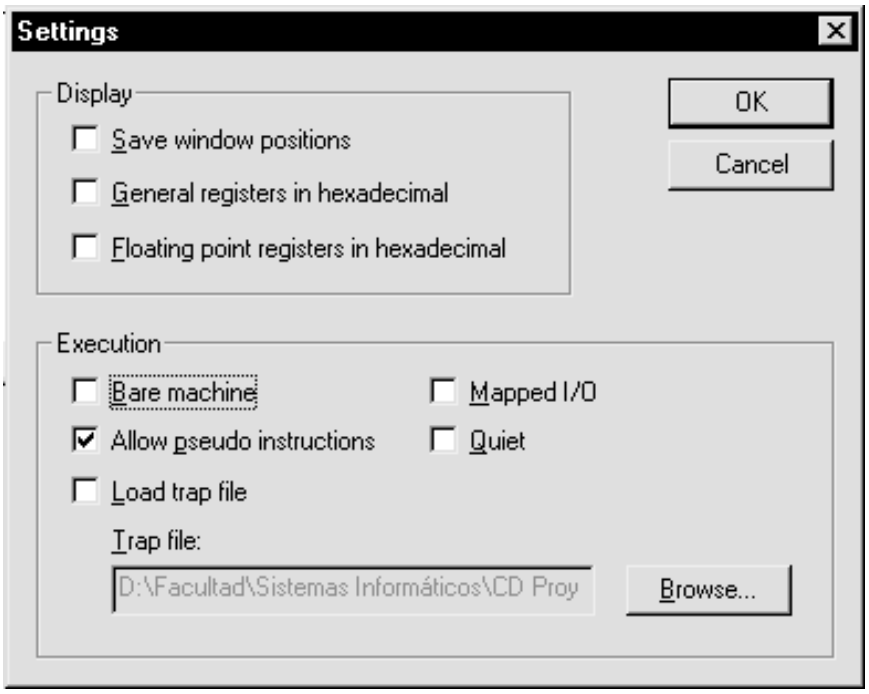
PCSpim es un simulador sobre el cual pueden correr programas para la familia MIPS R2000/3000 RISC. Es un software que lee y ejecuta programas escritos en ensamblador para esta familia de procesadores.

La versión utilizada (6.2) ha permitido generar archivos intermedios (*.log) entre los archivos fuente ensamblador del R2000 (*.asm) y los que finalmente se descargarían a la placa (*.paca). Estos archivos contienen información acerca de las direcciones de memoria en las cuales se

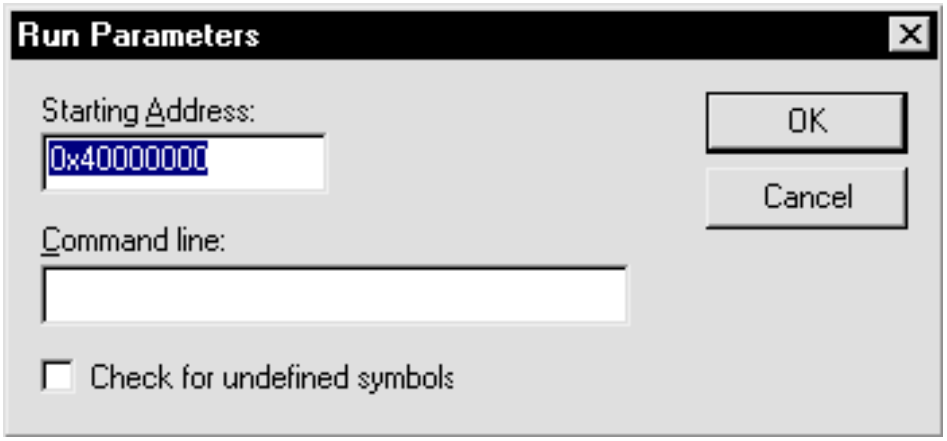
ubicarán datos y programas y son una primera traducción al lenguaje máquina del R2000.

Para generar un archivo .log a partir de uno escrito en ensamblador hemos de seguir el siguiente proceso:

1. En *Simulator -> Settings* activamnos exclusivamente la opción de permitir el uso de pseudoinstrucciones (esto nos permitirá utilizar todo el repertorio de instrucciones):



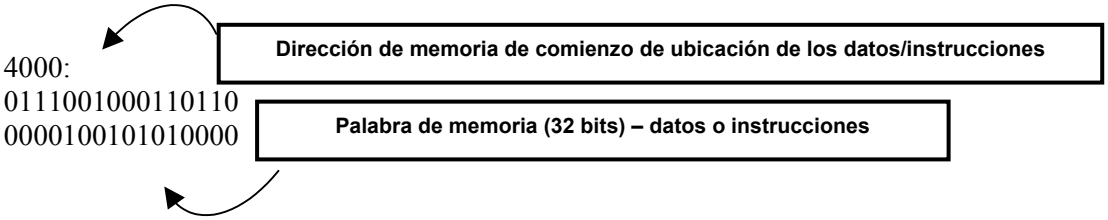
2. A continuación ejecutamos el programa: abrimos el fichero y seguidamente clickeamos *Simulator -> Go* (o bien pulsamos F5). Mostrará una ventana preguntando por la dirección de comienzo del programa y una línea de comandos. Como dirección daremos la "0x40000000" y dejaremos en blanco la línea de comandos:



3. Una vez ejecutado el programa guardamos el .log de la ejecución del mismo mediante *File -> Save Log File*.

2.2.5 ConSpim.

Software utilizado para la generación de los programas en código máquina (*.paca) a partir del *.log generado con el PCSpim. El formato genérico de un fragmento de estos archivos es el siguiente:



El programa se ejecuta sobre una ventana de MS-DOS bajo el siguiente comando:

conspim [nombre_archivo].log > [nombre_archivo].paca

2.2.6 ModelSim.

[MoWe] [XIMo] [MSMa]

Una potente herramienta de simulación digital de circuitos bajo VHDL. El análisis de las formas de onda para ver transiciones en señales de cualquier entidad de la arquitectura, la posibilidad de realizar simulaciones eléctricas a partir de ficheros de análisis generados por otras herramientas como Xilinx Foundation o la cantidad de librerías de distintos fabricantes compatibles con este software son caracrísticas que nos han llevado a trabajar con este simulador.

En su versión 5.4 ha sido utilizada para realizar simulaciones eléctricas y funcionales del MIPS. No explicamos en detalle los pasos a seguir para realizar una simulación ya que la página de ModelTech (www.model.com) contiene toda la información relevante a esta herramienta y no es un paso imprescindible en la generación de un archivo .bit funcional sobre la placa.

2.3 PROGRAMACIÓN DE LA PLACA XSV800

2.3.1. Programación de la Flash RAM y configuración de la FPGA.

Los 16Mbits de Flash RAM que posee la placa XSV pueden ser usados par guardar configuraciones de la FPGA. Esta operación se realiza en tres pasos:

1. Configurar el CPLD para que comunique la Flash con el puerto paralelo.

2. Guardar en la Flash una configuración de la FPGA.

3. Configurar el CPLD con un circuito que cargue la FPGA con la configuración almacenada en la memoria Flash.

Para guardar una configuración en Flash debemos:

1. Obtener un archivo Bitstream (.bit) valido que contenga una configuración de la FPGA válida.

2. Generar un archivo .EXO, .MCS, .HEX o .XES. La aplicación GXSLOAD toma estos tipos de archivos para cargar datos en la Flash RAM.

Para generar un archivo MCS ejecutamos en la línea de comandos:

promgen -u 0 file.bit -p mcs -s 2048

Esta es una llamada a la aplicación que genera el archivo "file.mcs" a partir de "file.bit".

-u NUM indica que el archivo se cargara a partir de la dirección NUM (en hexadecimal).
-p TYPE indica el tipo de archivo q se generará. En este caso es un archivo MCS.
-s SIZE indica el tamaño de la PROM en Kbytes (debe ser potencia de 2)

El archivo MCS generado se carga con la aplicación GXSLOAD (ver 2.2.2.3). Esto produce los tres pasos arriba indicados, ya que al principio se configura el CPLD para que actúe de interfaz entre el puerto paralelo y la Flash; y después de cargar la Flash se configura con un programa que carga en la FPGA, nada más encender la placa, la configuración almacenada a partir de la dirección 0 de la Flash.

Las dos configuraciones de la CPLD se encuentran ampliamente explicadas en [VAND01]

2.3.2 Programación de la Flash RAM con varias configuraciones.

Para cargar dos archivos de configuración, se intentó generar un archivo MCS que contuviera las dos configuraciones.

A partir de esta fase se vio que no se podían utilizar los archivos MCS, debido a que la versión del PROMGEN incluida en el Xilinx Foundation 3.1 no es capaz de generar archivos de este tipo para tamaños de más de 1 Mbyte. Como cada archivo de configuración total de la FPGA ocupa 575 Kbytes, no podíamos albergar dos de estos archivos en un mismo MCS. (Más tarde se vio que la versión 4 del Xilin Foundation no tenía esta restricción.)

Por esta razón, a partir de este momento se empezó a usar archivos tipo EXO de forma similar a los archivos MCS.

La forma en la que se generan estos archivos de programación de la Flash es:

promgen -u 0 file1.bit -u 100000 file2.bit -p exo -s 2048

Esta orden genera un archivo EXO a partir del cual se cagara en la Flash RAM el archivo de configuración **file1** a partir de la posición 0 de memoria, y el archivo **file2** a partir de la posición 1Mbyte, como se ve en lo mostrado por pantalla tras la ejecución:

PROMGEN: Xilinx Prom Generator D.19
Copyright (c) 1995-2000 Xilinx, Inc. All rights reserved.

promgen -u 0 file1.bit -u 100000 file2.bit -p exo -s 2048

PROM sum.prm map: Mon Aug 26 10:18:40 2002

Format Exormax
Size 2048K
PROM start 00000000
PROM end 001fffff

Addr1	*Addr2 File(s)*
00000000	*0008fe8b file1.bit*
00100000	*0018fe8b file2.bit*

Cargamos este archivo de forma similar a los anteriores usando el GXSLOAD.

En este momento, debido a la configuración que se carga en el CPLD al final del proceso por el cual GXSLOAD programa la Flash RAM, sólo se carga el primer archivo de configuración al encender la placa. Por ello debemos modificar la configuración del CPLD. Para ello generaremos nuevos archivos SVF.

2.3.3 Configuración de la FPGA desde la Flash RAM.

XESS proporciona el código de un interfaz para la comunicación y configuración de la FPGA a través de la Flash RAM. El funcionamiento de este interfaz, analizado con detalle en [VAND01], es el siguiente:

1. Se configura la Virtex FPGA en modo **SelectMap** para que pueda ser programada desde la Flash RAM. En este modo, la FPGA acepta bytes de configuración en el flanco de subida del reloj de configuración , mientras que las señales **chip-select** y **write-enable** están activas.

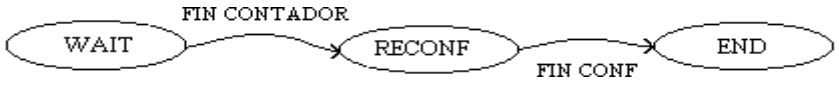
2. Se realiza una división de frecuencia de la señal de reloj. Esto es debido a que la Flash tiene un tiempo de acceso de 85 ns, mientras que el oscilador de la placa puede alcanzar una frecuencia de 100 MHz. Por ello se divide la frecuencia del oscilador entre 16 y se usa un reloj más lento para la configuración de la Virtex.

3. Tras el encendido de la placa es necesario esperar un tiempo antes de que ésta esté lista para que la configuración comience. Para ello se implementa un contador hardware y se deshabilita la configuración hasta que éste haya finalizado.

4. A partir de este momento el circuito simplemente incrementa el contador de direcciones para leer el siguiente byte de la Flash Ram y lo envía a la Virtex FPGA.

5. Cuando la FPGA indica que la configuración ha finalizado, el CPLD cesa sus operaciones. En este momento se ponen todas las líneas usadas por el CPLD que están compartidas con la FPGA a alta impedancia para que puedan ser utilizadas por el nuevo circuito cargado en la Virtex.

El siguiente diagrama muestra los pasos necesarios en la configuración de la Virtex:



2.3.4 Generación de archivos .SVF para la configuración.

Los archivos SVF son con los que, a través del GXSLOAD, se configura el circuito del CPLD.

Para generarlos con la herramienta Xilinx Foundation 3.1 se debe:

1. Crear un proyecto que incluya los archivos vhdl con el funcionamiento del circuito.

2. Antes de llevar a cabo la fase de síntesis hay que elegir como modo de codificación de estados el modo binario. Para ello:

- Pulsar en el menú superior el botón de *Synthesis* y:
  Options: FSM Synthesis: Default encoding: Binary. OK

- Forzar el análisis de todos los ficheros fuente:
  Synthesis → Force Analysis of all sources

3. Realizar la síntesis y el análisis eligiendo correctamente la familia y el chip concreto de CPLD. En nuestro caso son:
- ***Family:*** *XC9500*
- ***Device:*** *95108TQ100*

4. Después de lograr una implementación libre de errores se debe abrir la aplicación JTAG Programmer. Para ello hay que seleccionar en el menú:

- *Implementation → Tools → JTAG Programmer*.

O elegir la opción de *"Programming"* en la ventana principal de la herramienta.

5. Una vez abierta la aplicación hay que realizar los siguientes pasos:

1. En el menú se selecciona: *Output → Create SVF File...*
2. Elegir la opción *"Through Test-Logic-Reset"*
3. De nuevo en el menú: *Operations → Program*,
   donde hay que marcar las opciones: - *Erase Before Programing*
   - *Verify*

6. Tras estos pasos, en la carpeta **\xproj\ver1\rev1** del proyecto se podrá encontrar el archivo SVF generado.

Este archivo puede ser cargado mediante el GXSLOAD (ver 2.2.2.3) Es importante saber que cada vez que se produce la reconfiguración del CPLD, la placa es reiniciada, pasando automáticamente a funcionar el interfaz recién cargado.

2.3.5 Mapas de bits.

La memoria interna de configuración de la Virtex está dividida en segmentos llamados Frames. Las porciones de bitstream (mapas de bits) que son escritas en la memoria de configuración son los "Data Frame". El número y tamaño de los Frames depende del modelo de FPGA que estemos usando. Así en nuestro caso al usar una Virtex 800 tenemos 4333 Frames de 1088 bits cada uno. Además todos los datos de configuración, excepto la palabra de sincronización y las dummy word, son escritos en los registros de configuración. La explicación de qué hace cada uno de los registros de configuración la podemos encontrar en [XILI00], aunque a continuación se incluye cuales son dichos registros y sus direcciones:

Symbol	Register Name	Address
CMD	Command	0100b
FLR	Frame Length	1011b
COR	Configuration Option	1001b
MASK	Control Mask	0110b
CTL	Control	0101b
FAR	Frame Address	0001b
FDRI	Frame Data Input	0010b
CRC	Cyclic Redundancy Check	0000b
FDRO	Frame Data Output	0011b
LOUT	Daisy-chain Data Output (DOUT)	1000b

Por lo tanto la memoria de configuración de la Virtex se puede imaginar como un array rectangular de bits. Los bits son agrupados en frames verticales que son de un bit de anchura y van desde la parte superior del array hasta la parte inferior. Un frame es la unidad mínima de configuración, es decir es la porción más pequeña de la memoria de configuración que puede ser escrita o leída. Los frames son agrupados en columnas. Existen, sin embargo, distintos tipos de columnas. A continuación se incluye un cuadro que contiene el número de frames por columna y el número de columnas que podemos encontrar de cada tipo en la Virtex.

Column Type	# of Frames	# per Device
Center	8	1
CLB	48	# of CLB columns
IOB	54	2
Block SelectRAM Interconnect	27	# of Block SelectRAM columns
Block SelectRAM Content	64	# of Block SelectRAM columns

El primer punto para la elaboración del .bit parcial consiste en la identificación de la columna 6 dentro de laura1.bit. Para esto debemos tener en cuenta las siguientes definiciones y ecuaciones. Estas tablas las podemos encontrar también en [XILI00b].

Term	Definition
Chip_Cols	# of CLB columns on the Virtex device
Chip_Rows	# of CLB rows on the Virtex device
Chip_Rams	# of Block SelectRAM columns on the Virtex device
RAM_Space	Spacing of Block SelectRAM columns (in terms of CLB columns)
FL	# of 32-bit words in the frame.
RW	1 for Read, 0 for Write
CLB_Col	Column number of the desired CLB
CLB_Row	Row number of the desired CLB
Slice	0 or 1
FG	0 for the F-LUT, 1 for the G-LUT
lut_bit	The desired bit from the given LUT. Bits in the LUT are indexed from 0 to 15.
XY	0 for the X Flip-Flop, 1 for the Y Flip-Flop
RAM_Col	Column number of the desired Block SelectRAM
RAM_Row	Row number of the desired Block SelectRAM
ram_bit	The desired bit from the given Block SelectRAM. Bits are indexed from 0 to 4095.

Term	Definition
MJA	Frame Major Address
MNA	Frame Minor Address
fm_st_wd	The index of the word within a full configuration segment that corresponds to the starting word of the desired frame. A full configuration segment is defined as the following: 1) for CLB/IOB, all CLB, IOB, and RAM interconnect frames beginning at MJA=0, MNA=0 and 2) for Block SelectRAM, all RAM content frames for the given RAM column. Words are numbered starting at 0.

Term	Definition
fm_wd	The index of the 32-bit word within the frame that contains the desired bit. Words in a frame are numbered starting at 0.
fm_wd_bit_idx	The bit index of the desired bit within frame word fm_wd. Words are indexed in the "big-endian" style, with bit 31 on the left and bit 0 on the right.
fm_bit_idx	Bit index within a frame of the desired bit. Numbered starting with 0 as the left-most (first) bit. Bit numbering within a frame continues across all the words in the frame.

Term	Definition
MJA	if (CLB_Col ≤ Chip_Cols/2), then Chip_Cols – CLB_Col × 2 + 2 else 2 × CLB_Col – Chip_Cols – 1
MNA	lut_bit + 32 – Slice × (2 × lut_bit + 17)
fm_bit_idx	3 + 18 × CLB_Row – FG + RW × 32
fm_st_wd	FL × (8 + (MJA –1) × 48 + MNA) + RW × FL
fm_wd	floor(fm_bit_idx/32)
fm_wd_bit_idx	31 + 32 × fm_wd – fm_bit_idx

La variable que debemos utilizar para encontrar la primera dirección de memoria por donde debemos empezar a cortar el archivo es **fm_st_wd.** Esta variable sirve para localizar la primera dirección de memoria de un frame determinado.

3. MODIFICACIÓN DE LA ARQUITECTURA MIPS

3.1 VERSIONES DE LA CACHE IMPLEMENTADAS

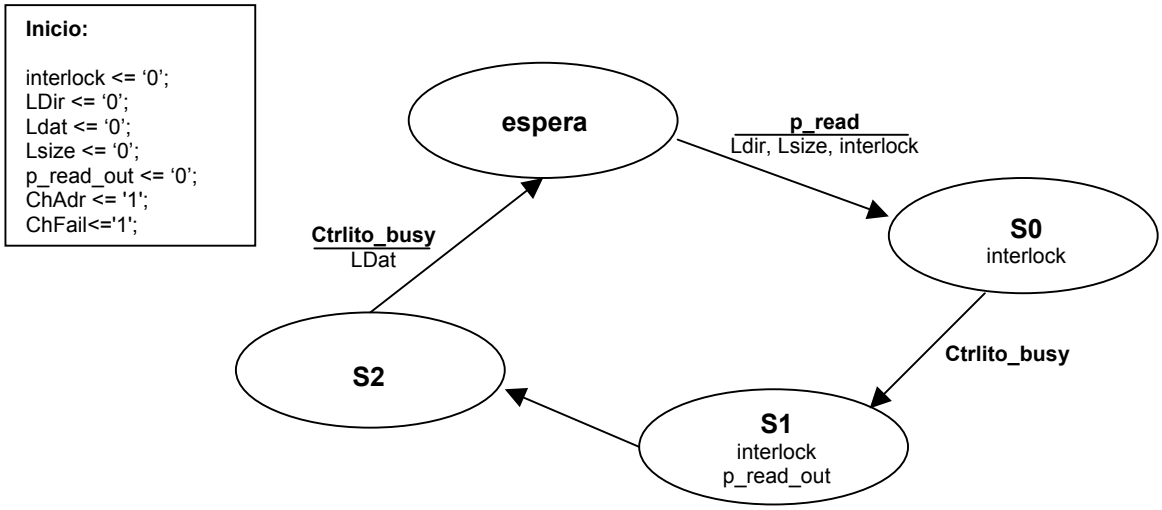
Las dos versiones funcionales de la cache no son memorias cache habituales sino que actúan como buffers intermedios de lectura entre el micro y la memoria principal. La diferencia entre las dos versiones radica en la cantidad de veces que el módulo importa la palabra de memoria principal antes de suministrársela al micro:

- **Cache:** Una vez.

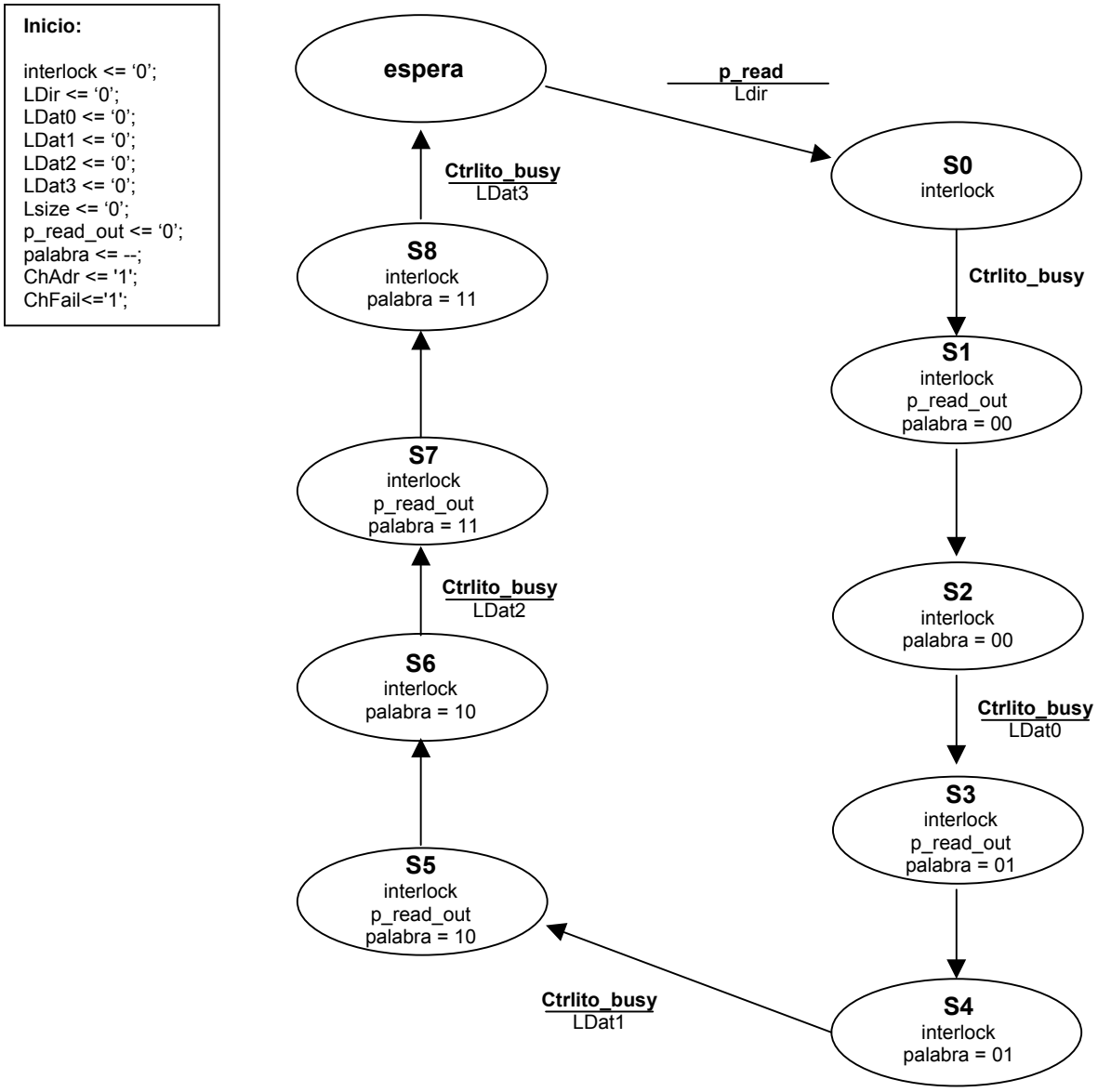
- **Cache_palabras:** Cuatro veces. Los motivos de esta versión vienen dados por las pruebas previas (apartado 1.3) en las que se intentó traer un bloque (cuatro palabras) por medio de la transmisión con memoria principal de tamaño bloque. Posteriormente se intentó trabajar con pseudobloques (4 palabras, transmisión palabra a palabra) pero con el mismo resultado, de modo que finalmente y para tener dos versiones cuyas diferencias fuesen "palpables" de cara a las pruebas de reconfiguración se optó por trabajar con esta última.

Los siguientes diagramas muestran las máquinas de estados de ambas versiones:

Cache:



Cache_palabras:

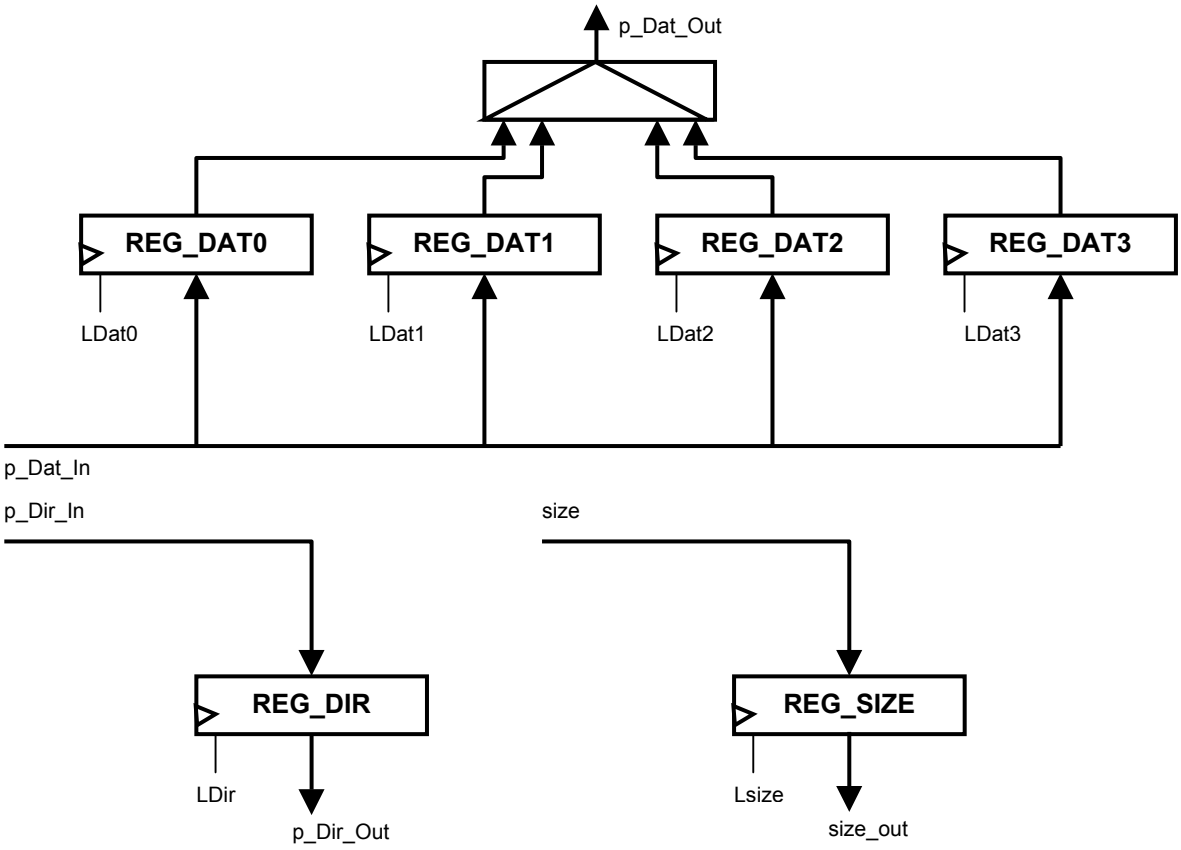


Señales:

p_read: señal de petición de lectura proveniente del micro.
p_read_out: señal de petición de lectura de la cache al controlito.
LDat (LDat0, LDat1, LDat2, LDat3) : señales de carga de los registros de datos.
LDir: señal de carga del registro de dirección (dir. de lectura).
Lsize: señal de carga del registro de tamaño de los datos.
interlock: señal de inhibición del micro.
palabra: entrada de control del multiplexor que direcciona los datos de salida (en esta versión no es funcional y los datos que se suministran al micro son los que amlacena el registro **REG_DAT0**).

ChAdr: indica si la dirección es cacheable o no. Tomamos todas las direcciones como cacheables.
ChFail: indica que ha habido un fallo de cache.

ELEMENTOS ESTRUCTURALES / RUTA DE DATOS (CACHE_PALABRAS):



3.2 CAMBIO DE PROTOCOLO E/S (CORE R2000)

3.2.1 Protocolo. Comunicación.

El nuevo protocolo es un sencillo protocolo tipo strobe que funciona del siguiente modo:

a) Lecturas

La CPU activa la señal **p_read** indicando la petición de una nueva lectura de tamaño **size**. La dirección de comienzo del dato es **p_DirOut**. Los multiplexores de la ruta determinan quién atenderá la petición:

- **Reconfig = '0' :** La cache está en fase de reconfiguración, lo cual indica que las peticiones de lectura por parte de la CPU serán atendidas directamente por el controlito, que traerá los datos de memoria principal y los proporcionará al micro. Durante los ciclos en los que el controlito se comunica con la RAM para proporcionar los datos a la CPU, ésta queda inhibida mediante la señal **interlock**, que en este caso depende exclusivamente del estado del controlito.

- **Reconfig = '1' :** La cache es funcional y atiende las peticiones de la CPU. Una vez detectada la señal **p_read** la cache inhibe al micro activando la señal de **interlock**. Si la cache posee los datos que se requieren los suministra y desactiva la línea de **interlock**. En caso contrario realiza una petición al controlito mediante la activación de la señal **p_readOut**. La dirección de comienzo de la lectura es la orginal, propagada a través de la línea **p_DirOut.** Una vez el controlito le ha suministrado los datos, la cache los proporciona a la CPU y la desinhibe.

b) Escrituras

La CPU activa la señal **p_write** indicando la petición de una nueva escritura de tamaño **size**. La ubicación del dato está indicada en **p_DirIn** y el dato en **p_DatOut**. Los multiplexores de la ruta determinarán siempre que las peticiones de escritura las atienda el controlito, ya que en esta versión del proyecto sólo están implementadas las lecturas a través de la cache.

3.2.2 Protocolo. Entidades afectadas.

Las entidades que han pasado una etapa de rediseño para la adaptación al nuevo protocolo han sido las siguientes:

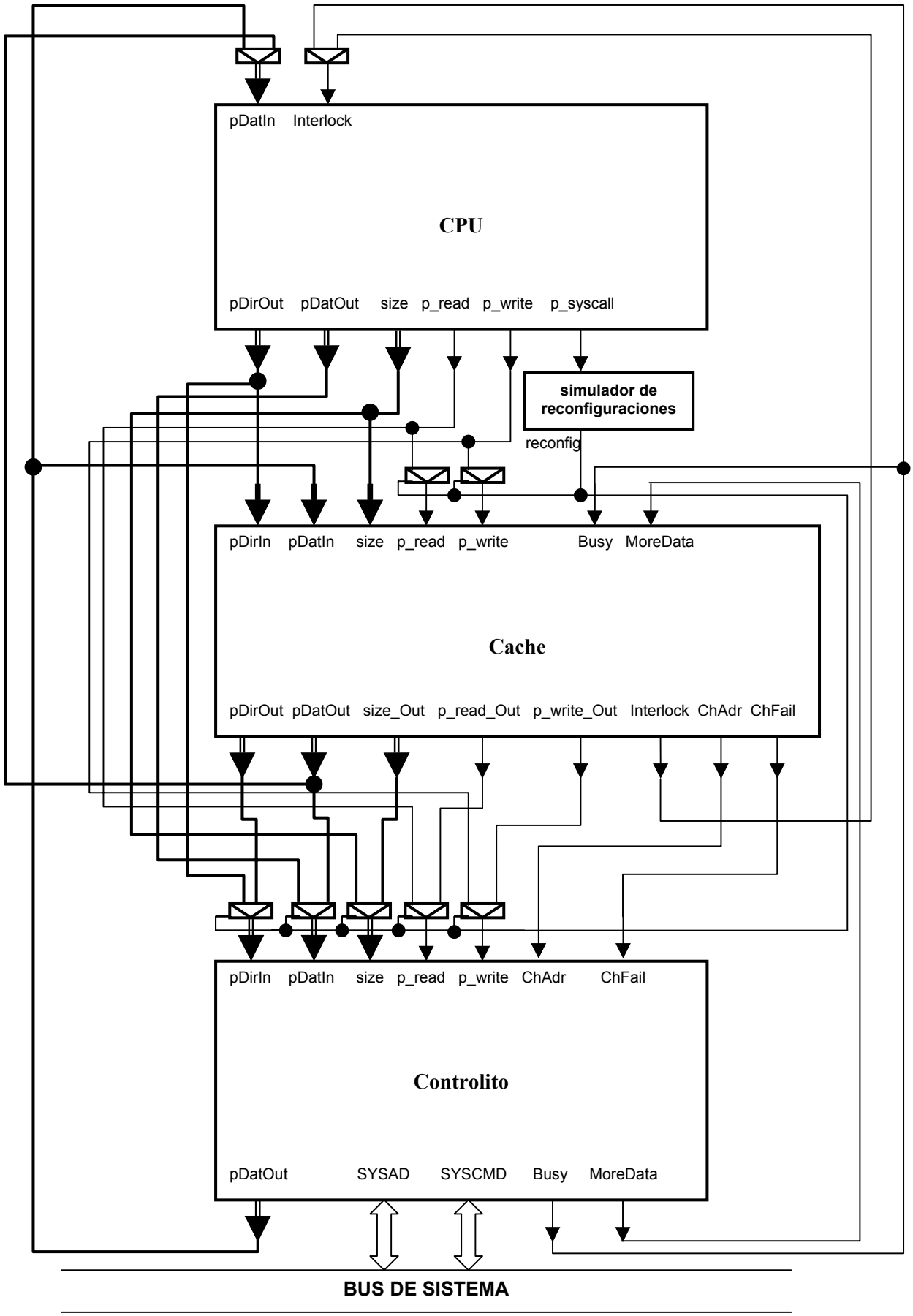
- **CPU y unidad de control:** cambios de consideración: no trabajar con buses bidireccionales o no emplear un bus de control para indicar datos de relevancia como el tamaño de la transmisión y otros de no tanta como el origen de la petición o el dato (información que podemos omitir si interpretamos el protocolo a modo maestro-esclavo donde todas las peticiones las realiza el micro) han dado como consecuencia modificaciones importantes.

- **Controlito y unidad de control del controlito**: en una medida similar a las dos entidades anteriores.

- **Ruta de datos:** Reestructuración de líneas.

3.3 NUEVA RUTA DE DATOS. RECONFIGURACIÓN

3.3.1 Nueva ruta de datos (señales competentes al protocolo).



Las modificaciones en la ruta de datos para la implementación de la reconfiguración se han llevado a cabo mediante la introducción de multiplexores que regulan el flujo de datos entre el micro y la cache o el controlito y la introducción de señales de control y elementos de almacenamiento que registran el estado actual del sistema.

3.3.2 Sincronismo con Interlock.

Uno de los problemas que plantea la reconfiguración es el instante de tiempo en que ha de tomarse otra cache como parte funcional y dejar de utilizar la que está funcionando actualmente.

La altenativa elegida ha sido la siguiente:

- Una vez la instrucción de programa correspondiente haya solicitado el cambio de cache el micro lo traslada a la ruta y ésta no registra la petición hasta que la señal de **interlock** no esté desactivada, ya que en caso contrario significaría que el micro está bloqueado y que por lo tanto la cache está actualmente cursando una operación de E/S. De este modo nos aseguramos que la cache ha concluído su operación en el momento de ser "retirada" de la ruta. Lo mismo ocurre en cuanto al final de la reconfiguración, **interlock** actúa como señal de carga del registro que indica el estado actual de la máquina (reconfigurando o no).

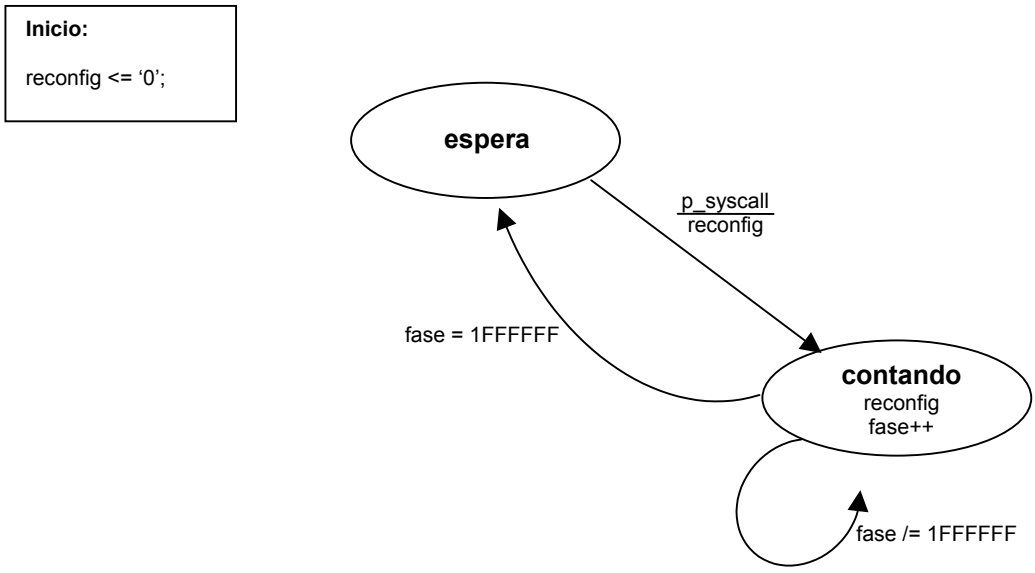
3.3.3 Pruebas preliminares (switches).

Para una primera aproximación existe una versión que reconfigura seleccionando y tomando las peticiones mediante del primer switch de la placa.

3.3.4 Simulador de reconfiguraciones.

Este módulo hace las funciones de CPLD-Flash a modo de simulador. No es más que un contador que permanece en espera por un periodo de algo más de 2 segundos. Su objetivo: introducir un retardo en el cual el R2000 esté trabajando sin cache cada vez que existe una petición de reconfiración. Una vez pasado este período de tiempo se volverá a trabajar con la cache que corresponda.

DIAGRAMA DE ESTADOS DEL SIMULADOR DE RECONFIGURACIONES:



Señales:

p_syscall: señal externa que indica la petición de reconfiguración.
reconfig: activa en alta, indica que se está llevando a cabo la "reconfiguración"
fase: estado del contador. Cuando alcance el valor hexadecimal "1FFFFFF" volverá a cero dando por concluido el período de reconfiguración. Teniendo en cuenta que el reloj oscila a una frecuencia de 12.5 Mhz este período es equivalente a 2'68 segundos.

3.4 INSTRUCCIÓN DE RECONFIGURACIÓN

El repertorio del MIPS consta de un gran número de instrucciones con una funcionalidad determinada, pero ninguna de ellas está pensada para abordar el problema de la reconfiguración. A tal efecto, se hace necesario enriquecer el repertorio de instrucciones del MIPS.

El problema se puede enfocar desde dos perspectivas distintas: la primera de ellas, consiste en pensar una nueva instrucción con un identificador que no esté recogido en el repertorio de instrucciones; la segunda, cambiar la funcionalidad de una instrucción recogida en dicho documento, para que actúe ordenando la reconfiguración.

Incluir una nueva instrucción con un identificador nuevo conlleva implícitamente realizar modificaciones en el Pcspim [La97], que es el traductor de código ensamblador a código máquina que se emplea, para que reconozca la instrucción y la codifique convenientemente. De otra manera, la traducción de esta instrucción debe hacerse a mano, incluyendo ceros y unos con el formato adecuado de la instrucción, directamente en los archivos .paca que se quieren probar en el MIPS. Sin embargo, tomando la vía de reutilizar una instrucción ya conocida por el MIPS, y por tanto por el pcspim, se ahorra el tener que realizar nuevas versiones del pcspim, o el tener que incluir código binario en archivos .paca. Por esta razón, se toma la decisión de utilizar una instrucción incluída ya en el repertorio del MIPS, pero no implementada en el proyecto inicial.

Una vez decidido el camino a tomar, el siguiente problema a considerar es cuál de estas instrucciones puede ser candidato a ejercer tal función, la decisión no debe ser azarosa, debe tomarse con una cierta lógica para que el uso del espacio útil del computador sea lo más eficiente posible (registros...), y para que la repercusión en los programas sea mínima.

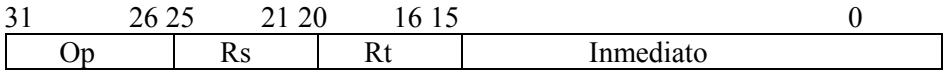
3.4.1 Estudio del repertorio de instrucciones.

Se busca una operación que tenga unas características adecuadas al uso que se va a hacer de ella, en cuanto al formato de la instrucción, modo de direccionamiento, etc...

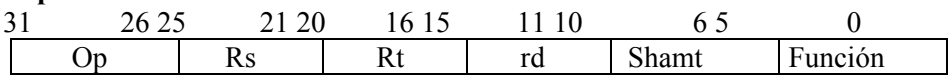
Ha de tenerse en cuenta que las únicas necesidades de dicha operación serán: por una parte el código de la operación, que está asociado al identificador de la instrucción, y un único operando, que se utiliza para escoger una cache determinada.

Los modos de direccionamiento utilizados por las operaciones son los siguientes:

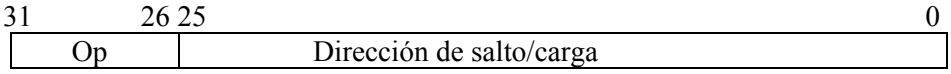
Tipo I:



Tipo R:



Tipo J:



Las instrucciones del MIPS se dividen en cinco grupos:

a) Aritméticas, lógicas, desplazamientos

Utilizan dos modos de direccionamiento: Tipo R y tipo I.

Ambos modos de direccionamiento son inadecuados para una instrucción que pretende tener sólo un operando, puesto que ambos formatos incluyen dos.

b) Cargas y descargas

Sólo soportan un modo de direccionamiento: Tipo I.

Este formato parece inadecuado puesto que también contiene dos campos, de modo que uno de ellos sería desperdiciado, y añadiríamos una complejidad a la instrucción innecesaria.

c) Saltos

Tienen tres modos de direccionamiento: Tipo R, tipo I, tipo J.

Podría valer cualquiera de las que tienen direccionamiento de tipo J, el inconveniente surge del hecho de que las instrucciones que utilizan este modo de direccionamiento están ya incluídas en el repertorio de instrucciones, y no es deseable privar al computador de funcionalidades que ya contenía.

d) De coprocesador

Tipo de direccionamiento : Tipo I.

Este modo de direccionamiento contiene más parámetros de los que son necesarios en la operación que se quiere implementar.

e) Especiales

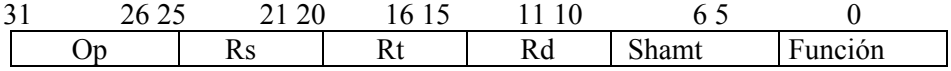
Modo de direccionamiento: Tipo R, con variaciones según la instrucción concreta que estemos explorando.

En principio no parecen tener un modo de direccionamiento adecuado.

Ahora, puesto que el modo de direccionamiento más adecuado es de tipo J, y como todas las instrucciones que contemplan este modo de direccionamiento ya han sido implementadas, se decide revisar las instrucciones especiales, puesto que tienen un formato especial, tipo R con variantes.

Algunas de las instrucciones denominadas como especiales son :

Break, syscall.



Para Break se tiene:

Op = 000000 (Especial)
Función = 001101 (Break)

Resto = Rs+Rt+Rd+Shamt son interpretados como un único campo que indica el código de excepción.

Para Syscall se tiene:

Op = 000000 (Especial)
Función = 001100 (Syscall)
Resto = Rs+Rt+Rd son interpretados como un único campo que indica el código de la llamada al sistema.

Existen otras instrucciones especiales : Teq (trap if equal), Tltu (trap if less than unsigned), Tge (Trap if greater than or equal),… Pero hacen una interpretación más literal del direccionamiento de tipo R, así que incluyen más parámetros de los necesarios. Por tanto, se deja syscall y break como candidatos.

El siguiente paso es considerar el lenguaje ensamblador que se utiliza, para terminar de concretar la elección.

El lenguaje ensamblador para syscall y break :

En el caso de syscall se observa que existen distintas llamadas al sistema, y que su codificación en lenguaje ensamblador pasa por el uso de distintos registros, por ejemplo:

Para escribir un entero :

```
li $v0,1    (código de llamada al sistema para print-int)
li $a0,5    (entero a escribir)
syscall     (ejecutar syscall)
```

Para escribir un carácter:

```
li $v0, 4    (código de llamada al sistema para print-str)
la $a0, str  (dirección del string a escribir)
syscall      (ejecutar syscall)
```

El lenguaje ensamblador que se utiliza para el break es el siquiente:

Break n

Por tanto, por simplicidad se escoge el break, puesto que bastaría utilizar el parámetro **n** como código de la cache que se quiere poner en funcionamiento.

3.4.2 Ciclo de instrucción.

Una vez escogida la instrucción hay que decidir cómo se va a implementar, esto es, fijar su comportamiento en cada ciclo de instrucción. En un primer lugar se muestra el cauce de las instrucciones del MIPS, para explicar las decisiones tomadas a ese respecto.

IF : Se trae de memoria la instrucción actual.

IS : Se ha terminado el proceso de traer la instrucción actual desde memoria.

RF : Se decodifica la instrucción y se extraen los operandos de los registros adecuados.

EX : Ejecución.

La alu actúa en función de la instrucción que se está ejecutando de la siguiente manera:

- En caso de operaciones aritméticas o lógicas: Calcula el valor de la operación.
- En caso de cargas o descargas: Calcula la dirección en la que se va a guardar la información, o desde la que se va a traer.
- En caso de saltos: Calcula las condiciones de los mismos, y las direcciones de salto.

DF : En este caso también es la operación la que marca la manera de actuar:

- Si la instrucción es de carga o descarga, la dirección implicada se halla en este paso.
- Se actualizan las direcciones de salto, para los saltos.
- No se hace nada para las instrucciones registro a registro.

DS : Según la instrucción que tengamos sucederá :

- Se ha completado la traducción de las direcciones para instrucciones de carga y descarga.
- Se ha terminado de actualizar las direcciones de salto para los saltos.
- En este ciclo tampoco sucede nada para instrucciones registro a registro.

WB : Los resultados de las instrucciones con formato registro a registro son escritos en el banco de registros. Los saltos condicionales con la condición falsa y las cargas y descargas no hacen nada en este ciclo de instrucción.

Algunos de estos ciclos de instrucción son comunes para todas las instrucciones, la fase de decodificación de la instrucción por ejemplo o la fase de ejecución, por tanto la instrucción nueva pasará por IF, IS y EX, de la misma manera que lo hacen todas.

La discusión tendría que centrarse pues en el resto de fases:

RF : Es la fase de extracción de operandos, y por tanto el momento en que se extrae el operando para break, este operando tiene como significado asignar el número de cache a la que se utilizar.

DF: En esta fase se tratan direcciones de memoria, por tanto como esta instrucción no implica direcciones de memoria en este ciclo no hace nada, igual que nop, o las operaciones registro a registro.

DS : En esta fase tampoco se hace nada, se actúa como si se tratase de un nop.

WB : La operación introducida no implica reescritura de resultado en el banco de registro puesto que no se calcula ningún resultado en ella, por tanto en esta fase también se comportará como un nop.

Por tanto, habrá que modificar el control microprogramado para que actúe en consecuencia con las decisiones tomadas. Para esto se introduce una entrada nueva en la tabla de la rom del microprograma, que se corresponde con la fase DF; y en el resto de ciclos, para los comunes no

es necesario hacer nada, puesto que por defecto la nueva instrucción pasará por ahí igual que todas, y para aquellos en los que se comporta como un nop se introduce el código de operación de la instrucción en las condiciones.

Respecto a la inclusión de una nueva entrada en la tabla, cabe señalar que además se hizo necesario incluir un nuevo bit de código de condición para indicar la reconfiguración (bitRegConfig). Esta solución fue adoptada por sencillez, aunque otra opción posible hubiera sido concebir una configuración no utilizada de los bits de configuración para ello. Esta señal se mantendrá a '0' siempre excepto en la fase RF para el break.
Se muestra la nueva entrada, con los bits de condición que se ven afectados (por problemas de espacio).

	DIR	M1	M2	M3	M4	BitRegConfig
24	RF break	00	00	01	010	1

3.4.3 Selección de cache.

La propia instrucción lleva explícito un campo que indica la memoria cache que se ha escogido, por tanto, simplemente se analiza este campo y se selecciona la memoria escogida.

3.4.4 Archivos vhdl implicados.

Los ficheros vhdl que se ven afectados por la nueva instrucción, son aquellos que tienen que ver con el control microprogramado, por razones evidentes explicadas con antelación, así como la propia cpu, debido a que es ésta al final quien va a ordenar la reconfiguración.

Estos archivos son : *cpu.vhd, uprogrom.vhd, uc.vhd*.

Los cambios realizados en uprogrom son los descritos antes, puesto que es la unidad encargada del control microprogramado.

En cuanto a la unidad de control, los cambios introducidos son para propagar la nueva señal.

En la cpu se introduce un módulo para calcular el número de cache que se quiere utilizar. Se podría haber hecho vía Alu, pero ésta no está dotada de la funcionalidad necesaria para realizar dicho cálculo, de manera que se hubiera tenido que modificar también.
Además, esta señal (numCache) debe ser propagada, junto con la señal que indica que va a haber reconfiguración, a otros módulos.
Por último, se añade una señal de entrada que indica a la cpu que la reconfiguración ha terminado, y por tanto, que se puede disponer de una cache de nuevo.

4. RECONFIGURACIÓN

4.1 RECONFIGURACIÓN PARCIAL

4.1.1 Introducción a la reconfiguración parcial.

Un importante adelanto dentro de la arquitectura de la Virtex es la capacidad de reconfigurar una porción de la FPGA mientras el resto sigue operativo. La reconfiguración parcial es útil para aplicaciones que requieren diferentes diseños dentro de la misma zona de la placa o la flexibilidad de cambiar porciones del diseño sin tener que resetear completamente la placa. Con esta capacidad, se abre la posibilidad de nuevas aplicaciones:

1. Nos permite poner al día y modernizar el hardware que se encuentra en lugares alejados

2. Reconfiguración en tiempo de ejecución

3. Adaptar el hardware a nuevos algoritmos

La **reconfiguración parcial activa** consiste en cargar nuevos datos de reconfiguración en un área específica de la placa, mientras el resto del diseño está operativo. Para las placas actuales, los datos son cargados en columnas básicas, que contienen las unidades mínimas de reconfiguración, los frames, cuyo tamaño varía dependiendo del modelo de la FPGA. La reconfiguración parcial activa normalmente se la llama **reconfiguración parcial**.

Reconfiguración parcial

La r**econfiguración parcial dinámica** se lleva a cabo mientras la placa está activa. Ciertas áreas del diseño se reconfiguran mientras el resto permanecen operativos y no se ven afectadas por la reconfiguración.

Los distintos **estilos de reconfiguración parcial** se pueden dividir en:

1. Reconfiguración parcial de **múltiples columnas, diseños independientes:** este tipo de reconfiguración es apta para módulos que son completamente independientes, es decir, que no hay comunicación entre los módulos. En este caso la reconfiguración de un módulo no afecta a ningún otro.

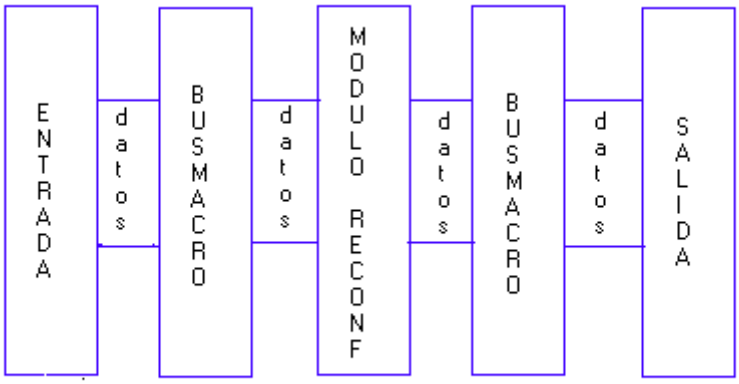
2. Reconfiguración parcial de **múltiples columnas, comunicación entre los diseños:** este tipo de reconfiguración es aplicable para aquellos módulos que se comunican con los demás. Por lo tanto aparece implicada en el diseño una **bus macro** que permite que las señales atraviesen las zonas que se reconfiguran. Sin esta bus macro la comunicación entre los módulos no sería posible porque no sería posible garantizar el enrutamiento entre módulos. Por lo tanto esta macro implementa un bus fijo para la comunicación intermodular. Así siempre que se diseña la reconfiguración parcial, la bus macro es utilizada para establecer un enrutamiento invariable de los canales de comunicación entre los módulos, **garantizando conexiones correctas**.

3. **Pequeñas manipulaciones de bits:** este tipo de reconfiguración se consigue realizando pequeñas variaciones en el diseño (normalmente con la FPGA-editor) y obteniendo mapas de bits que reflejan estas pequeñas variaciones únicamente.

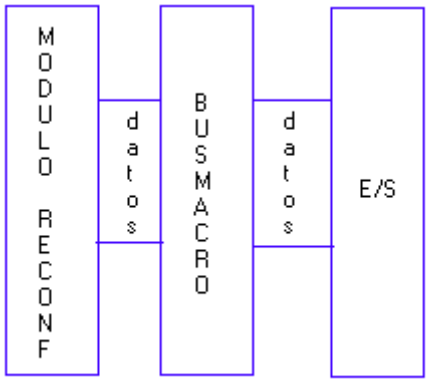
4.1.2 Descripción del diseño de pruebas.

El circuito que hemos utilizado para las pruebas de reconfiguración consta de cuatro entradas introducidas por los botones de la placa. Estas entradas alimentan las ocho entradas de dos puertas and o dos puertas or, cada una de ellas de cuatro entradas. Estas puertas devuelven dos salidas que son mostradas por el display de siete segmentos.

El siguiente diseño modular corresponde al circuito utilizado para las pruebas.



Aunque el circuito fue inicialmente diseñado así, se hizo luego la siguiente simplificación. Esta simplificación fue debida especialmente a que tanto las entradas como salidas del circuito venían de la parte derecha de la placa. Nos ahorramos de esta manera varias bus macro. Ya que las señales no tenían porque atravesar inútilmente la parte reconfigurable del circuito. Recordamos entonces que *cualquier señal que atraviese la parte reconfigurable tiene que estar conectada a través de bus macros.*



Módulos presentes en el diseño:

1. **Entrada:** Este módulo contiene las entradas de nuestro diseño. A pesar de que en el diseño modular simplificado la entrada y la salida se incluyen como un único módulo, los podemos explicar por separado.

2. **Módulo reconfigurable:** Este módulo contiene, dependiendo de la versión del diseño con la que estemos trabajando, dos puertas **and** o dos puertas **or** de cuatro entradas. Así en los proyectos ***laura18*** contendrá las puertas and y en ***laura19*** y ***laura1*** contendrá las puertas **or**. Estos proyectos serán explicados más adelante. Este módulo será la parte reconfigurable de nuestro circuito, es decir, la parte que podrá ser sustituida por otra en tiempo de ejecución del circuito.

3. **Salida:** Esta parte del circuito corresponde al display de siete segmentos. En este display se presentará el cero lógico como **0** y el uno lógico como una **F**.

4. **Bus Macro:** Este módulo requiere una explicación más amplia que los anteriores debido a su importancia para la aplicación de la reconfiguración parcial.

NOTA: Los códigos vhdl y el ucf que los acompaña se pueden encontrar en el **APÉNDICE A**. Estos módulos fueron insertados en los proyectos en forma de archivos **.edf**. Los proyectos que fueron implementados para la obtención de estos archivos se encuentran en el disquete adjunto.

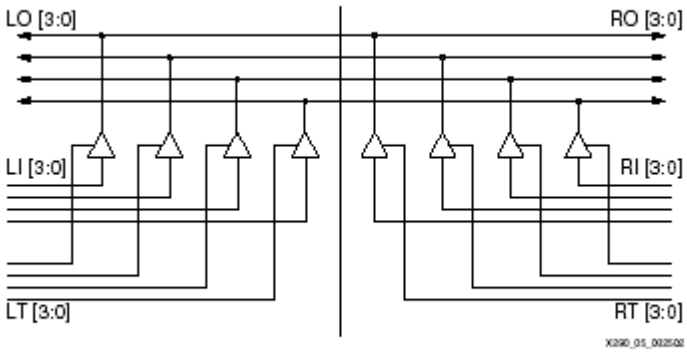
4.1.3 Comunicación mediante bus macro.

Para facilitar la comunicación a través de módulos reconfigurables y otras partes del circuito, y para garantizar que el routing quede completamente fijo y estático a través de dichas zonas, es necesario el uso de un bus especial llamado *bus macro.*

La reconfiguración parcial requiere del uso de señales para la comunicación entre o a través de los módulos reconfigurables, y requieren que el routing de dichas señales quede en la misma posición una vez que sustituimos un módulo por otro, es decir que quede fijo y estático a pesar de la reconfiguración. Los recursos de routing utilizados para las señales que comunican módulos reconfigurables con otras partes del diseño, no deben cambiar al reconfigurar un módulo.

La ***bus macro*** es un **puente** con un **routing fijo** que **comunica** dos partes del diseño, una de esas partes, o las dos, son **módulos reconfigurables**. Así para cada diseño, no habrá ningún tipo de variación en el routing de las bus macro.

La implementación física de una bus macro es la siguiente.



Esta implementación de la bus macro utiliza ocho buffers triestado (**TBUFS**). Permite que un

bit de información viaje de izquierda a derecha y de derecha a izquierda usando un TBUF, dicho buffer estará conectado a **línea larga** de la placa. Cada fila de la Virtex puede soportar cuatro bits de una bus macro. Así el número de bus macro que puede haber en un diseño estará limitado por el número de líneas horizontales disponibles en la FPGA.

Para las pruebas utilizamos distintas formas de generar las bus macro.

Mediante una **hard macro.** Para la creación de dicha macro deberán seguirse los siguientes pasos:

1. Después del routing del diseño abrimos el archivo **.ncd** con el FPGA Editor. Salvamos el diseño como una macro con *File -> Save As*. Y por último ponemos el modo de la FPGA Editor en escritura con *File -> Main Properties -> Edit Mode = Read Write*.

2. Debemos eliminar todos los componentes de **pad**. Para borrar un pad debemos seleccionarlo. Ir a *Tools -> Place -> Unplace*. Esto borrara el pad y la señal que llega a él.

3. Cuando aparezca pintada de verde un pin que previamente habremos borrado, deberemos añadir pines externos a la macro. Para hacerlo deberemos situarnos encima del pin. Ir a *Edit -> Add Macro External Pin*. El nombre que le demos en External Name será el utilizado en la instanciación de la macro.

4. Eliminamos los pad que estén un-placed y las nets un-routed de la lista que aparece en la ventana de la derecha. Seleccionaremos los nombres de las componentes y pulsaremos el botón de Delete.

5. Ahora debemos seleccionar un punto de referencia para mantener siempre las posiciones relativas de los componentes que aparecen en la macro. Para hacer esto debemos seleccionar una CLB que utilizaremos como referencia, e iremos a *Edit -> Set Macro Referente Comp*. Esto creará un **.rpm.**

6. Ahora ya podrá ser instanciada la macro en cualquier componente que queramos. El nombre del archivo de la macro será el nombre de la componente para ser instanciada.

7. En la fase de implementación de la herramienta **Xilinx Foundation 4** le indicaremos la ruta de la carpeta que contiene nuestras macros.

   La herramienta (**Xilinx Foundation**) **no nos permitió introducir la macro dentro de nuestro diseño**. Surgían errores en el proceso de implementación, a pesar de haber seguido todos los puntos tal cual vienen explicados en las *aplications notes*.

8. Utilización de un archivo **.edf** obtenido en el diseño de la bus macro. Para conseguir que el routing fuera fijo de una instanciación a otra, se debe utilizar siempre el mismo **.ucf** que hace referencia a las instancias del archivo edif. Una de las formas más sencillas de ver el nombre de las instancias de los diseños, para luego poder utilizar dichas instancias en los ucfs, es utilizar el **FloorPlanner**. Esta herramienta también es apropiada para generar ucfs sencillos, y utilizarlos como base para restricciones más complicadas.

   Para la generación del edf bastará con realizar la fase de diseño de la herramienta **Xilinx Foundation 3**, en esta fase se genera el archivo deseado.

   Este modo de implementar las bus macro resultó ser el **más adecuado** para las pruebas.

   El código vhdl que utilizamos en la generación de las bus macro lo podemos ver en los

apéndices de este documento.

4.1.4 Proyectos utilizados para las pruebas.

Para las distintas pruebas realizadas en el estudio de la reconfiguración parcial, fueron desarrollados varios proyectos. Los proyectos que al final resultaron completos para su estudio y valoración fueron cuatro: laura18, laura 19, laura1 y laura182. Los tres primeros proyectos fueron desarrollados con la herramienta **Xilinx Foundation F3.1.** El último de los proyectos fue desarrollado con **Xilinx Foundation F4.2**.

A continuación aparece la explicación de cada uno de estos proyectos. Es decir, en que se distinguen de los anteriores y que pruebas fueron realizadas con ellos.

4.1.4.1 Laura18.

Este proyecto sigue exactamente el diseño modular explicado anteriormente. Como módulo reconfigurable tiene dos puertas and de cuatro entradas cada una de ellas. Los distintos módulos fueron introducidos en forma de .edf para que sus instancias se mantuvieran constantes y fueran tratadas por la herramienta de forma estática en cada instanciación.

Una vez que el módulo fue diseñado e implementado se añadieron las *restricciones de usuario* en forma de archivo **.ucf**. Este ucf se puede ver en el apéndice de este documento. Este archivo fue diseñado de tal manera que la parte reconfigurable del diseño fuera totalmente almacenada en la columna 6 de la Virtex. Esto se hizo siguiendo los requisitos de los módulos reconfigurables, que deben estar totalmente implementados en columnas donde no haya ninguna parte más del diseño, tal como se explicó al comienzo del documento.

A pesar de incluir estas restricciones surgió un problema en la implementación del diseño. En la columna 6 no debía aparecer nada excepto el módulo reconfigurable, pero, sin embargo, algunas señales atravesaban dicha columna. Se incluyo en el .ucf del proyecto la sentencia PROHIBIT incluyendo la columna 6 de la placa, pero aún con esta restricción siguieron apareciendo señales atravesando la columna.

Estas señales fueron eliminadas de dicha columna con ayuda del FPGA Editor. Las señales implicadas fueron marcadas, y con la columna de Tools fueron **unrouted** y después **autorouted** de nuevo. Estas dos operaciones deben ser realizadas tantas veces como fuera necesario, hasta que ninguna señal aparezca cruzando la columna reconfigurable.

4.1.4.2 Laura19.

Este proyecto fue desarrollado exactamente igual que el anterior, pero sustituyendo el módulo de puertas and por el módulo de puertas or.

Este proyecto fue diseñado para hacer una comparación entre los archivos de **mapas de bits** (**.bit**) de los proyectos laura18 y laura19.

Una vez entendido en qué consiste el mapa de bits es posible explicar para que se debe realizar la comparación entre los mapas de bits de los dos proyectos.

En ambos proyectos se colocó el módulo de reconfiguración en la columna 6. Esta es la única parte del diseño en el que se distinguían los dos proyectos.

Fueron generados los archivos **.rbt** de ambos proyectos en el momento de la implementación. Este archivo no es más que la traducción a ASCII de los mapas de bits. Teniendo estos archivos en ASCII se puede realizar la comparación de los dos proyectos con un editor de texto común. Una vez realizada la comparación de los dos proyectos se llegó a la conclusión de que ambos proyectos se distinguían en más de una columna, no sólo la columna 6 estaba afectada por diferencias. Estas diferencias a lo largo de todo el archivo se debían al problema que hubo con las señales que atravesaban la columna de reconfiguración y hubo que volver a hacer un **autorouted**. Al dejar que algunas señales fueran enrutadas de forma diferente en uno y otro proyecto se produjeron demasiadas diferencias como para poder cortar la columna 6 del .bit de uno de los proyectos y cargarlo en un .bit parcial sobre el mapa de bit del otro proyecto.

Debido a este gran número de diferencias fue necesario seguir otra línea de investigación para obtener el .bit parcial buscado para cargar sobre la placa.

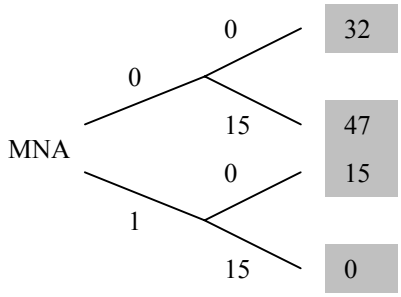
4.1.4.3 Laura1.

Este proyecto es en principio igual a laura18. La diferencia entre ambos fue elaborada con la FPGA Editor. Sobre laura18.ncd se editaron las puertas and. Estas puertas fueron rescritas y transformadas en puertas or. De esta manera ahora sí que los cambios no se distinguían en más de una columna. La columna 6 la única afectada por los cambios es la única columna distinta en ambos proyectos.

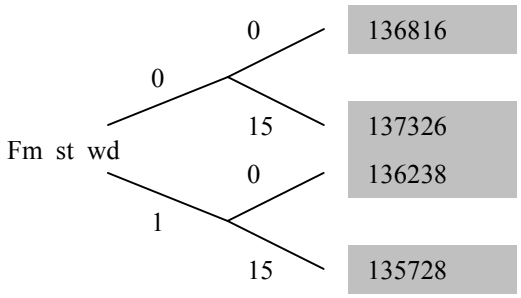
Llegado a este punto el paso siguiente consistía en elaborar el .bit parcial. Una vez elaborado el archivo parcial debemos cargarlo en la placa donde previamente habríamos cargado el .bit total laura18.bit. Por lo tanto la columna 6 que debemos cortar es la que se encuentra en laura1.bit y a partir de esta columna generaríamos el archivo parcial.

Para estar seguros de que no había ningún problema en el circuito una vez cargado el .bit parcial, antes de cargar la columna 6 hicimos pruebas sobre la columna 1. En ambos circuitos esta columna estaba vacía, sin ningún cable ni ningún otro componente que lo afectara. A continuación se incluyen los cálculos para las direcciones de dicha columna 1. Para realizar estos cálculos es necesario utilizar las tablas de ecuaciones descritas en el punto 2.1.18.

MJA = 84



Las distinciones de casos que se han hecho para el MNA son: si estamos considerando el slice 0 ó el l y si estamos considerando el lut_bit 0 (el más pequeño) o el 15 (el mayor).



Las distinciones de casos en esta variable vuelven a ser las mismas.

Ahora ya sabemos entre que direcciones se encuentra la columna 1. Debemos tomar la dirección más pequeña y la dirección mayor, teniendo en cuenta que a la mayor deberemos sumarle 34 palabras ya que solo nos indica donde comienza el último frame, por lo tanto deberemos ver donde acaba dicho frame.

- Primera dirección: 135728
- Última dirección: 137360

Ahora ya casi se está en disposición de cortar la parte del mapa de bit que guarda la columna 1. Pero antes deberemos tener en cuenta los desplazamientos debidos a la cabecera que se incluyen siempre en el archivo .bit. Debido a que el editor de texto binario numera todas las líneas deberemos realizar ese desplazamiento.

En el .bit de nuestro caso deberemos tener en cuenta que hay un desplazamiento de 25 palabras de 32 bits. Estas palabras guardan la cabecera propiamente dicha utilizada para reconocer que se trata de un mapa de bits de una xcv800 válido, y también los comandos utilizados para cargarla en la placa.

Con este desplazamiento tenemos que las dos direcciones anteriores se transforman en 135753 y 137385.

Una vez efectuados los desplazamientos debemos tener en cuenta cómo numera las líneas el editor de texto binario. En nuestro caso el editor numera las palabras de 8 bits, es decir, bytes, y almacena en cada línea 10 bytes.

Teniendo en cuenta que nosotros trabajamos con palabras de 32 bits deberemos multiplicar por 4 nuestras direcciones para obtener las direcciones válidas en el editor.

Así las dos direcciones que deberemos tener en cuenta son: 543012 y 549540. Estas direcciones expresadas en hexadecimal: **84924** y **862A4.** Estas son las dos direcciones que guardan la columna 1 expresadas adecuadamente en el formato que utiliza el editor común binario. La diferencia entra ambas direcciones es de 1632 palabras, exactamente el contenido de 48 frames de 34 palabras, es decir, una columna.

Una vez cortada la columna 1 debemos construir el .bit parcial. Es decir, debemos incluir el conjunto de comandos válidos para que el mapa de bits se cargue apropiadamente en la placa.

A continuación se incluye el aspecto de los comandos de un .bit parcial con los comandos básicos, este aspecto se dedujo teniendo en cuenta las XAPPs.

Cabecera	
FF FF FF FF	***dummy word***
AA 99 55 66	***palabra de sincronismo***
30 00 80 01	***registro CMD***
00 00 00 07	***reset del CRC***
30 00 20 01	***escribir en el registro FAR***
00 A8 00 00	***Nos situamos en la primera palabra (MJA)***
30 00 80 01	***registro CMD***
00 00 00 01	***indicamos la opción WCFG (escritura de configuración)***
30 00 40 00	***registro FDRI***
50 00 0660	***escribir 1632 palabras sobre la placa***
...	
Datos	
...	
FF FF FF FF	***dummy word***
30 00 80 01	***registro CMD***
00 00 00 00	***NOP***
FF FF FF FF	***dummy word***

Para obtener una cabecera válida podremos cortarla de un mapa de bits completo.

La dummy word que se incluye justo después de los datos es debida a que el registro FDRI es un registro de desplazamiento y para que se cargue la última palabra es necesario que esta se desplace con otra palabra que venga a continuación.

Esta primera prueba sobre la placa resultó fallida, la placa se quedaba en estado de reconfiguración, con los dos displays encendidos. Por lo tanto se comenzaron a realizar distintas pruebas.

1. La primera prueba que hicimos fue resetear de nuevo el CRC una vez que se habían cargado los datos.

...	
Datos	
...	
FF FF FF FF	***dummy word***
30 00 80 01	***registro CMD***
00 00 00 07	***reset del CRC***
30 00 80 01	***registro CMD***
00 00 00 00	***NOP***
FF FF FF FF	***dummy word***

   Con esta prueba la placa salía del estado de reconfiguración pero no realizaba ninguna operación.

   Esta prueba nos llevó a la conclusión de que el problema se encontraba en el CRC. En las XAPPs que hablan sobre reconfiguración parcial se dice que no es necesario calcular el CRC para un .bit parcial, pero sin embargo el problema de que la placa no realizará ninguna operación parece estar en este punto. La placa al no detectar un CRC válido al final del archivo resetea su contenido.

2. Se desarrolló un programa en C que implementa el algoritmo del cálculo del CRC. Dicho programa está incluido en los apéndices de este documento. Este programa sigue al pie de la letra el algoritmo que está incluido en la [XILI00b]. Después de diversas pruebas no se consiguió que el programa calculara un crc válido sobre un .bit total, a pesar de que se siguieron exactamente las especificaciones incluidas en el documento de

Xilinx.

Por lo tanto se abandonó esta línea de investigación después de realizar muchas pruebas con el programa y no obtener ningún resultado válido.

Las pruebas se realizaron sobre un .bit total donde fueron cortados todos los valores después del cálculo del primer crc, esto es debido a que en un .bit se calculan 2 crc´s. Uno antes del último frame y otro después. Se eliminó también la cabecera ya que podría contener valores que dieran resultados inválidos, aunque también se hicieron pruebas con la cabecera incluida…

3. La siguiente prueba que se realizó fue incluir un dummy frame al final de la carga de los datos. Esta idea se sacó de la [XILI00]. Un dummy frame está totalmente formado por 1´s. Para incluir este dummy frame habrá que modificar el número de palabras que se van a cargar en el FDRI.

```
...
30 00 40 00          registro FDRI
50 00 0682           escribir 1632+34 palabras sobre la placa
...
Datos
...
FF FF FF FF          Dummy frame formado por 34 dummy words
...
...
FF FF FF FF
FF FF FF FF          dummy word
30 00 80 01          registro CMD
00 00 00 07          reset del CRC
30 00 80 01          registro CMD
00 00 00 00          NOP
FF FF FF FF          dummy word
```

Con este cambio tampoco se consiguió que la placa realizara operaciones una vez que se cargaba el archivo sobre la placa.

4. Otra modificación que se intentó con respecto al crc fue cargar una palabra formada por 0´s, para que cuando el circuito hiciera la comprobación, es decir, una xor con el valor correcto del crc, produjera una suma válida.

```
30 00 00 01          escribir sobre el crc
00 00 00 00          palabra nula
```

Tampoco el resultado fue satisfactorio en este caso. Su inclusión no modificaba en absoluto el comportamiento de la placa.

5. Aunque con el tipo de reconfiguración que estamos utilizando no sería necesario realizar una secuencia de **start up** una de las pruebas que realizamos fue forzar su realización. Dicha secuencia consiste en pasar la placa de estado de configuración a estado operacional. Este paso no sería necesario en principio ya que nosotros estamos trabajando siempre sobre una placa activa.

Después de resetear el crc se incluyó la siguiente secuencia:

```
30 00 80 01          registro CMD
```

00 00 00 05 ***start shut down***

A pesar de la introducción de estas líneas de comandos no fue posible hacer volver la placa a estado operativo.

4.1.5 Conclusiones.

Aunque se hicieron amplios avances sobre el estudio de la reconfiguración parcial en una Virtex, no se consiguieron resultados favorables a la hora de cargar el .bit parcial, que hubiera sido la mejor recompensa al trabajo llevado a cabo. Pero queda enfocada la línea de trabajo para cualquier intento que se lleve en el futuro.

La herramienta que hubiera facilitado el trabajo es ***Modular Design*** una herramienta que complementa el ***Foundation*** y que permite elaborar .bits parciales sólo con una llamada en la línea de comandos. Sin embargo, aunque esta herramienta fue solicitada no se obtuvo respuesta por parte de Xilinx para su obtención a modo de prueba, ya que era necesario su compra y no se proporcionaban demos. El manejo y utilización de esta herramienta queda explicada en la [XILI02]. Inicialmente se pensó que los avances de esta herramienta estaban incluidos en Foundation 4, pero una vez instalado, este programa no contaba con los adelantos del diseño modular.

4.2 RECONFIGURACIÓN TOTAL

4.2.1 Objetivos.

Esta parte del proyecto trata de obtener reconfiguración dinámica total de la FPGA por petición del propio circuito configurado en ella.

Para ello se fueron fijando una serie de objetivos parciales:

1. Configurar la FPGA a partir de un archivo de configuración (Bitstream) cargado en la Flash RAM en lugar de realizarse a través del puerto paralelo.

2. Programar la Flash RAM con varios archivos de configuración.

3. Configurar la FPGA con cualquiera de los archivos cargados en la Flash (determinado en tiempo de compilación)

4. Configurar cuando el CPLD recibiese una señal en vez de realizarse nada más encender la placa.

5. Configurar cuando el CPLD recibiese la señal cargando el archivo de configuración indicado por otra señal.

6. Conseguir que las señales fueran activadas por el propio circuito configurado en la FPGA.

4.2.2 Configurar la FPGA con el 2º archivo cargado en la Flash.

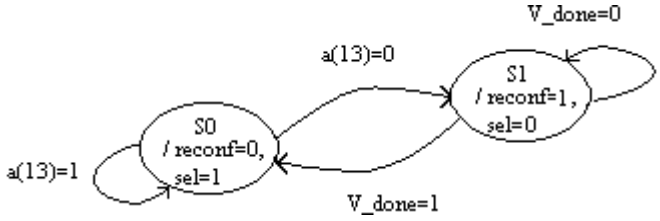
Partiendo del código proporcionado por XESS para la comunicación y configuración de la FPGA a través de la Flash RAM, hemos ido generando nuevas configuraciones del CPLD hasta llegar al objetivo marcado.

En primer lugar modificamos el interfaz de la Flash visto en el punto 2.3.3 para cargar la FPGA con la segunda configuración cargada en la Flash RAM. Para ello hubo que modificar la dirección inicial de lectura de la Flash para que se comenzase desde la dirección 1000000x0 en lugar de la primera palabra.

4.2.3 Configuración de la FPGA por petición.

Una vez probada la posibilidad de configurar la FPGA con Bitstreams (archivos de configuración) guardados en cualquier zona de la Flash RAM, intentamos conseguir que no se reconfigure al encender la placa, sino que espere a una señal generada en principio desde uno de los switches de la placa. Para ello se modifico la lógica del circuito para que no comenzara a reconfigurar hasta que se hubiera recibido la señal.

Para ello se añade una máquina de estados al circuito:



Los dos estados que contiene son:

1. S0: estado en el que se espera a que se active la señal de reconfiguración.
   Esta señal proviene del switch 1, línea compartida con la 13 del bus de direcciones de la Flash. Esperamos que se ponga a 0, valor que toma un switch cuando esta en ON.
   Las salidas de este estado son **reconf** =’0’ para que no reconfigure y **sel**=’1’. La señal **sel** se utiliza para saber cuando nos encontramos en el estado que debe poner en alta impedancia las señales compartidas por el CPLD y la FPGA

2. S1: estado de reconfiguración. Nos mantenemos en él hasta que recibimos desde la FPGA la señal que nos indica que a terminado la configuración.
   Las salidas de este estado son **reconf** =’1’ para que pueda comenzar la configuración, como hemos visto en el proceso anterior y **sel**=’0’ para indicar que las señales compartidas deben pasar a alta impedancia.

Es importante incidir en el hecho de que, en principio, todas las señales de los switches están siendo utilizadas como líneas de dirección de la Flash, ya que estas líneas están compartidas por estos componentes. En principio podríamos pensar que no podemos hacer que el CPLD use un switch para saber cuando tiene que reconfigurar. Pero como se ha indicado arriba, y gracias a la máquina de estados, podemos distinguir dos momentos independientes; la reconfiguración, en la que si necesitamos esas líneas para acceder a la Flash, y la espera de la señal de reconfiguración, en la que no necesitamos acceder a la Flash y podemos recibir datos de los switches.

Se modificó ligeramente la lógica del programa para que no solo se esperara a que expirara el contador para comenzar la configuración, sino que también fuera necesario que la señal de reconfiguración estuviera activa.

En esta prueba se consiguió que la FPGA no se reconfigurara hasta que no se pusiera el switch 1 a ON, pero una vez finalizada la configuración no se podía volver a configurar la placa. El problema podía ser debido a:

1. No se vuelve al estado de espera de la señal de reconfiguración.

2. La FPGA no se queda preparada para una nueva reconfiguración.

Para comprobar la fuente de error se modifico ligeramente el código para mostrar por la barra de leds tanto el estado actual de la máquina de estado como el estado de la placa a través de las señales **V_done** y **V_initb**. Para ello solo fue necesario añadir dos nuevas salidas a la máquina de estados para poder visualizar en cual de ellos no encontrábamos (al haber sólo dos estados), con una única señal hubiera bastado. Con esto se comprueba que la transición de estados es correcta y que el problema está en el estado de la FPGA.

4.2.4 Programación de la FPGA eligiendo la configuración.

Antes de abordar este problema se probó a seleccionar la configuración a cargar desde otro de los switches. Para ello solo hizo falta hacer una pequeña modificación en el código y de la lógica del circuito.

Como estamos cargando archivos de configuración al principio y en la mitad de la Flash RAM, sólo necesitamos una señal de un bit para indicar cual de ellos queremos cargar. Por ello añadimos una señal proveniente de otro switch. Con modificar el bit más significativo de la dirección ya realizamos la selección de la configuración a realizar, por ello se asigna como bit más significativo de la dirección el valor proveniente del switch.

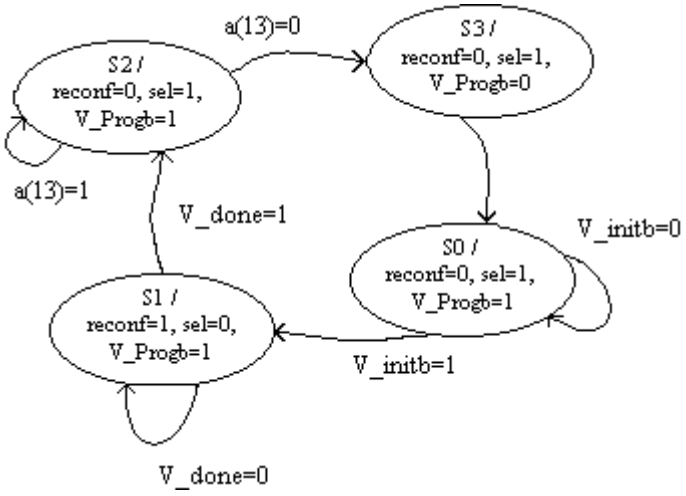
NOTA: Hay que asegurarse que el valor del switch no es modificado durante la operación de reconfiguración.

4.2.5 Reconfiguración cíclica por petición.

Volviendo al problema del estado de la FPGA, comprobamos que para que esté lista para configurar de la misma forma que lo esta al encender la placa deben sucederse una serie de pasos.

A partir de los pasos de esta secuencia se puede entender por qué la FPGA no podía ser reconfigurada. Es necesario activar la señal **/PROG** de la Virtex para que esta se borre y vuelva a estar preparada para una nueva configuración.

Esto nos obliga a realizar modificaciones sobre nuestra máquina de estados. El diagrama de la nueva máquina de estados es el siguiente:



Los estados que contiene son:

1. S2: estado en el que se espera a que se active la señal de reconfiguración.
   Esta señal proviene del switch 1, línea compartida con la 13 del bus de direcciones de la Flash. Esperamos que se ponga a 0, valor que toma un switch cuando esta en ON.
   Las salidas de este estado son **reconf** =’0’ para que no reconfigure y **sel**=’1’. La señal **sel** se utiliza para saber cuando nos encontramos en el estado que debe poner en alta impedancia las señales compartidas por el CPLD y la FPGA. Valor de la señal **V_progb** sigue a 1 para permitir el funcionamiento del último circuito cargado en la FPGA, ya que si no seria borrado inmediatamente tras su programación.

2. S3: Estado de borrado de la FPGA.
   En este estado se pone a 0 la señal **V_progb** para que se comience a borrar la FPGA.
   Automáticamente se pasa al estado S2. La señal **sel** vale 1, ya que seguimos necesitando acceder a las señales del bus de direcciones de la Flash, que son compartidas con la FPGA y **reconf** se mantiene a cero, para evitar que comience una reconfiguración que aun no es posible.

3. S0: estado en el que se espera a que se active la señal de la FPGA que indique que esta lista para la configuración (**V_initb.**
   En este estado la señal **V_progb** pasa de nuevo a valer 1 para que la FPGA termine de ser borrada y llegue a estar lista para la configuración. La señal **sel** vale 1, ya que seguimos necesitando acceder a las señales del bus de direcciones de la Flash, que son compartidas con la FPGA y **reconf** se mantiene a cero, para evitar que comience una reconfiguración que aun no es posible.

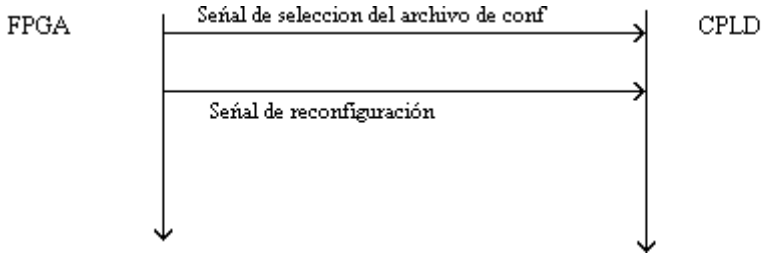
4. S1: estado de reconfiguración. Nos mantenemos en él hasta que recibimos desde la FPGA la señal que nos indica que ha terminado la configuración.
   Las salidas de este estado son **reconf** =’1’ para que pueda comenzar la configuración, y, como hemos visto en el apartado anterior, **sel**=’0’ para indicar que las señales compartidas deben pasar a alta impedancia. **V_progb** se mantiene a 1 para evitar el borrado de la FPGA.

Una vez conseguido esto, tenemos un circuito en el CPLD que recibe dos señales, una de reconfiguración y otra de selección, gracias a las cuales podemos indicar qué y cuándo configurar. Ahora debemos conseguir que estas señales provengan del circuito cargado en la FPGA.

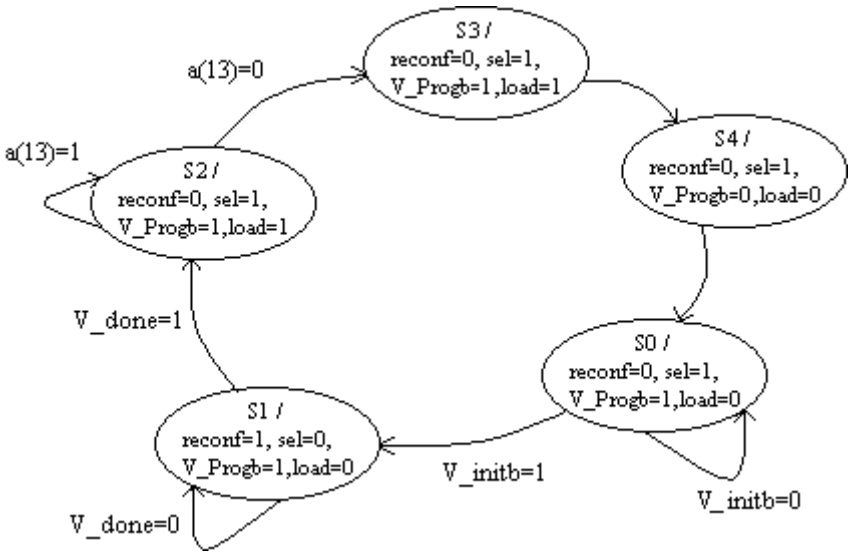
4.2.6 Reconfiguración cíclica por petición desde la FPGA.

Para poder tener el mismo funcionamiento que en el caso anterior pero con la diferencia de que las señales provengan de la FPGA debemos hacer unas leves modificaciones. Antes, la señal que nos indicaba que archivo de la Flash queríamos utilizar para realizar la reconfiguración estaba siempre con un valor valido antes de comenzar la reconfiguración. Pero este caso surge un problema: antes de empezar a reconfigurar tenemos que borrar el contenido de la FPGA para que esté preparada para la configuración, perdiendo así el valor del archivo a cargar en la Flash.

Este inconveniente tiene una muy sencilla solución: almacenar ese valor en un biestable en el momento que se indica que se quiere reconfigurar para tenerlo accesible hasta el momento en el que comienza la reconfiguración. La señal de carga del biestable se mantiene activa mientras que nos encontramos en el estado que espera la señal de reconfiguración. Con hacer que la señal que indica que configuración cargar tiene un valor valido un ciclo de reloj antes de la generación de la señal de configuración se asegura el correcto funcionamiento del interfaz. Si la carga del biestable fuera asíncrona no sería necesario cumplir esta condición.



Para tener que evitar al diseñador del circuito de la FPGA esta restricción, se creo una última máquina de estados para el circuito del CPLD. Esta máquina añade un nuevo estado entre los actuales S2 y S3 que mantiene durante un ciclo más de reloj la señal de carga del biestable activa. De esta forma con asegura que con qué el circuito de la FPGA active al mismo tiempo las dos señales que necesita el interfaz del CPLD, el funcionamiento de dicho interfaz va a ser correcto:



4.2.7 Circuitos cargados en la FPGA para las pruebas.

Para los distintos interfaces cargados en el CPLD se iban creando distintas configuraciones para la FPGA según las necesidades.

a) Sumador-restador

En primer lugar se diseño un circuito sumador-restador en el que se podía elegir la operación a realizar. Tanto los datos de entrada como el tipo operación se introducían por teclado y los datos eran mostrados por los displays de 7 segmentos de la placa XSV.

b) Sumador y restador

Para realizar las pruebas de carga de archivos de configuración y del interfaz que recibían las señales desde los switches se crearon dos circuitos, uno para el sumador y otro para el restador.

Para realizarlos se reutilizó el circuito anterior, fijando el valor de la señal de operación al valor adecuado. Estos dos archivos de configuración se agruparon en un archivo.EXO como se explica en el apartado 4.2.3. y cargados utilizando la herramienta GXSLOAD.

c) Sumador y restador con señales de configuración

Para comprobar el funcionamiento del interfaz que recibía las señales de la propia FPGA, hubo lógicamente que modificar el sumador y el restador, añadiéndoles funcionalidad para generar dichas señales.

Para generar un funcionamiento análogo al que podría tener el MIPS respecto a al señal de reconfiguración, se introdujo en el sumador y el restador un proceso que activaba dicha señal cuando el resultado de la operación era igual a dos. En cada uno de los dos diseños se fijaba la señal de selección de tal manera que, al producirse la configuración, se cargara el otro circuito.

d) Conclusión

Debido al funcionamiento requerido, cualquier circuito de la FPGA que desee utilizar el interfaz de reconfiguración creado para el CPLD solo debe generar la señal (o señales según cuantos archivos se hayan cargado en la Flash) que seleccione el archivo a configurar y a continuación activar la señal de configuración.

NOTA: La señal de reconfiguración deber ser fijada al pin 50 de la placa, y la de selección al 59.

5. BIBLIOGRAFÍA

[DUFI00] Dunlap, Chris; Fischaber, Tom "XAPP079: Configuring Xilinx FPGAs Using an XC9500 CPLD and Parallel PROM" http://www.xilinx.com/apps/epld.htm

[HPCA] J. Hennessy, D. Patterson "Computer Architecture: A quantitative approach (2nd. Ed.)". Morgan Kaufmann Publishers Inc. 1996.

[JH94] Joe Heinrich. "MIPS R4000 Microprocessor User's Manual". MIPS Technologies, Inc. 1994.

[KLEI02] Kleitz, William "XILINX Foundation CPLD software tutorial"

[La97] James R. Larus.Computer Sciences Department. "SPIM S20: A MIPS R2000 Simulator". University of Wisconsin. 1997.
http://www.cs.utexas.edu/users/skeckler/cs352/handouts/xspim/spim.pdf

[MaBP] Ivan Magán Barroso, Javier Basilio Pérez Ramas y Miguel Péon Quirós. "Diseño e implementación de un computador autoreconfigurable". Universidad Complutense de Madrid. 2001.

[MoWe] Model Technology website. http://www.model.com

[MSMa] "ModelSim SE User's Manual". Model Technology. 2001.

[VAND00] Vanden Bout, D. "XSV Parallel Port Interface" http://www.xess.com/ho03000.html#Tutorials

[VAND01] Vanden Bout, D. "XSV Flash Programming and Virtex Configuration" http://www.xess.com/ho03000.html#Tutorials

[XESS01] "XSV Board V1.1 Manual" http://www.xess.com/manuals.html

[XILI99] "XAPP153: Status and Control Semaphore Registers Using Partial Reconfiguration" http://www.xilinx.com/apps/virtexapp.htm#appnotes

[XILI00] "XAPP138: Virtex FPGA Series Configuration and Readback" http://www.xilinx.com/apps/virtexapp.htm#appnotes

[XILI00b] "XAPP151: Virtex Series Configuration Architecture User Guide" http://www.xilinx.com/apps/virtexapp.htm#appnotes

[XILI00c] "DS003: Virtex™ 2.5 V Field Programmable Gate Arrays" http://www.xilinx.com/apps/virtexapp.htm#appnotes

[XILI02] "XAPP290: Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulations" http://www.xilinx.com/apps/virtexapp.htm#appnotes

[XIMo] "XAPP100: HDL Simulation FPGA Design Methodology" http://www.xilinx.com/apps/virtexapp.htm#appnotes