ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Aceleración por Hardware de Algoritmos Basados en Soft-Computing Mediante Dispositivos Programables y Reconfiguración Dinámica

AUTOR: Juan Quero Llor
DIRECTOR: Fco. Javier Garrigós Guerrero
CODIRECTOR: Javier Toledo Moreo

Julio / 2008

Autor	Juan Quero Llor
E-mail del Autor	juanquerollor@terra.es
Director	Francisco Javier Garrigós Guerrero
E-mail del Director	javier.garrigos@upct.es
Codirector	Javier Toledo Moreo
Título del PFC	Aceleración por Hardware de Algoritmos Basados en Soft-Computing Mediante Dispositivos Programables y Reconfiguración Dinámica
Descriptores	Fpga, reconfiguración parcial, auto-reconfiguración parcial

Resumen

Las FPGAs son dispositivos reprogramables que permiten configurar en su interior el sistema hardware deseado, en función de lo que se haya descargado en su memoria de programación.

La Reconfiguración parcial dinámica consiste en cambiar la configuración de una parte de la FPGA sin que deje de funcionar el resto de ella.

Además, esta reconfiguración parcial puede realizarse desde un sistema instalado dentro de la FPGA (microprocesador), en lo que se conoce como auto-reconfiguración parcial, de tal manera que la FPGA se reconfigura a si misma.

La reconfiguración parcial puede usarse en múltiples aplicaciones: aceleración hardware, auto-reparación, algoritmos adaptativos hardware, encriptación… y proporciona grandes ventajas como son: configuración a distancia, ahorro en costes, espacio y consumo…

En el presente proyecto se presentan los distintos tipos de Reconfiguración parcial y la metodología de trabajo necesaria para obtenerlos, con objeto de servir de manual para futuras aplicaciones con FPGAs.

Titulación	Ingeniero de Telecomunicación
Intensificación	Sistemas y Redes de Telecomunicación
Departamento	Electrónica, Tecnología de Computadoras y Proyectos
Fecha de Presentación	Julio / 2008

ÍNDICE

ÍNDICE DE FIGURAS:

CAPÍTULO 1
Introducción

1.1. Planteamiento Inicial del Proyecto

En los últimos años, los fabricantes de dispositivos lógicos programables han sabido aprovechar la progresión exponencial en la capacidad de los circuitos integrados. En particular, las *FPGAs (Field-Programmable Gate Arrays)* han pasado de ser mera lógica de soporte para otros chips a integrar verdaderos sistemas, con capacidades de millones de puertas, que se pueden configurar al antojo del usuario. En la actualidad, una *FPGA* puede implementar sin problema uno o más procesadores, una parte de la memoria del sistema a partir de bloques de *RAM* internos, varios periféricos, y aun así quedar espacio para incluir coprocesadores que aumenten la velocidad de las tareas software.

Las *FPGAs* son circuitos programables en los que es posible cambiar la funcionalidad de sus elementos y el conexionado (rutado) entre ellos para que el dispositivo cumpla con la función hardware que se desee. Ello se consigue activando o desactivando cada uno de sus múltiples puntos de programación, conectándolos a un '1' o '0' lógicos. La *FPGA* incluye por tanto una memoria de configuración, donde se almacena el estado de todos los puntos de programación. Cambiando el contenido de esta memoria, se consigue reprogramar o cambiar la funcionalidad del circuito.

Una de las características más atractivas de las *FPGAs* es su reprogramabilidad (se puede programar cuantas veces se desee). En el modo más habitual de funcionamiento, la configuración es leída de una memoria estática externa al arrancar el sistema. Una vez reprogramada, la *FPGA* entra en funcionamiento y para volver a programarla hay que pararla y programarla de nuevo por completo. Sin embargo, con las técnicas de Reconfiguración parcial de *FPGAs* es posible cambiar la funcionalidad de una parte del chip, mientras el resto del hardware sigue su operación, en lo que se conoce como reconfiguración parcial activa o dinámica.

Para reconfigurar la *FPGA* se necesita un circuito "inteligente", que sea capaz de decidir el momento preciso y cargar la configuración adecuada en la memoria de la FPGA, por lo que normalmente se utiliza un ordenador. Sin embargo, la reconfiguración puede ser realizada por un dispositivo implementado en la propia *FPGA*, sin que sea necesario ningún microprocesador externo adicional ni ordenador, esta técnica es llamada auto-reconfiguración parcial. La utilización de *Soft-Core Processors (SCP)* como dispositivos encargados de realizar la reconfiguración de una parte de la *FPGA,* añade grandes posibilidades a esta técnica. Gracias a la auto-reconfiguración estos *SCPs* pueden reprogramar una parte de la *FPGA* con el coprocesador o hardware elegido, obteniendo una mejora de rendimiento apreciable y circuitos hardware capaces de re-adaptarse, evolucionar o auto-repararse.

Estas técnicas, no obstante, se encuentran en un temprano estadio de desarrollo, por lo que no existen herramientas estándar que permitan tener aplicaciones con ciclos de desarrollo relativamente cortos. Aún así se observa un gran potencial en la Reconfiguración parcial, con multitud de posibles aplicaciones futuras que revolucionarán la tecnología y la industria, con considerables reducciones de espacio, de costes, de tiempo

y de consumo, y las grandes posibilidades que tiene que un chip sea capaz de cambiarse físicamente a si mismo, en una especie de metamorfosis en la que el microprocesador (el "cerebro") y otras partes de la *FPGA* no dejan de funcionar en ningún momento.

Por ello, dentro de este marco, el objetivo principal del presente proyecto es el desarrollo de los procedimientos y la metodología de trabajo que permitan beneficiarse de las técnicas de reconfiguración parcial y auto-reconfiguración parcial en el diseño de aceleradores hardware para aplicaciones específicas.

1.2. Ventajas de la Reconfiguración

La reconfiguración parcial es útil para aplicaciones que requieren cargar diferentes diseños en la misma área del dispositivo, o que posibiliten el cambio de una parte de un diseño sin hacer un reset o reconfigurar completamente el dispositivo. Con esta capacidad, nuevas áreas de aplicación son posibles, en particular, actualizaciones de hardware y reconfiguración en tiempo de ejecución.

Estas técnicas resultan de interés en aplicaciones donde es posible multiplexar en el tiempo la utilización de determinados recursos (de forma similar las técnicas software de cambio de contexto que se utilizan en sistemas multitarea). Dentro de un mismo chip podrían irse intercalando tareas hardware que no sean necesarias ejecutar al mismo tiempo, para conseguir una reducción del tamaño del chip, y por tanto del coste y del consumo del mismo.

Una ventaja esencial es que se podría actualizar el hardware a distancia. Estamos acostumbrados a actualizar software para obtener nuevas aplicaciones, ventajas o reparación de *bugs*. Con la reconfiguración parcial se podría conseguir poder actualizar el hardware sin cambiarlo, incluso a distancia, con la consiguiente ventaja para las aplicaciones espaciales o en lugares comprometidos y sin que otras partes del mismo dejasen de funcionar.

Podría llegar el caso en el que el propio chip detectase fallos en su interior (por ejemplo por radiación en el espacio) y procediese a "instalar" la parte dañada en otro lugar del mismo chip, con el ahorro en dinero, consumo y volumen que supondría el no tener que enviar tanta electrónica redundante al espacio.

En ocasiones el cambio o reparación de un circuito electrónico supone tener que apagar una instalación y reiniciarla desde el principio, mientras que con reconfiguración se podría hacer manteniendo en servicio el hardware.

Otra posible aplicación podría ser la utilización de comunicaciones reconfigurables o sistemas criptográficos, que permitiría cambios de cifrado hardware o incluso de forma de comunicación manteniendo el mismo circuito y a distancia.

Es además una ventaja para el trabajo con las propias *FPGAs*, puesto que reconfigurar parcialmente tarda menos tiempo que configurar la *FPGA* con un *bitstream* completo, y el *bitstream* parcial ocupa menos.

En aplicaciones de carácter adaptativo o evolutivo, se podría hablar de una adaptación hardware de los algoritmos, con la correspondiente mejora en la velocidad. Estas características son de particular importancia en sistemas fundamentalmente paralelos y con carácter evolutivo, como las redes neuronales, los sistemas *neuro-fuzzy* y los algoritmos genéticos.

Cabe decir que el desarrollo de arquitecturas de hardware de cómputo eficientes (teniendo en cuenta restricciones de velocidad de cómputo, tamaño, consumo, etc.) es una de las principales tareas de los ingenieros de Telecomunicación involucrados por ejemplo, en el desarrollo de nuevos y más potentes dispositivos móviles de comunicaciones (teléfonos, PDAs, sensores remotos, etc.).

1.3. Objetivos Detallados del Proyecto

- Estudio de las diferentes arquitecturas y capacidades de (auto)reconfiguración parcial de los dispositivos programables comercializados por diversos fabricantes.

- Estudio y desarrollo de los diferentes flujos de trabajo propuestos para el uso de la (auto)reconfiguración parcial propuestos por el fabricante, observando las ventajas e inconvenientes de cada uno de ellos.

- Desarrollo de las distintas técnicas de reconfiguración: reconfiguración parcial estática, reconfiguración parcial estática conservando el estado, reconfiguración activa, auto-reconfiguración activa y auto-reconfiguración activa de un periférico de un microprocesador embedido, con los distintos flujos de trabajo.

- Estudio de cores de procesadores de propósito general (*MicroBlaze*, *PowerPC*, etc.) optimizados para su uso dentro de dispositivos programables. Es fundamental la adaptación de sus características de cara a realizar la reconfiguración de la *FPGA*, así como para realizar otras funciones de supervisión de alto nivel del sistema: control, *E/S*, interfaz con el usuario, etc.

- Estudio del software de trabajo en *FPGAs* (*ISE*, *EDK*, *PlanAhead*, *HyperTerminal*…) para su utilización en los distintos procesos de reconfiguración.

- Análisis de las técnicas de reconfiguración en las distintas tecnologías que la permiten: ***Virtex-II***, ***Virtex-II pro*** y ***Virtex-4***.

- Integración o desarrollo en su caso de técnicas eficientes para la (auto)reconfiguración parcial de *FPGAs* sobre la arquitectura seleccionada.

- Desarrollo de una arquitectura de cómputo completa en un chip (*SoC, System on a Chip*) formada por: microprocesador, memoria, controladores de buses e interrupciones, periféricos de E/S estándar (*RS232*) y uno o varios coprocesadores genéricos.

- Verificación de la funcionalidad y la versatilidad de la metodología de desarrollo propuesta con ejemplos y aplicaciones características.

- Recopilación de la información para la obtención de manuales o guías de trabajo, que permitan realizar (auto)reconfiguración parcial de una manera fácil, rápida y efectiva.

- Estudio de las ventajas y posibles aplicaciones futuras en las que se pueda utilizar la reconfiguración parcial en cualquiera de sus facetas.

1.4. Fases del Proyecto

En primer lugar se realizará un estudio de distintos dispositivos comerciales tipo *FPGA*, con el objetivo de decidir cuál es el más idóneo para su utilización en técnicas de reconfiguración parcial. La propuesta inicial es utilizar las arquitecturas ***Virtex-II*** o ***Virtex-II Pro*** del fabricante *Xilinx Inc.,* frente a otras como las ***Virtex-4***, ***Virtex-5*** o ***Spartan-3***.

Tras ello se procederá al estudio de la metodología de diseño y de los distintos flujos de trabajo, así como de las herramientas proporcionadas por los fabricantes (*ISE, EDK, PlanAhead*) que permitan las técnicas de reconfiguración.

A continuación se utilizarán tanto las herramientas en las que se ha trabajado, como la metodología y los flujos de trabajo recopilados, para realizar ejemplos que comprueben el correcto funcionamiento de la reconfiguración parcial en sus distintas versiones y las prestaciones obtenidas frente a otras soluciones tradicionales. Una vez conseguida la reconfiguración parcial se procederá a estudiar las técnicas de auto-reconfiguración parcial.

Se procederá entonces al estudio de las características de los *cores* y *drivers* de procesadores existentes adecuados para llevar a cabo la auto-configuración. Se optará, en principio, por el uso de la herramienta *Xilinx EDK* y el procesador *MicroBlaze*, por ser un procesador 100% sintetizable y libre de *royalties* sobre dispositivos *FPGA* del fabricante, lo que permite que pueda ser utilizado en cualquier *FPGA* de esta marca, y por su integración en dicha herramienta. La arquitectura *RISC* tipo *Harvard* de 32 bits de este microprocesador, la posibilidad de ampliar la memoria con dispositivos externos a la *FPGA* e introducir diferentes cachés ofrece a priori unas garantías suficientes en cuanto a que las prestaciones que se pueden obtener de este procesador sean suficientes para nuestros requerimientos. Posee además una conectividad específica para el core de auto-reconfiguración de la *FPGA (hwicap)* que facilita el diseño de la estrategia de reconfiguración.

Como en el caso de la reconfiguración, se buscará obtener una metodología de trabajo y unas herramientas que permitan la auto-reconfiguración de elementos no conectados al microprocesador, así como algún ejemplo que corrobore su correcto funcionamiento.

A continuación se procederá a intentar conseguir la auto-reconfiguración parcial de periféricos del propio microprocesador que reconfigura, obteniendo también una metodología de trabajo y ejemplos que permitan comprobar su adecuado funcionamiento.

Finalmente se procederá a recopilar toda la documentación, metodologías de trabajo, herramientas y ejemplos en un documento que explique y facilite la futura creación de proyectos reconfigurables de todo tipo.

CAPÍTULO 2
Estado de la Técnica

2.1. Dispositivos Reprogramables

La *FPGA* (del inglés *Field Programmable Gate Array*) es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad se puede programar y reprogramar un número indefinido de veces. La lógica programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional, hasta complejos sistemas en un chip (*System-on-a-chip*).

Las *FPGAs* frente a los *ASICs* tienen las ventajas de ser reprogramables (lo que añade una enorme flexibilidad al flujo de diseño), sus costes de desarrollo y adquisición son mucho menores para pequeñas cantidades de dispositivos y el tiempo de desarrollo es también menor. Sin embargo, son más lentas, tienen un mayor consumo de potencia y no pueden abarcar sistemas tan complejos como ellos, aunque las últimas tecnologías ***Virtex-4*** y ***Virtex-5*** cada vez tienen un menor tamaño, un mayor número de puertas, un menor consumo de potencia y una mayor velocidad.

La metodología de diseño para asegurar diseños *FPGAs* exitosos y fiables es más relajada y menos restrictiva que en el caso de los *ASICs,* debido a la facilidad y bajo coste de implementar cambios de diseño en las *FPGAs*. La evolución de la tecnología de las *FPGAs* en los últimos años ha llevado a unos dispositivos más pequeños, más rápidos y más potentes.

La arquitectura interna de las *FPGAs* consiste en una matriz de *CLBs* (*Configurable Logic Blocks*) rodeada por bloques de Entrada/Salida programables, todo ello interconectado por una amplia jerarquía de recursos de rutado. Además, contiene dos grandes bloques de memoria *SelectRAM* que complementan la memoria distribuida que se puede obtener en los *CLBs*.

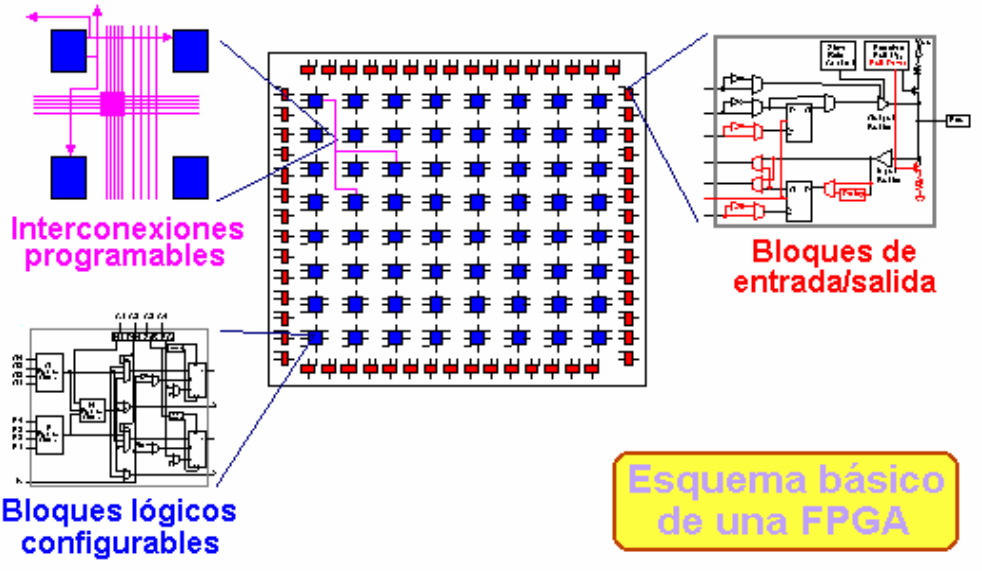


Figura 1. Esquema básico de una *FPGA*

La configuración de las *FPGAs Virtex* está basada en memorias *SRAM* donde se cargan los datos de configuración. De esta forma, la memoria de configuración se puede ver como una matriz rectangular de bits que se agrupan en líneas verticales llamadas *frames*. Los *frames* se agrupan a su vez formando unidades mayores llamadas columnas, de arriba a abajo de la *FPGA*. Mientras en las ***Virtex*** y ***Virtex-II*****,** la columna es la porción mínima de memoria que puede ser configurada, en las ***Virtex-4*** esta porción se reduce a nivel de *frame*.

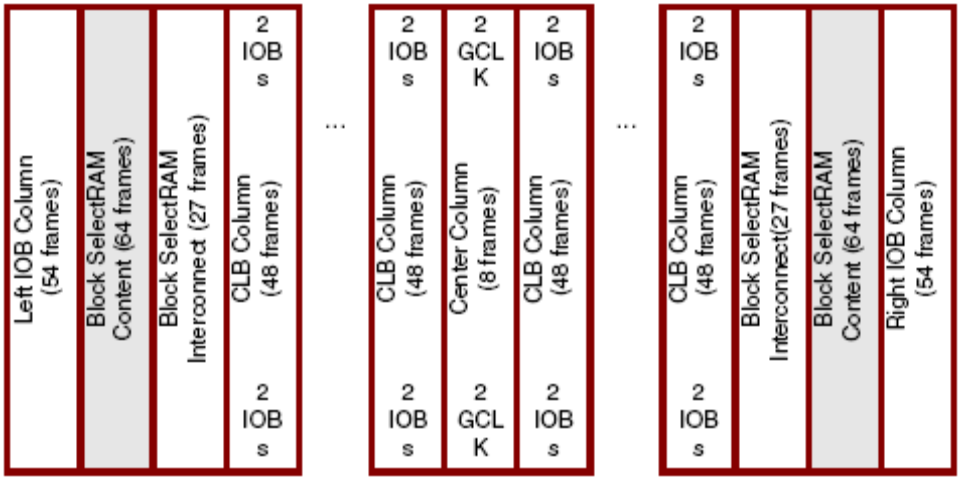


Figura 2. Disposición de las columnas de la memoria de configuración

Las *FPGAs* se configuran descargando a la placa los datos de configuración (llamados *bitstream*). Como la memoria de configuración de las *FPGAs* de *Xilinx* es volátil, debe ser configurada cada vez que se conecta la alimentación. El *bitstream* es descargado en la placa a través de unos pines especiales de configuración, que sirven de interfaz para varios modos de configuración (*Master-serial*, *Slave-serial*, *SelecMap*, *Jtag/Boundary-scan*…). En principio para los diseños reconfigurables se utilizará la opción *Master-serial* (normalmente *M2=M1=M0=0* en los *jumpers* de programación de la placa).

El *bitstream* contiene una parte de direccionamiento, indicando de esta manera qué *frame* debe ser reconfigurado en la memoria de configuración cuando el dispositivo vaya a ser programado o reprogramado.

2.1.1. Placas Utilizadas

2.1.1.1. Virtex-II 1000

En los inicios del proyecto se empezó a trabajar con la placa de *Avnet Xilinx Virtex-II Evaluation Kit*, que contiene la *FPGA* ***Virtex-II*** *xc2v1000-4fg256*.

Dispone de una *EEPROM* para configuración, un puerto *RS-232, 8 switchs, 8 LEDS*, 2 pulsadores y dos *displays* de 7 segmentos. Su reloj interno es de 40MHz

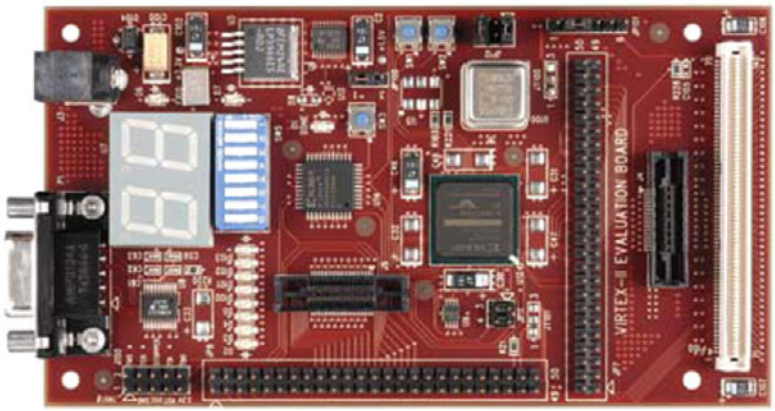


Figura 3. Virtex II Evaluation Kit

Esta placa se comenzó a utilizar para el Flujo Antiguo de reconfiguración parcial (Capítulo 3), así como para el Flujo *EA_PR* para reconfiguración y auto-reconfiguración parcial, pero tuvo que ser sustituida por la ***Virtex-II pro*** cuando se detectó la necesidad de que la placa dispusiera de memoria *RAM*.

2.1.1.2. Virtex-II Pro 7

A continuación se utilizó la placa de *Avnet Xilinx Virtex-II pro Evaluation Kit*, que contiene la *FPGA* ***Virtex-II Pro*** *xc2vp7-ff896*.

La placa está dotada de una memoria *DDR SDRAM* de 64 MB, dos dispositivos *PROM* para configuración, un puerto serie *RS-232, 8 switchs* y 8 *LEDS*. Su reloj interno es de 100MHz.

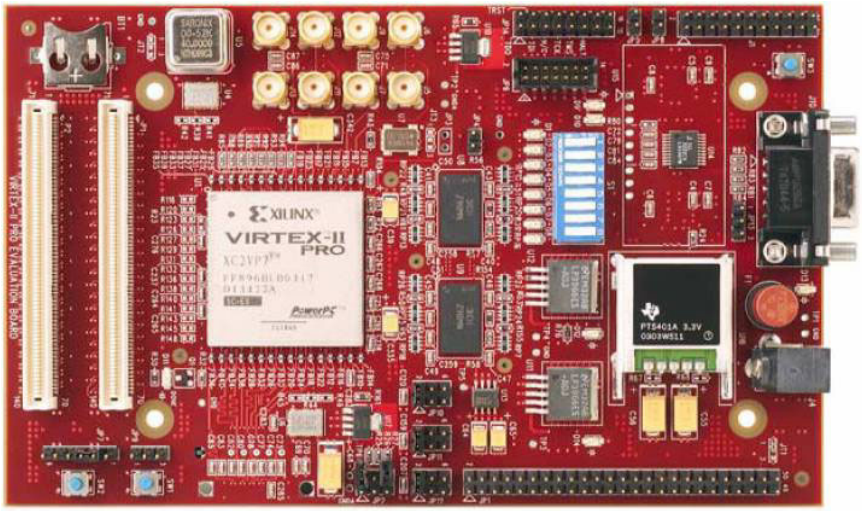


Figura 4. Virtex-II Pro Evaluation Kit

Esta placa se comenzó a utilizar con el nuevo flujo *EA_PR*. Tras conseguir la reconfiguración y la auto-reconfiguración parcial y que *Xilinx* lanzase una nueva versión del parche *EA_PR* que soportaba ***Virtex-4***, se decidió cambiar de nuevo de placa, para trabajar con la última tecnología disponible.

2.1.1.3. Virtex-4 fx12

Finalmente se trabajó con la placa *Digilent FX12 board*, que contiene la *FPGA* ***Virtex-4*** *xc4vfx12-10sf363*.

La placa dispone de una memoria *DDR SDRAM* de 64 MB, dos memorias *Flash ROM*, una pantalla *LCD, 8 LEDs*, 4 pulsadores y un puerto serie *RS-232*. Su reloj interno es configurable, y puede llegar hasta los 350MHz.

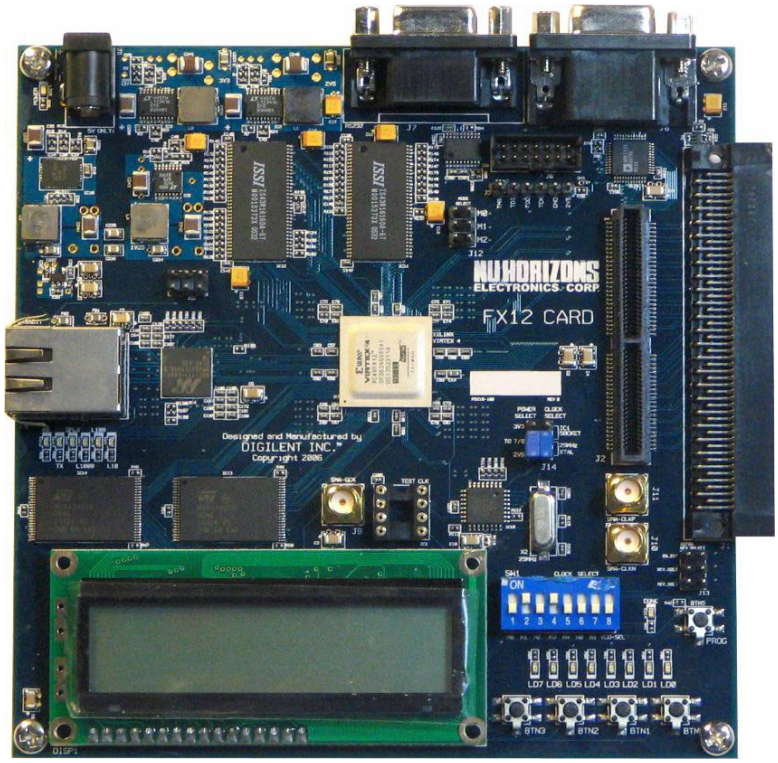


Figura 5. FX12 board

Esta placa, como las demás ***Virtex-4***, permite comunicarse con la zona reconfigurable con *buses macro* por arriba y por debajo de esta, no sólo por los lados.

2.2. Reconfiguración Parcial de FPGAs

2.2.1. Definiciones

Dentro de la reconfiguración parcial cabe definir ciertos términos utilizados habitualmente:

- **Reconfiguración parcial:**

Configuración de un cierto número de *frames* de la *FPGA* sin activar *PROG* (que borra la memoria de configuración), ni apagar y encender la *FPGA*.

-Área Reconfigurable: zona de la *FPGA* que va a ser reconfigurada.

-Área Estática o No Reconfigurable: zona de la *FPGA* que no se va a modificar.

- **Reconfiguración parcial estática:**

Reconfiguración parcial de la *FPGA* parando el funcionamiento de la parte estática o no reconfigurable. Esto impide obviamente la auto-reconfiguración. Existen dos tipos:

-Reconfiguración parcial estática sin conservar el estado: se reconfigura sin mantener el estado de la *FPGA* (*luts, flip-flops*, máquinas de estado… se resetean)

-Reconfiguración parcial estática conservando el estado: se mantiene el estado del resto de la *FPGA* al realizar la reconfiguración.

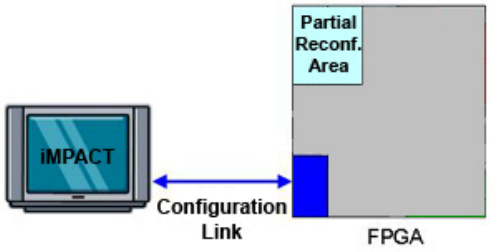


Figura 6. Reconfiguración Parcial Estática

- **Reconfiguración parcial dinámica (o activa):**

Reconfiguración parcial de un determinado número de *frames* de la *FPGA* sin parar el funcionamiento del resto de ella, con el consiguiente peligro de cortocircuito si no se toman las medidas necesarias.

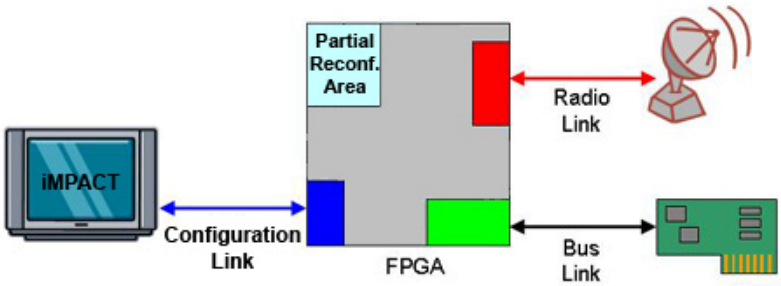


Figura 7. Reconfiguración parcial dinámica o activa

- **Auto-Reconfiguración parcial:**

Un microprocesador embedido se comunica con la memoria de programación de la *FPGA* y la reconfigura. El microprocesador debe formar parte del área estática (aunque sus periféricos no tienen por qué). Existen dos tipos:

-Auto-Reconfiguración parcial de una zona independiente: el microprocesador reconfigura un área de la *FPGA* que no está conectada a él.

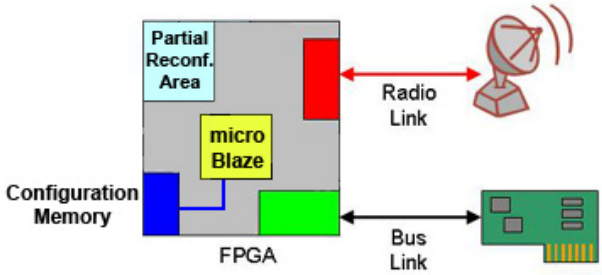


Figura 8. Auto-Reconfiguración parcial de una zona independiente

-Auto-Reconfiguración parcial de un periférico: se reconfigura un periférico que está conectado al microprocesador que está reconfigurando.

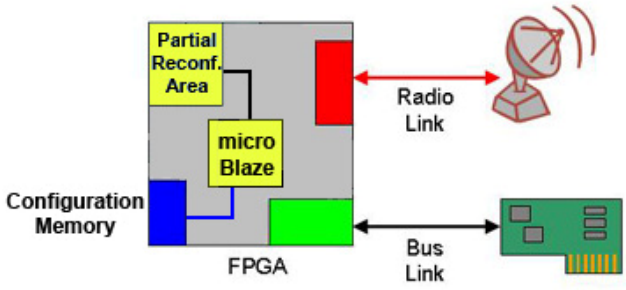


Figura 9. Auto-Reconfiguración parcial de periférico

- **Flujo de Reconfiguración:**

Metodología de trabajo a seguir para conseguir la reconfiguración parcial en cualquiera de sus versiones.

- **Bitstream parcial:**

Bitstream de menor tamaño que el completo, producto de alguno de los flujos de reconfiguración que, al ser descargado en la placa, permite reconfigurar una parte de la *FPGA*.

- **Buses macro y buffers triestado:**

Elementos situados entre el área reconfigurable y el área estática para prevenir cortocircuitos durante la reconfiguración.

2.2.2. Flujo Antiguo:

Dentro de lo que se ha llamado el flujo antiguo de *Xilinx* se encuentran dos flujos bien distintos: el flujo en diferencias y el flujo en módulos, que se explican a continuación.

Ambos flujos han sido abandonados por *Xilinx* debido a los errores y las restricciones que tenían, lanzando posteriormente el flujo actual *EA_PR*.

A lo largo del Capítulo 3 se explican con detalle los pasos a seguir para conseguir la auto-reconfiguración parcial mediante el flujo diferencial. El flujo en módulos se abandonó ya que el flujo *EA_PR* es una evolución de éste, cuyos manuales para conseguir auto-reconfiguración parcial se encuentran en los capítulos 4, 5 y 6.

2.2.2.1. Flujo en Diferencias:

El flujo en diferencias se basa en obtener el *bitstream* parcial a partir de los *frames* de diferencia entre dos diseños.

Esto se consigue gracias a la opción de *bitgen –r*, que obtiene el *bitstream* parcial comparándolo con otro obtenido anteriormente.

Para diseños en los que la reconfiguración sea activa, es decir, que se realice sin parar el funcionamiento del resto de la placa, es conveniente añadir *buffers* triestado que separen la parte reconfigurable de la parte estática, para prevenir cortocircuitos.

Gracias a este flujo también se reduce notablemente el tamaño de los *bitstreams* cuando dos diseños similares deben descargarse en la placa.

Los pasos del flujo son los siguientes:

1. Se sintetiza, implementa y se obtiene el *bitstream* completo de un diseño.
2. Se parte de ese diseño y se modifica para obtener el segundo diseño.
3. Se obtiene el *bitstream* parcial mediante la opción de *bitgen –r.*
4. Se descarga el *bitstream* completo del primer diseño
5. Se reconfigura con el *bitstream* parcial en diferencias obtenido.

Este flujo sólo se debe utilizar para diseños pequeños, en los que haya pocos cambios entre uno y otro, dejando los diseños de mayor complejidad para el Flujo en módulos.

2.2.2.2. Flujo en Módulos:

Este flujo de diseño está basado en la Metodología de Flujo Modular de *Xilinx*. Consiste en separar en distintas regiones la parte estática y la parte reconfigurable del diseño, de tal manera que se obtienen unos *bitstreams* parciales de cada uno de los módulos reconfigurables.

Existirá por tanto una región reconfigurable en la que se descargará un *bitstream* parcial de un módulo reconfigurable u otro, consiguiéndose con ello la reconfiguración parcial. En el caso de que este área no se comunique con la parte estática no es necesario añadir nada más, pero en la mayoría de los casos si que existe dicha comunicación y se deben colocar *buses macro* para las señales que las interconectan. En la imagen se puede apreciar los *buses macro* situados entre la parte reconfigurable (*PR Logic*) y la parte estática (*Fixed Logic*):

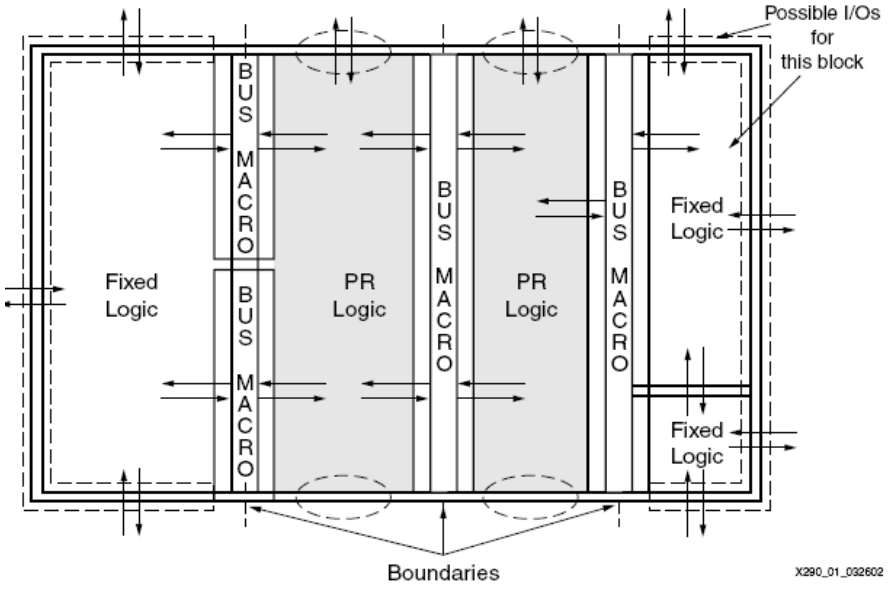


Figura 10. Estructura del la FPGA en el flujo modular

Se deben cumplir una serie de estrictas normas para el funcionamiento del flujo modular:

1. La altura de los módulos reconfigurables debe ser la de la *FPGA*..
2. Los rangos de anchura de los módulos reconfigurables son de un mínimo de 4 *slices* a un máximo del ancho completo de la *FPGA*, en incrementos de 4 *slices*.
3. Las fronteras verticales de las áreas reconfigurables deben situarse siempre en múltiplos de 4: x=0, 4, 8…
4. Todos los recursos que se encuentren dentro del área reconfigurable se consideran como parte del módulo reconfigurable, incluyendo *slices, TBUFs, block RAMs*, multiplicadores, *IOBS,* así como todo el rutado, por lo tanto el rutado de la parte estática no puede atravesar la parte reconfigurable sin incluir *buses macro*.
5. La lógica del reloj (*BUFGMUX, CLIKOB*) siempre debe estar separada del módulo reconfigurable.
6. Los *IOBs* situados en las fronteras superior e inferior de un módulo reconfigurable pertenecen a la parte reconfigurable.
7. Si una región reconfigurable se encuentra a la izquierda o a la derecha de la *FPGA*, los *IOBs* que se encuentren en esa zona pertenecerán a la parte reconfigurable.
8. Para minimizar los problemas relacionados con la complejidad del diseño, el número de módulos reconfigurables debe ser minimizado (si es posible uno solo).
9. La frontera del área reconfigurable no puede cambiarse. La posición y región ocupadas por un módulo reconfigurable es siempre fija.
10. Si el módulo reconfigurable se comunica con la parte estática o con otro módulo reconfigurable debe hacerlo a través de *buses macro*.
11. El diseño debe realizarse de tal manera que la parte estática no interfiera en el estado del módulo que se está reconfigurando.
12. El estado de los elementos de almacenamiento situados en el interior del área reconfigurable es preservado durante y después de la reconfiguración.

El flujo de reconfiguración modular sigue los siguientes pasos:

1. Se escribe y sintetiza el código *vhdl* según las normas descritas con anterioridad.
2. Se realiza *floorplanning* (se sitúan los módulos en la *FPGA* con *PACE* o *PlanAhead*), se sitúa la lógica y se crean las restricciones de tiempo para el *top* y para cada módulo.
3. Se implementa el *top*, la parte estática y cada una de las regiones reconfigurables con cada uno de los módulos reconfigurables por separado.
4. Se unen los diseños y se verifican.
5. Se crea el *bitstream* del diseño completo (con el módulo reconfigurable inicial).
6. Se crean los *bitstreams* parciales de cada uno de los módulos reconfigurables.
7. Se descarga el *bitstream* completo
8. Se reconfigura o auto-reconfigura con los *bitstreams* parciales.

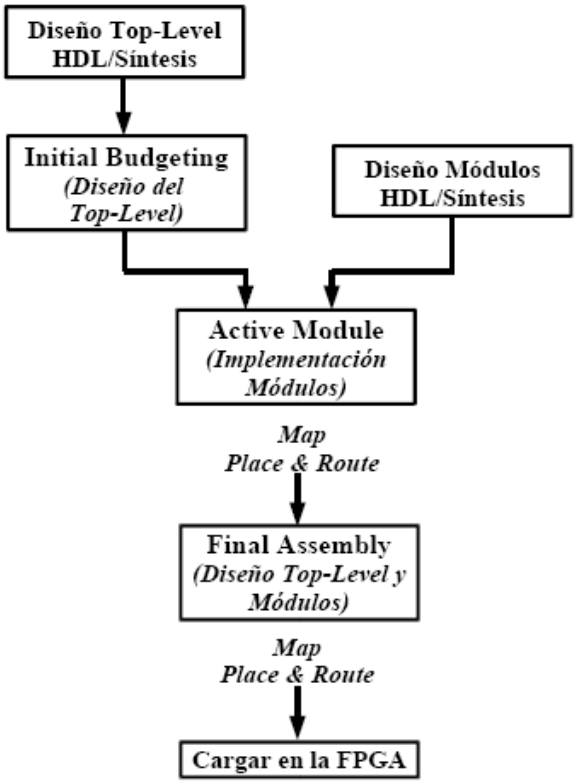


Figura 11. Flujo de diseño Modular

2.2.3. Flujo Early Access Partial Reconfiguration (EA_PR)

Debido a la complejidad de diseño de los dos flujos anteriores, así como a la gran cantidad de fallos que se producían en la síntesis, implementación y obtención de *bitstreams* con los ejecutables habituales de *ISE, Xilinx* sacó un nuevo flujo basado en el flujo en Módulos, pero que permite una mayor flexibilidad en sus restricciones.

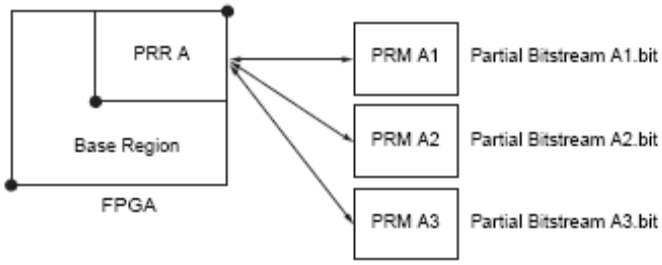


Figura 12. Distribución de módulos en el flujo EA_PR

Este flujo no está disponible comercialmente en los productos actuales de *ISE*, y no está previsto que lo esté hasta por lo menos la versión 11 de *ISE*, ya que se trata de un parche para investigadores en la materia que sigue produciendo gran cantidad de fallos y tiene múltiples actualizaciones para las distintas versiones de *ISE*, y que además invalida el *ISE* para otras tareas distintas a la reconfiguración.

El parche se puede obtener, junto con la documentación necesaria, en la página web de Xilinx http*://www.xilinx.com/support/prealounge/protected/index.htm*, previo registro. La instalación de estas herramientas se explica en el Anexo 1.

Los capítulos 4, 5 y 6 son manuales que adaptan el flujo *EA_PR* en tres casos distintos: la auto-reconfiguración parcial en ***Virtex-4***, la auto-reconfiguración parcial en ***Virtex-II*** y ***Virtex-II Pro*** y la auto-reconfiguración parcial de periféricos en ***Virtex-4***.

Las principales diferencias entre el presente flujo y el flujo en módulos son:

1. Las regiones reconfigurables no tienen por qué tener todo el alto de la *FPGA*.
2. Las señales de la parte estática (que no estén conectadas a la parte reconfigurable) pueden cruzar las fronteras de una zona reconfigurable sin utilizar *buses macro*.
3. Se soporta ***Virtex-4***

El flujo de diseño es básicamente el mismo que el del diseño modular.

1. Generación del código y síntesis
2. Establecimiento de restricciones
3. Implementación del diseño no reconfigurable.
4. Análisis de tiempo y localización
5. Implementación del *top* y la parte estática
6. Implementación de los módulos reconfigurables
7. Unión y obtención de *bitstreams*

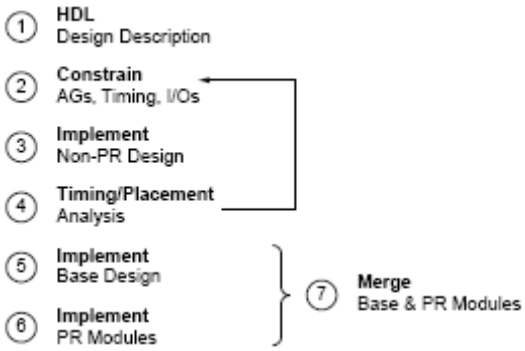


Figura 13. Flujo de diseño EA_PR

Para realizar estos pasos es conveniente crear una estructura de directorios que faciliten la tarea:

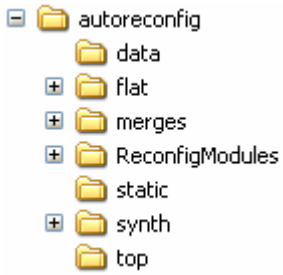


Figura 14. Estructura de directorios del Flujo EA_PR

- *data*: contiene los ficheros ***.vhd***, ***.ucf*** y ***.nmc***, de aquí se instancian y se copian para los procesos de síntesis e implementación (pasos 1, 2, 3, 4, 5 y 6).
- *flat*: contiene el proyecto no reconfigurable (paso 3 y 4)
- *merges*: en el se realiza la unión de todos los módulos y se obtienen los distintos bitstreams.

- *ReconfigModules*: contiene una carpeta con cada uno de los módulos reconfigurables implementados (paso 6)
- *static*: contiene la implementación de la parte estática (paso 5)
- *synth*: contiene una carpeta para la síntesis del *top*, otra para la síntesis de la parte estática y otra por cada uno de los módulos reconfigurables (paso 1)
- *top*: contiene la implementación del *top* (paso 5).

Los pasos del 5 al 7 pueden realizarse automáticamente a través de un *script,* que se verá en capítulos posteriores para cada caso.

2.2.3.1. Generación del Código y Síntesis:

Se genera el código *vhdl* o *verilog*, y se sintetiza individualmente cada una de las partes. El diseño debe ser jerárquico, existiendo un *top level* y una serie de submódulos entre los que estarán los que conforman la parte estática y la parte reconfigurable.

Se deben de cumplir los siguientes requisitos:

1. La herramienta de síntesis debe estar configurada de tal forma que no se realice optimización entre módulos distintos, pudiéndose utilizar la instrucción *KEEP_HIERARCHY* en *ISE*, o bien realizar distintas *netlist* para cada uno de los módulos o utilizar *PlanAhead*.
2. El *top level* debe contener solamente instancias del resto de módulos que lo compondrán (no debe haber lógica sintetizable), conteniendo solamente: *I/O*, primitivas de reloj (*DCM y BUFG*), instancias de la parte estática, instancias de los módulos reconfigurables, declaraciones de señales e instancias de *buses macro*.
3. Todas las señales no globales que comuniquen la parte estática con la parte reconfigurable deben conectarse a través de *buses macro*. Por tanto el reloj podrá ir conectado directamente.
4. Los módulos de nivel inferior situados en el *top level* no pueden contener primitivas relacionadas con el reloj o el *reset* (por ejemplo *BUFG, DCM*…). Esto deberá ser tenido en cuenta al utilizar *EDK*, puesto que *MicroBlaze* incluye *dcm* y *bufg* y no es *top level*.

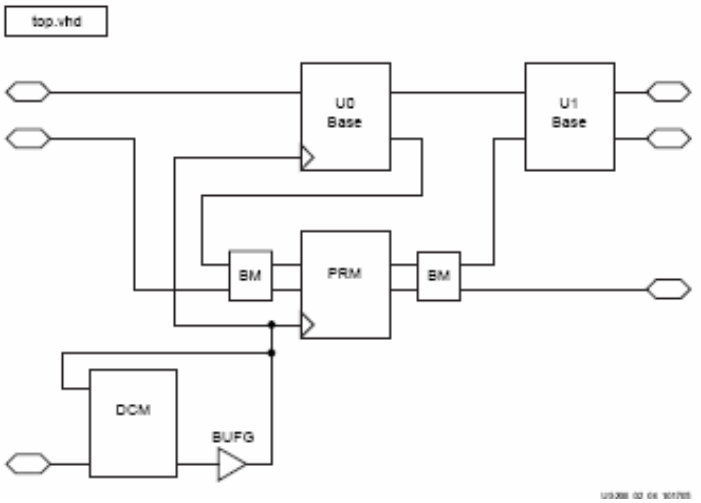


Figura 15. Ejemplo de Diseño reconfigurable con el flujo *EA_PR*

5. Los módulos reconfigurables deben tener los mismos puertos y el mismo nombre de *entity*.

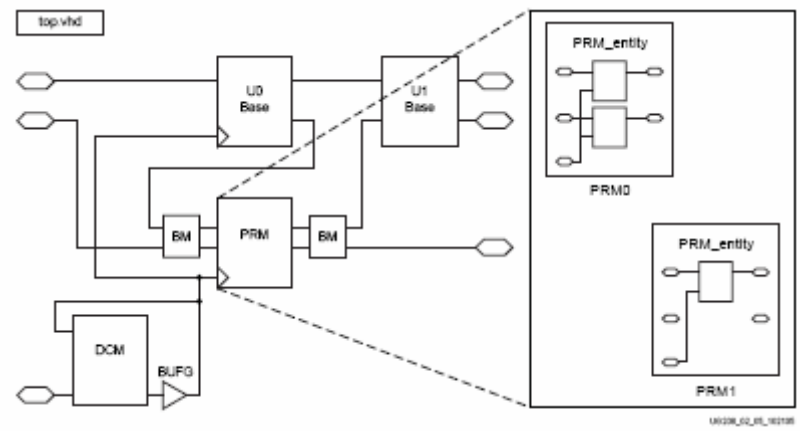


Figura 16. Utilización de los mismos puertos

2.2.3.2. Establecimiento de Restricciones

Se deben de añadir una serie de restricciones de área en el fichero ***.ucf.***

En primer lugar se deben separar claramente la parte estática y la parte reconfigurable. Por ejemplo:

```
INST "sysace_compactflash" AREA_GROUP = "static" ;
INST "rs232_uart_1" AREA_GROUP = "static" ;
INST "reset_block" AREA_GROUP = "static" ;

INST "prm_1" AREA_GROUP = "reconfig";
```

Además se debe situar el área reconfigurable en la *FPGA*, mientras que la parte estática ocupara el resto. Se deben incluir en la parte reconfigurable todos los elementos de *bram* contenidos en dicho área, por ejemplo:

```
AREA_GROUP "reconfig" RANGE = SLICE_(minX)(minY):SLICE_(maxX)(maxY);
AREA_GROUP "reconfig" RANGE = RAMB16_(minX)(minY):RAMB16(maxX)(maxY);
```

Donde *(minX,minY)* son las coordenadas del *slice* inferior izquierdo, y *(maxX,maxY)* son las coordenadas del *slice* superior derecho. El rango de *slices* nunca puede caer entre los dos *slices* de un *CLB*, por lo tanto:

- *(minX,minY)* son siempre pares.
- *(maxX,maxY)* son siempre impares.

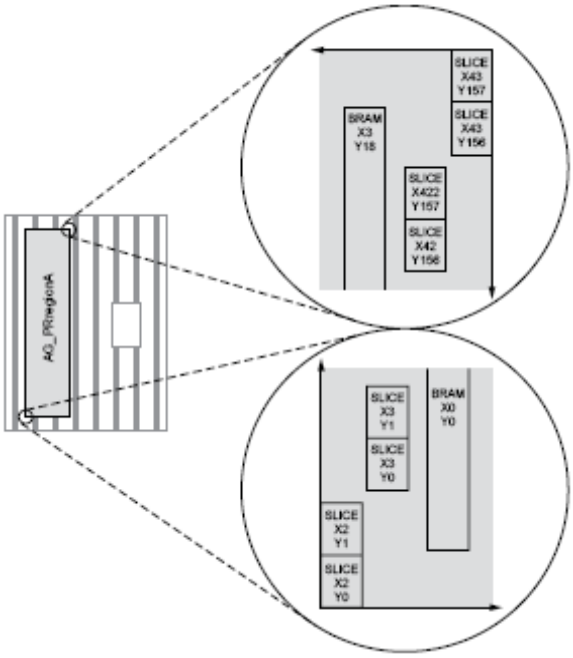


Figura 17. Localización de los módulos reconfigurables

Las herramientas de *Floorplanning* de *Xilinx*, tanto de *ISE (PACE)* como *PlanAhead*, facilitan bastante esta labor y se explica su uso en los siguientes capítulos.

La disposición de las áreas reconfigurables se debe basar en que los *bitstreams* parciales resultantes ocupen lo menos posible. Por ello es recomendable que las áreas reconfigurables de las ***Virtex-II*** sean a lo alto (dado que así ocuparán un menor número de *frames*), mientras que para las ***Virtex-4*** esto es indiferente puesto la organización de los frames no está dispuesta verticalmente.

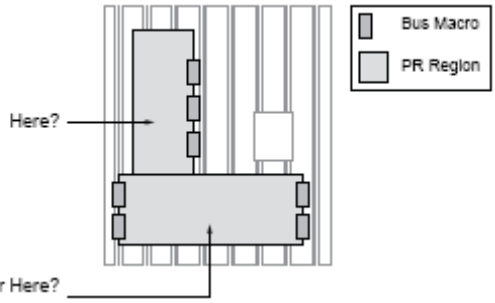


Figura 18. Disposición de los módulos reconfigurables

Para las ***Virtex-4*** es recomendable que las regiones reconfigurables no atraviesen la columna central. Si esto ocurriese, todos los elementos deberían ser instanciados en la parte estática y situados en el ***.ucf.***

El atributo *MODE=RECONFIG* debe ser definido en todas las áreas reconfigurables.

```
AREA_GROUP "reconfig" MODE=RECONFIG;
```

Se tienen que localizar en el fichero ***.ucf*** todos los pins y las primitivas de reloj (*DCM* y *BUFG*), así como los *buses macro*.

```
NET "sys_rst_pin" LOC = "AH5" ;
NET "sys_clk_pin" LOC = "AJ15" ;
NET "fpga_0_SysACE_CompactFlash_SysACE_WEN_pin" LOC = "AC16";

INST "DCM_0" LOC = "DCM_X0Y1";
INST "BUFG_0" LOC = "BUFGMUX2P";
```

La localización de los *buses macro* puede verse en el Apartado 2.3.1

2.2.3.3. Implementación del Diseño no Reconfigurable

Aunque este paso no es necesario, es recomendable realizar una implementación del diseño con *ISE* sin seguir el flujo de reconfiguración parcial.

Este paso puede ser crucial para la depuración de diseño y de sus fallos. También puede unirse al paso anterior y utilizarse para obtener de una forma más automatizada las restricciones, utilizando la herramienta *PACE*.

Para la implementación del diseño no reconfigurable es necesario reemplazar la directiva *MODE=RECONFIG* por *GROUP=CLOSED*:

```
AREA_GROUP "reconfig" GROUP = CLOSED
```

Esto impide que elementos que no estén situados en la parte reconfigurable se introduzcan dentro de ésta al implementar siguiendo el flujo habitual de *ISE*.

2.2.3.4. Análisis de Tiempo y Localización

Se trata de otro paso no necesario, pero que permite mejorar la eficiencia del diseño reconfigurable. Observando los resultados del análisis temporal y de localización se puede modificar la localización de los *buses macro* o de la parte reconfigurable, si con ello se mejoran estos tiempos.

2.2.3.5. Implementación del Top y la Parte Estática

Para la implementación del *top* y de la parte estática es importante recordar volver a incluir la directiva *MODE=RECONFIG* en el fichero ***.ucf***.

Los archivos ***.nmc*** que definen los *buses macro* deben estar situados en cada uno de los directorios de implementación.

2.2.3.6. Implementación de los Módulos Reconfigurables

Una vez que se ha realizado la implementación del *top* y de la parte estática se procede a la implementación de cada uno de los módulos reconfigurables por separado.

Se debe copiar el fichero *static.used* de la carpeta de la implementación de la parte estática y renombrarlo a *arcs.exclude*.

También deben de incluirse las definiciones de los *buses macro*.

2.2.3.7. Unión y Obtención de los Bitstreams

El último paso del flujo *EA_PR* consiste en la unión del *top*, de la parte estática y de los módulos reconfigurables.

Durante este paso se crea un *bitstream* completo y uno parcial con cada uno de los módulos reconfigurables, así como un *bitstream* con el área reconfigurable en blanco y *bitstreams* en diferencias.

Para ejecutar esta parte existen unos *scripts* especiales del *EA_PR* llamados *PR_verifydesign* y *PR_assemble*.

2.3. Elementos Utilizados en la Reconfiguración

2.3.1. Buses Macro

Los b*uses macro* proporcionan un modo de fijar el rutado entre la parte reconfigurable y la parte estática, haciendo que los puertos del módulo reconfigurable sean compatibles con el diseño de la parte estática.

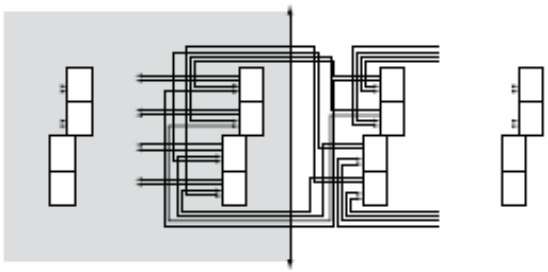


Figura 19. Interior de un *bus macro*

Por tanto todas las conexiones entre la parte estática y el módulo reconfigurable tienen que pasar a través de un *bus macro*, exceptuando la señal de reloj (ya que es manejada automáticamente por las herramientas de implementación de forma transparente al usuario). Las señales de *reset* si que deberán atravesar los *buses macro*.

Hay distintos tipos de *buses macro*, dependiendo de:

- ✓ La dirección de la señal:
  - o De izquierda a derecha:
  - o De derecha a izquierda
  - o De arriba a abajo (sólo ***Virtex-4***)
  - o De abajo a arriba (sólo ***Virtex-4***)

- ✓ El ancho del *bus macro*
  - o Ancho (4 CLBs)
  - o Estrecho (2 CLBs)

- ✓ Si las señales que atraviesan los *buses macro* pasan por registros
  - o Síncrono
  - o Asíncrono

- ✓ Si el *bus macro* se puede activar o desactivar
  - o *Enable*

Las definiciones de los *buses macro* se encuentran en unos archivos de extensión ***.nmc***. Su título en inglés sigue los siguientes criterios:

```
busmacro_device_direction_synchronicity_[enable]_width.nmc

        device =          xc2vp     - Virtex-II Pro
                          xc2v      - Virtex-II
                          xc4v      - Virtex-4
        direction =       r2l       - right-to-left
                          l2r       - left-to-right
                          b2t       - bottom-to-top (Virtex-4 only)
                          t2b       - top-to-bottom (Virtex-4 only)
        synchronicity =   sync      - synchronous
                          async     - asynchronous
        width =           wide      - wide bus macro
                          narrow    - narrow bus macro
```

Las definiciones de los *buses macro* que van a ser utilizados deberán siempre copiarse a la carpeta principal del proyecto *ISE* que se realice, para que no se produzcan errores.

La localización y elección de los *buses macro* debe hacerse en función de si estarán situados en la frontera derecha o izquierda del área reconfigurable y de si las señales entran o salen de dicha área.

Para entradas a un área reconfigurable, si se entra por la frontera izquierda deberá elegirse un *l2r*, y por la frontera derecha un *r2l*:

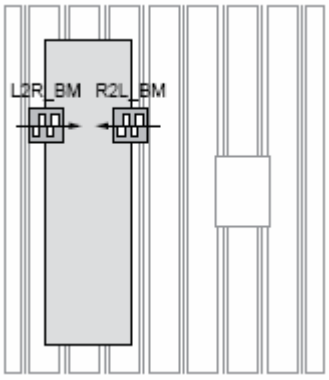


Figura 20. Elección de *buses macro* para entradas al módulo reconfigurable

Para salidas de un área reconfigurable, si se sale por la frontera izquierda deberá elegirse un *r2l*, y por la frontera derecha un *l2r*:

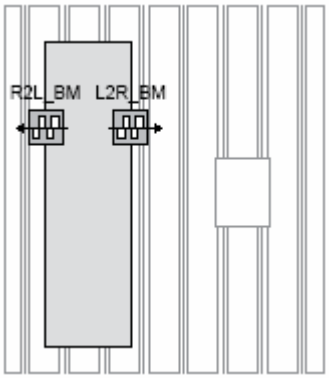


Figura 21. Elección de *buses macro* para salidas de un módulo reconfigurable

Los *buses macro* además de ser declarados e instanciados en el código *vhdl* deben obviamente localizarse en la *FPGA* mediante el fichero de restricciones ***.ucf***. Por ejemplo un *bus macro r2l* localizado en la fila *Y38*, si la frontera derecha del área reconfigurable se extiende hasta la columna *X41* deberá localizarse de la siguiente forma:

```
INST "busmacro_l2r_async_narrow_0" LOC = "SLICE_X40Y38";
```

En ocasiones las declaraciones de los *buses macro* que se consiguen tienen delimitadores de bus distintos a los que soporta el sintetizador, o los nombres de sus entradas y salidas no corresponden con los que nos interesan. En el Anexo 4 se especifica un método para poder editar los archivos ***.nmc*** en modo texto.

2.3.2. MicroBlaze

MicroBlaze es un microprocesador embedido de arquitectura *RISC (Reduced Instruction Set Computer)* optimizado para trabajar en *FPGAs* de *Xilinx*. Una de sus principales ventajas es su alto grado de configurabilidad, lo que permite al usuario seleccionar un determinado juego de características específicas dependiendo de lo que el diseño le exija.

Para ello se dispone de la herramienta de *Xilinx EDK (Embedded Development Kit)*, que facilita considerablemente su configuración, implementación y descarga en la *FPGA*, así como el desarrollo de los programas que se le instalarán. En el Anexo 2 se explica como crear un proyecto de *EDK* para Reconfiguración parcial.

Cabe destacar la configuración de buses:

- *OPB (On-Chip Peripheral Bus):* indicado para la conexión con periféricos situados tanto en el chip como fuera de él y la memoria.

- *LMB (Local Memory Bus):* especialmente diseñado para una comunicación eficiente con la memoria interna.

- *FSL (Fast Simplex Link):* lleva a cabo una comunicación rápida y no arbitrada, útil para la interconexión con coprocesadores de propósito específico

- *XCL (Xilinx Cache Link):* interfaz planeada para la conexión entre cachés y controladores de memoria externos.

- *DCR: (Device Control Register)*: Bus síncrono diseñado para una conexión tipo anillo de periféricos con ancho de banda muy bajo. Solo soporta un maestro y está diseñado para minimizar el uso de lógica interna.

MicroBlaze presenta una arquitectura de memoria *Harvard*. En ella los accesos a datos e instrucciones se realizan en espacios separados de memoria. Cada espacio tiene un rango de 32 bits, lo que posibilita el manejo de hasta 4 gigabytes de instrucciones y de datos de memoria respectivamente.

Uno de los periféricos que puede añadirse al bus *OPB* es el *hwicap*, un dispositivo que permite que *MicroBlaze* acceda a la memoria de configuración de la *FPGA* y la pueda reconfigurar.

MicroBlaze constituye por tanto una solución interesante a la hora de asignar un procesador a la *FPGA* para dirigir el proceso de auto-reconfiguración, debido a sus altas prestaciones, buena configurabilidad y poco espacio ocupados.

Además, para la auto-reconfiguración de periféricos de *MicroBlaze*, *Xilinx* ha proporcionado con el flujo *EA_PR* el módulo *opb_socket_brigde*, que permite conectar el bus *OPB* al periférico reconfigurable a través de los *buses macro* correspondientes.

2.3.3. Hwicap

El *hwicap* es un periférico del bus *OPB,* que permite a un microprocesador embedido (como *MicroBlaze* o *PowerPC*), leer y escribir en la memoria de configuración de la *FPGA* en tiempo de ejecución, a través del *ICAP* (*Internal Configuration Access Port*). Esto facilita al usuario escribir programas para el microprocesador que modifiquen la estructura de circuitos y la funcionalidad de la *FPGA* en tiempo real.

Este elemento permite leer y escribir a nivel de *LUT*, de *Flip-Flop*, *frame*, e incluso permite la descarga de *bitstreams* parciales que reconfiguren una parte de la *FPGA*.

La interfaz del *ICAP* es similar a la del *SelectMAP*, sólo que permite ser conectada desde pines de uso general. Para el correcto funcionamiento del *hwicap* es importante que los *jumpers* que seleccionan el modo de configuración no se encuentren colocados en modo *JTAG o Boundary Scan*, puesto que esto desconecta la interfaz del *ICAP*. Por tanto, como se indicó en el Apartado 2.1, es importante utilizar *Master Serial* (*M2=M1=M0=0),* y conectar la alimentación después de realizar los cambios en la placa.

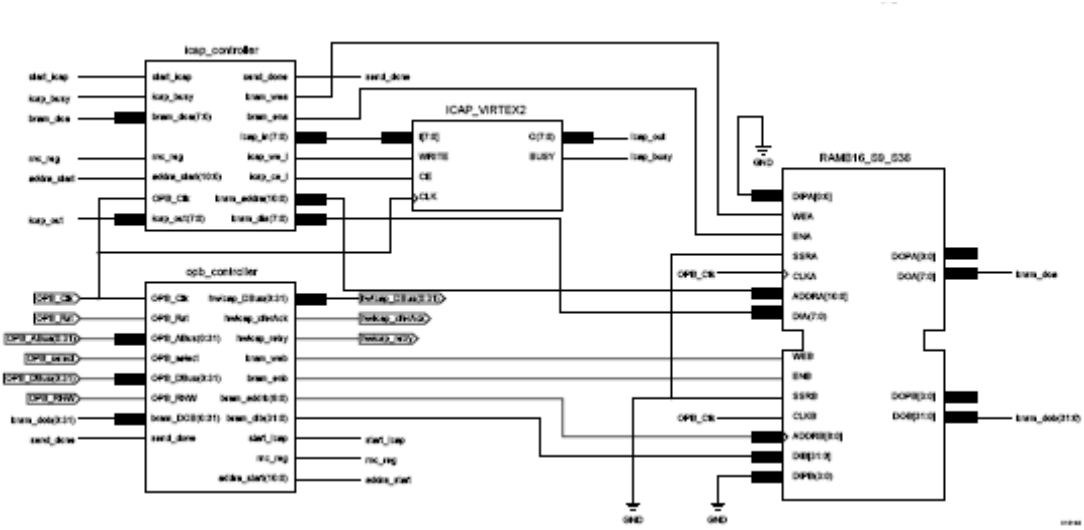


Figura 22. Estructura del *hwicap*

Los *drivers* del *hwicap* disponen de las siguientes funciones:

- *XStatus* ***XHwIcap_Initialize*** *(XHwIcap *InstancePtr, Xuint16 DeviceId, Xuint32 DeviceIdCode):*
  - Inicializa la instancia del *XHwIcap* (es necesario realizarlo antes de ejecutar cualquier otra función).
- *XStatus* ***XHwIcap_DeviceRead*** *(XHwIcap *InstancePtr, Xuint32 Offset, Xuint32 NumInts):*
  - Lee un número de bytes del *ICAP* y los sitúa en el *buffer* de almacenamiento
- *XStatus* ***XHwIcap_DeviceWrite*** *(XHwIcap *InstancePtr, Xuint32 Offset, Xuint32 NumInts):*
  - Escribe un número de bytes del *buffer* de almacenamiento en el *ICAP*.

- *void* ***XHwIcap_StorageBufferWrite*** *(XHwIcap *InstancePtr, Xuint32 Address, Xuint32 Data):*
  - Escribe datos en una dirección del *buffer* de almacenamiento
- *Xuint32* ***XHwIcap_StorageBufferRead*** *(XHwIcap *InstancePtr, Xuint32 Address):*
  - Lee datos de una dirección del *buffer* de almacenamiento
- *XStatus* ***XHwIcap_DeviceReadFrame*** *(XHwIcap *InstancePtr, Xint32 Block, Xint32 MajorFrame, Xint32 MinorFrame):*
  - Lee un *frame* de la *FPGA* y lo coloca en el *buffer* de almacenamiento.
- *XStatus* ***XHwIcap_DeviceWriteFrame*** *(XHwIcap *InstancePtr, Xint32 Block, Xint32 MajorFrame, Xint32 MinorFrame):*
  - Escribe un *frame* del *buffer* de almacenamiento en la *FPGA*
- *XStatus* ***XHwIcap_SetConfiguration*** *(XHwIcap *InstancePtr, Xuint32 *Data, Xuint32 Size).*
  - Carga un *bitstream* parcial de la memoria del sistema en la FPGA.
- *XStatus* ***XHwIcap_CommandDesync*** *(XHwIcap *InstancePtr):*
  - Envía un comando *Desync* al puerto *ICAP*
- *XStatus* ***XHwIcap_CommandCapture*** *(XHwIcap *InstancePtr):*
  - Envía un comando *Capture* al puerto *ICAP*
- *Xuint32* ***XHwIcap_GetConfigReg*** *(XHwIcap *InstancePtr, Xuint32 ConfigReg):*
  - Devuelve el valor del registro de configuración especificado.
- *XStatus* ***XHwIcap_SetClbBits*** *(XHwIcap *InstancePtr, Xint32 Row, Xint32 Col, const Xuint8 Resource[][2], const Xuint8 Value[], Xint32 NumBits):*
  - Establece los bits contenidos en un *CLB* especificados por las coordenadas fila y columna. El *CLB* superior izquierdo es (1,1). Hay 4 *slices* por *CLB.*
- *XStatus* ***XHwIcap_GetClbBits*** *(XHwIcap *InstancePtr, Xint32 Row, Xint32 Col, const Xuint8 Resource[][2], Xuint8 Value[], Xint32 NumBits):*
  - Obtiene los bits contenidos en un *CLB* especificados por las coordenadas fila y columna. El *CLB* superior izquierdo es (1,1). Hay 4 slices por *CLB.*

El *hwicap* será por tanto utilizado para la Auto-Reconfiguración Parcial, es decir, cuando un microprocesador embedido en la *FPGA* sea el que va a reconfigurar parcialmente a la propia *FPGA.*

La función que se utilizará esencialmente para reconfigurar es *XStatus* ***XHwIcap_SetConfiguration*** *(XHwIcap *InstancePtr, Xuint32 *Data, Xuint32 Size),* donde **InstancePtr* es un puntero a la variable que contiene el *HwIcap, *Data* es un puntero a la dirección donde está contenido el *bitstream* parcial, y *Size* es el número de bytes del *bitstream.*

Durante el proyecto se han probado la mayoría de estas funciones, con un resultado exitoso. En el Anexo 6 se utilizan, a modo de ejemplo, varias de estas funciones, para leer y escribir *frames* y *CLBs*.

CAPÍTULO 3
Flujo Antiguo

3.1. Flujo en Diferencias

3.1.1. Reconfiguración Parcial Estática sin Conservar el Estado

Para acercarse al mundo de la reconfiguración parcial, cabe pensar en que lo primero que se puede conseguir es: parar la *FPGA*, reconfigurarla y que se reinicie con la nueva función desde cero, es decir, sin continuar con el estado anterior de la placa, como si se hubiese reseteado.

Esta función puede ser válida aunque los diseños sean completamente distintos, ya que al tratarse de una reconfiguración estática en la que se para la *FPGA*, no podrán darse cortocircuitos, y será como si se hubiese programado la *FPGA* de nuevo.

También puede ser de utilidad en el caso de que se quiera obtener un *bitstream* de menor tamaño que el habitual, o que se quiera tardar menos tiempo en configurar la placa.

Para que se produzca este *reset* automático hay que utilizar la opción de *bitgen*:

```
-g GSR_cycle:4 -g GWE_cycle:4
```

-g GSR_cycle:4: establece en qué momento de la secuencia de arranque se procede a realizar el reset global.

-g GWE_cycle:4: establece en qué momento de la secuencia de arranque se impide cambiar de estado a los biestables y a las *bram* cuando se está escribiendo.

El ejemplo se ha realizado en la placa *Virtex-II Evaluation Board* de *Avnet*, *FPGA xc2v1000-4fg256* de Xilinx, con el software *ISE 7.1*. Se puede encontrar en el cd en la carpeta *xc2v1000/flujodiferencias/noconservaestado*.

En este ejemplo se pretende hacer que un contador cambie de sentido ascendente a descendente. El esquema utilizado es el siguiente:

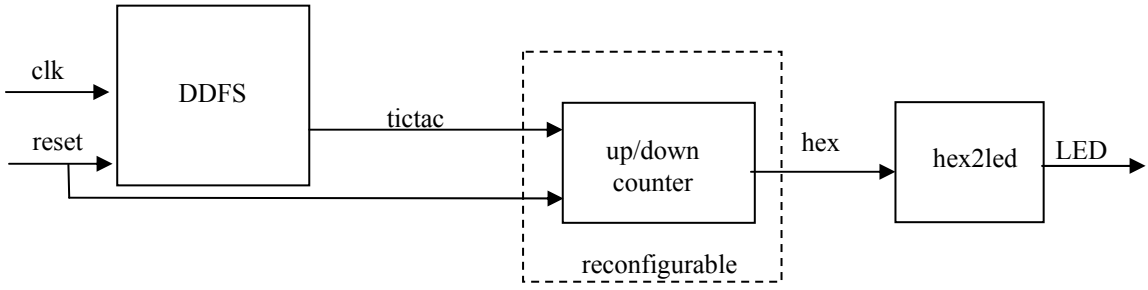


Figura 23. Diagrama de bloques del ejemplo de Reconfiguración Estática con *Virtex-II*

En primer lugar se crea un módulo divisor de frecuencias *ddfs* que se encarga de dividir el reloj para crear una señal llamada *tictac* visible a la vista humana:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity DDFS is
       Port          (      reset : in std_logic;
                            sel   : in std_logic_vector(1 downto 0);
                            clk   : in std_logic;
                            reloj_dividido : out std_logic);
end DDFS;

architecture Behavioral of DDFS is
       -- La fórmula general del Direct Digital Frequency Synthesizer es:
       -- nº veces que sumo el factor antes de desbordar acumulador =
       -- (2^n)/factor = frec_ini/frec_dividida.
       -- En nuestro caso:
       -- 40MHz/1Hz = 2^30/factor =>factor = 26.84;
       -- 40MHz/2Hz = 2^30/factor =>factor = 53.68;
       -- 40MHz/4Hz = 2^30/factor =>factor = 107.37;
       -- 40MHz/8Hz = 2^30/factor =>factor = 214.75;
       constant precision: positive:=30;
       signal acumm: unsigned(precision - 1 downto 0);
       type const_array is array (0 to 3) of integer;
       constant factor: const_array:= (27,54,107,215);

begin

       process(reset, clk)
       begin
              if (reset='1') then
                acumm <= (others=>'0');
              elsif (clk = '1' and clk'event) then
                acumm <= acumm + to_unsigned(factor(to_integer(unsigned(sel))),precision);
              end if;
       end process;

       reloj_dividido <= acumm(precision - 1);

end Behavioral;
```

Por otro lado se tiene un *top level* llamado *conv_frec.vhd,* que obtiene la señal de reloj dividido (*tictac*) y la envía al módulo reconfigurable contador, que será ascendente en primer lugar. De éste sale la cuenta (*HEX*), que irá conectada a un decodificador hexadecimal a *display* 7 segmentos, que permitirá ver el contador en la placa:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity conv_frec is
    Port ( reset : in std_logic; -- pin:
           clk : in std_logic;   -- pin:
           sel : in std_logic_vector(1 downto 0); -- pin:
           LED : out std_logic_vector(6 downto 0)); -- pin:
end conv_frec;

architecture Behavioral of conv_frec is

component DDFS
```

```
Port         (      reset : in std_logic;
                            clk   : in std_logic;
                            sel   : in std_logic_vector(1 downto 0);
                            reloj_dividido : out std_logic);
end component;

signal HEX: unsigned(3 downto 0);
signal tictac: std_logic;

begin

       -- Divisor de frecuencia para generar el parpadeo
       ddfs1 : ddfs
       port map (
                     reset => reset,
                            sel => sel,
                            clk => clk,
                            reloj_dividido => tictac
                            );

       -- Contador que se configurará como asc/desc
       -- con reconfiguración estática
       process(reset, tictac)
       begin
              if(reset='1') then
                     HEX <= (others => '0');
              elsif(tictac='1' and tictac'event) then
                     HEX <= HEX + 1;
              end if;
       end process;

       -- Conversor hex2led para visualizar la cuenta en un 7segmentos
       process(HEX)
       begin
              case HEX is
                     when "0001" => LED <= "0110000";  --1
                     when "0010" => LED <= "1101101";  --2
                     when "0011" => LED <= "1111001";  --3
                     when "0100" => LED <= "0110011";  --4
                     when "0101" => LED <= "1011011";  --5        --       6
                     when "0110" => LED <= "1011111";  --6        --      ---
                     when "0111" => LED <= "1110000";  --7        --  1 |   | 5
                     when "1000" => LED <= "1111111";  --8        --      ---   <- 0
                     when "1001" => LED <= "1111011";  --9        --  2 |   | 4
                     when "1010" => LED <= "1110111";  --A        --      ---
                     when "1011" => LED <= "0011111";  --B        --       3
                     when "1100" => LED <= "1001110";  --C
                     when "1101" => LED <= "0111101";  --D
                     when "1110" => LED <= "1001111";  --E
                     when "1111" => LED <= "1000111";  --F
                     when others => LED <= "1111110";  --0
              end case;
       end process;

end Behavioral;
```

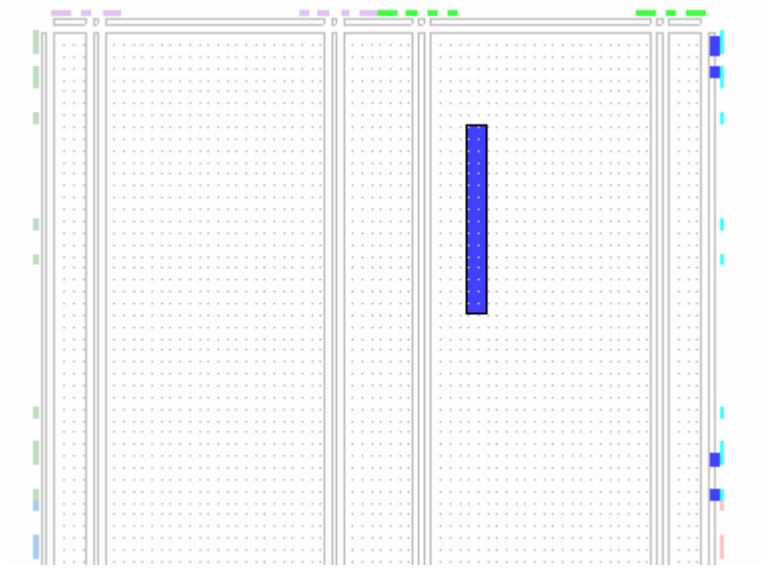
En primer lugar se crea un proyecto de *ISE* en una carpeta (*rec1*), al que se le añaden los archivos anteriores y el ***.ucf***, y se sintetiza con normalidad.

En proyectos parecidos entre si, como es el caso, se recomienda que se haga *floorplanning* con *PACE*, delimitando una zona específica para el proyecto, de tal forma que habrá menos *frames* de diferencia entre uno y otro *bitstream*, por lo tanto el *bitstream* parcial en diferencias será mucho menor.

Para ello se selecciona el fichero ***.ucf*** en la pestaña *Module View*, y se hace doble clic en *Create Area Constraints*.

Una vez se ha abierto *PACE* se despliega el árbol *Logic* en la ventana *Design Browser*:

Se selecciona el módulo completo (en este caso *conv_frec*), arrastrándolo hacia el lugar elegido en la *FPGA*, extendiéndolo todo lo que se estime necesario:



Se guarda el resultado, de tal forma que el fichero ***.ucf*** queda:

```
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 25 ns HIGH 50 %;

#PACE: Start of PACE I/O Pin Assignments
NET "clk"  LOC = "T9"  ;
NET "LED<0>"  LOC = "D16"  ;
NET "LED<1>"  LOC = "D14"  ;
NET "LED<2>"  LOC = "D15"  ;
NET "LED<3>"  LOC = "E13"  ;
NET "LED<4>"  LOC = "E14"  ;
NET "LED<5>"  LOC = "H13"  ;
NET "LED<6>"  LOC = "H14"  ;
```

```
NET "reset"  LOC = "R10"  ;
NET "sel<0>"  LOC = "H15"  ;
NET "sel<1>"  LOC = "H16"  ;

#PACE: Start of PACE Area Constraints
AREA_GROUP "AG_conv_frec" RANGE = SLICE_X40Y71:SLICE_X41Y56 ;
INST "/" AREA_GROUP = "AG_conv_frec" ;
```

Finalmente se implementa y se obtiene el *bitstream* con normalidad.

A continuación se copia la carpeta del proyecto y se renombra a *rec2*, se abre y se hacen los cambios deseados en el código, en este caso se cambia el siguiente proceso, haciendo que la cuenta sea descendente:

```
-- Contador que se configurará como asc/desc
-- con reconfiguración estática
process(reset, tictac)
begin
      if(reset='1') then
            HEX <= (others => '0');
      elsif(tictac='1' and tictac'event) then
            HEX <= HEX - 1;
      end if;
end process;
```

Se sintetiza e implementa el proyecto. Se abre una ventana de comandos, se accede al directorio de este segundo proyecto y se teclea la siguiente línea de comandos:

```
bitgen –w -r ..\rec1\conv_frec.bit conv_frec.ncd conv_frec2.bit -g GWE_cycle:4
```

Donde, como se ha explicado, la opción *–r* genera el *bitstream* parcial en diferencias, y *-g GWE_cycle:4* hace que se impida cambiar de estado a los biestables y a las *bram* cuando se está reconfigurando. *Bitgen* indica que hay 17 frames de diferencia entre un diseño y otro.

Se puede observar que no se ha incluido la directiva *-g GSR_cycle:4*, ya que para las placas ***Virtex-II*** no existe esa opción, se resetea globalmente por defecto.

A continuación se descarga mediante *iMPACT* el primer *bitstream*, que realiza el contador ascendente y posteriormente el *bitstream* parcial, observando que la placa se reconfigura correctamente, parándose y reseteando a 0 antes de comenzar el contador descendente.

3.1.2. Reconfiguración Parcial Estática Conservando el Estado:

En algunas ocasiones será necesario que se mantenga el estado anterior en la *FPGA*, aunque ésta se pare para pasar de un diseño a otro.

Esta opción impide hacer auto-reconfiguración ya que si no la *FPGA* se pararía cuando se estuviera reconfigurando, pero permite otras opciones entre dos diseños similares.

Si se quiere mantener el estado es necesario que los dos diseños sean lo más parecidos posibles, forzando al software a que implemente las partes comunes utilizando los mismos recursos en la *FPGA*.

Para ello existe la opción *exact* en el mapeado y enrutado de un proyecto, al que se le pone como referencia otro proyecto ya mapeado y enrutado, para que todas las partes comunes se sitúen igual en la *FPGA*.

El ejemplo se ha realizado en la placa *Virtex-II Evaluation Board* de *Avnet*, *FPGA xc2v1000-4fg256* de Xilinx, con el software *ISE 7.1*. Se puede encontrar en el cd en la carpeta *xc2v1000/flujodiferencias/conservaestado*.

Se ha utilizado el mismo ejemplo de contador ascendente/descendente. Como se indicó en el apartado anterior, las ***Virtex-II*** se resetean por defecto al reconfigurar estáticamente, por lo que este método en principio sólo funciona con tarjetas ***Virtex***.

Para tratar de conseguir que no se produjese el reseteo en las ***Virtex-II*** se retiró el reset global del módulo reconfigurable y se realizó el proyecto:

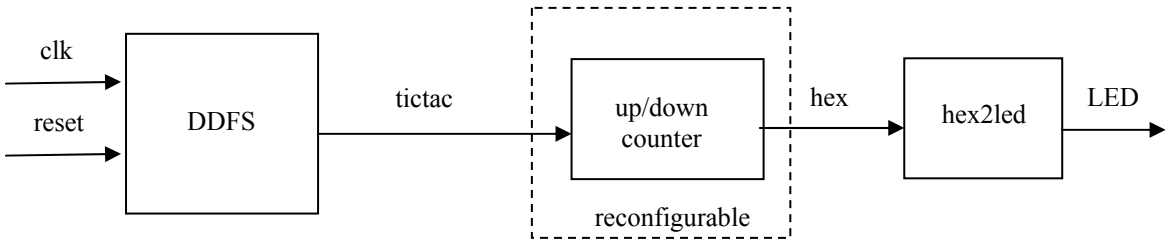
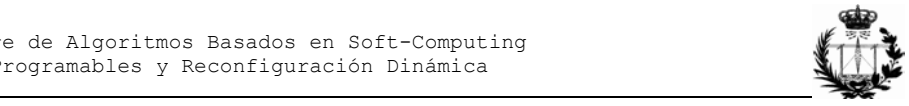


Figura 24. Diagrama de bloques del ejemplo de Reconfiguración conservando el estado

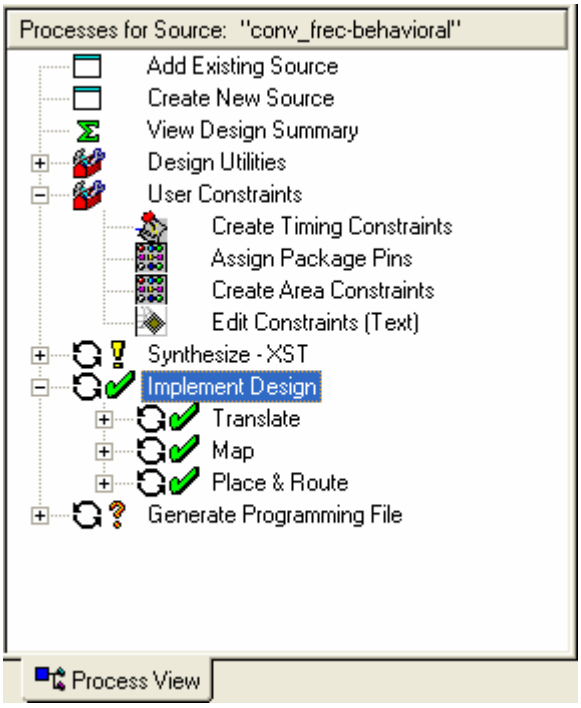
El módulo reconfigurable quedó de la siguiente forma:

```
-- Contador que se configurará como asc/desc
-- con reconfiguración estática
process(tictac)
begin
if(tictac='1' and tictac'event) then
            HEX <= HEX + 1;
      end if;
end process;
```

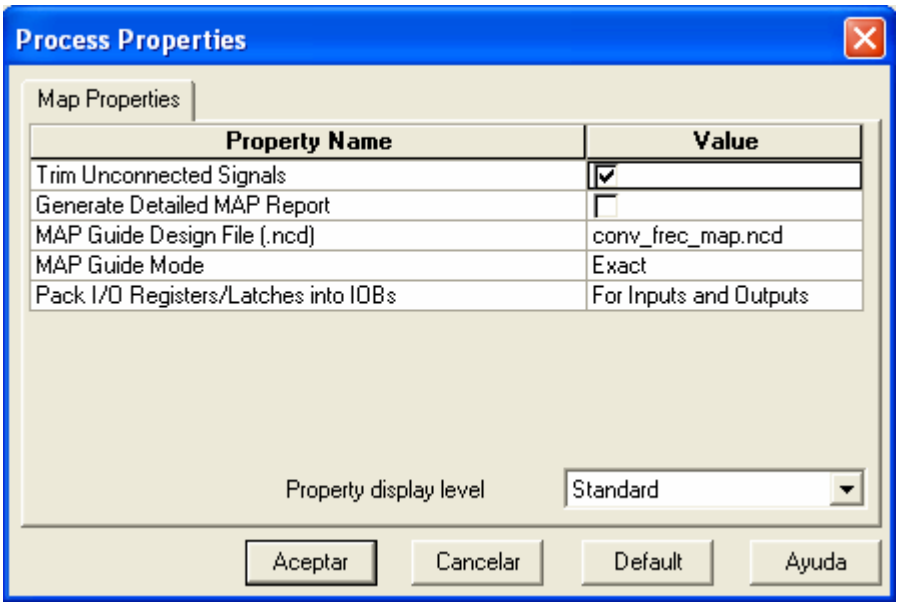
En primer lugar se crea un proyecto (*rec1* en este caso), que se sintetiza, se implementa y se obtiene el *bitstream* tal y como se hizo en el caso anterior (también es conveniente hacer el *floorplannig*).



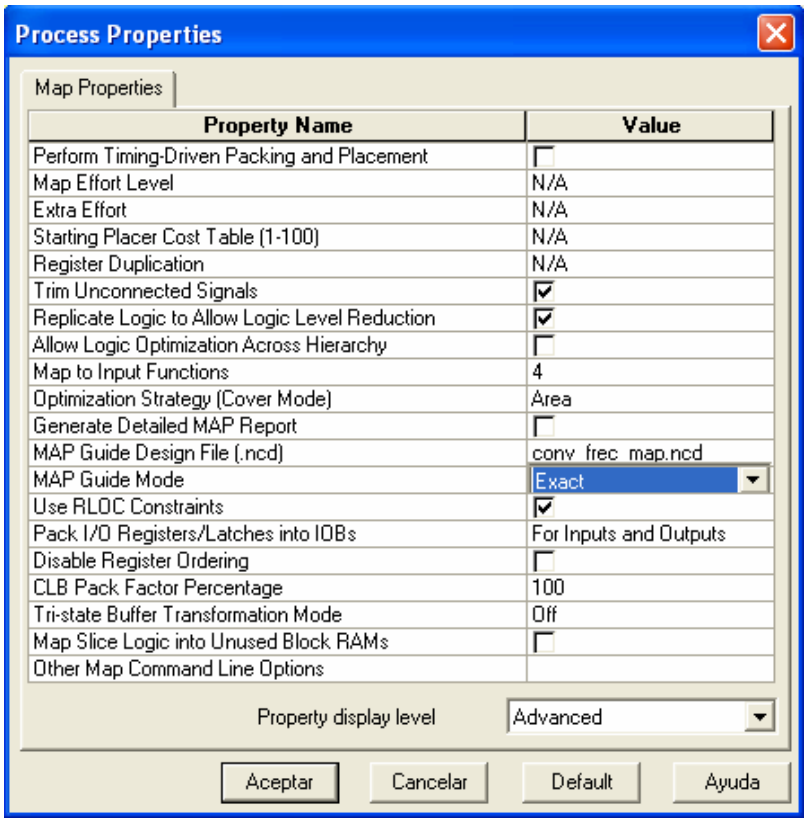
A continuación se cierra el proyecto y se copia esa carpeta renombrándola a *rec2*. Se abre el nuevo proyecto y se edita el contador reconfigurable para hacerlo descendente. Se selecciona el fichero *conv_frec.vhd* y en la pestaña *Process View* se hace clic en *Implement Design*:



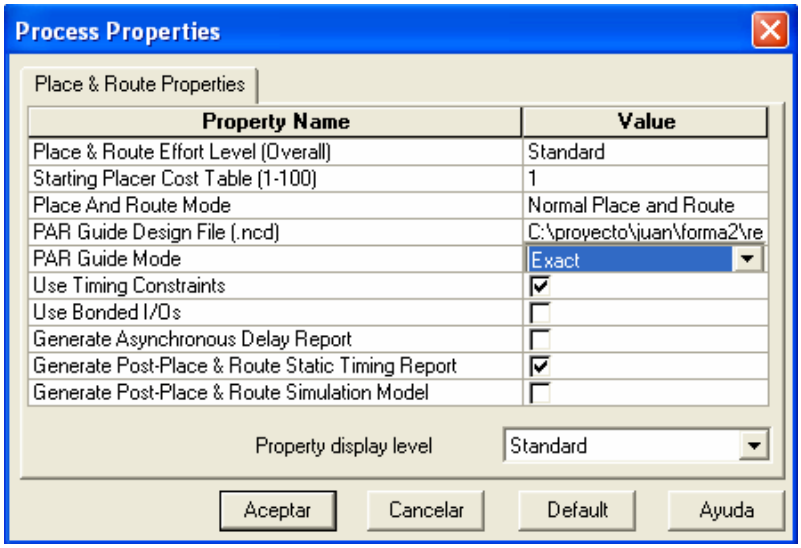
A continuación se hace clic con el botón derecho sobre *Map* y se selecciona *Properties*, abriéndose la siguiente ventana.



En *Property display level* se selecciona *Advanced*, y en las nuevas propiedades que aparecen se busca *MAP Guide mode* y se selecciona *Exact*. A continuación en el apartado *Map Guide Design File (.ncd)* se hace clic y se busca el archivo *conv_frec.ncd* del primer proyecto (que es el fichero que se utilizará para hacer que este mapeado sea lo más parecido posible al del anterior proyecto). Finalmente se acepta.



En la pestaña *Process View* se hace clic con el botón derecho encima de *Place & Route,* seleccionando *Properties*. Se selecciona el *PAR Guide Mode* como *Exact*, y en *PAR Guide Design (.ncd)* se selecciona nuevamente el archivo *conv_frec.ncd* del proyecto anterior.



Finalmente se acepta y se procede a sintetizar e implementar el proyecto. Se abre una ventana de comandos y se accede a la carpeta *rec2*, tecleando el siguiente código para generar el *bitstream* parcial en diferencias:

```
bitgen -w -r ..\rec1\conv_frec.bit conv_frec.ncd conv_frec2.bit
```

A continuación se descarga mediante *iMPACT* el primer bitstream, que instala el contador ascendente en la placa y posteriormente el *bitstream* parcial, observando que se reconfigura correctamente, parándose pero conservando el estado.

Con este método el número de frames de diferencia se reduce de 17 a 5, confirmando el buen funcionamiento de la directiva *exact*.

3.1.3. Reconfiguración Parcial Activa

En el caso de que no se desee que la *FPGA* se pare durante la reconfiguración, existe la posibilidad de la reconfiguración parcial activa. Para ello además de la opción diferencial *–r,* se ha de añadir el comando *-g ActiveReconfig:Yes*.

Como la placa sigue funcionando durante la reconfiguración, existe riesgo de cortocircuito, por lo que hay que tomar ciertas medidas a la hora de realizar el diseño. Habrá que hacer *floorplanning* con *PACE* y separar la parte estática de la parte reconfigurable. Entre una zona y otra habrá que colocar *buffers* triestado que permitan aislar cuando se produce la reconfiguración.

Debe también existir una señal de *reset* parcial, que resetee e inicialice el estado de la parte reconfigurable una vez reconfigurada. Esto se hará conectando la salida del *buffer* triestado correspondiente a *PULL UP*.

El ejemplo escogido es similar a los casos anteriores, el contador ascendente/descendente, salvo que se añaden nuevos elementos para hacer ciertas comprobaciones que se explican más adelante. El diagrama de bloques del diseño queda de la siguiente forma:

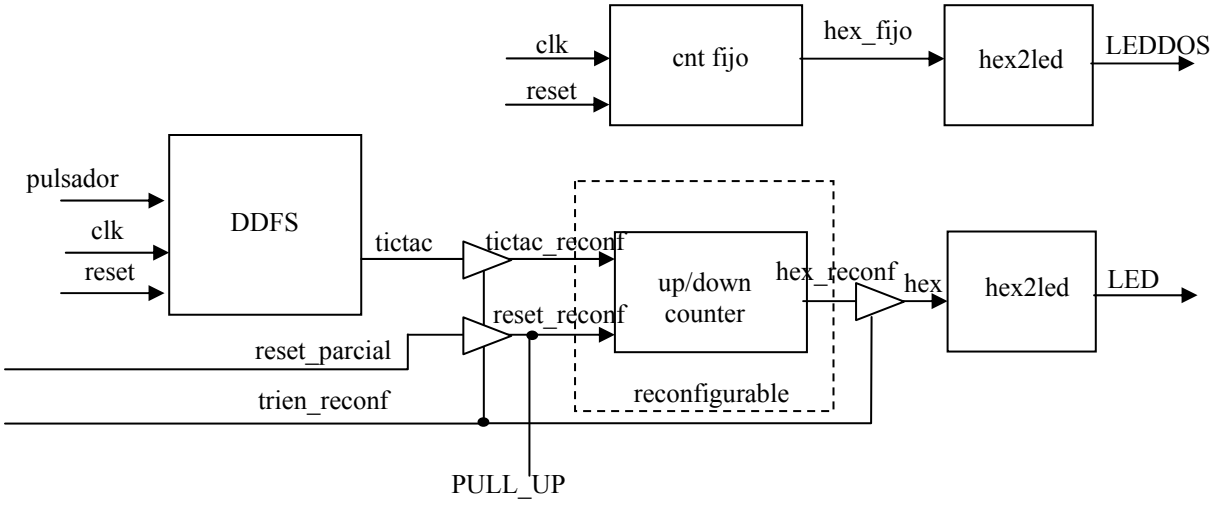


Figura 25. Diagrama de bloques del ejemplo de Reconfiguración Parcial Activa

El ejemplo se ha realizado en la placa *Virtex-II Evaluation Board* de *Avnet*, *FPGA xc2v1000-4fg256* de Xilinx, con el software *ISE 7.1*. Se puede encontrar en el cd en la carpeta *xc2v1000/flujodiferencias/reconfiguracionactiva* y en el vídeo demostrativo "Reconfiguración parcial flujo diferencias xc2v1000".

En primer lugar, para facilitar la separación entre parte estática y parte reconfigurable, se han introducido todos los procesos que antes estaban en el *top level*, en distintos módulos con sus entradas y salidas respectivas.

Para comprobar que la reconfiguración parcial es activa, es decir, que no se para la *FPGA* durante la reconfiguración, se ha realizado un proyecto en el que se añade otro contador idéntico pero a velocidad de reloj (es decir, en todo momento se debe ver un ocho, ya que la velocidad es más rápida que la de la vista humana). Cuando se proceda a reconfigurar se comprobará que el 8 permanezca sin parpadear en ningún momento (en ejemplos similares realizados en los casos anteriores de reconfiguración parcial estática si que se podía observar que el ocho desaparecía unos instantes durante la reconfiguración).

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity conv_frec is
    Port ( reset             : in std_logic;  -- pin:
           clk          : in std_logic; -- pin:
           pulsador     : in std_logic; -- pin:
                     reset_parcial: in std_logic;
                     trien_reconf : in std_logic;
           LED          : out std_logic_vector(6 downto 0); -- pin:
                     LEDDOS       : out std_logic_vector(6 downto 0));
end conv_frec;

architecture Behavioral of conv_frec is

      component DDFS
      Port         (     reset : in std_logic;
                               clk   : in std_logic;
                               pulsador : in std_logic;
                               reloj_dividido: out std_logic);
      end component;

      component reconf_wrapper
      Port         (--reconf-specific I/O:
                        trien_reconf : in std_logic;
                        reset_parcial: in std_logic;
                        --reset_reconf : out std_logic;
                        rst : out std_logic;
                        --general purpose DataIn:
                        tictac : in std_logic;
                        tictac_reconf: out std_logic;
                        --general purpose DataOut:
                        hex_reconf: in std_logic_vector(3 downto 0);
                          hex : out std_logic_vector(3 downto 0));
      end component;

            --Declaration of user reconfigurable module
      component cnt
      Port         (     reset  : in std_logic;
                               tictac : in std_logic;
                               count  : out std_logic_vector(3 downto 0));
      end component;
```

```
            --Declaracion del contador fijo
      component cntfijo
      Port         (     reset  : in std_logic;
                         clk : in std_logic;
                         count  : out std_logic_vector(3 downto 0));
      end component;

      component hex2led
      Port         (     HEX : in std_logic_vector(3 downto 0);
                         LED : out std_logic_vector(6 downto 0));
      end component;

signal tictac : std_logic;
signal hex  : std_logic_vector(3 downto 0);
signal reset_reconf: std_logic;
signal tictac_reconf: std_logic;
signal hex_reconf: std_logic_vector(3 downto 0);
signal hex_fijo: std_logic_vector(3 downto 0);

begin

      -- Divisor de frecuencia para generar el parpadeo
      ddfs1 : ddfs
      port map (
                  reset => reset,
                           pulsador => pulsador,
                           clk => clk,
                       reloj_dividido => tictac
                      );

      --Wrapper para los triestados del contador reconfigurable
      rw1 : reconf_wrapper
      port map(--reconf-specific I/O:
                          trien_reconf => trien_reconf,
                        reset_parcial => reset_parcial,
                            --reset_reconf => reset_reconf,
                            rst => reset_reconf,
                      --general purpose DataIn:
                               tictac => tictac,
                        tictac_reconf => tictac_reconf,
                      --general purpose DataOut:
                           hex_reconf => hex_reconf,
                                  hex => hex
                      );

      -- Contador que se configurará como asc/desc
      -- con reconfiguración dinámica(activa)
      cnt1 : cnt
      port map ( --reconf-specific I/O:
                    reset => reset_reconf,
                          --general purpose DataIn:
                          tictac => tictac_reconf,
                          --general purpose DataOut:
                           count => hex_reconf
                      );

      --convertidor hex2led para mostrar la salida del contador
reconfigurable
      hex2led1 : hex2led
      port map (
                   hex => hex,
                          led => led
                          );

      cont_fijo : cntfijo
      port map (   reset => reset,
                                clk => clk,
                             count => hex_fijo
```

```
                              );

       --convertidor hex2led para mostrar la salida del contador fijo
       hex2led_fijo : hex2led
       port map (
                     hex => hex_fijo,
                            led => leddos
                            );

end Behavioral;
```

Se ha creado un *wrapper* alrededor del contador reconfigurable en el que van situados los *buffers* triestado, que serán controlados desde un pulsador externo tipo *switch*. Además también se encuentra ahí el *PULL UP* que se encargará de resetear la parte reconfigurable para que no se produzcan cortocircuitos:

```
entity reconf_wrapper is
Port          (--reconf-specific I/O:
                              trien_reconf : in std_logic;
                              reset_parcial: in std_logic;
                              --reset_reconf : out std_logic;
                              rst : out std_logic;
                              --general purpose DataIn:
                              tictac : in std_logic;
                              tictac_reconf: out std_logic;
                              --general purpose DataOut:
                              hex_reconf: in std_logic_vector(3 downto 0);
                                hex : out std_logic_vector(3 downto 0));
end reconf_wrapper;

architecture Behavioral of reconf_wrapper is

signal reset_reconf : std_logic;

begin

       --tristates-wrapper for reconfigurable module I/O:
       process(trien_reconf, reset_parcial, tictac, hex_reconf)
       begin
              if trien_reconf='1' then
                     reset_reconf <= reset_parcial;
                     tictac_reconf <= tictac;
                     hex <= hex_reconf;
              else
                     reset_reconf <= 'Z';
                     tictac_reconf <= 'Z';
                     hex <= (others=>'Z');
              end if;
       end process;

       process(reset_reconf)
       begin
              if reset_reconf='Z' then
                     rst <= '1';
              else
                     rst <= reset_reconf;
              end if;
       end process;

end Behavioral;
```

Se ha modificado el divisor de frecuencias *ddfs* para que cambie la velocidad del contador según se desee mediante un pulsador externo que controla una máquina de estados. Esto permitirá observar si la parte estática de la placa se resetea, ya que si se conserva el estado, la velocidad del contador será la misma antes y después de la reconfiguración. La máquina de estados y el código utilizados es el siguiente:

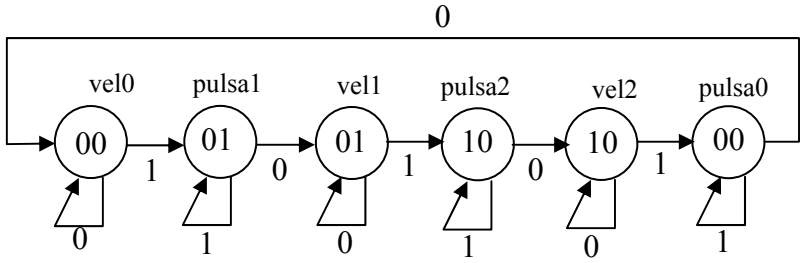


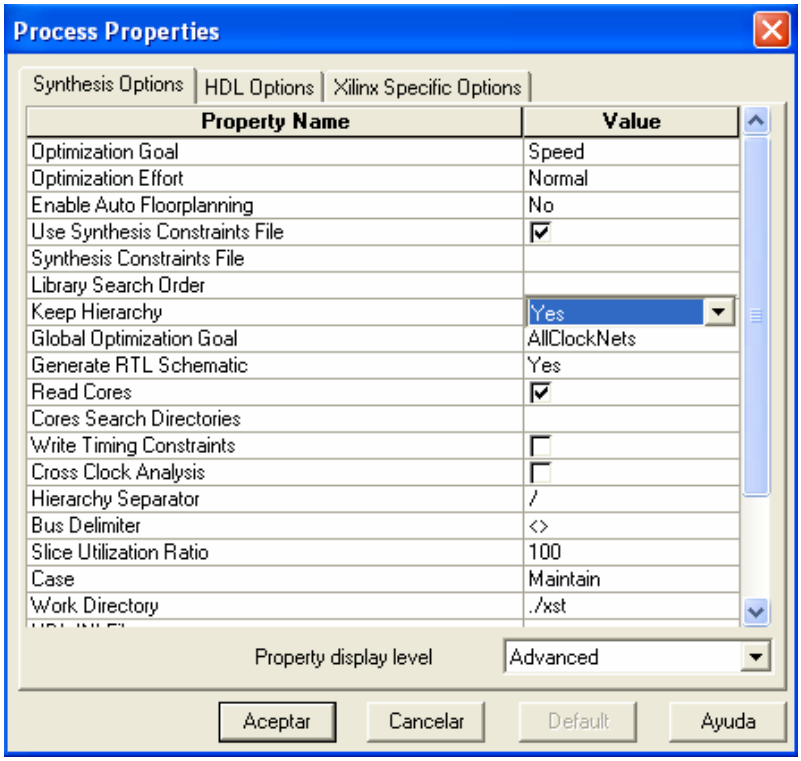
Figura 26. Máquina de estados del divisor de frecuencias *ddfs*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity DDFS is
      Port         (     reset : in std_logic;
                         pulsador : in std_logic;
                         clk   : in std_logic;
                         reloj_dividido: out std_logic);
end DDFS;

architecture Behavioral of DDFS is

      constant precision: positive:=30;
      signal acumm: unsigned(precision - 1 downto 0);
      type const_array is array (0 to 3) of integer;
      constant factor: const_array:= (27,54,107,215);
      signal sel: std_logic_vector(1 downto 0);
      --señales para la FSM:
      type type_state is (vel0,pulsa1,vel1,pulsa2,vel2,pulsa0);
      signal present_state, next_state: type_state;

begin

      --FSM para controlar el pulsador de selección de velocidad:
      process(reset, acumm(precision - 1))
      begin
            if reset='1' then
                  present_state <= vel0;
            elsif rising_edge(acumm(precision - 1)) then    --para hacerlo +lento y evitar los
rebotes
                  present_state <= next_state;
            end if;
      end process;

      process(present_state, pulsador)
      begin
            case present_state is
            when vel0 =>
                  if pulsador='0' then next_state <= vel0; else next_state <= pulsa1; end if;
                  sel <= "00";
            when pulsa1 =>
                  if pulsador='0' then next_state <= vel1; else next_state <= pulsa1; end if;
                   sel <= "01";
            when vel1 =>
                  if pulsador='0' then next_state <= vel1; else next_state <= pulsa2; end if;
```

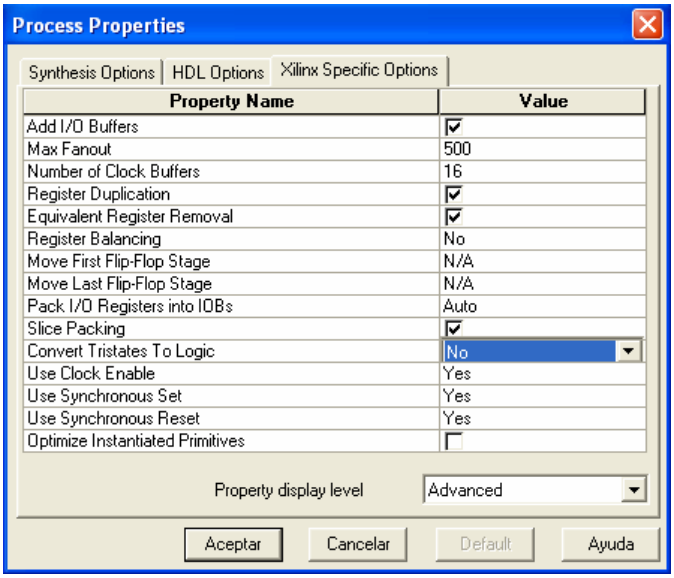
```
                sel <= "01";
        when pulsa2 =>
                if pulsador='0' then next_state <= vel2; else next_state <= pulsa2; end if;
                sel <= "10";
        when vel2 =>
                if pulsador='0' then next_state <= vel2; else next_state <= pulsa0; end if;
                sel <= "10";
        when pulsa0 =>
                if pulsador='0' then next_state <= vel0; else next_state <= pulsa0; end if;
                sel <= "00";
        end case;
end process;


--El ddfs propiamente dicho comienza aqui:
process(reset, clk)
begin
        if (reset='1') then
                acumm <= (others=>'0');
        elsif (clk = '1' and clk'event) then
                acumm <= acumm + to_unsigned(factor(to_integer(unsigned(sel))),precision);
        end if;
end process;

reloj_dividido <= acumm(precision - 1);

end Behavioral;
```

Una vez creado el proyecto (situado en este caso en la carpeta *chip1*), y añadido los ficheros de código, se hace clic con el botón derecho dentro de la pestaña *Processes View en Synthesize – XST* y se selecciona *Properties*.

En el apartado *Property Display Level* se selecciona *Advanced*, y dentro de la pestaña *Synthesis Options,* se marca *Yes* en la casilla *Keep Hierarchy*, para que se respete que la lógica permanezca dentro de los módulos y no se mezcle con la de otros módulos.

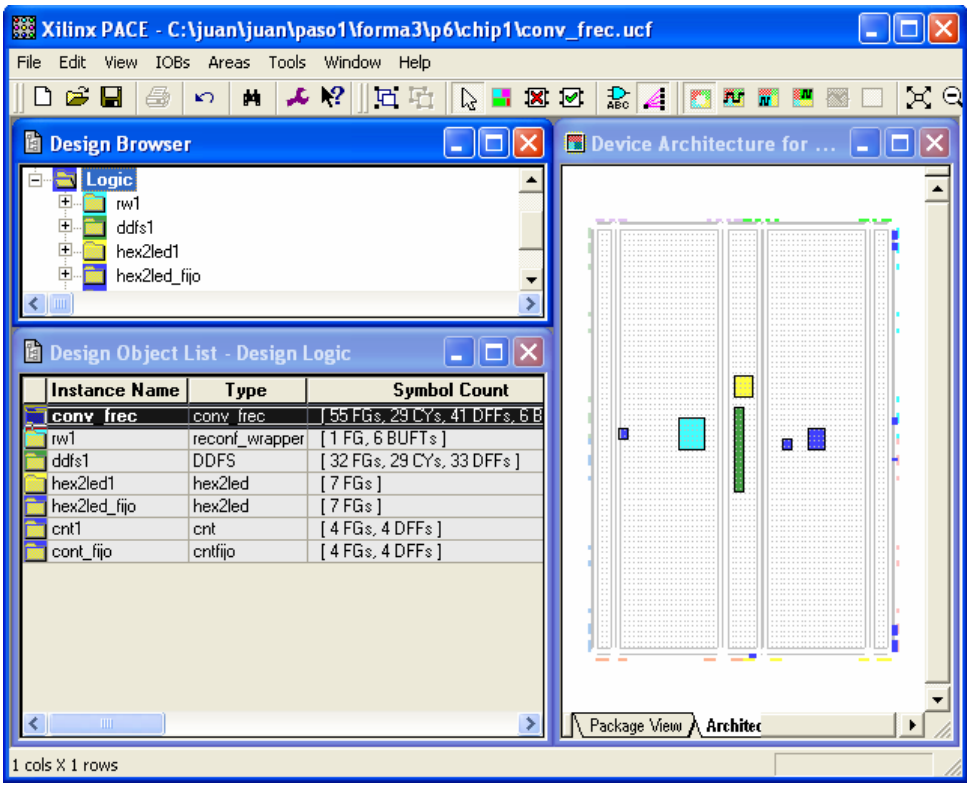


A continuación, en la pestaña *Xilinx Specific Options*, se selecciona el valor *No* para *Convert Triestates to Logic*. Esto es necesario para que se utilicen triestados reales y no triestados hechos a partir de lógica, que podrían dañar la placa en caso de cortocircuito..



Se selecciona el fichero ***.ucf*** y se hace clic en *Create Area Constraints*, de tal forma que se abre *PACE*. Tal y como se indicó en el apartado 3.1.1, se seleccionan todos los módulos y se van situando sobre la *FPGA*, cumpliendo las siguientes normas:

1. La parte reconfigurable no debe tener ningún módulo por encima o por debajo, ya que con este flujo se reconfigura el *frame* completo.
2. El posicionamiento horizontal del módulo reconfigurable debe comenzar en un *slice* múltiplo de 4: 0, 4, 8, …



El código del fichero de restricciones ***.ucf*** quedará:

```
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 25 ns HIGH 50 %;

#PACE: Start of PACE I/O Pin Assignments
NET "clk"  LOC = "T9"  ;
NET "LED<0>"  LOC = "D16"  ;
NET "LED<1>"  LOC = "D14"  ;
NET "LED<2>"  LOC = "D15"  ;
NET "LED<3>"  LOC = "E13"  ;
NET "LED<4>"  LOC = "E14"  ;
NET "LED<5>"  LOC = "H13"  ;
NET "LED<6>"  LOC = "H14"  ;
NET "LEDDOS<0>"  LOC = "J13"  ;
NET "LEDDOS<1>"  LOC = "M14"  ;
NET "LEDDOS<2>"  LOC = "M13"  ;
NET "LEDDOS<3>"  LOC = "N15"  ;
NET "LEDDOS<4>"  LOC = "N14"  ;
NET "LEDDOS<5>"  LOC = "N16"  ;
NET "LEDDOS<6>"  LOC = "P16"  ;
NET "pulsador"  LOC = "T10"  ; #pushbutton()

NET "reset"  LOC = "R10"  ;
NET "reset_parcial"  LOC = "M15"  ; #switch(7)
NET "trien_reconf"  LOC = "M16"  ; #switch(6)

#PACE: Start of PACE Area Constraints
AREA_GROUP "AG_cnt1" RANGE = SLICE_X4Y41:SLICE_X5Y40 ;
INST "cnt1" AREA_GROUP = "AG_cnt1" ;
AREA_GROUP "AG_cont_fijo" RANGE = SLICE_X40Y39:SLICE_X41Y38 ;
INST "cont_fijo" AREA_GROUP = "AG_cont_fijo" ;
AREA_GROUP "AG_ddfs1" RANGE = SLICE_X30Y45:SLICE_X31Y30 ;
INST "ddfs1" AREA_GROUP = "AG_ddfs1" ;
AREA_GROUP "AG_hex2led1" RANGE = SLICE_X30Y51:SLICE_X33Y48 ;
INST "hex2led1" AREA_GROUP = "AG_hex2led1" ;
AREA_GROUP "AG_hex2led_fijo" RANGE = SLICE_X46Y41:SLICE_X49Y38 ;
INST "hex2led_fijo" AREA_GROUP = "AG_hex2led_fijo" ;
AREA_GROUP "AG_rw1" RANGE = SLICE_X18Y43:SLICE_X23Y38 ;
AREA_GROUP "AG_rw1" RANGE = TBUF_X18Y43:TBUF_X22Y38 ;
INST "rw1" AREA_GROUP = "AG_rw1" ;
```

Finalmente ya se puede implementar y obtener el *bitstream* del primer proyecto.

A continuación se copia y se renombra la carpeta a *chip2*, se realizan los cambios para hacer que el contador sea descendente (sólo el contador reconfigurable, no el contador fijo) y se procede de la misma forma que en el apartado anterior, haciendo los procesos de *MAP* y *PAR* de forma *exact*, con mucho cuidado de seleccionar el archivo *conv_frec.ncd* de la carpeta *chip1,* y se procede a la implementación y a la obtención del *bitstream* completo de forma normal.

Se abre una ventana de comandos, accediendo a la carpeta que contiene a *chip1* y *chip2* y se ejecutan las siguientes sentencias:

```
bitgen -r ./chip1/conv_frec.bit -g ActiveReconfig:Yes ./chip2/conv_frec.ncd
chip1_2_chip2.bit -w

bitgen -r ./chip2/conv_frec.bit -g ActiveReconfig:Yes ./chip1/conv_frec.ncd
chip2_2_chip1.bit -w
```

Con esto se habrán obtenido dos *bitstreams* parciales en diferencias, que permitirán cambiar del diseño uno al dos y del dos al uno respectivamente.

Para su comprobación se descarga el primero de los *bitstreams* completos, el del contador ascendente. Una vez funcionando, se pulsa el primer *switch* (que acciona los *buffers* triestado) y el segundo (que acciona el reset parcial de la parte reconfigurable), y se procede a descargar mediante *iMPACT* el *bitstream* parcial *chip1_2_chip2.bit*. A continuación se vuelven a poner los *switchs* en su posición normal y se observa como el contador ha pasado a ser descendente, consiguiendo con ello la reconfiguración parcial activa, tal y como se puede ver en el vídeo demostrativo.

Finalmente se pueden realizar varias pruebas para confirmar que se ha realizado correctamente la reconfiguración parcial activa:

- En primer lugar para confirmar que la reconfiguración ha sido activa, es decir, que la *FPGA* no se ha parado, basta con observar que el segundo contador no ha parpadeado cuando se producía la reconfiguración.

- Por último, y teniendo en cuenta que se puede producir un cortocircuito y que va en contra de lo indicado en el flujo diferencial, se puede proceder a comprobar que también se guarda el estado de la parte reconfigurable, si al realizar la reconfiguración únicamente se pulsa el *switch* de los *buffers* triestado, y no el del reset parcial, observando que la cuenta comienza a descender desde el número por el que iba.

3.1.4. Auto-Reconfiguración Parcial Activa

Para conseguir la auto-reconfiguración parcial activa se instancia un procesador *MicroBlaze* con *hwicap,* que realizara la reconfiguración.

El ejemplo se ha realizado en la placa *Virtex-II Evaluation Board* de *Avnet*, *FPGA xc2v1000-4fg256* de Xilinx, con el software *ISE 7.1 y EDK 7.1 SP1.* Se puede encontrar en el cd en la carpeta *xc2v1000/flujodiferencias/autoreconfiguracion* y en el vídeo demostrativo y en el vídeo demostrativo "Auto-Reconfiguración parcial flujo diferencias xc2v1000".

Se utilizó el mismo ejemplo que en el apartado anterior, un contador ascendente/descendente y un contador fijo, pero en este caso se instancia un microprocesador MicroBlaze para que realice la auto-reconfiguración.

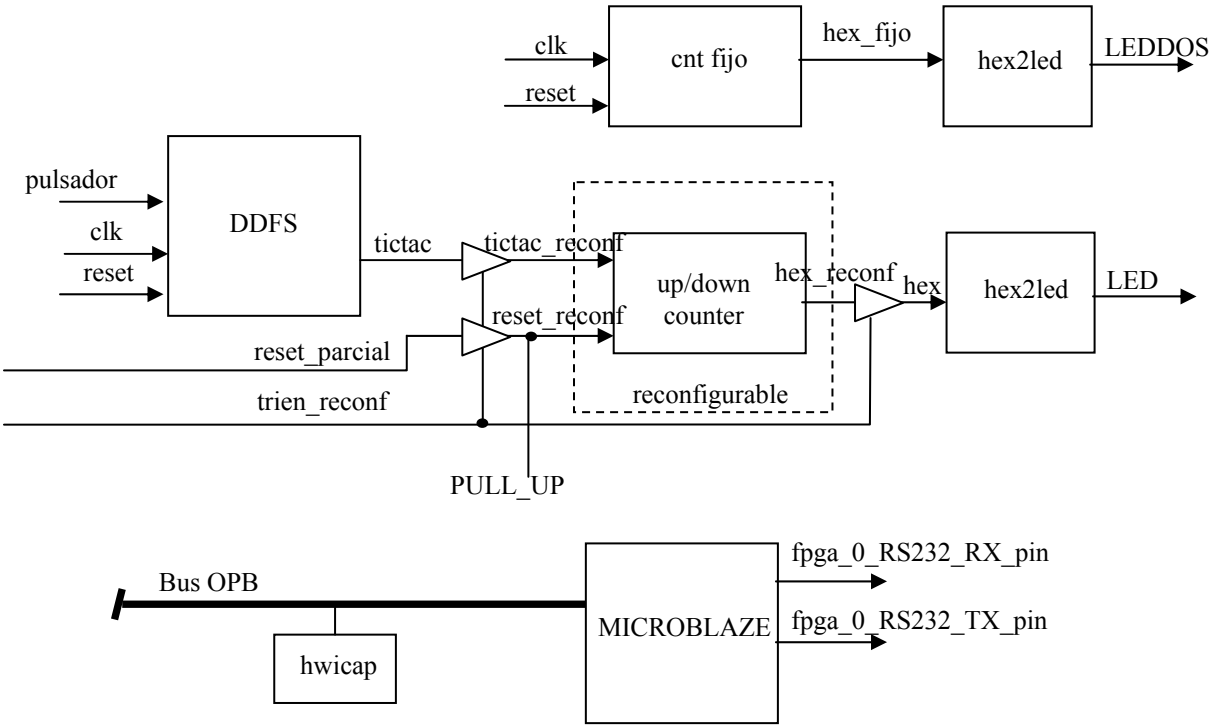
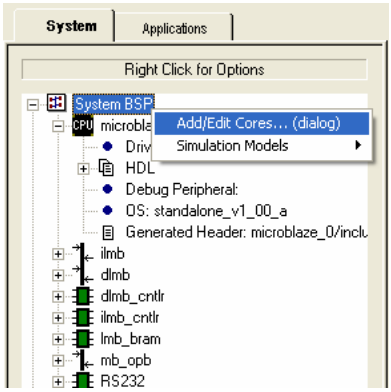


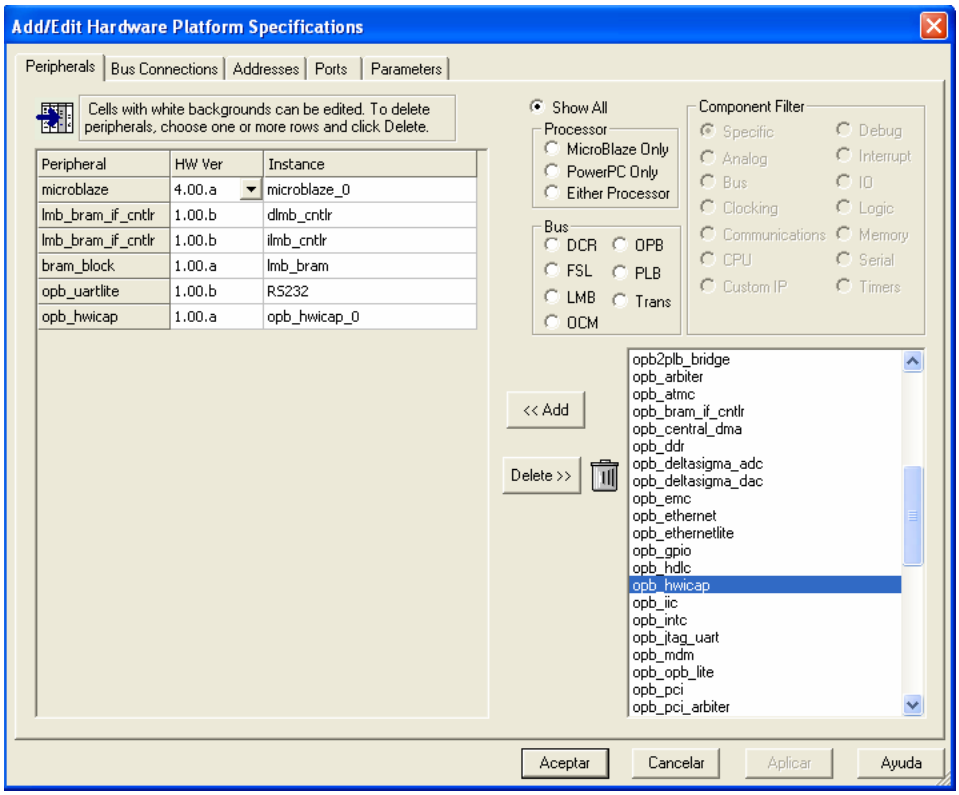
Figura 27. Diagrama de bloques del ejemplo de Auto-Reconfiguración con el flujo en diferencias

En primer lugar se crea un proyecto *EDK* con *UART* utilizando el *wizard* del *EDK,* tal y como se indica en el Anexo 2, pero teniendo en cuenta que este proyecto es de *EDK 7.1* y no de *EDK 8.2*.

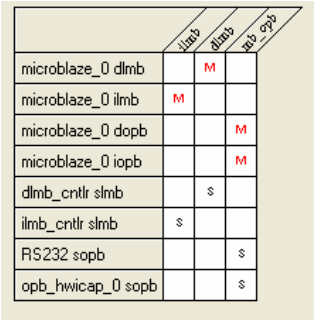
A continuación se hace clic con el botón derecho en el apartado *System BSP* de la pestaña *System*, seleccionando *Add/Edit Cores*.



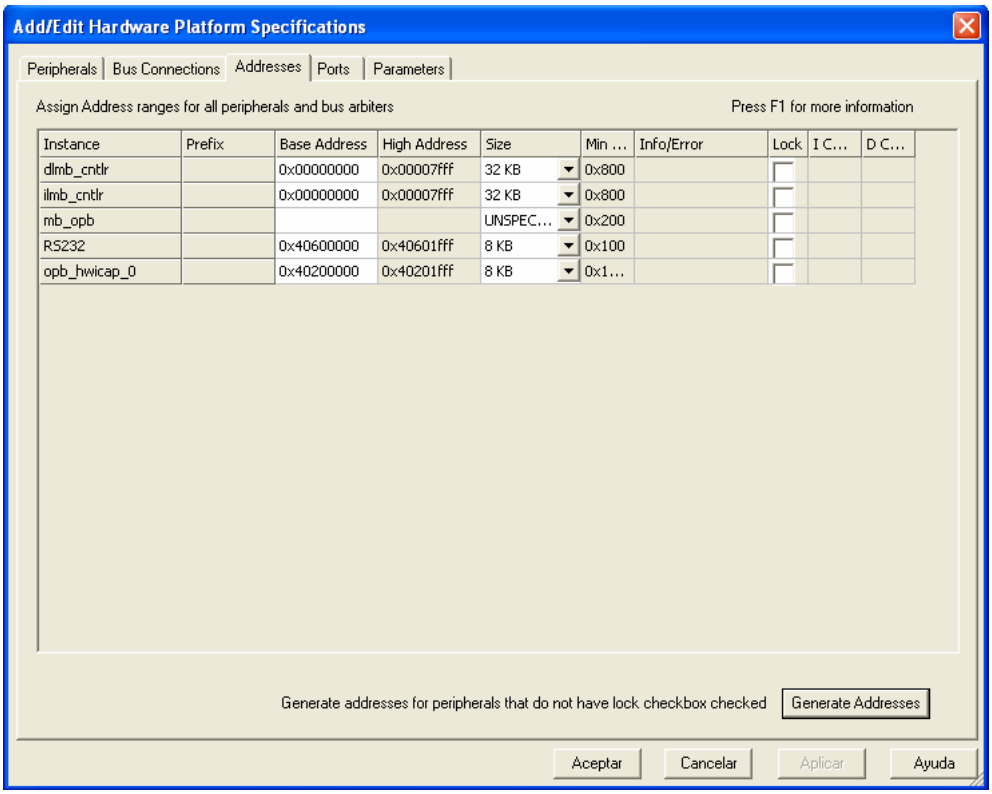
En la ventana que aparece se selecciona de la lista el *opb_hwicap* y se hace clic en *Add*.



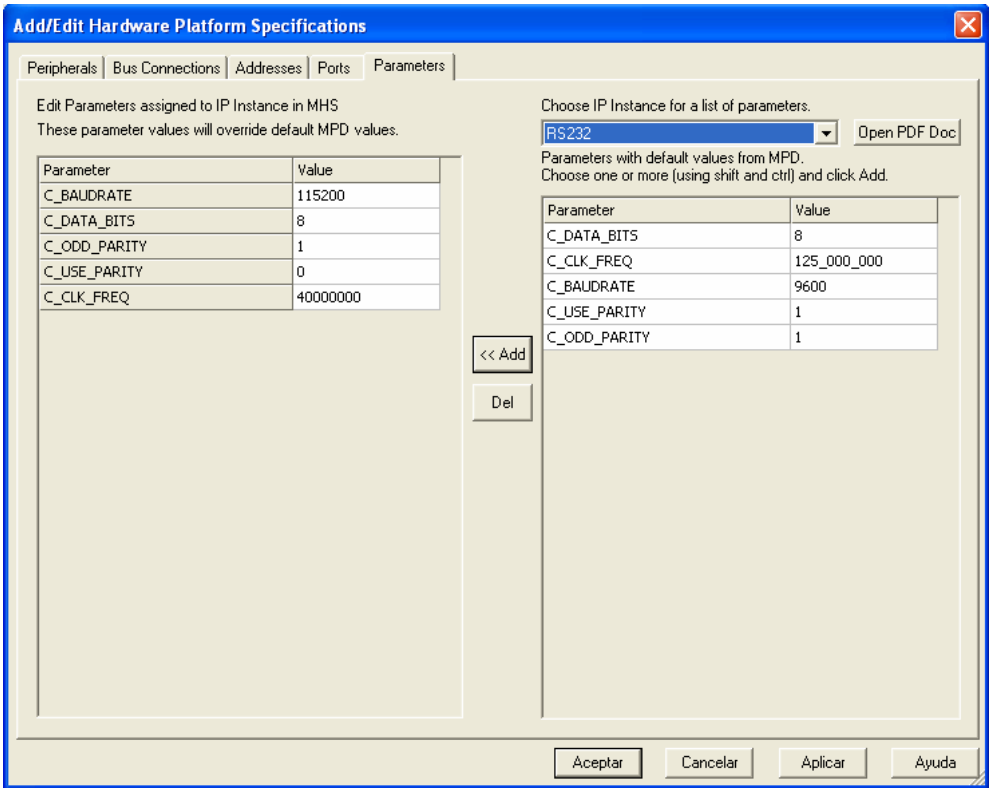
En la pestaña *Bus Connections* se hace clic en la columna *mb_opb* de la fila *opb_hwicap_0 sopb*, lo que hace que se conecte el *hwicap* al bus *opb* como *slave*.



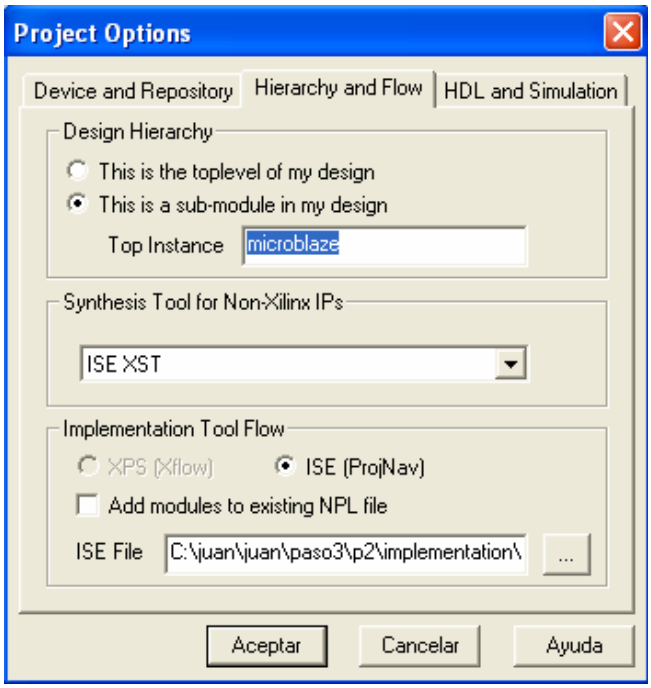
En la pestaña *Addresses* se especifica un tamaño de 8KB en el apartado *Size* del *opb_hwicap_0*, y se hace clic en *Generate Addresses*:



Finalmente en la pestaña *Parameters* se especifican las propiedades del *RS232* y se hace clic en Aceptar.



Se hace clic en el menú *Options / Project Options…* y se accede a la pestaña *Hierarchy and Flow*, donde se selecciona *This is a sub-module in my design*, así como la casilla *ISE (ProjNav)*, guardando el proyecto *ISE* en la carpeta *implementation* del proyecto *EDK*.



A continuación se hace clic en el menú *Tools / Generate Netlist* y cuando finaliza la síntesis, en el menú *Tools / Export to ProjNav*, que genera en la carpeta *implementation* un proyecto *ISE* con el *MicroBlaze*.

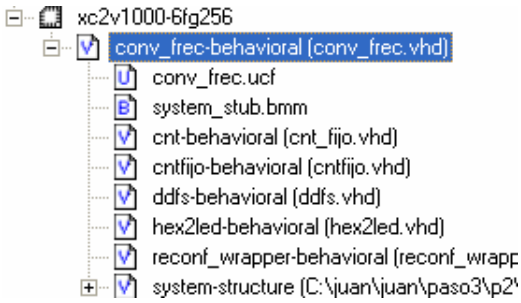
A continuación se abre dicho proyecto con *ISE*, y se sustituye el *top* por el archivo *conv_frec.vhd*, copiando en él la declaración y la instancia del *MicroBlaze*, así como los puertos de entrada y salida, quedando el archivo *conv_frec.vhd* de la siguiente manera:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity conv_frec is
    Port ( reset             : in std_logic;  -- pin:
           clk          : in std_logic; -- pin:
           pulsador     : in std_logic; -- pin:
                     reset_parcial: in std_logic;
                     trien_reconf : in std_logic;
           LED           : out std_logic_vector(6 downto 0); -- pin:
                     LEDDOS       : out std_logic_vector(6 downto 0);
               fpga_0_RS232_req_to_send_pin : out std_logic;
             fpga_0_RS232_RX_pin : in std_logic;
             fpga_0_RS232_TX_pin : out std_logic);

end conv_frec;

architecture Behavioral of conv_frec is

      component DDFS
      Port         (      reset : in std_logic;
                                clk   : in std_logic;
```

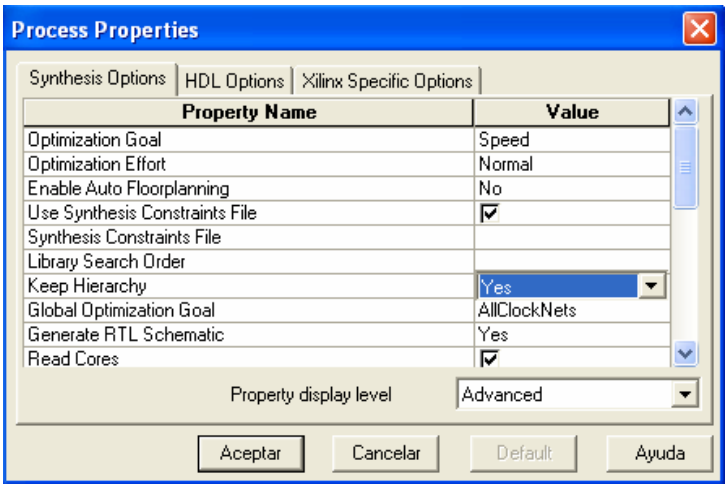
```
                                pulsador : in std_logic;
                                reloj_dividido: out std_logic);
      end component;


    component system is
    port (
      fpga_0_RS232_req_to_send_pin : out std_logic;
      fpga_0_RS232_RX_pin : in std_logic;
      fpga_0_RS232_TX_pin : out std_logic;
      sys_clk_pin : in std_logic;
      sys_rst_pin : in std_logic
    );
  end component;

      component reconf_wrapper
      Port          (--reconf-specific I/O:
                           trien_reconf : in std_logic;
                           reset_parcial: in std_logic;
                           --reset_reconf : out std_logic;
                           rst : out std_logic;
                           --general purpose DataIn:
                           tictac : in std_logic;
                           tictac_reconf: out std_logic;
                           --general purpose DataOut:
                           hex_reconf: in std_logic_vector(3 downto 0);
                             hex : out std_logic_vector(3 downto 0));
      end component;


            --Declaration of user reconfigurable module
      component cnt
      Port          (     reset  : in std_logic;
                                tictac : in std_logic;
                                count  : out std_logic_vector(3 downto 0));
      end component;

            --Declaracion del contador fijo
      component cntfijo
      Port          (     reset  : in std_logic;
                                clk : in std_logic;
                                count  : out std_logic_vector(3 downto 0));
      end component;


      component hex2led
      Port          (     HEX : in std_logic_vector(3 downto 0);
                                LED : out std_logic_vector(6 downto 0));
      end component;

signal tictac : std_logic;
signal hex  : std_logic_vector(3 downto 0);
signal reset_reconf: std_logic;
signal tictac_reconf: std_logic;
signal hex_reconf: std_logic_vector(3 downto 0);
signal hex_fijo: std_logic_vector(3 downto 0);

begin

      -- Divisor de frecuencia para generar el parpadeo
      ddfs1 : ddfs
      port map (
                     reset => reset,
                              pulsador => pulsador,
                              clk => clk,
                           reloj_dividido => tictac
                          );
```

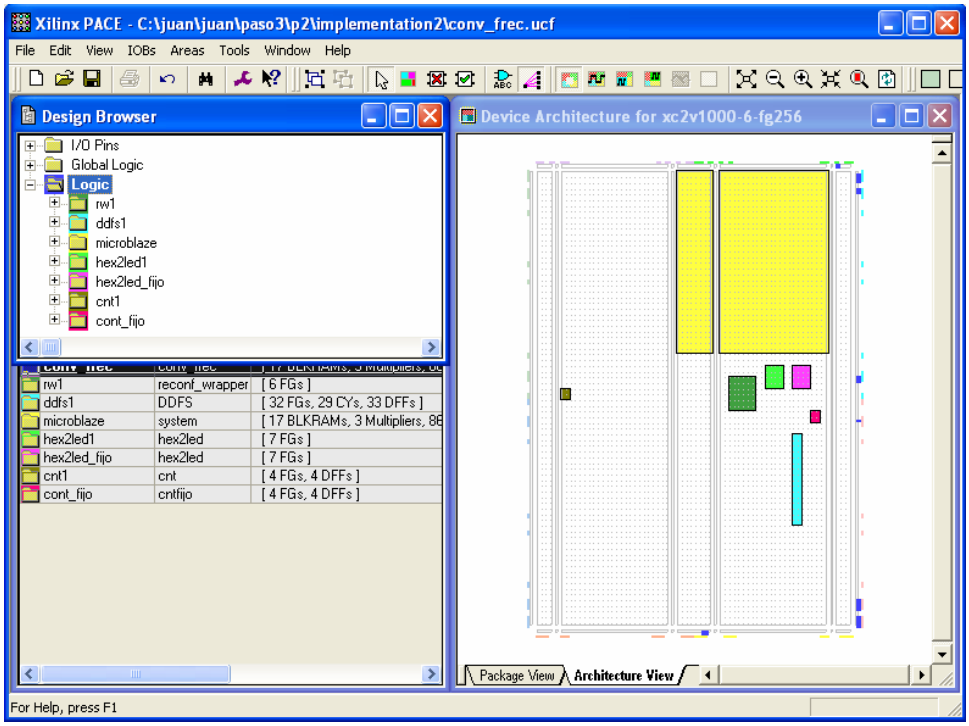
```
       microblaze : system
    port map (
      fpga_0_RS232_req_to_send_pin => fpga_0_RS232_req_to_send_pin,
      fpga_0_RS232_RX_pin => fpga_0_RS232_RX_pin,
      fpga_0_RS232_TX_pin => fpga_0_RS232_TX_pin,
      sys_clk_pin => clk,
      sys_rst_pin => reset
    );

       --Wrapper para los triestados del contador reconfigurable
       rw1 : reconf_wrapper
       port map(--reconf-specific I/O:
                             trien_reconf => trien_reconf,
                           reset_parcial => reset_parcial,
                                rst => reset_reconf,
                           --general purpose DataIn:
                                   tictac => tictac,
                           tictac_reconf => tictac_reconf,
                           --general purpose DataOut:
                                hex_reconf => hex_reconf,
                                       hex => hex
                           );

       --Instantiation of user reconfigurable module:
       -- Contador que se configurará como asc/desc
       -- con reconfiguración dinámica(activa)
       cnt1 : cnt
       port map ( --reconf-specific I/O:
                        reset => reset_reconf,
                              --general purpose DataIn:
                              tictac => tictac_reconf,
                              --general purpose DataOut:
                               count => hex_reconf
                           );

       --convertidor hex2led para mostrar la salida del contador
reconfigurable
       hex2led1 : hex2led
       port map (
                     hex => hex,
                            led => led
                            );

       cont_fijo : cntfijo
       port map (   reset => reset,
                                  clk => clk,
                              count => hex_fijo
                           );

       --convertidor hex2led para mostrar la salida del contador fijo
       hex2led_fijo : hex2led
       port map (
                     hex => hex_fijo,
                            led => leddos
                            );

end Behavioral;
```

También se añade al archivo *conv_frec.ucf* todo lo contenido en el archivo ***.ucf*** del proyecto *EDK*. Además se añaden todos los demás códigos de los submódulos que contenía *conv_frec*, quedando:



Se realiza la implementación como en los proyectos anteriores, seleccionando *Keep Hierarchy: Yes en* las opciones de síntesis, realizando *floorplanning* y se obtiene el *bitstream* completo:





Se cierra el proyecto y se copia la carpeta *implementation* renombrándola a *implementation2*. Se abre el proyecto *ISE* y se hacen los cambios pertinentes en las propiedades del *Map* y el *Place and Route*, tal y como se hizo en el apartado anterior, escogiendo como guía el fichero *conv_frec.ncd* de la carpeta *implementation*.

Se implementa y finalmente se obtiene el *bitstream* parcial entrando en la línea de comandos a la carpeta *implementation2* y ejecutando:

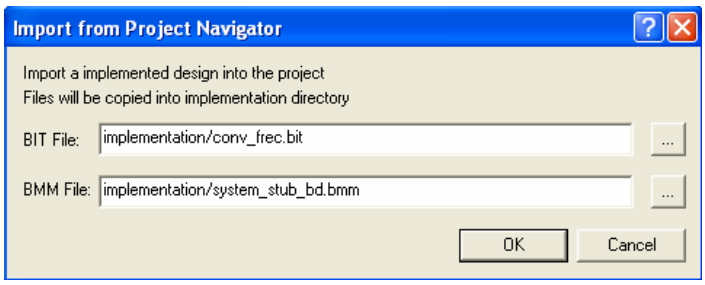
```
bitgen –b -r conv_frec.bit -g ActiveReconfig:Yes conv_frec.ncd conv_frec1a2.bit
```

Al añadir la opción *–b* se generará un fichero de texto *ASCII* en el que viene incluido el *bitstream* parcial en binario. Mediante *Matlab*, y utilizando la técnica que se describe posteriormente en el apartado 5.2.3, se convierte este código binario en un *array* de palabras hexadecimales que se puede incluir en el código *C++* de *MicroBlaze*.

Se obtuvo un *bitstream* parcial de un tamaño excesivo comparado con los obtenidos con anterioridad, lo cual no tenía sentido ya que se añadió únicamente parte estática (parte no reconfigurable que permanecería intacta tras la reconfiguración). Se cree que esto es debido a los fallos del flujo diferencial con proyectos complejos, sobre todo con *hwicap*, así que en lugar de abandonar se decidió realizar una prueba para la obtención de un *bitstream* parcial correcto.

Se utilizó el exactamente el mismo proyecto, con las mismas restricciones *ucf* y *floorplanning*, pero sin *hwicap* en el *MicroBlaze*, obteniendo un *bitstream* en diferencias de 2 *frames*, siendo este el tamaño esperado. Como en la teoría se cumple que en el *bitstream* parcial sólo debe incluir las diferencias de los módulos reconfigurables, y dado que las restricciones y el *floorplanning* del proyecto es exactamente el mismo, se decide utilizar este *bitstream* parcial de dos *frames* para reconfigurar el proyecto anteriormente realizado que incluye *hwicap*.

Se abre el proyecto inicial de *EDK* y se importan los ficheros *conv_frec.bit* y *system_stub_bd.bmm*, haciendo clic en *Tools / Import from ProjNav*.



Luego se selecciona la pestaña *Applications* y ahí en *Sources*, se abre el archivo de código fuente y se incluye lo siguiente:

```
/*FPGAs PARTIAL RECONFIGURATION*/
/*Non-Peripheral partial Self-Reconfiguration*/
/*Difference based flow*/
/*Juan Quero Llor*/

#include <xuartlite_l.h>
```

```
#include "xparameters.h"
#include <xHwIcap.h>
#include "xstatus.h"
#include "xhwicap_i.h"

static Xint32 bitstream [552] = {0xFFFFFFFF,
0xAA995566,
0x30008001,
……
0x30008001,
0x0000000D
};

XHwIcap InstIcap;

int main (void) {
       char option;

       XStatus status;
       Xint32 step, address;
       step=0;
       print("\r\n-- Entering main() --"); //this lines executes but is not seen

       while(1) {
              print("\fFPGAs PARTIAL RECONFIGURATION");
              print("\r\nNon-Peripheral partial self-reconfiguration");
              print("\r\nVirtex 2 (xc2v1000)");
              print("\r\nDifference Based Flow");
              print("\r\n\tJuan Quero Llor");

              print("\r\n\nSelect an option [1-2]:");
              if(step>0)
                     print("\r\n\t 1. Initialize hwicap (DONE)");
              else
                     print("\r\n\t 1. Initialize hwicap.");
              print("\r\n\t 2. Perform Partial Reconfiguration");

              option = (char)inbyte();

              switch(option){
              case '1'://hwicap initialization
                     xil_printf("\r\n\r\nExecuting option %c.\r\n",option);
                     if(step==0){
                            status = XHwIcap_Initialize(&InstIcap,
                                   XPAR_OPB_HWICAP_0_DEVICE_ID,
                                   XHI_READ_DEVICEID_FROM_ICAP);

                            if (status != XST_SUCCESS) {
                                   xil_printf("\r\nXHwICAP Initialization failed. Device ID
read: %x status %x", InstIcap.DeviceIdCode, status);
                                   //exit(1);
                            } else {
                            xil_printf("\r\nXHwICAP Initialization success. Device ID read: %x",
                                          InstIcap.DeviceIdCode);
                            }

                            xil_printf("\r\n Full info received from ICAP: \
                            \r\n\t IsReady: %x \
                            \r\n\t DeviceIdCode: %x \
                            \r\n\t DeviceId: %x \
                            \r\n\t Rows: %x \
                            \r\n\t Cols: %x \
                            \r\n\t BramCols: %x \
                            \r\n\t BytesPerFrame: %x \
                            \r\n\t WordsPerFrame: %x \
                            \r\n\t ClbBlockFrames: %x \
                            \r\n\t BramBlockFrames: %x \
                            \r\n\t BramIntBlockFrames: %x", \
```

```
                        InstIcap.IsReady, InstIcap.DeviceIdCode, InstIcap.DeviceId, \
                        InstIcap.Rows, InstIcap.Cols, InstIcap.BramCols,
InstIcap.BytesPerFrame, \
                        InstIcap.WordsPerFrame, InstIcap.ClbBlockFrames, \
                        InstIcap.BramBlockFrames, InstIcap.BramIntBlockFrames);
                        step++;
                  } else {
                        print("\r\nHwicap Initialization already done");
                  }
                  print("\r\n\r\nPress any key to continue...");       option =
(char)inbyte();
                  break;

            case '2': //FPGA Partial reconfiguration

                  if(step==0)
                  {
                        print("\r\n\r\nPlease initialize Hwicap before(step 1)");
                        print("\r\nPress any key to continue...");
                        option = (char)inbyte();
                        break;
                  }

                  xil_printf("\r\n\r\nExecuting option %c.",option);

                  address=(Xuint32)&bitstream;
                  XHwIcap_SetConfiguration(&InstIcap, address, sizeof(bitstream));

                  if (status != XST_SUCCESS)
                        xil_printf("\r\nFPGA Reconfiguration failed.\r\n\nIMPORTANT: Please
download the initial full bitstream again", InstIcap.DeviceIdCode, status);
                  else
                        xil_printf("\r\nFPGA Reconfiguration success. Device ID read: %x
status %x\r\n", InstIcap.DeviceIdCode, status);
                  step=1;
                  print("\r\nPress any key to continue...");    option = (char)inbyte();
            break;

            default:
                  xil_printf("\r\n The character selected %c (ASCII %d) was not valid
option.",option,option);
                  print("\r\nPress any key to continue...");
                  option = (char)inbyte();

            }
      }

      //this line never executes
      print("\r\n-- Exiting main() --\r\n");
      return 0;
}
```

El método *main()* tiene dos opciones (tecleando números del 1 al 2):

1. Se inicializa el *hwicap*.
2. Se reconfigura, leyendo el *bitstream* parcial del array descargándolo en la placa a través del *hwicap* mediante *XHwIcap_SetConfiguration()*.

Finalmente se compila haciendo clic en *Tools / Build all User Applications* y se descarga con *Tools / Download* (habiendo abierto previamente una ventana de *HyperTerminal* y configurándola tal y como se especifica en el Anexo 3)

```
FPGAs PARTIAL RECONFIGURATION
Non-Peripheral partial self-reconfiguration
Virtex 2 (xc2v1000)
Difference Based Flow
        Juan Quero Llor

Select an option [1-2]:
         1. Initialize hwicap.
         2. Perform Partial Reconfiguration
```

Se inicializa el *hwicap* tecleando la opción 1 y a continuación se activan los *buffers* triestado y el *reset* de la parte reconfigurable mediante los *switchs*, se procede a realizar la auto-reconfiguración parcial tecleando 2, y se desactivan los *switchs*, observando que funciona correctamente, ya que el contador pasa de ser ascendente a descendente tal y como se puede comprobar en el vídeo demostrativo.

No se pudo comprobar si funcionaba correctamente con el *bitstream* parcial obtenido inicialmente (el de mayor número de *frames* de diferencia) ya que no cabía en la memoria *bram* de la *FPGA*.

Se trató de repetir el proyecto en *ISE 8.1* y *EDK 8.1*, por comprobar si se obtenían los 2 *frames* de diferencias, pero se obtuvieron más *frames* aún que con la otra versión. Aún así también funcionó la auto-reconfiguración parcial en diferencias con *ISE 8.1* y *EDK 8.1* utilizando los 2 *frames* del proyecto anterior.

3.2. Flujo en Módulos

Tal y como se explicó en el capítulo anterior, el flujo en módulos surgió a la par que el flujo en diferencias, y consistía en realizar submódulos con la parte reconfigurable, que estuviesen conectados con la parte estática a través de *buses macro*.

El *bitstream* parcial generado no es una diferencia entre los dos *bitstreams* completos, si no que es el *bitstream* únicamente del módulo reconfigurable en cuestión. Se crean por tanto unos módulos intercambiables que cambian esa parte de la *FPGA* si se descarga uno u otro *bitstream*.

Tras conseguir la reconfiguración parcial activa con el flujo en diferencias, así como la auto-reconfiguración parcial activa utilizando otro *bitstream* parcial, se empezó a trabajar en la reconfiguración con el flujo en módulos, buscando conseguir un mayor número de objetivos.

En ese momento *Xilinx* lanzó el primer parche *Early Access Partial Reconfiguration (EA_PR)* para la versión *ISE 8.1*, creando con ello un nuevo flujo de trabajo similar al flujo en módulos, pero con un menor número de restricciones y con mayor probabilidad de que funcionara, por lo que se decidió comenzar a trabajar con el flujo *EA_PR* y abandonar el obsoleto flujo en módulos.

CAPÍTULO 4
Auto-Reconfiguración Parcial en Virtex-4

4.1. Introducción

Debido a los problemas y fallos generados con los flujos diferencial y modular, *Xilinx* sacó un parche llamado *EA_PR (Early Access Partial Reconfiguration),* destinado a los investigadores en esta materia, que no está disponible en la herramienta comercial, creando con ello un nuevo flujo de trabajo.

Se comenzó trabajando en este flujo sobre las placas con ***Virtex-II*** y ***Virtex-II Pro,*** *xc2v1000* y *xc2vp7* respectivamente, obteniendo resultados satisfactorios en la reconfiguración parcial y la auto-reconfiguración parcial de elementos no conectados a *MicroBlaze* en ambas placas.

En el momento en el que salio la versión 5 del *EA_PR* para *ISE 8.2.01i*, que permitía la reconfiguración parcial en ***Virtex-4*** se optó por trabajar con esta placa, ya que se trataba de la última tecnología y el último software disponible.

Es por esta última razón por lo que se ha decidido colocar en primer lugar este capítulo, especificando paso a paso como realizar la auto-reconfiguración parcial de elementos que no sean periféricos, y detrás el capítulo referente al trabajo con las ***Virtex-II***, aclarando en él únicamente las diferencias, los ejemplos realizados y los pasos seguidos para llegar a obtener estos resultados.

En este caso se va a realizar, a modo de ejemplo, un proyecto que va encendiendo los 8 *leds* de la placa hacia la derecha, hacia la izquierda y en ambos sentidos, según el bitstream parcial descargado. Para la reconfiguración se utilizará el *hwicap* y el módulo *debug* de un procesador *MicroBlaze* que se instanciará únicamente para estos cometidos. La parte estática estará compuesta por el microprocesador y por un módulo divisor de frecuencia.

La placa utilizada fue la *Virtex-4 FX Card* de *Nu Horizonst*, *FPGA xc4vfx12*-10sf363 de Xilinx, con el software *ISE 8.2 Service Pack 1, EDK 8.2* y *EA_PR5* para *ISE 8.2.01i.*

El diseño de auto-reconfiguración realizado ha sido el siguiente:

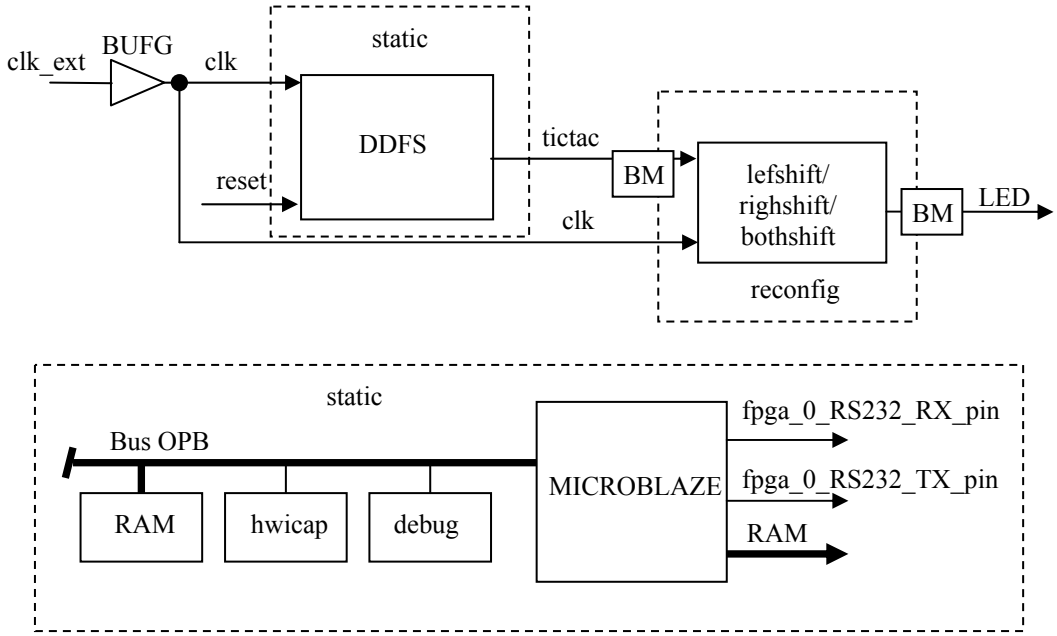


Figura 28. Diagrama de bloques de la Auto-Reconfiguración parcial de no periféricos

4.2. Configuración de la Placa

Es necesario añadir algunas líneas de código para permitir que la memoria *RAM* funcione correctamente en la placa *v4fx12*.

Tras ejecutar el *wizard* en el *EDK* para crear el proyecto tal y como se especifica en el Anexo 2, y añadir y conectar el hwicap, se abre el ***.mhs*** y se añade el siguiente código tras el *dcm_0*:

```
BEGIN dcm_module
 PARAMETER INSTANCE = dcm_1
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_CLK0_BUF = FALSE
 PARAMETER C_CLK90_BUF = TRUE
 PARAMETER C_CLKIN_PERIOD = 10.000000
 PARAMETER C_CLK_FEEDBACK = 1X
 PARAMETER C_EXT_RESET_HIGH = 0
 PORT CLKIN = clk_90_s
 PORT CLK0 = dcm_1_fb
 PORT CLK90 = internal_feedback
 PORT CLKFB = dcm_1_fb
 PORT RST = dcm_0_lock
 PORT LOCKED = dcm_1_lock
END

BEGIN util_vector_logic
 PARAMETER INSTANCE = dcm_1_clk90_inv
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_SIZE = 1
 PARAMETER C_OPERATION = not
 PORT Op1 = internal_feedback
 PORT Res = internal_feedback_n
END
```

A continuación se añade el siguiente código dentro de la instancia de la memoria *RAM*, antes del *end* final:

```
# clk feedback shifted 90
 PORT DDR_Clk90_in = internal_feedback
# not
 PORT DDR_Clk90_in_n = internal_feedback_n
```

También es conveniente, aunque no imprescindible, añadir las siguientes líneas en el fichero ***.ucf***:

```
CONFIG PROHIBIT = C17;
CONFIG PROHIBIT = E19;
CONFIG PROHIBIT = L19;
CONFIG PROHIBIT = H19;
```

4.3. Estructura del Proyecto Reconfigurable:

En primer lugar se procede a la estructuración en carpetas para los distintos pasos de la realización del proyecto:

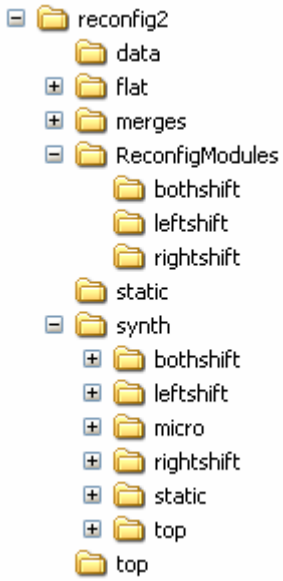


Figura 29. Estructura de carpetas del proyecto reconfigurable

Para ello se crean bajo el directorio principal los siguientes subdirectorios:

- *data*: contiene los archivos ***.vhd***, el ***.ucf*** y los *buses macro*.
- *flat*: en él se crean los proyectos *EDK* necesarios para comprobar que los módulos reconfigurables funcionan.
- *merges*: generación de los *bitstreams* del proyecto.
- *ReconfigModules*: implementación de los módulos reconfigurables
- *static*: implementación de la parte estática del proyecto
- *synth*: síntesis de los módulos reconfigurables, de la parte estática y del conjunto de ambas (*top*)
- *top*: implementación del proyecto

En la carpeta *flat* se proceden a realizar todos aquellos proyectos de *ISE* que sean necesarios para obtener el código *vhdl* del proyecto y los módulos que se quieran realizar, comprobando su correcto funcionamiento independientemente. Una vez hecho esto se copian en la carpeta *data* los archivos ***.vhd*** de los módulos reconfigurables, y de la parte estática (sin contar el *MicroBlaze*), así como el *top*, que se renombra a *top_partial.vhd* y el archivo ***.ucf*** que se renombra a *top_partial.ucf.*

4.4. Síntesis del Proyecto Reconfigurable:

4.4.1. Síntesis de MicroBlaze

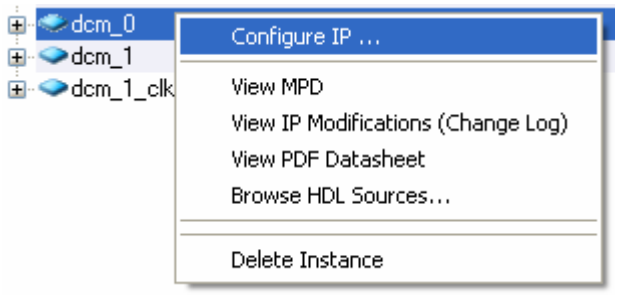
La versión del parche de reconfiguración parcial de *Xilinx EA_PR5* para *ISE 8.2.01i* debe instalarse con el *EDK 8.2* sin ningún *Service Pack* (ya que el *SP1* y el *SP2* del *EDK 8.2* requiere el *SP2* del *ISE 8.2*, que no se puede instalar para el *EA_PR5*). Es en el *Service Pack 2* del *EDK* cuando *Xilinx* añade el *hwicap* válido para las *Virtex 4 y 5*.

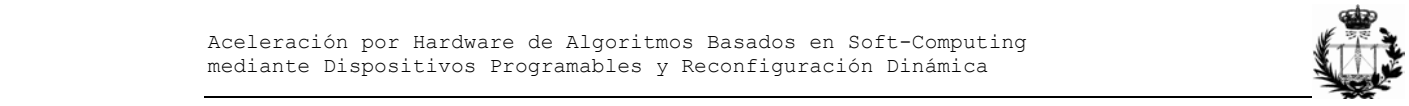
Por lo tanto deberá conseguirse dicho *hwicap* de un ordenador que tenga instalado el *SP2*. Se deben obtener las carpetas *hwicap_v1_00_a* y *opb_hwicap_v1_00_b* de la carpeta de instalación *C:\EDK\hw\XilinxProcessorIPLib\pcores*, así como la carpeta *hwicap_v1_00_a* de la carpeta de instalación *C:\EDK\sw\XilinxProcessorIPLib\drivers*

Estas carpetas obtenidas de un ordenador que tenga instalado el *EDK 8.2 SP2* podrán añadirse de dos maneras: sustituyendo las carpetas del *hwicap* anteriormente citadas en los directorios de instalación del *EDK 8.2* sin *Service Pack*, o incluyéndolos en cada proyecto reconfigurable que se cree. Para esta última opción habría que añadir las carpetas *hwicap_v1_00_a* y *opb_hwicap_v1_00_b* al directorio *pcores* del proyecto *EDK* y la carpeta *hwicap_v1_00_a* al directorio *drivers*.

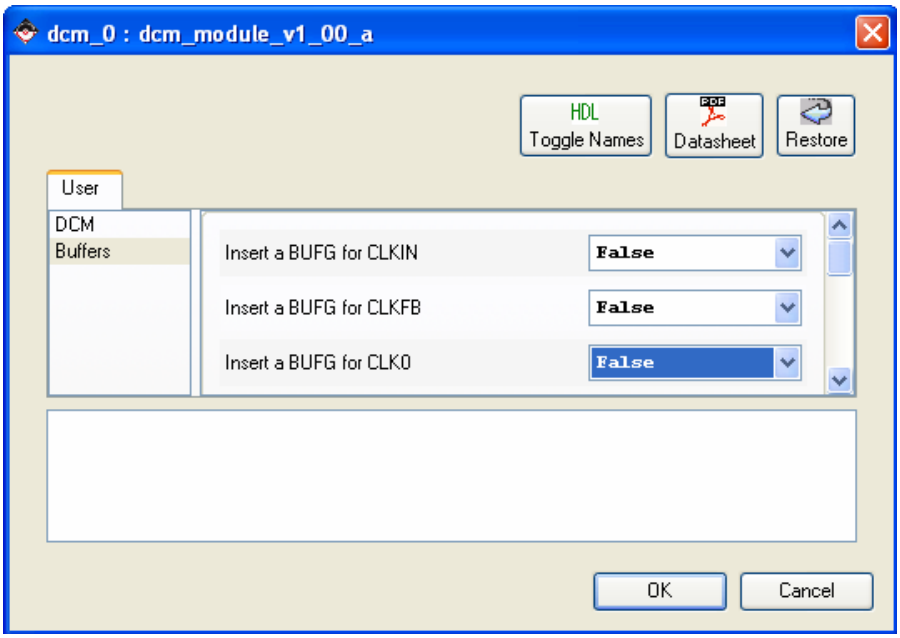
Dentro de la carpeta *synth* se crea una carpeta llamada *micro*, que contenga un proyecto *EDK* con *hwicap*, *debugger hardware* y memoria *RAM*. Para crearlo se puede seguir el Anexo 2.

Se hace clic con el botón derecho encima del módulo *dcm_0* en la pestaña *Bus Interface*, y se selecciona *Configure IP*:



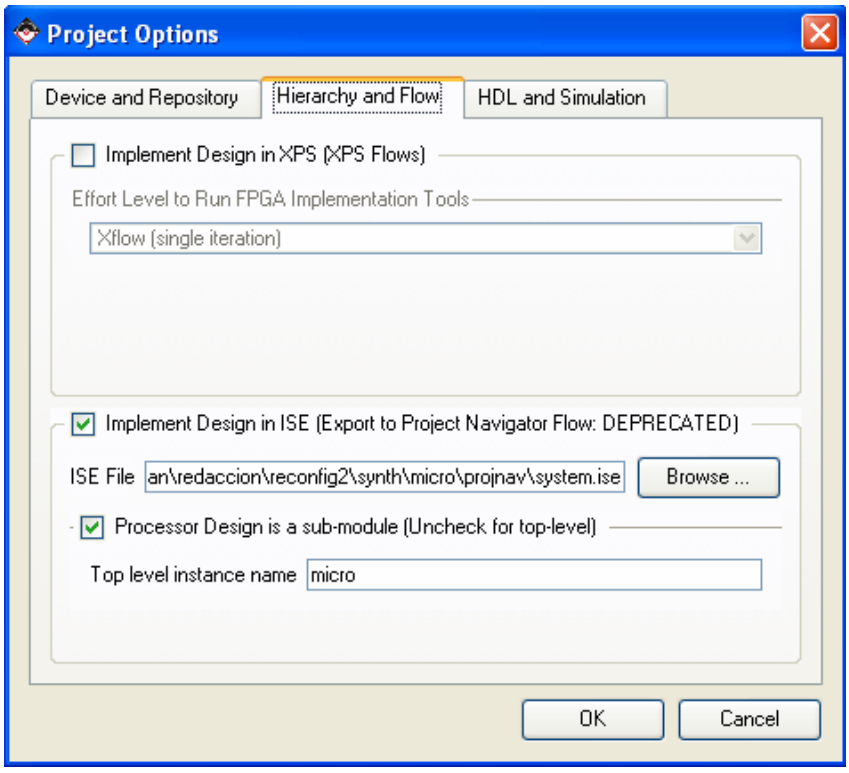


En la pestaña *buffers* se selecciona el valor *False* para el apartado *Insert a BUGF for CLK0*:



Esto se hace así porque el *BUFG* del *dcm_0* debe instanciarse en el *top* del proyecto, tal y como se vio en en Capítulo 2.

En el menú *Project* / *Project Options* se hace clic en la pestaña *Hierachy and Flow* y se seleccionan las casillas *Implement Design in ISE* y *Processor Design is a sub-module*.



A continuación se exporta el proyecto a *ISE,* haciendo clic en *Project Options / Export to Project Navigator* y aceptando la advertencia que aparece. De esta forma se habrá generado un proyecto *ISE* dentro de la carpeta *synth/micro/projnav.*

Se abre el proyecto *ISE*, haciendo doble clic en *synth/micro/projnav/system.ise.* Se hace clic con el botón derecho encima de *system_stub* en la ventana *Sources* y se selecciona *Remove*, para borrar el *top* que ha generado *EDK* y que se sintetizará posteriormente una vez unido al *top* del proyecto completo.

Se accede a la carpeta *synth/micro/vhd* y se copia el archivo *system_stub.vhd* en la carpeta *data*, renombrándolo como *top.vhd*. Este código será la base del *top* del proyecto reconfigurable.

Una vez realizado esto quedará *system* en la ventana *Sources*, es decir, el microprocesador. Se abre su código *vdhl* y se ha de cortar la declaración del componente *IOBUF* así como todas sus instancias y pegarlas en el citado archivo *top.vhd* de la carpeta *data*. Esto es así porque, tal y como se indicó en el Capítulo 2, los *buffers* deben estar situados en el *top* del proyecto.

Esta modificación provoca que sea necesario alterar código de las entradas y salidas del *MicroBlaze* para que no se generen errores.

La principal modificación se produce en la declaración del componente *system*. La declaración automáticamente realizada por *EDK* es la siguiente:

```
entity system is
  port (
    fpga_0_RS232_Uart_RX_pin : in std_logic;
    fpga_0_RS232_Uart_TX_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_Clk_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_Clkn_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_Addr_pin : out std_logic_vector(0 to 12);
    fpga_0_DDR_SDRAM_16Mx32_DDR_BankAddr_pin : out std_logic_vector(0 to 1);
    fpga_0_DDR_SDRAM_16Mx32_DDR_CASn_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_CKE_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_CSn_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_RASn_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_WEn_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_DM_pin : out std_logic_vector(0 to 3);
    fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_pin : inout std_logic_vector(0 to 3);
    fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_pin : inout std_logic_vector(0 to 31);
    sys_clk_pin : in std_logic;
    sys_rst_pin : in std_logic
  );
end system;
```

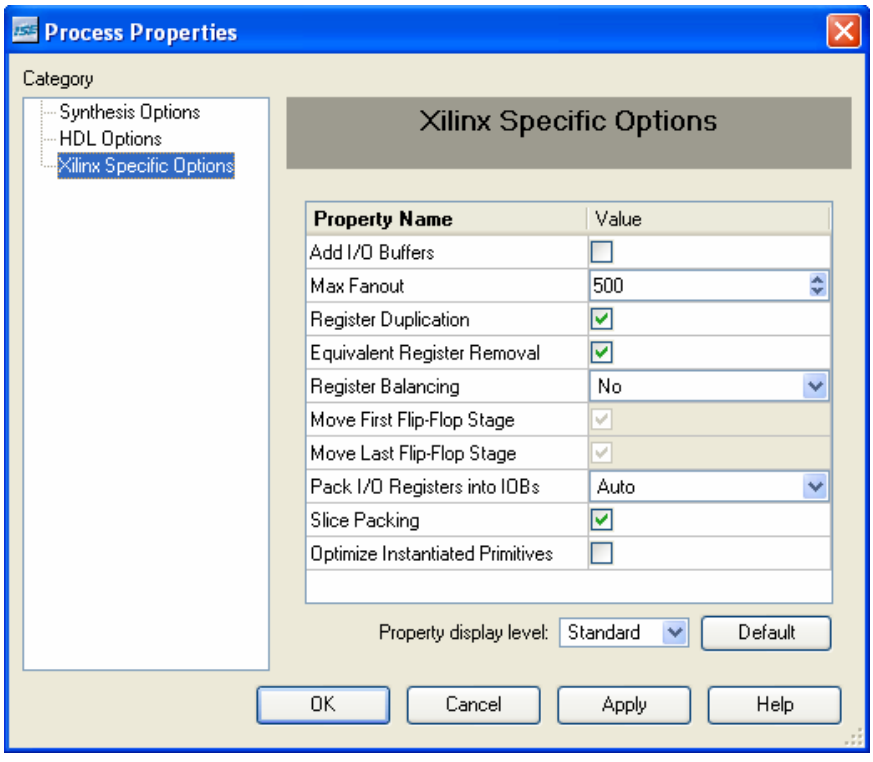
Los pines de entrada/salida (*inout*) *fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_pin* y *fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_pin*, no pueden seguir existiendo debido a que sus señales entrantes y salientes iban directamente a los *buffers IOBUF* (que se han tenido que retirar al *top*) y de ahí a la memoria *RAM.* Ahora tendrán que ir directamente a la *RAM.*

Es por ello que habrá que comentar ambas señales *inout* y crear otras nuevas, con el nombre de las señales que conectaban las entradas y salidas de los *buffers IOBUF* con las entradas y salidas de la memoria *RAM*, de tal forma que se producirá una correcta

conexión, ya que simplemente se habrán extraído los *buffers* al *top*, pero seguirán realizando su papel. De esta forma la declaración quedará:

```
entity system is
  port (
    fpga_0_RS232_Uart_RX_pin : in std_logic;
    fpga_0_RS232_Uart_TX_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_Clk_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_Clkn_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_Addr_pin : out std_logic_vector(0 to 12);
    fpga_0_DDR_SDRAM_16Mx32_DDR_BankAddr_pin : out std_logic_vector(0 to 1);
    fpga_0_DDR_SDRAM_16Mx32_DDR_CASn_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_CKE_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_CSn_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_RASn_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_WEn_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_DM_pin : out std_logic_vector(0 to 3);
  --  fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_pin : inout std_logic_vector(0 to 3);
  --  fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_pin : inout std_logic_vector(0 to 31);
    sys_clk_pin : in std_logic;
    sys_rst_pin : in std_logic;
   fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_I : in std_logic_vector(0 to 3);
   fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_O : out std_logic_vector(0 to 3);
   fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_T : out std_logic_vector(0 to 3);
   fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_I : in std_logic_vector(0 to 31);
   fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_O : out std_logic_vector(0 to 31);
   fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_T : out std_logic_vector(0 to 31)
  );
end system;
```

Para finalizar con la síntesis del *MicroBlaze* se selecciona *system* en la ventana *Sources* y se hace clic con el botón derecho encima de *Synthesize – XST* en la ventana *Processes,* pinchando en *Properties.* En la pestaña *Xilinx Specific Options* se deselecciona la casilla *Add I/O Buffers*:



A continuación se vuelve a hacer clic con el botón derecho en *Synthesize-XST* y se pincha en *Run*, sintetizando con ello el microprocesador. El proyecto *ISE* debe cerrarse, ya que la implementación no se realizará en este entorno.

Si se produce algún error durante la síntesis del microprocesador o en un futuro en la implementación puede ser por:

- Mantener alguna instancia de un *IOBUF* en el *MicroBlaze*.
- No haber quitado las entradas *inout*.
- No haber añadido las entradas y salidas correspondientes por quitar las entradas *inout*, con el nombre de las entradas a la memoria *RAM*.
- Mantener activada la casilla *Add I/O Buffers* de las propiedades de la síntesis.
- Haber seleccionado *Yes* en la casilla *Keep Hierarchy* de las propiedades de la síntesis.

4.4.2. Síntesis del Top

Se crea un proyecto *ISE* en la carpeta *synth/top,* al que se le añade el anteriormente citado archivo *top.vhd* situado en la carpeta *data*, que actualmente sólo contiene una instancia a MicroBlaze. Hay que deseleccionar la casilla *Copy to project*, para que no se generen dos archivos *top.vhd* que puedan producir confusión.

Es la hora de unir este *top.vhd* que contiene la instancia de *MicroBlaze,* con el *top_partial.vhd* del proyecto que se quiere que sea reconfigurable, que previamente se realizó en la carpeta *flat* y se ha copiado en *data*. El *top* final contendrá una parte estática y una reconfigurable, así como la instancia de los *buses macro* y de los *buffers*.

En primer lugar se cambia el nombre de la *entity* a *top* y se añaden las entradas y salidas del archivo *top_partial.vhd.* En este caso bastará con añadir la salida *LED*, que está conectada a los *leds* de la placa. También es necesario cambiar el nombre de las entradas del reloj y el *reset* para que no se conecten automáticamente con el microprocesador, ya que han de pasar primero por los *buffers* correspondientes. La declaración quedará de la siguiente forma:

```
entity top is
    Port (
    fpga_0_RS232_Uart_RX_pin : in std_logic;
    fpga_0_RS232_Uart_TX_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_Clk_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_Clkn_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_Addr_pin : out std_logic_vector(0 to 12);
    fpga_0_DDR_SDRAM_16Mx32_DDR_BankAddr_pin : out std_logic_vector(0 to 1);
    fpga_0_DDR_SDRAM_16Mx32_DDR_CASn_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_CKE_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_CSn_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_RASn_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_WEn_pin : out std_logic;
    fpga_0_DDR_SDRAM_16Mx32_DDR_DM_pin : out std_logic_vector(0 to 3);
    fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_pin : inout std_logic_vector(0 to 3);
    fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_pin : inout std_logic_vector(0 to 31);
    reset_pin    : in std_logic; -- pin:
    clk_ext      : in std_logic; -- pin:
    LED          : out std_logic_vector(7 downto 0)
       );
end top;
```

Es importante modificar las entradas y salidas en la declaración y la instancia del componente *system* (el *MicroBlaze*), debido a los cambios que se han realizado por el hecho de pasar los *IOBUF* al *top*. También cabe comprobar que se han añadido correctamente la declaración y las instancias de los *IOBUF* a este archivo.

```
micro : system
  port map (
    fpga_0_RS232_Uart_RX_pin => fpga_0_RS232_Uart_RX_pin,
    fpga_0_RS232_Uart_TX_pin => fpga_0_RS232_Uart_TX_pin,
    fpga_0_DDR_SDRAM_16Mx32_DDR_Clk_pin => fpga_0_DDR_SDRAM_16Mx32_DDR_Clk_pin,
    fpga_0_DDR_SDRAM_16Mx32_DDR_Clkn_pin => fpga_0_DDR_SDRAM_16Mx32_DDR_Clkn_pin,
    fpga_0_DDR_SDRAM_16Mx32_DDR_Addr_pin => fpga_0_DDR_SDRAM_16Mx32_DDR_Addr_pin,
    fpga_0_DDR_SDRAM_16Mx32_DDR_BankAddr_pin => fpga_0_DDR_SDRAM_16Mx32_DDR_BankAddr_pin,
    fpga_0_DDR_SDRAM_16Mx32_DDR_CASn_pin => fpga_0_DDR_SDRAM_16Mx32_DDR_CASn_pin,
    fpga_0_DDR_SDRAM_16Mx32_DDR_CKE_pin => fpga_0_DDR_SDRAM_16Mx32_DDR_CKE_pin,
    fpga_0_DDR_SDRAM_16Mx32_DDR_CSn_pin => fpga_0_DDR_SDRAM_16Mx32_DDR_CSn_pin,
    fpga_0_DDR_SDRAM_16Mx32_DDR_RASn_pin => fpga_0_DDR_SDRAM_16Mx32_DDR_RASn_pin,
    fpga_0_DDR_SDRAM_16Mx32_DDR_WEn_pin => fpga_0_DDR_SDRAM_16Mx32_DDR_WEn_pin,
    fpga_0_DDR_SDRAM_16Mx32_DDR_DM_pin => fpga_0_DDR_SDRAM_16Mx32_DDR_DM_pin,
--  fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_pin => fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_pin,
--  fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_pin => fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_pin,
    sys_clk_pin => clk,
    sys_rst_pin => reset,
    fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_I => fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_I,
    fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_O => fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_O,
    fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_T => fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_T,
    fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_I => fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_I,
    fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_O => fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_O,
    fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_T => fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_T

  );
```

Habrá que añadir las señales intermedias que conexionarán las entradas y salidas de los *IOBUF* con las entradas y salidas del *MicroBlaze*:

```
  signal fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_I : std_logic_vector(0 to 3);
  signal fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_O : std_logic_vector(0 to 3);
  signal fpga_0_DDR_SDRAM_16Mx32_DDR_DQS_T : std_logic_vector(0 to 3);
  signal fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_I : std_logic_vector(0 to 31);
  signal fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_O : std_logic_vector(0 to 31);
  signal fpga_0_DDR_SDRAM_16Mx32_DDR_DQ_T : std_logic_vector(0 to 31);
```

Es el momento de añadir el *BUFG* que se retiró del *dcm_0* del *MicroBlaze*. Para ello se añade el siguiente código en las declaraciones:

```
 component BUFGP
  port ( I : in std_logic;
         O : out std_logic
       );
      end component;

  component IBUF is
    port (
      I : in std_logic;
      O : out std_logic
    );
  end component;

signal clk : std_logic;
signal reset : std_logic;
```

Y el siguiente código dentro del *begin*:

```
-- global clock buffer for the design
       bufgp_0: BUFGP
  port map (
               I      => clk_ext,
               O      => clk
              );

  ibuf_4 : IBUF
    port map (
      I => reset_pin,
      O => reset
    );
```

Este código conecta las patillas de entrada del reloj y el reset a unos *buffers,* y sus salidas a las señales generales de *clk* y *reset* que van a todos los componentes.

Se incluyen en *top.vhd* las declaraciones de componentes del archivo *top_partial.vhd*, así como las señales intermedias, instancias, conexiones y todo el resto del código contenido en él.

En este caso se añadirán las instancias de la parte estática (el divisor de frecuencia que genera la señal *tictac*) y del módulo reconfigurable *shift* (recibe la señal *tictac* y el reloj y tiene como salida un *array* con los valores que hacen que los *leds* se enciendan hacia la derecha, hacia la izquierda o en ambos sentidos).

```
       static1: static
    Port Map( reset => reset,
              clk => clk,
              tictac => tictac
             );

       --reconfigurable
       shift1 : shift
       port map ( tictac => tictac_reconf,
                  clk=>clk,
                  LED => LED_reconf
                 );
```

Es el momento de conectar la parte reconfigurable (en este caso el módulo *shift*) con la parte estática a través de los *buses macro*. Cabe destacar que el reloj global (*clk*) no debe pasar por ningún *bus macro* aunque esté conectado con la parte reconfigurable.

En primer lugar se añaden las declaraciones de los *buses macro*:

```
component busmacro_xc4v_l2r_async_narrow is
       port (
              input0 : in std_logic;
              input1 : in std_logic;
              input2 : in std_logic;
              input3 : in std_logic;
              input4 : in std_logic;
              input5 : in std_logic;
              input6 : in std_logic;
```

```
              input7 : in std_logic;
              output0 : out std_logic;
              output1 : out std_logic;
              output2 : out std_logic;
              output3 : out std_logic;
              output4 : out std_logic;
              output5 : out std_logic;
              output6 : out std_logic;
              output7 : out std_logic
       );
       end component;

component busmacro_xc4v_r2l_async_narrow is
       port (
              input0 : in std_logic;
              input1 : in std_logic;
              input2 : in std_logic;
              input3 : in std_logic;
              input4 : in std_logic;
              input5 : in std_logic;
              input6 : in std_logic;
              input7 : in std_logic;
              output0 : out std_logic;
              output1 : out std_logic;
              output2 : out std_logic;
              output3 : out std_logic;
              output4 : out std_logic;
              output5 : out std_logic;
              output6 : out std_logic;
              output7 : out std_logic
       );
       end component;
```

A continuación se añaden las señales intermedias que conexionan entradas y salidas con los *buses macro*. Para evitar fallos o confusiones, todo lo que esté conectado con el módulo reconfigurable llevará el apéndice _reconf.

```
signal tictac : std_logic;
signal tictac_reconf: std_logic;
signal LED_reconf: std_logic_vector(7 downto 0);
```

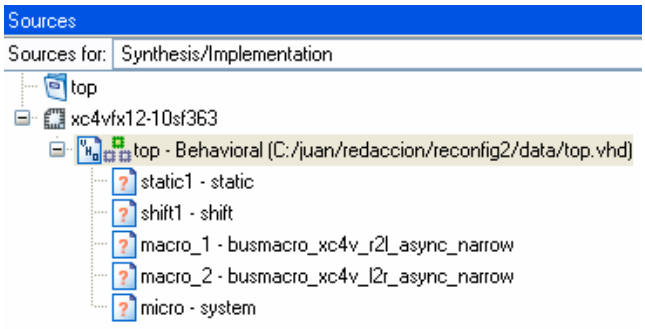
Se han utilizado dos *buses macro l2r* y *r2l*. Estos buses van a ser situados en la pared derecha del módulo reconfigurable, por lo que el *l2r* deberá contener las salidas del módulo reconfigurable (que irán de dentro-izquierda hacia fuera-derecha) y el *r2l* sus entradas (que irán de fuera-derecha hacia dentro-izquierda), tal y como se vio en el apartado 2.3.1.

Las salidas sobrantes de los *buses macro* deberán dejarse en abierto mediante *open*, y las entradas conectadas a 1. Por tanto la instancia de los *buses macro* quedará:

```
-- bus macro to transfer data from the control module to the reconfig_module

      macro_1 : busmacro_xc4v_r2l_async_narrow
    port map (  input0 => tictac,
                input1 => '1',
                input2 => '1',
                input3 => '1',
                input4 => '1',
```

```
                input5 => '1',
                input6 => '1',
                input7 => '1',
               output0 => tictac_reconf,
               output1 => open,
               output2 => open,
               output3 => open,
               output4 => open,
               output5 => open,
               output6 => open,
               output7 => open
             );

-- bus macro to transfer data from the reconfig_module to the control module

    macro_2 : busmacro_xc4v_l2r_async_narrow
    port map (  input0 => LED_reconf(0),
                input1 => LED_reconf(1),
                input2 => LED_reconf(2),
                input3 => LED_reconf(3),
                input4 => LED_reconf(4),
                input5 => LED_reconf(5),
                input6 => LED_reconf(6),
                input7 => LED_reconf(7),
               output0 => LED(0),
               output1 => LED(1),
               output2 => LED(2),
               output3 => LED(3),
               output4 => LED(4),
               output5 => LED(5),
               output6 => LED(6),
               output7 => LED(7)
             );
```

Una vez realizados todos estos cambios, se procede a guardar el archivo *top.vhd*, de tal forma que la ventana *Sources* quedará:



Es importante no añadir los ficheros de código *vhdl* de los componentes del *top* al proyecto (ni siquiera el *MicroBlaze*), ya que se va a sintetizar todo por separado. Esto puede comprobarse porque una vez guardado el fichero *top.vhd*, todos los componentes aparecerán con una interrogación.

Antes de proceder a la síntesis, se debe comprobar que la casilla *Add I/O Buffers* de las propiedades de *Synthesize*-XPS se encuentra activada (con el *top* será la única vez que esta casilla deberá activarse). Para sintetizar, se hace doble clic en *Synthesize-XPS*.

Es normal que se produzcan errores en esta síntesis, ya que se ha de añadir mucho código propio. Para ver donde está el fallo es esencial que primero se haya probado independientemente el proyecto en la carpeta *flat* (sin *MicroBlaze*, e incluso sin *buses macro*), ya que entonces el probable error se habrá producido seguro en estos últimos cambios.

Si se produce algún error durante la síntesis o en un futuro en la implementación o ejecución puede ser por:

- No haber cambiado el nombre de las entradas *clk* y *reset.*
- No haber añadido los *buffers* de las entradas *clk* y *reset*.
- No haber añadido las entradas/salidas del *top_partial.vhd*.
- No haber quitado las señales *inout* de la declaración o instancia del *MicroBlaze*.
- No haber añadido las señales correspondientes a la declaración o instancia del *MicroBlaze* por haber quitado las señales *inout*.
- No haber añadido los buffers *IOBUF* de la memoria *RAM*.
- No haber indicado las señales intermedias que conectan los *buffers IOBUF* con el *MicroBlaze*.
- No haber colocado las instancias de los *buses macro*.
- Haber errado al discernir qué *bus macro* (*r2l* o *l2r*) es necesario para el proyecto y como se deben poner las entradas/salidas del módulo reconfigurable.
- Conectar al revés las entradas o salidas con apéndice *_reconfig*.
- No haber utilizado *open* y *1* para las entradas/salidas no utilizadas de los *buses macro*.
- No mantener activada la casilla *Add I/O Buffers* de las propiedades de la síntesis.
- Haber seleccionado *Yes* en la casilla *Keep Hierarchy* de las propiedades de la síntesis.

4.4.3. Síntesis de la Parte Estática

Aunque el *MicroBlaze* también es parte estática, este apartado se refiere a la parte estática del proyecto inicial sin *MicroBlaze*, en este caso el divisor de frecuencia contenido en *static.vhd* en la carpeta *data*.

Se crea un proyecto *ISE* en la carpeta *synth/static* al que se añade el fichero *data/static.vhd* (sin añadir una copia al proyecto).

Se hace clic en las propiedades de *Synthesize – XPS* y se deselecciona la casilla *Add I/O Buffers*. Finalmente se sintetiza.

Si la parte estática contuviera otros submódulos, también deberían haberse añadido sus ficheros ***.vhd*** al proyecto. Si la parte estática estuviera compuesta por más de un módulo, se procedería de la misma forma para cada uno de ellos (pudiéndose realizar si se prefiere la síntesis dentro de un único proyecto *ISE*, estableciendo cada vez uno de ellos como *Top Module*).

Si se produce algún error en esta parte es debido a que no se realizó y comprobó bien el primer proyecto en la carpeta *flat*, o por no haber deseleccionado la casilla *Add I/O Buffers*.

4.4.4. Síntesis de los Módulos Reconfigurables

Se crea un proyecto *ISE* en la carpeta *synth/leftshift* al que se añade el fichero *data/lefthshift.vhd* (sin añadir una copia al proyecto).

Se hace clic en las propiedades de *Synthesize – XPS* y se deselecciona la casilla *Add I/O Buffers*. Finalmente se sintetiza.

Si se produce algún error en esta parte es debido a que no se realizó y comprobó bien el primer proyecto en la carpeta *flat*, o por no haber deseleccionado la casilla *Add I/O Buffers*.

Estos pasos se repiten para todos los módulos reconfigurables, en este caso *leftshift.vhd*, *rightshift.vhd* y *bothshift.vhd*.

4.5. Implementación del Proyecto Reconfigurable:

Lo primero antes de proceder a la implementación es unir los ficheros *system.ucf* (que proviene del proyecto *EDK* y contiene las restricciones del *MicroBlaze*) y *top_partial.ucf* (que proviene del proyecto inicial realizado en la carpeta *flat*).

Para ello se sitúa el fichero *system.ucf* de la carpeta *synth/micro/data* en la carpeta *data*, renombrándolo a *top.ucf*. Abriéndolos con un editor de textos se copian todas las líneas del archivo *top_partial.ucf* que sean necesarias, evitando repetir el reloj y el reset (hay que recordar que se les había cambiado el nombre para que las señales pasasen por un *buffer* en el *top*).

En este caso únicamente bastará con añadir al fichero generado por *EDK* los *pins* de los *LEDS*, y renombrar el reloj y el reset:

```
NET     "LED<0>" LOC="u2"| drive = 2 | slew = slow | IOstandard=LVCMOS25;
NET     "LED<1>" LOC="u3"| drive = 2 | slew = slow | IOstandard=LVCMOS25;
NET     "LED<2>" LOC="t3"| drive = 2 | slew = slow | IOstandard=LVCMOS25;
NET     "LED<3>" LOC="t4"| drive = 2 | slew = slow | IOstandard=LVCMOS25;
NET     "LED<4>" LOC="r5"| drive = 2 | slew = slow | IOstandard=LVCMOS25;
NET     "LED<5>" LOC="r6"| drive = 2 | slew = slow | IOstandard=LVCMOS25;
NET     "LED<6>" LOC="r3"| drive = 2 | slew = slow | IOstandard=LVCMOS25;
NET     "LED<7>" LOC="r4"| drive = 2 | slew = slow | IOstandard=LVCMOS25;
```

Por último es necesaria la creación de los *Area_Groups* así como localizar los *buses macro* y el *BUFG*.

Se debe crear un *Area_Group* con la parte estática (todos los nombres de las instancias que no son reconfigurables, en este caso *static1* y *micro*). No es necesario localizar este *Area_Group*, ya que se situará en la parte de la *FPGA* en la que no hay zona reconfigurable.

Por otra parte se crea un *Area_Group* con la parte reconfigurable, que se tiene que localizar en un rango que debe cumplir que vaya desde un *slice* con coordenadas x e y

pares hasta otro *slice* con coordenadas x e y impares. Además se tiene que añadir la propiedad *MODE=RECONFIG*, para indicar que ese es el módulo reconfigurable.

Finalmente se localiza el *BUFG* en la posición *BUFGCTRL_X0Y0* y los *buses macro,* de tal forma que su mitad quede partida por la frontera derecha del área reconfigurable.

```
INST "static1" AREA_GROUP = "StaticAreaGrp" ;
INST "micro" AREA_GROUP = "StaticAreaGrp" ;
INST "shift1" AREA_GROUP = "ReconfAreaGrp" ;
AREA_GROUP "ReconfAreaGrp" RANGE = SLICE_X28Y8:SLICE_X37Y23 ;

AREA_GROUP "ReconfAreaGrp" MODE = RECONFIG;
#AREA_GROUP "ReconfAreaGrp" GROUP = CLOSED;

#PACE: Start of PACE I/O Pin Assignments
INST "bufgp_0" LOC = "BUFGCTRL_X0Y0"  ;

#Bus Macro
INST "macro_1" LOC = "SLICE_X36Y16";
INST "macro_2" LOC = "SLICE_X36Y14";
```

4.5.1. Implementación del Top

Se copian en la carpeta *top* los siguientes ficheros:

- *synth/top/top.ngc* (síntesis del *top* realizado con *ISE*)
- *data/top.ucf* (fichero de restricciones)
- *data/*.nmc* (archivos de los *buses macro*)

Y se ejecuta la siguiente sentencia en la línea de comandos, que implementa el *top*:

```
ngdbuild -modular initial -p xc4vfx12-10-sf363 top.ngc
```

4.5.2. Implementación de la Parte Estática

Se copian en la carpeta *static*, los siguientes ficheros:

- *synth/static/*.ngc* (síntesis de la parte estática realizada con *ISE*)
- *data/top.ucf* (fichero de restricciones)
- *data/*.edn* si los hubiera
- *synth/micro/implementation/system_stub.bmm* (espacio para el programa)
- *synth/micro/implementation/*.ngc* (síntesis de módulos de *MicroBlaze*)
- *synth/micro/projnav/*.ngc* (sintesis de *MicroBlaze*)

Se ejecutan las siguientes sentencias en la línea de comandos, que implementan la parte estática, teniendo en cuenta el fichero *system_stub.bmm* (que reserva espacio para luego poder descargar el programa del *MicroBlaze* en el *bitstream*).

```
ngdbuild -modular initial -p xc4vfx12-10-sf363 -bm system_stub.bmm ../top/top.ngo
map top.ngd
par -w top.ncd top_routed.ncd
```

4.5.3. Implementación de los Módulos Reconfigurables

Se copian en la carpeta *ReconfigModules/leftshift*, los siguientes ficheros:

- *synth/lefthift/shift.ngc* (síntesis del módulo reconfigurable *leftshift*)
- *data/top.ucf* (fichero de restricciones)
- *data/*.nmc* (archivos de los *buses macro*)

Se copia de la carpeta *static* el fichero *static.used* (que generó la implementación de la parte estática), que se renombra a *arcs.exclude.*

Se ejecutan las siguientes sentencias en la línea de comandos, que implementan el módulo reconfigurable *lefthshift*:

```
ngdbuild -modular module -p xc4vfx12-10-sf363 -active shift ../../top/top.ngo
map top.ngd
par -w top.ncd top_routed.ncd
```

A continuación se realizan exactamente los mismos pasos para el resto de módulos reconfigurables, cada uno en su carpeta dentro de *ReconfigModules*.

4.5.4. Implementación Mediante Script

Se puede simplificar todo el proceso de implementación del proyecto mediante un *script* que automatice todos los pasos anteriormente descritos.

En este caso, se ejecutaría el siguiente *script* en una ventana de comandos, una vez situados en la carpeta del proyecto y si se han seguido las indicaciones descritas con anterioridad para la estructura de los directorios:

```
echo IMPLEMENTACION DEL TOP

cd top
del /Q *.*

copy ..\synth\top\top.ngc
copy ..\data\top.ucf
copy ..\data\*.nmc

ngdbuild -modular initial -p xc4vfx12-10-sf363 top.ngc

cd..


ECHO IMPLEMENTACION DEL STATIC

cd static
del /Q *.*

copy ..\synth\static\*.ngc
copy ..\data\top.ucf
copy ..\data\*.edn
copy ..\synth\micro\implementation\system_stub.bmm
copy ..\synth\micro\implementation\*.ngc
copy ..\synth\micro\projnav\*.ngc
```

```
ngdbuild -modular initial -p xc4vfx12-10-sf363 -bm system_stub.bmm
../top/top.ngo
map top.ngd
par -w top.ncd top_routed.ncd

cd..


ECHO IMPLEMENTACION DEL leftshift

cd ReconfigModules\leftshift
del /Q *.*

copy ..\..\synth\leftshift\shift.ngc
copy ..\..\data\top.ucf
copy ..\..\data\*.nmc
copy ..\..\static\static.used
ren static.used arcs.exclude

ngdbuild -modular module -p xc4vfx12-10-sf363 -active shift ../../top/top.ngo
map top.ngd
par -w top.ncd top_routed.ncd


cd..

ECHO IMPLEMENTACION DEL rightshift

cd rightshift
del /Q *.*

copy ..\..\synth\rightshift\shift.ngc
copy ..\..\data\top.ucf
copy ..\..\data\*.nmc
copy ..\..\static\static.used
ren static.used arcs.exclude

ngdbuild -modular module -p xc4vfx12-10-sf363 -active shift ../../top/top.ngo
map top.ngd
par -w top.ncd top_routed.ncd

cd..


ECHO IMPLEMENTACION DEL BOTHSHIFT

cd bothshift
del /Q *.*

copy ..\..\synth\bothshift\shift.ngc
copy ..\..\data\top.ucf
copy ..\..\data\*.nmc
copy ..\..\static\static.used
ren static.used arcs.exclude

ngdbuild -modular module -p xc4vfx12-10-sf363 -active shift ../../top/top.ngo
map top.ngd
par -w top.ncd top_routed.ncd

cd..
cd..
```

4.6. Generación del Bitstream:

Para generar los *bitstreams* tanto parciales como finales, se copian en la carpeta *merges* los siguientes ficheros:

- *static/top_routed.ncd* renombrándolo a *static.ncd*
- *ReconfigModules/*/top_routed.ncd* renombrándolos a **.ncd*
- *static/system_stub.bmm*

El *script* que lo realizaría sería:

```
ECHO UNIENDO LOS MODULOS

cd merges
del /Q *.*

copy ..\static\top_routed.ncd static.ncd
copy ..\ReconfigModules\leftshift\top_routed.ncd leftshift.ncd
copy ..\ReconfigModules\rightshift\top_routed.ncd rightshift.ncd
copy ..\ReconfigModules\bothshift\top_routed.ncd bothshift.ncd
copy ..\static\system_stub.bmm
```

Se ejecutan los siguientes *scripts* del *EA_PR* en la línea de comandos (primero uno y después otro, ya que al ser *scripts* se detiene la ejecución al finalizar el primero):

```
PR_assemble.bat static.ncd rightshift.ncd

PR_verifydesign.bat static.ncd leftshift.ncd rightshift.ncd bothshift.ncd
```

De esta forma se habrán generado los siguientes *bitstreams* en la carpeta *merges*:

- *static_full.bit* (*bitstream* completo con *leftshift*)
- *rightshift_partial.bit* (*bitstream* parcial de *rightshift*)
- *leftshift_partial.bit* (*bitstream* parcial de *leftshift*)
- *bothshift_partial.bit* (*bitstream* parcial de *bothshift*)
- *reconfareagrp_blank.bit* (*bitstream* parcial en blanco)

4.7. Carga del Programa de Reconfiguración en el Bitstream

El código *C++* del proyecto reconfigurable podrá crearse y compilarse en cualquier proyecto *EDK* que se haya realizado en la carpeta *flat*, aunque es mejor utilizar el proyecto situado en la carpeta *synth/micro*.

En la pestaña *Applications* de la ventana *Project Information Area* se encuentra el archivo de código principal (*TestApp_Peripheral.c*) que se utiliza para la reconfiguración con *hwicap*:

```
/*FPGAs PARTIAL RECONFIGURATION*/
/*Non-Peripheral partial reconfiguration*/
/*Juan Quero Llor*/

#include <xuartlite_l.h>
#include "xparameters.h"

#include <xHwIcap.h>
#include "xstatus.h"
#include "xhwicap_i.h"

#define MEM_REG (*(unsigned*)MEM_ADDR)
unsigned MEM_ADDR;
XHwIcap InstIcap;

unsigned RAM_start_addr=0x24000000;
unsigned RAM_end_addr=0x24100000;

int main (void) {
	char option;

	XStatus status;
	Xint32 step, words;
	step=0;
	unsigned start_addr, end_addr;
	print("\r\n-- Entering main() --"); //this lines executes but is not seen

	while(1) {
		print("\fFPGAs PARTIAL RECONFIGURATION");
		print("\r\nNon-Peripheral partial reconfiguration");
		print("\r\nVirtex 4 (xc4vfx12)");
		print("\r\n\tJuan Quero Llor");

		print("\r\n\nSelect an option [1-3]:");
		if(step>0)
			print("\r\n\t 1. Initialize hwicap (DONE)");
		else
			print("\r\n\t 1. Initialize hwicap.");
		print("\r\n\t 2. New partial bitstream downloaded to system RAM");
		print("\r\n\t 3. Perform Partial Reconfiguration");

		option = (char)inbyte();

		switch(option){
		case '1'://hwicap initialization
			xil_printf("\r\n\r\nExecuting option %c.\r\n",option);
			if(step==0){
				status = XHwIcap_Initialize(&InstIcap,
					XPAR_OPB_HWICAP_0_DEVICE_ID,
					XHI_XC4VFX12);
```

```
                        if (status != XST_SUCCESS) {
                              xil_printf("\r\nXHwICAP Initialization failed. Device ID
read: %x status %x",
                                                      InstIcap.DeviceIdCode, status);
                              //exit(1);
                        } else {
                        xil_printf("\r\nXHwICAP Initialization success. Device ID read: %x",
                                          InstIcap.DeviceIdCode);
                        }

                        xil_printf("\r\n Full info received from ICAP: \
                                          \r\n\t IsReady: %x \
                                          \r\n\t DeviceIdCode: %x \
                                          \r\n\t DeviceId: %x \
                                          \r\n\t Rows: %x \
                                          \r\n\t Cols: %x \
                                          \r\n\t BramCols: %x \
                                          \r\n\t BytesPerFrame: %x \
                                          \r\n\t WordsPerFrame: %x \
                                          \r\n\t ClbBlockFrames: %x \
                                          \r\n\t BramBlockFrames: %x \
                                          \r\n\t BramIntBlockFrames: %x", \
                                          InstIcap.IsReady, InstIcap.DeviceIdCode,
InstIcap.DeviceId, \
                                          InstIcap.Rows, InstIcap.Cols, InstIcap.BramCols,
InstIcap.BytesPerFrame, \
                                          InstIcap.WordsPerFrame, InstIcap.ClbBlockFrames,
\
                                          InstIcap.BramBlockFrames,
InstIcap.BramIntBlockFrames);
                        step++;
                  } else {
                        print("\r\nHwicap Initialization already done");
                  }
                  print("\r\n\r\nPlease download partial bitstream with XMD and select
2\r\nPress any key to continue...");    option = (char)inbyte();
                  break;

            case '2':    //Recognize new partial bitstream downloaded to system RAM
                  if(step==0)
                  {
                        print("\r\n\r\nPlease initialize Hwicap before (step 1)");
                        print("\r\nPress any key to continue...");    option =
(char)inbyte();
                        break;
                  }

                  xil_printf("\r\n\r\nExecuting option %c.\r\n",option);

                  start_addr=RAM_start_addr;
                  end_addr=RAM_start_addr;
                  MEM_ADDR = start_addr;

                  while (MEM_REG!=0x0000000D && MEM_ADDR<RAM_end_addr){
                        MEM_ADDR=MEM_ADDR+4;
                        end_addr=MEM_ADDR;
                  }
                  if(MEM_ADDR>=RAM_end_addr){
                        print("\r\n\r\nThere is not a valid partial bitstream on RAM memory,
please download it with XMD");
                        print("\r\nPress any key to continue...");    option =
(char)inbyte();
                        break;
                  }
                  words=0;
                  for (MEM_ADDR = start_addr; MEM_ADDR <= end_addr; MEM_ADDR += 4) {
                        //xil_printf("value(0x%X)=%8X \r\n",MEM_ADDR,MEM_REG);
                        if(MEM_ADDR == start_addr || MEM_ADDR == end_addr)
```

```
                                    xil_printf("value(0x%X)=%8X \r\n",MEM_ADDR,MEM_REG);
                              words++;
                        }
                        step=2;
                        print("\r\nPartial Bitstream is ready.\r\nPress any key to continue...");
      option = (char)inbyte();
                        break;

                case '3': //FPGA Partial reconfiguration

                        if(step==0)
                        {
                              print("\r\n\r\nPlease initialize Hwicap before(step 1)");
                              print("\r\nPress any key to continue...");    option =
(char)inbyte();
                              break;
                        }
                        if(step==1)
                        {
                              print("\r\n\r\nPlease download partial bitstream with XMD (step
2)");
                              print("\r\nPress any key to continue...");    option =
(char)inbyte();
                              break;
                        }
                        xil_printf("\r\n\r\nExecuting option %c.",option);

                        status=XHwIcap_SetConfiguration(&InstIcap, start_addr, words);
                        if (status != XST_SUCCESS)
                              xil_printf("\r\nFPGA Reconfiguration failed.\r\n\nIMPORTANT: Please
download the initial full bitstream again",
                                                InstIcap.DeviceIdCode, status);
                        else
                              xil_printf("\r\nFPGA Reconfiguration success. Device ID read: %x
status %x\r\n",
                                                InstIcap.DeviceIdCode, status);
                        step=1;
                        print("\r\nPress any key to continue...");    option = (char)inbyte();
                break;

                default:
                        xil_printf("\r\n The character selected %c (ASCII %d) was not valid
option.",option,option);
                        print("\r\nPress any key to continue...");
                        option = (char)inbyte();

                }
        }

        //this line never executes
        print("\r\n-- Exiting main() --\r\n");
        return 0;
}
```

El método *main()* tiene tres opciones (tecleando números del 1 al 3):

3. Se inicializa el *hwicap*.
4. Una vez cargado el *bitstream* en la *RAM* con el *debugger*, se reconoce el inicio y el final del mismo.
5. Se reconfigura, leyendo el *bitstream* parcial de la memoria *RAM* (es importante haberlo cargado antes con el depurador) y descargándolo en la placa a través del *hwicap* mediante *XHwIcap_SetConfiguration()*.

Se procede a su compilación haciendo clic en el menú *Software* / *Build all User Applications*.

Se copia el archivo ya compilado (*executable.elf*) situado en la carpeta *synth/micro/TestApp_Peripheral*, en la carpeta merges y se ejecuta la siguiente línea de comandos, que se encarga de generar un archivo *download.bit* que ya lleva incluido el programa:

```
data2mem -bm system_stub_bd.bmm -bt static_full.bit -bd executable.elf -o b download.bit
```

Finalmente ya se puede descargar el *bitstream* en la *FPGA* mediante *iMPACT*.

Antes de reconfigurar, los *leds* de la *FPGA* se mueven hacia la derecha:

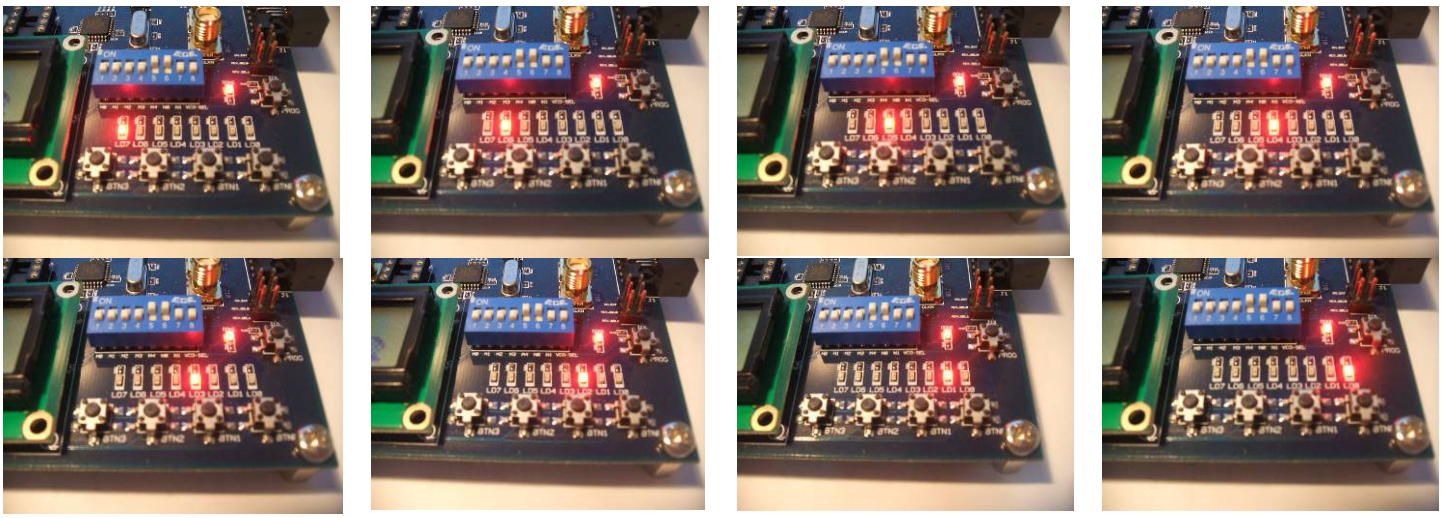


Figura 30. Placa *Virtex-4* con *bitstream* parcial *rightshift*

4.8. Ejecución del Programa de Auto-Reconfiguración Parcial

Se abre una ventana de *HyperTerminal* según el Anexo 3 y una vez descargado el *bitstream* aparece el menú:

```
FPGAs PARTIAL RECONFIGURATION
Non-Peripheral partial self-reconfiguration
Virtex 4 (xc4vfx12)
        Juan Quero Llor

Select an option [1-3]:
         1. Initialize hwicap.
         2. New partial bitstream downloaded to system RAM
         3. Perform Partial Reconfiguration
```

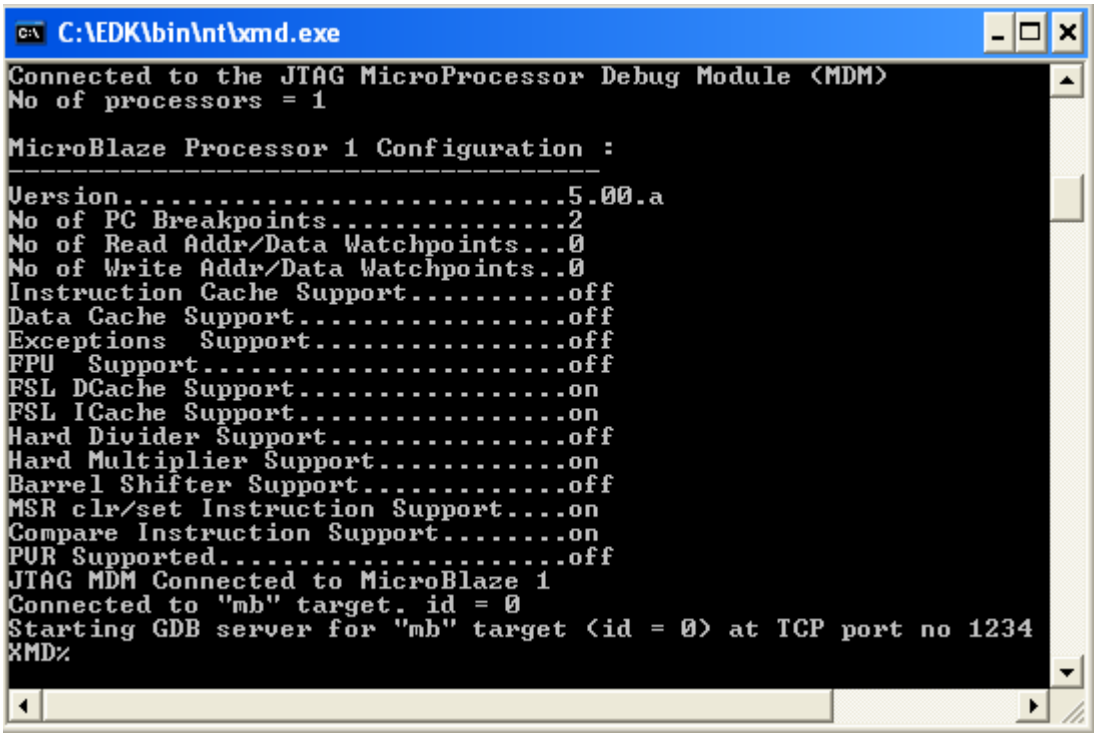
Seleccionando la primera opción se inicializa el *hwicap:*

```
Executing option 1.

XHwICAP Initialization success. Device ID read: 1E58093
 Full info received from ICAP:
        IsReady: 11111111
        DeviceIdCode: 1E58093
        DeviceId: 0
        Rows: 40
        Cols: 18
        BramCols: 3
        BytesPerFrame: A4
        WordsPerFrame: 29
        ClbBlockFrames: 248
        BramBlockFrames: C0
        BramIntBlockFrames: 42

Please download partial bitstream with XMD and select 2
Press any key to continue...
```

A continuación para descargar el *bitstream* parcial en la memoria *RAM*, es necesario ejecutar el depurador hardware en el *EDK*. En primer lugar se han de copiar los bitstreams parciales de la carpeta *merges* a la carpeta *synth/micro,* para poder descargarlos en la placa.

A continuación se establecen las opciones del depurador, para lo que hace clic en *Debug* /  *XMD Debug Options* y se marcan las opciones del cable por el que se va a depurar el microprocesador (en este caso el cable *USB* por el que se programa), pinchando finalmente en *Save*.

Para ejecutar el depurador *XMD* se hace clic en el menú *Debug* /  *Launch XMD...*



Se ejecuta la sentencia para descargar un archivo en una dirección del *MicroBlaze*: *dow –data bothshift_partial.bit 0x24000000*, donde *bothshift_partial.bit* es el *bitstream* parcial, y 0x24000000 es la dirección donde empieza la memoria *RAM*, lugar en el que el programa reconfigurable empieza a leer el *bitstream* para enviarlo al *ICAP*.

```
XMD% dow -data bothshift_partial.bit 0x24000000
XMD%
```

Se puede observar que el depurador *XMD* tarda unos segundos para la descarga, antes de aparecer la siguiente línea *XMD%*. Si no tarda quiere decir que no se está cargando correctamente el archivo en memoria *RAM*, posiblemente por un fallo de ésta, porque no se ha situado el *bitstream* parcial en la carpeta *synth/micro* o porque ha habido alguna errata al escribir el nombre del *bitstream.*

A continuación es necesario ejecutar la sentencia *"con",* para que el programa continúe ejecutándose:

```
XMD% con
Processor started. Type "stop" to stop processor
RUNNING> _
```

Volviendo al *HyperTerminal*, se ejecuta la opción 2 para comprobar que el *bitstream* parcial ha sido subido correctamente a la memoria, y se ha encontrado su principio y su final:

```
FPGAs PARTIAL RECONFIGURATION
Non-Peripheral partial self-reconfiguration
Virtex 4 (xc4vfx12)
        Juan Quero Llor

Select an option [1-3]:
         1. Initialize hwicap (DONE)
         2. New partial bitstream downloaded to system RAM
         3. Perform Partial Reconfiguration

Executing option 2.
value(0x24000000)=   90FF0
value(0x240040E8)=       D

Partial Bitstream is ready.
Press any key to continue...
```

Finalmente ya se puede realizar la auto-reconfiguración parcial de la *FPGA* pulsando en 3:

```
FPGAs PARTIAL RECONFIGURATION
Non-Peripheral partial self-reconfiguration
Virtex 4 (xc4vfx12)
        Juan Quero Llor

Select an option [1-3]:
         1. Initialize hwicap (DONE)
         2. New partial bitstream downloaded to system RAM
         3. Perform Partial Reconfiguration

Executing option 3.
FPGA Reconfiguration success. Device ID read: 1E58093 status 0

Press any key to continue...
```

Una vez realizada la reconfiguración, los *LEDS* de la placa se encienden en ambos sentidos, sin haber dejado ésta de funcionar (el led *DONE* sigue activo durante la reconfiguración):

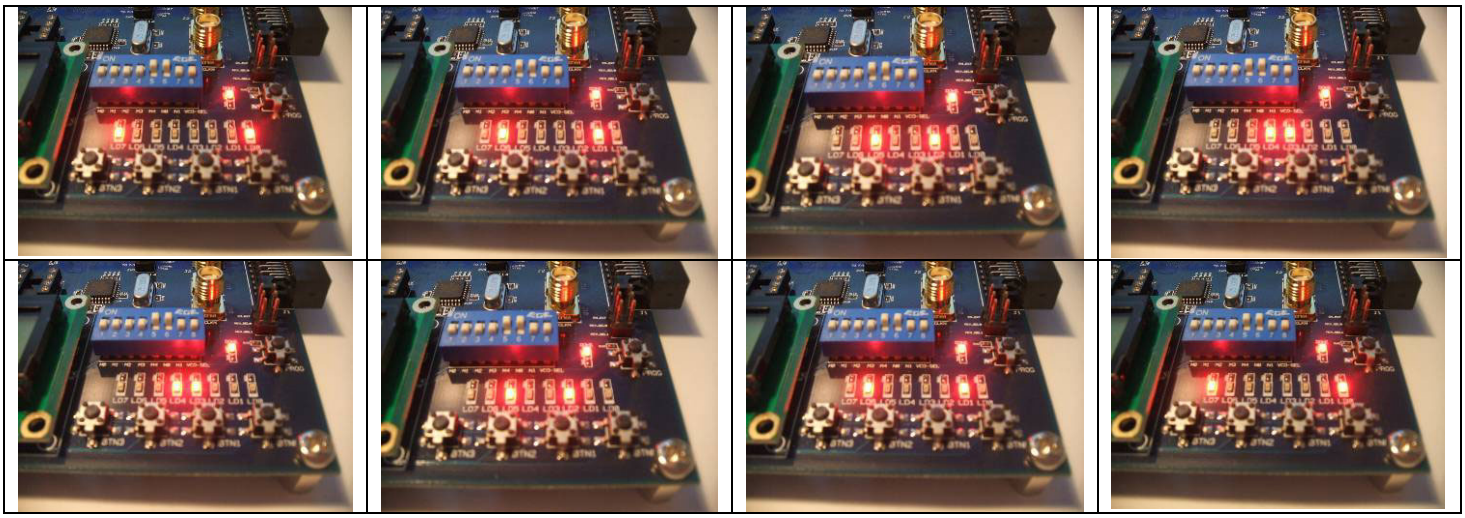


Figura 31. Placa *Virtex-4* con *bitstream* parcial *bothshift* tras la auto-reconfiguración

4.9. Reconfiguración Parcial

Si en lugar de auto-reconfiguración parcial se desea realizar simplemente reconfiguración parcial, es decir, que la *FPGA* no se reconfigure a ella misma con *MicroBlaze*, sino que se reconfigure parcialmente desde el exterior (con *iMPACT*), el proceso a seguir es exactamente el mismo al realizado con anterioridad, sólo que no es necesario incluir el *MicroBlaze*, ni trabajar con *EDK*.

Los pasos serían los mismos: se realizan los proyectos previos en *flat* comprobando que funcionen por separado, se crea el *top* en el que se instancian los *buses macro* (esta vez no es necesario unirlo al *top* de *MicroBlaze*) y el fichero ***.ucf*** y se ejecuta el proceso de síntesis, implementación y generación del *bitstream* parcial a partir del *script* anterior.

Para finalizar, una vez se esté descargado el *bitstream* completo en la placa mediante *iMPACT* , se procede a descargar a través del mismo programa el *bitstream* parcial. Es importante saber que sólo se podrá descargar un *bitstream* parcial desde una instalación de *Xilinx* que contenga el parche *EA_PR*.

La reconfiguración parcial también puede realizarse perfectamente con el ejemplo anterior pero utilizando *iMPACT* para descargar el *bitstream* parcial, el *MicroBlaze* seguirá funcionando y los *leds* cambiarán de sentido, habiéndose realizado reconfiguración parcial en lugar de auto-reconfiguración parcial.

CAPÍTULO 5
Auto-Reconfiguración Parcial en Virtex-II y Virtex-II Pro

5.1. Introducción

Aunque en la presente memoria se ha estimado conveniente explicar detalladamente la creación de un proyecto auto-reconfigurable con las herramientas EA_PR sobre ***Virtex-4*** y con *ISE* y *EDK 8.2* (se trataba de la última tecnología disponible), durante la realización del proyecto se llevó a cabo en primer lugar sobre ***Virtex-II*** y ***Virtex-II pro*** y con *ISE* y *EDK 8.1*, obteniendo también resultados satisfactorios.

Es por ello que a continuación se explican brevemente los ejemplos realizados y las diferencias encontradas cuando se trabajó con las placas con *FPGAs* ***Virtex-II*** *xc2v1000* y ***Virtex-II pro*** *xc2vp7*.

5.2. Auto-Reconfiguración Parcial en Virtex-II xc2v1000

5.2.1. Introducción

La ***Virtex-II*** *xc2v1000* fue la primera placa en la que se trabajó con el parche *EA_PR* de Xilinx para reconfiguración parcial.

Se trató de conseguir mediante el flujo de reconfiguración parcial por módulos y el nuevo flujo *EA_PR* resultados similares o mejores a los obtenidos mediante el flujo de reconfiguración basado en diferencias.

Para ello se partió del mismo ejemplo que se había utilizado antes, explicado en el Capítulo 3, una parte estática compuesta por un divisor de frecuencia que dividiese el reloj y lo hiciera visible al ojo humano, así como un decodificador hexadecimal-7 segmentos, y una parte reconfigurable que va incrementando o decrementando un contador según este reconfigurado o no el dispositivo.

La placa utilizada fue la *Virtex-II Evaluation Board* de *Avnet*, *FPGA xc2v1000-4fg256* de Xilinx, con el software *ISE 8.1 Service Pack 1, EDK 8.1* y *EA_PR v8* para *ISE 8.1.01i.*

Se consiguió tanto la reconfiguración parcial como la auto-reconfiguración parcial en esta placa.

5.2.2. Reconfiguración Parcial

El proyecto ejemplo de Reconfiguración Parcial con *xc2v1000* se encuentra en el cd en la carpeta *xc2v1000/reconfig*. El vídeo demostrativo se titula “Reconfiguración parcial xc2v1000”.

El esquema del diseño reconfigurable es el siguiente:

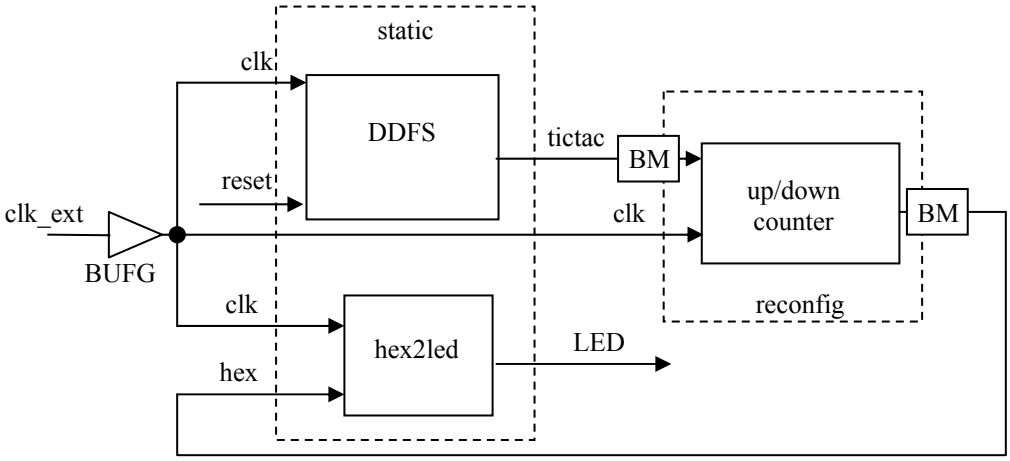


Figura 32. Diagrama de bloques del ejemplo de Reconfiguración parcial

Para comenzar se crea el sistema de directorios:

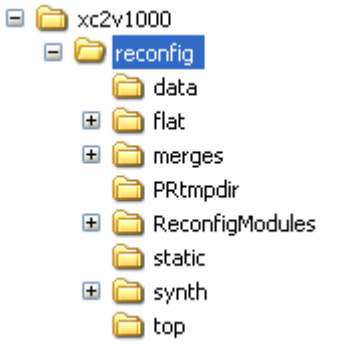


Figura 33. Estructura de directorios del proyecto reconfigurable

Partiendo de los ficheros utilizados con anterioridad *conv_frec.vhd, conv_frec.ucf, ddfs.vhd, hex2led.vhd, upcount.vhd* y *downcount.vhd* se procede a adaptarlos al flujo *EA_PR*.

Se sitúan en la carpeta *data*. Se crea un fichero *static.vhd* que contenga toda la parte estática (instancias a *ddfs* y a *hex2led*). Como se indicó en el apartado 4.4.3, se ha demostrado que esto no es necesario y se puede realizar la síntesis de los distintos módulos de la parte estática por separado, aunque de esta forma queda todo más claro.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity static is
    Port ( reset             : in std_logic;  -- pin:
           clk                  : in std_logic;      -- pin:
           hex          : in std_logic_vector(3 downto 0);
            tictac       : out std_logic; -- pin:
           LED          : out std_logic_vector(6 downto 0));
end static;
```

```
architecture Behavioral of static is

       component DDFS
       Port          (      rst : in std_logic;
                            clk   : in std_logic;
                             enable: out std_logic);
       end component;


       component hex2led
       Port          (      HEX : in std_logic_vector(3 downto 0);
                            LED : out std_logic_vector(6 downto 0));
       end component;

begin

-- Divisor de frecuencia para generar el parpadeo
       ddfs1 : ddfs
       port map (
                       rst => reset,
                             --   pulsador => pulsador,
                                clk => clk,
                             enable => tictac
                            );


--conversor hex2led para mostrar la salida del contador reconfigurable
       hex2led1 : hex2led
       port map (
                      hex => hex,
                      led => led
              );

end Behavioral;
```

Se modifica el fichero *conv_frec.vhd*, añadiendo las declaraciones del *buffer* del reloj *BUFGP*, de los *buses macro* y las señales intermedias entre los *buses macro* y el módulo reconfigurable. En este caso los buses elegidos son *busmacro_xc2v_r2l_async_narrow* y *busmacro_xc2v_l2r_async_narrow*.

```
       component BUFGP
  port ( I : in std_logic;
         O : out std_logic
       );
       end component;

       -- slice macros to communicate with reconfigurable module.  This goes
from the left of the chip to the right of the chip
component busmacro_xc2v_l2r_async_narrow is
       port (
              input0 : in std_logic;
              input1 : in std_logic;
              input2 : in std_logic;
              input3 : in std_logic;
              input4 : in std_logic;
              input5 : in std_logic;
              input6 : in std_logic;
              input7 : in std_logic;
              output0 : out std_logic;
              output1 : out std_logic;
              output2 : out std_logic;
              output3 : out std_logic;
              output4 : out std_logic;
```

```
              output5 : out std_logic;
              output6 : out std_logic;
              output7 : out std_logic
       );
       end component;

       -- slice macros to communicate with reconfigurable module.  This goes
from the right of the chip to the left of the chip
component busmacro_xc2v_r2l_async_narrow is
       port (
              input0 : in std_logic;
              input1 : in std_logic;
              input2 : in std_logic;
              input3 : in std_logic;
              input4 : in std_logic;
              input5 : in std_logic;
              input6 : in std_logic;
              input7 : in std_logic;
              output0 : out std_logic;
              output1 : out std_logic;
              output2 : out std_logic;
              output3 : out std_logic;
              output4 : out std_logic;
              output5 : out std_logic;
              output6 : out std_logic;
              output7 : out std_logic
       );
       end component;

signal tictac : std_logic;
signal hex  : std_logic_vector(3 downto 0);
signal tictac_reconf: std_logic;
signal hex_reconf: std_logic_vector(3 downto 0);
```

A continuación se instancian los elementos declarados para separar la parte estática y reconfigurable: los *buses macro*, así como el *buffer* del reloj y se conectan pertinentemente al módulo reconfigurable *cnt*.

Como los dos *buses macro* van a ser situados en la frontera derecha del módulo reconfigurable, el bus *r2l* estará destinado a las señales de entrada al módulo, mientras que el *l2r* estará destinado a las señales de salida.

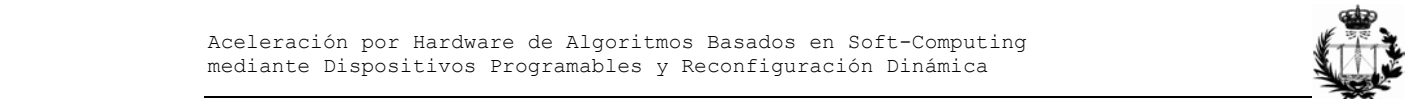
El reloj que entra a la *FPGA* irá directamente al buffer, y de ahí saldrá el reloj interno para todos los módulos.

```
      --Instantiation of user reconfigurable module:
       -- Contador que se configurará como asc/desc
       -- con reconfiguración dinámica

       cnt1 : cnt
       port map (
                              --general purpose DataIn:
                              tictac => tictac_reconf,
                              clk=>clk,
                              --general purpose DataOut:
                               count => hex_reconf
                            );
```

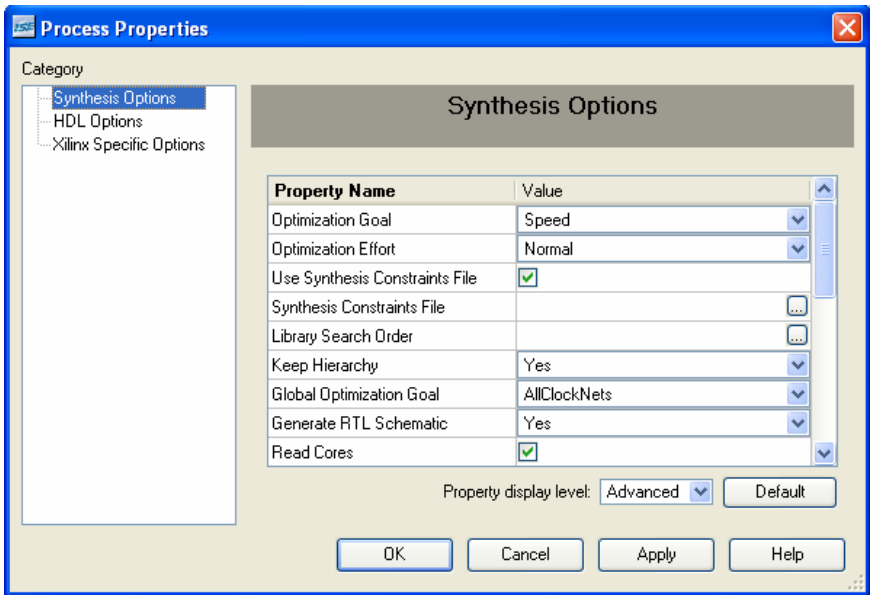
```
      bufgp_0: BUFGP
       port map (
               I      => clk_ext,
               O      => clk
               );

-- bus macro to transfer data from the control module to the reconfig_module
      macro_1 : busmacro_xc2v_r2l_async_narrow
    port map (  input0 => tictac,
                input1 => '1',
                input2 => '1',
                input3 => '1',
                input4 => '1',
                input5 => '1',
                input6 => '1',
                input7 => '1',
               output0 => tictac_reconf,
               output1 => open,
               output2 => open,
               output3 => open,
               output4 => open,
               output5 => open,
               output6 => open,
               output7 => open
             );

-- bus macro to transfer data from the reconfig_module to the control module
      macro_2 : busmacro_xc2v_l2r_async_narrow
    port map (  input0 => hex_reconf(0),
                input1 => hex_reconf(1),
                input2 => hex_reconf(2),
                input3 => hex_reconf(3),
                input4 => '1',
                input5 => '1',
                input6 => '1',
                input7 => '1',
               output0 => hex(0),
               output1 => hex(1),
               output2 => hex(2),
               output3 => hex(3),
               output4 => open,
               output5 => open,
               output6 => open,
               output7 => open
             );
```

Finalmente se procede a crear un proyecto de *ISE* en la carpeta *flat*, donde se incluyen (sin copiarlos) todos los ficheros ***.vhd*** y ***.ucf*** de la carpeta *data*, excepto el fichero del módulo reconfigurable del contador descendente (*downcount.vhd*). Lo que si es necesario es copiar los archivos ***.nmc*** de los *buses macro*.

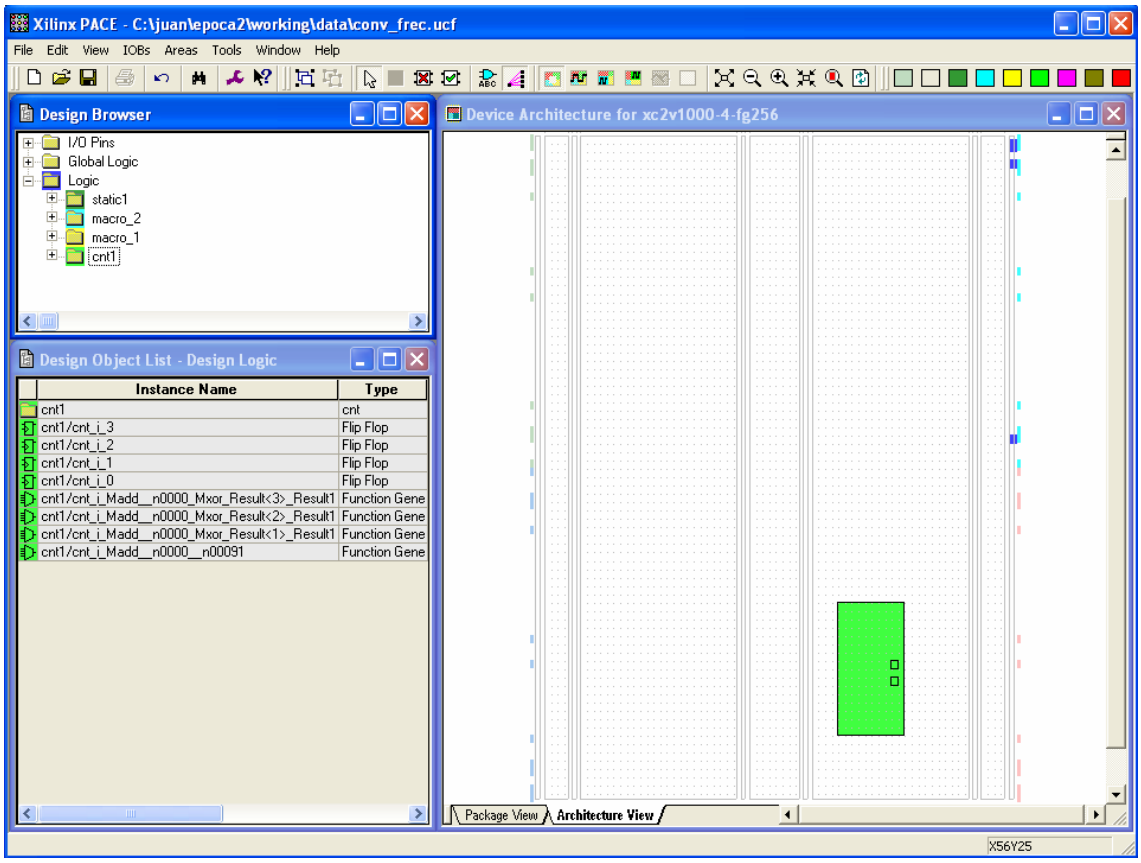
En ese momento se puede proceder a sintetizar el proyecto, observando los errores de código que se hayan podido producir en la adaptación al flujo *EA_PR*.



En esta placa, con la versión 8 de *EA_PR* para *ISE 8.1.01i*, si que se ha mantenido activada la casilla *Keep Hierarchy* en la síntesis de todos los proyectos *ISE*.



A continuación se abre el fichero ***.ucf*** con *PACE* y se sitúan los *buses macro* y el área reconfigurable.



Una vez guardado se edita el ***.ucf*** y en el que se deben incluir los *area_groups* de la parte estática y de la reconfigurable. Se localiza el *BUFG* del reloj en *BUFGMUX0P*, y los *buses macro* en la frontera derecha del área reconfigurable.

Se recuerda que para poder obtener el *bitstream* del proyecto de prueba de la carpeta *flat* en *ISE* es necesario comentar la directiva *MODE=RECONFIG* y añadir *GROUP=CLOSED*.

```
#PACE: Start of PACE Area Constraints
AREA_GROUP "ReconfAreaGrp" RANGE = SLICE_X40Y8:SLICE_X49Y23 ;
INST "cnt1" AREA_GROUP = "ReconfAreaGrp" ;
INST "static1" AREA_GROUP = "StaticAreaGrp" ;

AREA_GROUP "ReconfAreaGrp" MODE = RECONFIG;
#AREA_GROUP "ReconfAreaGrp" GROUP = CLOSED;

#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments
INST "bufgp_0" LOC = "BUFGMUX0P"  ;

#Bus Macro
INST "macro_1" LOC = "SLICE_X48Y16";
INST "macro_2" LOC = "SLICE_X48Y14";

NET "clk_ext" TNM_NET = "clk_ext";
TIMESPEC "TS_clk_ext" = PERIOD "clk_ext" 25 ns HIGH 50 %;

NET reset  LOC = R10  ;
Net reset TIG;

#NET "clk_ext"  LOC = "T9"  ;
NET "LED<0>"  LOC = "D16"  ;
NET "LED<1>"  LOC = "D14"  ;
NET "LED<2>"  LOC = "D15"  ;
NET "LED<3>"  LOC = "E13"  ;
NET "LED<4>"  LOC = "E14"  ;
NET "LED<5>"  LOC = "H13"  ;
NET "LED<6>"  LOC = "H14"  ;
```

Ya puede implementarse y obtenerse el bitstream, para comprobar su correcto funcionamiento en la placa.

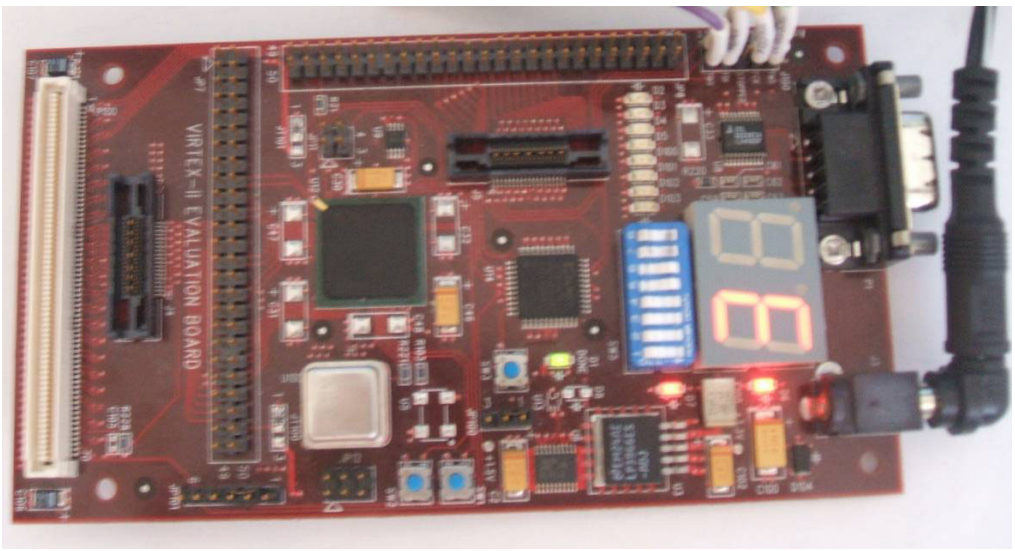


Figura 34. Contador ascendente en la placa *Virtex-II*

A continuación se vuelve a cambiar la directiva a *MODE=RECONFIG* en el archivo ***.ucf*** y se prosigue con el flujo *EA_PR*, realizando y sintetizando los diversos proyectos *ISE* de la carpeta *synth*, tal y como se vio en el apartado 4.4.

Finalmente se procede a implementar y obtener los *bitstreams* parciales, utilizando el flujo descrito. Para ello se utiliza el siguiente *script*:

```
echo IMPLEMENTACION DEL TOP

cd C:\juan\epoca2\working\top
del /Q *.*

copy ..\synth\top\conv_frec.ngc
copy ..\data\conv_frec.ucf
copy ..\data\*.nmc

ngdbuild -modular initial -p xc2v1000-4-fg256 conv_frec.ngc

ECHO IMPLEMENTACION DEL STATIC

cd C:\juan\epoca2\working\static
del /Q *.*

copy ..\synth\static\*.ngc
copy ..\data\conv_frec.ucf
copy ..\data\*.edn

ngdbuild -modular initial -p xc2v1000-4-fg256 ../top/conv_frec.ngo
map conv_frec.ngd
par -w conv_frec.ncd conv_frec_routed.ncd

ECHO IMPLEMENTACION DEL UPCOUNT

cd C:\juan\epoca2\working\ReconfigModules\upcount
del /Q *.*

copy ..\..\synth\upcount\cnt.ngc
copy ..\..\data\conv_frec.ucf
copy ..\..\static\static.used
ren static.used arcs.exclude

ngdbuild -modular module -p xc2v1000-4-fg256 -active cnt
../../top/conv_frec.ngo
map conv_frec.ngd
par -w conv_frec.ncd conv_frec_routed.ncd

ECHO IMPLEMENTACION DEL DOWNCOUNT

cd C:\juan\epoca2\working\ReconfigModules\downcount
del /Q *.*

copy ..\..\synth\downcount\cnt.ngc
copy ..\..\data\conv_frec.ucf
copy ..\..\static\static.used
ren static.used arcs.exclude

ngdbuild -modular module -p xc2v1000-4-fg256 -active cnt
../../top/conv_frec.ngo
map conv_frec.ngd
par -w conv_frec.ncd conv_frec_routed.ncd

ECHO UNIENDO LOS MODULOS

cd C:\juan\epoca2\working\merges
del /Q *.*
```

```
copy ..\static\conv_frec_routed.ncd static.ncd
copy ..\ReconfigModules\upcount\conv_frec_routed.ncd upcount.ncd
copy ..\ReconfigModules\downcount\conv_frec_routed.ncd downcount.ncd

PR_assemble.bat static.ncd upcount.ncd
```

Y a continuación:

```
echo GENERANDO LOS BITSTREAMS PARCIALES

PR_verifydesign.bat static.ncd upcount.ncd downcount.ncd

cd C:\juan\epoca2\working\
```

Una vez concluido se accede a la carpeta *merges*, donde se encuentran los ficheros *static_full.bit, upcount_partial.bit* y *downcount_partial.bit.* Se procede a descargar el *bitstream* completo *static_full.bit* y a continuación los *bitstreams* parciales con *iMPACT*, observando como se produce correctamente la reconfiguración parcial y el contador asciende o desciende según el módulo que se haya descargado. Puede comprobarse en el vídeo demostrativo que se adjunta en el cd.

Para comprobar que la reconfiguración parcial es activa con el nuevo flujo *EA_PR*, es decir, que no se para la *FPGA* durante la reconfiguración, se utiliza el mismo método que en el apartado 3.2.3, se añade otro contador a la velocidad del reloj y se reconfigura, observando que el segundo 8 siempre permanece sin parpadear. Por lo tanto se comprueba que la reconfiguración es activa, tal y como se esperaba.

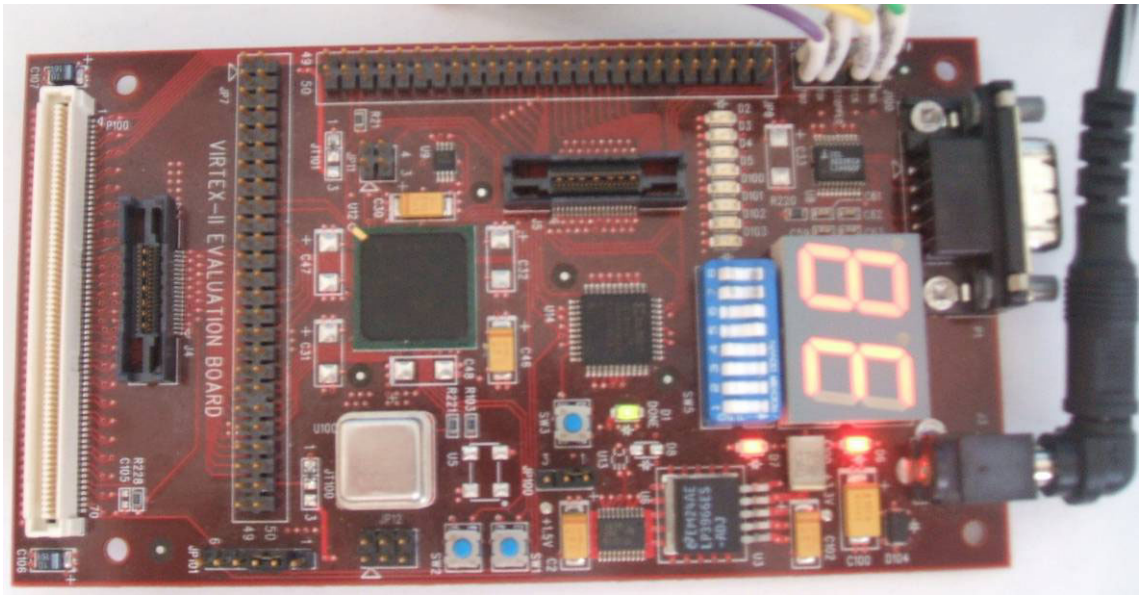


Figura 35. FPGA con el segundo contador a velocidad de reloj

5.2.3. Auto-Reconfiguración Parcial

El proyecto con el ejemplo de Auto-Reconfiguración Parcial con *xc2v1000* se encuentra en el cd en la carpeta *xc2v1000/autoreconfig*. También se puede ver su vídeo demostrativo titulado "Auto-Reconfiguración parcial xc2v1000".

Para realizar el ejemplo de auto-reconfiguración parcial con ***Virtex-II*** se procede de forma similar a la explicada en el Capítulo 4 para ***Virtex-4***, instanciando un microprocesador *MicroBlaze* con *hwicap*, que permita que la propia *FPGA* se auto-reconfigure.

Se trata de la primera prueba de auto-reconfiguración parcial que se realizó siguiendo el flujo *EA_PR*. En este caso se realiza la reconfiguración leyendo el *bitstream* parcial en el propio código del programa *C++* que se ejecuta en el *MicroBlaze,* no descargándolo a través del depurador en la memoria *RAM* como en la ***Virtex-4*.**

Se parte del proyecto reconfigurable *reconfig* del apartado anterior. En la carpeta *flat* se crea un proyecto *EDK* que contenga el *hwicap* y el puerto serie *RS-232.*

En este caso no se instancia el *debugger* ya que no se utilizará. Tampoco será necesario hacer ninguna modificación específica del archivo ***.mhs*** que genere *EDK* para la placa en cuestión, como ocurría en la ***Virtex-4***.

El diseño auto-reconfigurable será el siguiente:

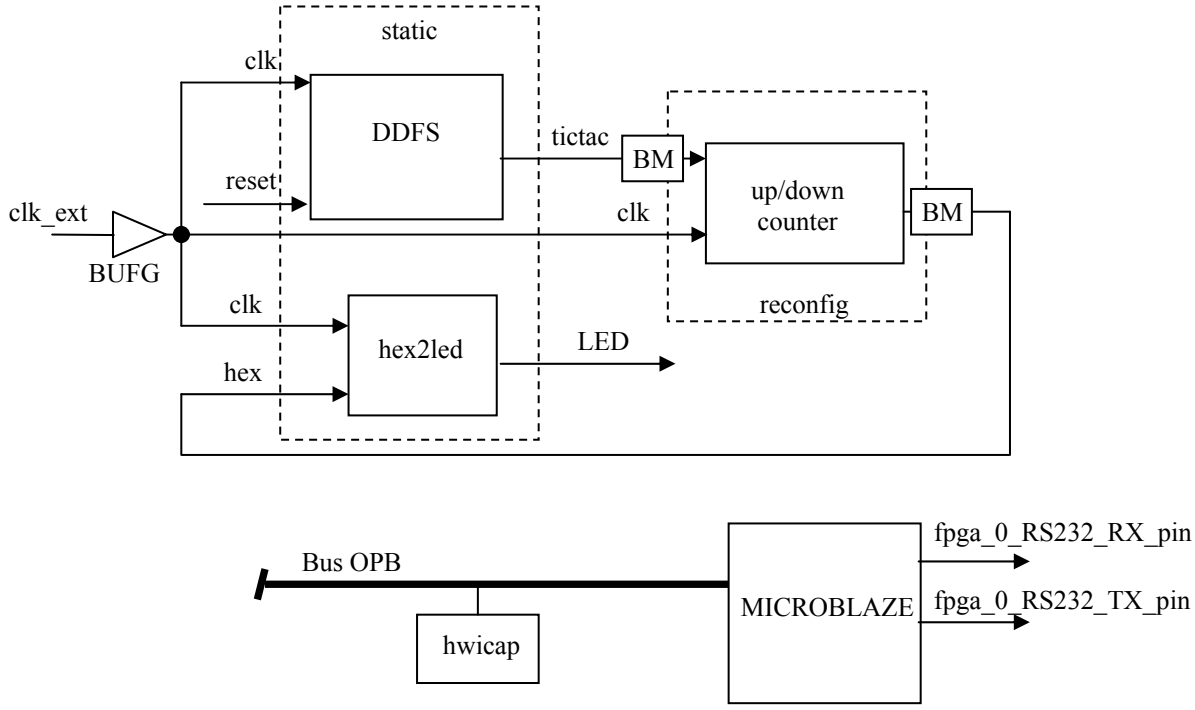


Figura 36. Diagrama de bloques de Auto-Reconfiguración con *Virtex-II*

Se quita el *BUFG* del *CLK0* del *dcm*, y se sintetiza el proyecto. Se exporta a *ISE* como sub-módulo y se mezcla el *top* con el archivo *conv_frec.vhd*, prosiguiendo tal y como se describe en los apartados 4.4, 4.5 y 4.6, hasta la obtención de los *bitstreams* parciales.

Para poder introducir el *bitstream* parcial en el código *C++* es necesario obtener el archivo ***.rbt*** (fichero *ASCII*) cuando se genera el ***.bit.***

Para ello hay que modificar los *scripts* del *EA_PR* que generan los ***.bit***: *PR_assemble.pl* y *PR_verifydesign.pl.* Ambos se encuentran en la carpeta *C:\Xilinx\bin\nt*. Se les quita con el botón derecho la opción de sólo lectura y se editan con *WordPad,* buscando la siguiente sentencia:

```
my($a_gclk) = "-g ActivateGclk:Yes";
```

Que se sustituye por esta otra sentencia, que incluye el comando *–b* que se encarga de generar los ***.rbt***, y se guardan los ficheros:

```
my($a_gclk) = "-b -g StartUpClk:jtagclk -g ActivateGclk:Yes"
```

Esta modificación es permanente, por lo que siempre que se obtengan los ***.bit*** se obtendrán también los ***.rbt***. Ya se puede por tanto ejecutar los *scripts* que generan los distintos *bitstreams.*

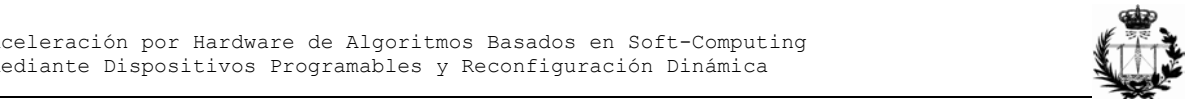
En este momento surge un problema. El *bitstream* parcial generado (*downcount_partial.bit*) es demasiado grande como para ponerlo en el *array* e incluirlo dentro del programa de *MicroBlaze*, ya que no cabe almacenado en las *bram* de la *FPGA*.

Debido a este problema se decide pasar a utilizar otra placa, la ***Virtex-II Pro*** *xcv2p7*, que posee un chip de memoria *RAM* en el que sí que cabrá el *bitstream* parcial tenga la longitud que tenga.

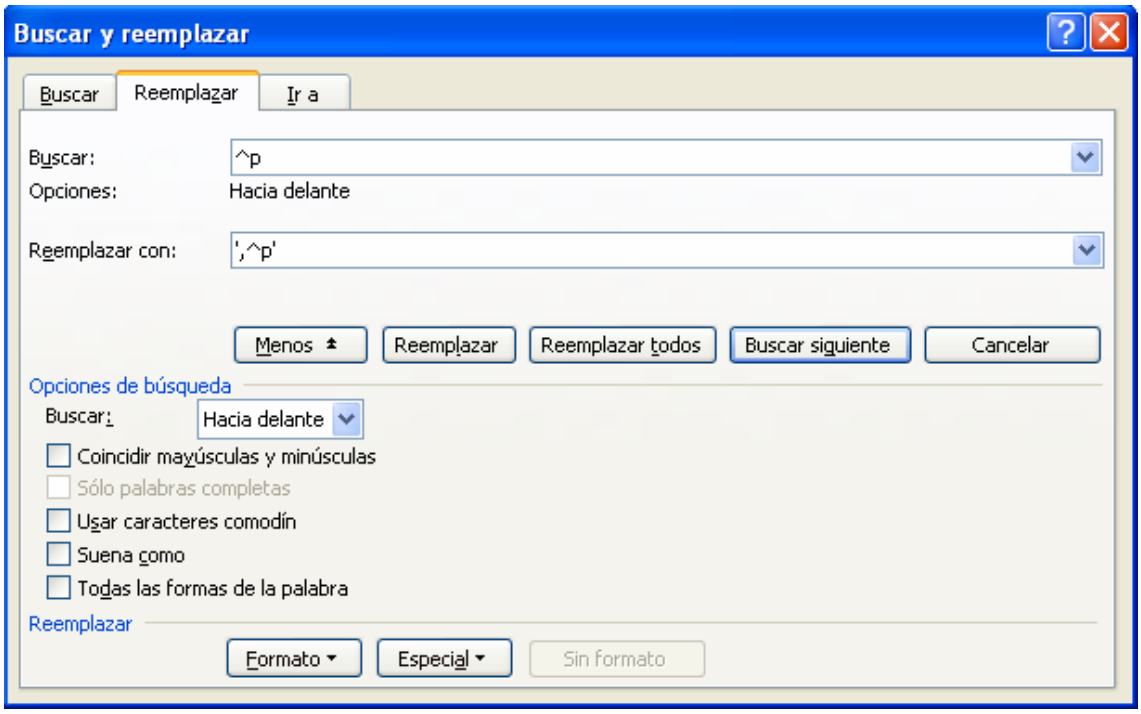
Pero en lugar de finalizar aquí con la placa *xc2v1000*, se procede a tratar de auto-reconfigurar con otro *bitstream* parcial que se genera con los scripts del *EA_PR*, el bitstream en diferencias (*downcount_full_framediff.bit*) que sólo incluye la diferencia entre un módulo reconfigurable y el otro, lo cual hace que sea de mucho menor tamaño y que quepa en las *bram*.

Para proceder a reconfigurar es necesario cambiar el formato de *downcount_full_framediff.rbt* para poder introducirlo en el código *C++* como un *array* de números hexadecimales.

```
00110000000000000010000000000001
00000000001011100000000000000000
00110000000000000100000000000000
01010000000000000001111111100010
00110000000000000000000000000000
```



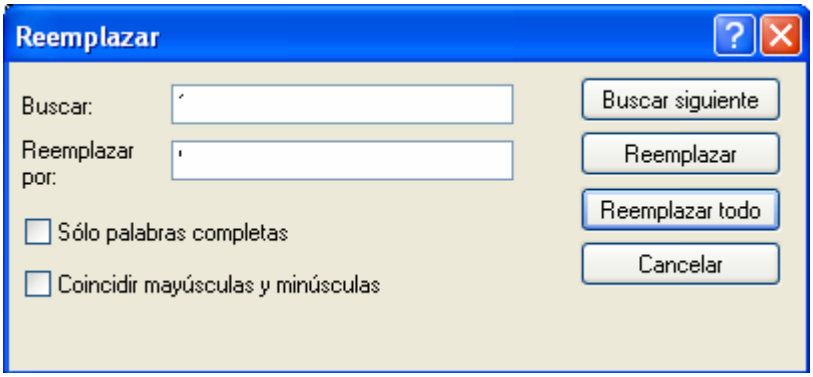
El método más automatizado encontrado ha sido abrir el ***.rbt*** con *MS Word*, y reemplazar todos los párrafos por comillas simples y comas de separación:



Quedando:

```
'00110000000000000010000000000001',
'00000000001011100000000000000000',
'00110000000000000100000000000000',
'01010000000000000001111111100010',
'00110000000000000000000000000000',
```

Pero estas comillas no son las aceptadas por *Matlab*, así que es necesario copiarlo todo a *WordPad* y reemplazar todas las comillas, copiando la comilla en la casilla superior, y poniendo una comilla simple en la inferior. Esto habrá que realizarlo dos veces, para las comillas delanteras y para las traseras.



Finalmente se copia el archivo resultante en un *script* de *Matlab* llamado *convierte.m* que lo convierte en números hexadecimales y los guarda en un fichero *hexadecimal.txt*.

```
binario=['11111111111111111111111111111111',
'10101010100110010101010101100110',
'00110000000000001000000000000001',
'00000000000000000000000000000111',
'00110000000000011100000000000001',
'00000010000001010000000010010011',
.....................................
'00110000000000000000000000000001',
'00000000000000001101101000100001',
'00110000000000001000000000000001',
'00000000000000000000000000001101'];

hexadecimal=[];
for i=1:length(binario)
    hexadecimal=[hexadecimal;  dec2hex(bin2dec(binario(i,:)),8)];
end

archivo1=fopen('hexadecimal.txt','w')

for i=1:length(hexadecimal)
        fprintf(archivo1,'0x%s,\n',hexadecimal(i,:));

end

fclose(archivo1);
```

Se abre el fichero *hexadecimal.txt* y se copia su contenido en el código *C++*. Este código reconfigurará la *FPGA* con el *array* en cuestión en el que se encuentra el *bitstream* parcial. Finalmente el código C++ quedará:

```
/*FPGAs PARTIAL RECONFIGURATION*/
/*Non-Peripheral partial Self-Reconfiguration*/
/*Juan Quero Llor*/

#include <xuartlite_l.h>
#include "xparameters.h"
#include <xHwIcap.h>
#include "xstatus.h"
#include "xhwicap_i.h"

static Xint32 bitstream [3426] = {
0xFFFFFFFF,
0xAA995566,
0x30008001,
……
0x0000DA21,
0x30008001,
0x0000000D
};

XHwIcap InstIcap;

int main (void) {
       char option;
       XStatus status;
       Xint32 step, address;
       step=0;
       print("\r\n-- Entering main() --"); //this lines executes but is not seen
```

```
	while(1) {
		print("\fFPGAs PARTIAL RECONFIGURATION");
		print("\r\nNon-Peripheral partial self-reconfiguration");
		print("\r\nVirtex 2 (xc2v1000)");
		print("\r\n\tJuan Quero Llor");

		print("\r\n\nSelect an option [1-2]:");
		if(step>0)
			print("\r\n\t 1. Initialize hwicap (DONE)");
		else
			print("\r\n\t 1. Initialize hwicap.");
		print("\r\n\t 2. Perform Partial Reconfiguration");

		option = (char)inbyte();

		switch(option){
		case '1'://hwicap initialization
			xil_printf("\r\n\r\nExecuting option %c.\r\n",option);
			if(step==0){
				status = XHwIcap_Initialize(&InstIcap,
					XPAR_OPB_HWICAP_0_DEVICE_ID,
					XHI_READ_DEVICEID_FROM_ICAP);

				if (status != XST_SUCCESS) {
					xil_printf("\r\nXHwICAP Initialization failed. Device ID read:
%x status %x", InstIcap.DeviceIdCode, status);
					//exit(1);
				} else {
				xil_printf("\r\nXHwICAP Initialization success. Device ID read: %x",
							InstIcap.DeviceIdCode);
				}

				xil_printf("\r\n Full info received from ICAP: \
							\r\n\t IsReady: %x \
							\r\n\t DeviceIdCode: %x \
							\r\n\t DeviceId: %x \
							\r\n\t Rows: %x \
							\r\n\t Cols: %x \
							\r\n\t BramCols: %x \
							\r\n\t BytesPerFrame: %x \
							\r\n\t WordsPerFrame: %x \
							\r\n\t ClbBlockFrames: %x \
							\r\n\t BramBlockFrames: %x \
							\r\n\t BramIntBlockFrames: %x", \
							InstIcap.IsReady, InstIcap.DeviceIdCode,
InstIcap.DeviceId, \InstIcap.Rows, InstIcap.Cols, InstIcap.BramCols, InstIcap.BytesPerFrame, \
InstIcap.WordsPerFrame, InstIcap.ClbBlockFrames, \ InstIcap.BramBlockFrames,
InstIcap.BramIntBlockFrames);
				step++;
			} else {
				print("\r\nHwicap Initialization already done");
			}
			print("\r\n\r\nPress any key to continue...");	option = (char)inbyte();
			break;

		case '2': //FPGA Partial reconfiguration

			if(step==0)
			{
				print("\r\n\r\nPlease initialize Hwicap before(step 1)");
				print("\r\nPress any key to continue...");	option = (char)inbyte();
				break;
			}

			xil_printf("\r\n\r\nExecuting option %c.",option);

			address=(Xuint32)&bitstream;
```

```
                    XHwIcap_SetConfiguration(&InstIcap, address, sizeof(bitstream));

                    if (status != XST_SUCCESS)
                          xil_printf("\r\nFPGA Reconfiguration failed.\r\n\nIMPORTANT: Please
download the initial full bitstream again",
                                                InstIcap.DeviceIdCode, status);
                    else
                          xil_printf("\r\nFPGA Reconfiguration success. Device ID read: %x
status %x\r\n",
                                                InstIcap.DeviceIdCode, status);
                    step=1;
                    print("\r\nPress any key to continue...");    option = (char)inbyte();
              break;

              default:
                    xil_printf("\r\n The character selected %c (ASCII %d) was not valid
option.",option,option);
                    print("\r\nPress any key to continue...");
                    option = (char)inbyte();

              }
       }

       //this line never executes
       print("\r\n-- Exiting main() --\r\n");
       return 0;
}
```

El método *main()* tiene dos opciones (tecleando números del 1 al 2):

6. Se inicializa el *hwicap*.
7. Se reconfigura, leyendo el *bitstream* parcial del array descargándolo en la placa a través del *hwicap* mediante *XHwIcap_SetConfiguration()*.

Una vez compilado se procede a cargarlo en el *bitstream* completo generado en la carpeta *merges*. El *EDK 8.1* tiene un *bug* que no permite importar correctamente los ficheros ***.bit*** y ***.bmm***, por lo que se debe hacer manualmente.

Se renombra *static_full.bit* a *system.bit* y se copia en la carpeta *implementation* del proyecto *EDK* junto con *system_stub_bd.bmm*. A continuación ya se puede descargar en la placa:

```
FPGAs PARTIAL RECONFIGURATION
Non-Peripheral partial self-reconfiguration
Virtex 2 (xc2v1000)
        Juan Quero Llor

Select an option [1-2]:
         1. Initialize hwicap.
         2. Perform Partial Reconfiguration
```

Se inicializa el *hwicap* y a continuación se procede a realizar la auto-reconfiguración parcial, que funciona correctamente ya que el contador pasa de ser ascendente a descendente tal y como se puede comprobar en el vídeo demostrativo.

5.3. Auto-Reconfiguración Parcial en Virtex-II Pro xc2vp7

5.3.1. Introducción

Debido a los problemas ocurridos con la placa ***Virtex-II***, que no disponía de memoria *bram* suficiente como para poder introducir el *bitstream* parcial, ni disponía de memoria *RAM* donde introducirlo, se decidió cambiar de placa, a una ***Virtex-II pro***, que si que disponía de memoria *RAM*.

En el proceso de invención del flujo con el que se introduciría el *bitstream* parcial en la placa, se realizó un proyecto en el que el programa no se ejecutaba en la memoria *bram*, sino en la memoria *RAM*. Este proyecto viene descrito en el Anexo 5.

Finalmente se decidió que el programa iría instalado en la memoria *bram* de la *FPGA* y el *bitstream* se descargaría en la memoria *RAM* de la placa a través del *debugger hardware* del *MicroBlaze*.

La placa utilizada fue la *Virtex-II pro Evaluation Board* de *Avnet*, *FPGA xc2vp7-ff896*, con el software *ISE 8.1 Service Pack 1, EDK 8.1* y *EA_PR v8* para *ISE 8.1.01i.*

5.3.2. Configuración de la Placa

5.3.2.1. Configuración de la Memoria RAM:

Para el correcto funcionamiento de la memoria *RAM* en la placa *xc2vp7*, es necesario incluir el siguiente código en el fichero ***.ucf.*** Si no se incluye la *RAM* no funcionará.

```
NET     "VREF_DUMMY<0>"         LOC = "C2";     #
NET     "VREF_DUMMY<1>"         LOC = "J1";     #
NET     "VREF_DUMMY<2>"         LOC = "L1";    #
NET     "VREF_DUMMY<3>"         LOC = "N1";     #
NET     "VREF_DUMMY<4>"         LOC = "R2";     #
NET     "VREF_DUMMY<5>"         LOC = "U2";     #
NET     "VREF_DUMMY<6>"         LOC = "W1";     #
NET     "VREF_DUMMY<7>"         LOC = "AA1";     #
NET     "VREF_DUMMY<8>"         LOC = "AB2";     #
NET     "VREF_DUMMY<9>"         LOC = "AH1";     #
##############################################################
##                HEADER I/O                                  #
##############################################################
##
NET     "UNUSED_OUT<0>"         LOC = "AD19";    #  "HDR_IO0"
NET     "UNUSED_OUT<1>"         LOC = "AC19";    #  "HDR_IO1"
NET     "UNUSED_OUT<2>"         LOC = "AD20";    #  "HDR_IO2"
NET     "UNUSED_OUT<3>"         LOC = "AC20";    #  "HDR_IO3"
NET     "UNUSED_OUT<4>"         LOC = "AE21";    #  "HDR_IO4"
NET     "UNUSED_OUT<5>"         LOC = "AC21";    #  "HDR_IO5"
NET     "UNUSED_OUT<6>"         LOC = "AG21";    #  "HDR_IO6"
NET     "UNUSED_OUT<7>"         LOC = "AF21";    #  "HDR_IO7"
NET     "UNUSED_OUT<8>"         LOC = "AD21";    #  "HDR_IO8"
NET     "UNUSED_OUT<9>"         LOC = "AF22";    #  "HDR_IO9"
```

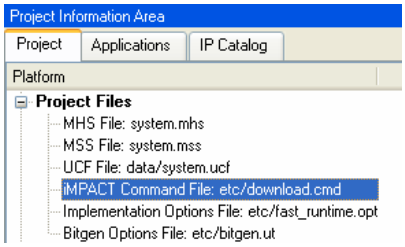
Esto es así porque ciertos pines con voltajes de referencia están conectados a las patillas de la *RAM*. Si no se incluyen estas conexiones a señales *dummy* (señales sin utilidad), en la síntesis e implementación del proyecto podrían quedar conectadas a cualquier otra señal que interfiriese en el correcto funcionamiento de la memoria *RAM*.

5.3.2.2. Descarga de Bitstreams en la Placa:

Cuando la placa *xc2vp7* se pretende configurar con el *Platform Cable USB*, se producen una serie de errores, tanto en *iMPACT* como en *EDK*, que hace que se reconozcan otros dispositivos que no existen y que no se reconozca la *FPGA*.

Esto es así porque la placa no está preparada para la velocidad normal del cable *USB*, y hay que reducirla a 750000 bps.

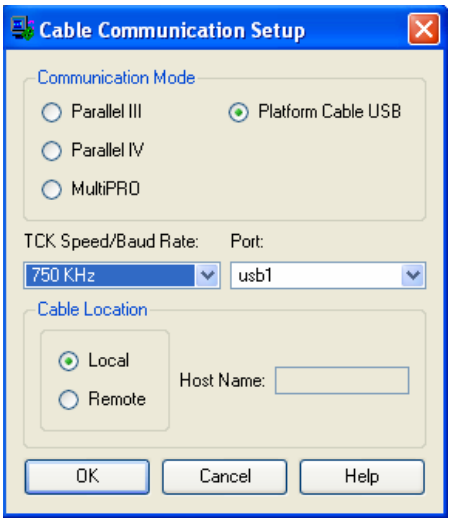
Para hacer esto en *EDK* hay que editar el fichero *etc/download.cmd*, haciendo doble clic en *iMPACT Command File* en la pestaña *Project* de la ventana *Project Information Area:*



En ese archivo se ha de añadir la directiva *setCableSpeed -speed 750000*:

```
setMode -bscan
setCable -p auto
setCableSpeed -speed 750000
identify
assignfile -p 3 -file implementation/download.bit
program -p 3
quit
```

Por otro lado si se pretende descargar a través de *iMPACT*, es necesario acudir al menú *Output/Cable Setup…* y cambiar la configuración a 750000:



5.3.3. Auto-Reconfiguración Parcial

El ejemplo realizado es el mismo que para la ***Virtex-4*** en el Capítulo4: encendido de los *leds* hacia la derecha, hacia la izquierda, o en ambos sentidos. Se decidió este ejemplo porque la placa no disponía de display de 7 segmentos como la anterior.

El proyecto ejemplo de Auto-Reconfiguración Parcial con *xc2vp7* se encuentra en el cd en la carpeta *xc2vp7/autoreconfig*, y su vídeo demostrativo se titula "Auto-Reconfiguración parcial xc2vp7".

El diseño de auto-reconfiguración realizado es el siguiente:

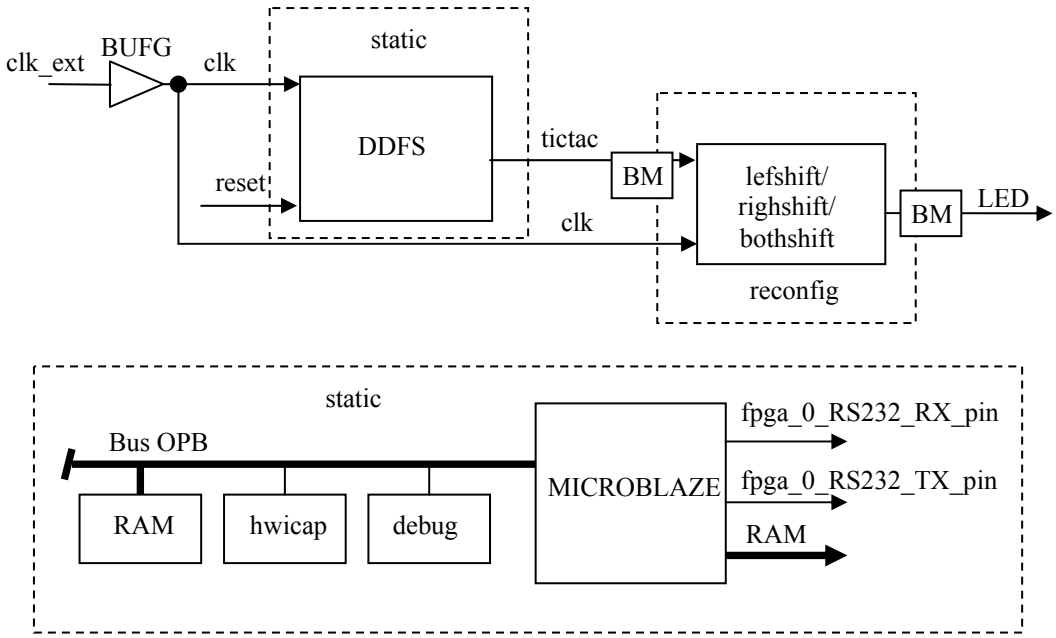


Figura 37. Diagrama de bloques de Auto-Reconfiguración con *Virtex-II Pro*

Para realizar la auto-reconfiguración parcial se siguen exactamente los mismos pasos que para la ***Virtex-4***. Se descargará el *bitstream* parcial en la memoria *RAM* a través del depurador y se procederá a la auto-reconfiguración parcial con un microprocesador *MicroBlaze* con *hwicap*.

La principal diferencia se encuentra obviamente en el fichero ***.ucf,*** que varía dependiendo de la placa:

```
##############################################################################
## This system.ucf file is generated by Base System Builder based on the
## settings in the selected Xilinx Board Definition file. Please add other
## user constraints to this file based on customer design specifications.
##############################################################################

INST "static1" AREA_GROUP = "StaticAreaGrp" ;
INST "micro" AREA_GROUP = "StaticAreaGrp" ;

INST "shift1" AREA_GROUP = "ReconfAreaGrp" ;
AREA_GROUP "ReconfAreaGrp" RANGE = SLICE_X40Y8:SLICE_X49Y23 ;

AREA_GROUP "ReconfAreaGrp" MODE = RECONFIG;
#AREA_GROUP "ReconfAreaGrp" GROUP = CLOSED;
```

```
#PACE: Start of PACE I/O Pin Assignments
INST "bufgp_0" LOC = "BUFGMUX0P"  ;

#Bus Macro
INST "macro_1" LOC = "SLICE_X48Y16";
INST "macro_2" LOC = "SLICE_X48Y14";

Net clk_ext LOC=C16;
Net reset_pin LOC=AG5;
## System level constraints
Net clk_ext TNM_NET = clk_ext;
TIMESPEC TS_clk_ext = PERIOD clk_ext 10000 ps;
Net reset_pin TIG;

## IO Devices constraints


NET     "LED<0>"            LOC = "AE15";
NET     "LED<1>"            LOC = "AD15";
NET     "LED<2>"            LOC = "AC15";
NET     "LED<3>"            LOC = "AB15";
NET     "LED<4>"            LOC = "AG14";
NET     "LED<5>"            LOC = "AF14";
NET     "LED<6>"            LOC = "AD14";
NET     "LED<7>"            LOC = "AE14";

#### Module RS232 constraints

Net fpga_0_RS232_RX_pin LOC=AB16;
Net fpga_0_RS232_TX_pin LOC=AC16;

#### Module DDR_SDRAM_1 constraints

Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<12> LOC=R26;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<12> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<11> LOC=R23;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<11> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<10> LOC=R24;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<10> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<9> LOC=R22;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<9> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<8> LOC=T29;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<8> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<7> LOC=R28;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<7> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<6> LOC=P28;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<6> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<5> LOC=P30;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<5> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<4> LOC=P29;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<4> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<3> LOC=M30;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<3> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<2> LOC=R25;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<2> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<1> LOC=M27;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<1> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<0> LOC=L26;
Net fpga_0_DDR_SDRAM_1_DDR_Addr_pin<0> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_BankAddr_pin<1> LOC=R27;
Net fpga_0_DDR_SDRAM_1_DDR_BankAddr_pin<1> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_BankAddr_pin<0> LOC=P27;
Net fpga_0_DDR_SDRAM_1_DDR_BankAddr_pin<0> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_CASn_pin LOC=P24;
Net fpga_0_DDR_SDRAM_1_DDR_CASn_pin IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_CKE_pin LOC=L27;
Net fpga_0_DDR_SDRAM_1_DDR_CKE_pin IOSTANDARD = SSTL2_I;
```

```
Net fpga_0_DDR_SDRAM_1_DDR_CSn_pin LOC=V23;
Net fpga_0_DDR_SDRAM_1_DDR_CSn_pin IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_CSn_1_pin LOC=V24;
Net fpga_0_DDR_SDRAM_1_DDR_CSn_1_pin IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_RASn_pin LOC=P26;
Net fpga_0_DDR_SDRAM_1_DDR_RASn_pin IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_WEn_pin LOC=P23;
Net fpga_0_DDR_SDRAM_1_DDR_WEn_pin IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_DM_pin<3> LOC=N26;
Net fpga_0_DDR_SDRAM_1_DDR_DM_pin<3> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_DM_pin<2> LOC=N25;
Net fpga_0_DDR_SDRAM_1_DDR_DM_pin<2> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_DM_pin<1> LOC=T25;
Net fpga_0_DDR_SDRAM_1_DDR_DM_pin<1> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_DM_pin<0> LOC=T26;
Net fpga_0_DDR_SDRAM_1_DDR_DM_pin<0> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_DQS_pin<3> LOC=N28;
Net fpga_0_DDR_SDRAM_1_DDR_DQS_pin<3> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQS_pin<2> LOC=N27;
Net fpga_0_DDR_SDRAM_1_DDR_DQS_pin<2> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQS_pin<1> LOC=T27;
Net fpga_0_DDR_SDRAM_1_DDR_DQS_pin<1> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQS_pin<0> LOC=T28;
Net fpga_0_DDR_SDRAM_1_DDR_DQS_pin<0> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<31> LOC=M23;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<31> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<30> LOC=M24;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<30> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<29> LOC=K25;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<29> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<28> LOC=K26;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<28> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<27> LOC=K27;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<27> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<26> LOC=J28;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<26> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<25> LOC=J29;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<25> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<24> LOC=K30;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<24> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<23> LOC=K29;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<23> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<22> LOC=H29;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<22> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<21> LOC=L29;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<21> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<20> LOC=J27;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<20> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<19> LOC=M26;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<19> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<18> LOC=M25;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<18> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<17> LOC=N24;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<17> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<16> LOC=N23;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<16> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<15> LOC=Y30;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<15> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<14> LOC=W29;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<14> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<13> LOC=U28;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<13> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<12> LOC=U27;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<12> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<11> LOC=U26;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<11> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<10> LOC=U24;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<10> IOSTANDARD = SSTL2_II;
```

```
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<9> LOC=U23;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<9> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<8> LOC=U22;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<8> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<7> LOC=T22;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<7> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<6> LOC=T23;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<6> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<5> LOC=T24;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<5> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<4> LOC=V25;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<4> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<3> LOC=V26;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<3> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<2> LOC=V27;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<2> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<1> LOC=V28;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<1> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<0> LOC=V29;
Net fpga_0_DDR_SDRAM_1_DDR_DQ_pin<0> IOSTANDARD = SSTL2_II;
Net fpga_0_DDR_SDRAM_1_DDR_Clk_pin<0> LOC=N29;
Net fpga_0_DDR_SDRAM_1_DDR_Clk_pin<0> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Clkn_pin<0> LOC=M29;
Net fpga_0_DDR_SDRAM_1_DDR_Clkn_pin<0> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Clk_pin<1> LOC=V30;
Net fpga_0_DDR_SDRAM_1_DDR_Clk_pin<1> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Clkn_pin<1> LOC=U30;
Net fpga_0_DDR_SDRAM_1_DDR_Clkn_pin<1> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Clk_pin<2> LOC=AA29;
Net fpga_0_DDR_SDRAM_1_DDR_Clk_pin<2> IOSTANDARD = SSTL2_I;
Net fpga_0_DDR_SDRAM_1_DDR_Clkn_pin<2> IOSTANDARD = SSTL2_I;

Net fpga_0_DDR_CLK_FB LOC=AF16;
Net fpga_0_DDR_CLK_FB IOSTANDARD = LVCMOS25;


NET     "VREF_DUMMY<0>"         LOC = "C2";     #
NET     "VREF_DUMMY<1>"         LOC = "J1";     #
NET     "VREF_DUMMY<2>"         LOC = "L1";    #
NET     "VREF_DUMMY<3>"         LOC = "N1";     #
NET     "VREF_DUMMY<4>"         LOC = "R2";     #
NET     "VREF_DUMMY<5>"         LOC = "U2";     #
NET     "VREF_DUMMY<6>"         LOC = "W1";     #
NET     "VREF_DUMMY<7>"         LOC = "AA1";     #
NET     "VREF_DUMMY<8>"         LOC = "AB2";     #
NET     "VREF_DUMMY<9>"         LOC = "AH1";     #
############################################################
##                HEADER I/O                                #
############################################################
##
NET     "UNUSED_OUT<0>"         LOC = "AD19";   #  "HDR_IO0"
NET     "UNUSED_OUT<1>"         LOC = "AC19";   #  "HDR_IO1"
NET     "UNUSED_OUT<2>"         LOC = "AD20";   #  "HDR_IO2"
NET     "UNUSED_OUT<3>"         LOC = "AC20";   #  "HDR_IO3"
NET     "UNUSED_OUT<4>"         LOC = "AE21";   #  "HDR_IO4"
NET     "UNUSED_OUT<5>"         LOC = "AC21";   #  "HDR_IO5"
NET     "UNUSED_OUT<6>"         LOC = "AG21";   #  "HDR_IO6"
NET     "UNUSED_OUT<7>"         LOC = "AF21";   #  "HDR_IO7"
NET     "UNUSED_OUT<8>"         LOC = "AD21";   #  "HDR_IO8"
NET     "UNUSED_OUT<9>"         LOC = "AF22";   #  "HDR_IO9"
```

El script utilizado para la implementación y generación de los *bitstreams* parciales es el siguiente:

```
echo IMPLEMENTACION DEL TOP

cd top
del /Q *.*

copy ..\synth\top\top.ngc
copy ..\data\top.ucf
copy ..\data\*.nmc

ngdbuild -modular initial -p xc2vp7-6-ff896 top.ngc

cd..

ECHO IMPLEMENTACION DEL STATIC

cd static
del /Q *.*

copy ..\synth\static\*.ngc
copy ..\data\top.ucf
copy ..\data\*.edn
copy ..\synth\micro\implementation\system_stub.bmm
copy ..\synth\micro\projnav\*.ngc
copy ..\synth\micro\implementation\*.ngc

ngdbuild -modular initial -p xc2vp7-6-ff896 -bm system_stub.bmm ../top/top.ngo
map top.ngd
par -w top.ncd top_routed.ncd

cd..

ECHO IMPLEMENTACION DEL leftshift

cd ReconfigModules\leftshift
del /Q *.*

copy ..\..\synth\leftshift\shift.ngc
copy ..\..\data\top.ucf
copy ..\..\data\*.nmc
copy ..\..\static\static.used
ren static.used arcs.exclude

ngdbuild -modular module -p xc2vp7-6-ff896 -active shift ../../top/top.ngo
map top.ngd
par -w top.ncd top_routed.ncd

cd..

ECHO IMPLEMENTACION DEL rightshift

cd rightshift
del /Q *.*

copy ..\..\synth\rightshift\shift.ngc
copy ..\..\data\top.ucf
copy ..\..\data\*.nmc
copy ..\..\static\static.used
ren static.used arcs.exclude

ngdbuild -modular module -p xc2vp7-6-ff896 -active shift ../../top/top.ngo
map top.ngd
par -w top.ncd top_routed.ncd
```

```
cd..

ECHO IMPLEMENTACION DEL BOTHSHIFT

cd bothshift
del /Q *.*

copy ..\..\synth\bothshift\shift.ngc
copy ..\..\data\top.ucf
copy ..\..\data\*.nmc
copy ..\..\static\static.used
ren static.used arcs.exclude

ngdbuild -modular module -p xc2vp7-6-ff896 -active shift ../../top/top.ngo
map top.ngd
par -w top.ncd top_routed.ncd

cd..
cd..

ECHO UNIENDO LOS MODULOS

cd merges
del /Q *.*

copy ..\static\top_routed.ncd static.ncd
copy ..\ReconfigModules\leftshift\top_routed.ncd leftshift.ncd
copy ..\ReconfigModules\rightshift\top_routed.ncd rightshift.ncd
copy ..\ReconfigModules\bothshift\top_routed.ncd bothshift.ncd
copy ..\static\system_stub.bmm

echo GENERANDO LOS BITSTREAMS PARCIALES

PR_verifydesign.bat static.ncd leftshift.ncd rightshift.ncd bothshift.ncd
copy PRtmpdir\rightshift_full.bit
copy ..\synth\micro\TestApp_Peripheral\executable.elf
data2mem -bm system_stub_bd.bmm -bt rightshift_full.bit -bd executable.elf tag
lmb_bram -o b download.bit
```

Finalmente una vez obtenidos los *bitstreams* parciales, se podrá proceder a cargar el programa en memoria (el código es similar al utilizado con la ***Virtex-4*** en el apartado 4.7) y a la ejecución del programa reconfigurable, tal y como se hizo en el apartado 4.8.

```
FPGAs PARTIAL RECONFIGURATION
Non-Peripheral partial self-reconfiguration
Virtex 2 pro (xc2vp7)
        Juan Quero Llor

Select an option [1-3]:
         1. Initialize hwicap.
         2. New partial bitstream downloaded to system RAM
         3. Perform Partial Reconfiguration
```

La ***Virtex-II Pro*** también se auto-reconfigura y se reconfigura correctamente, lo que demuestra la fiabilidad del método obtenido. Se puede comprobar en el vídeo demostrativo que se adjunta en el cd.

CAPÍTULO 6
Auto-Reconfiguración Parcial de Periféricos en Virtex-4

6.1. Introducción

Hasta ahora, todos los proyectos realizados con las distintas placas iban enfocados a lograr la auto-reconfiguración parcial utilizando el *MicroBlaze* exclusivamente como reconfigurador, es decir, su única misión era reconfigurar otra parte de la *FPGA*.

En muchos casos la parte importante de un proyecto es un periférico de *MicroBlaze*, por lo que llegados este momento se aprecia la necesidad de conseguir auto-reconfigurar periféricos que estén unidos al propio *MicroBlaze* reconfigurador a través del bus *OPB*.

El ejemplo a realizar es un periférico reconfigurable de *MicroBlaze* que realiza operaciones aritméticas: suma, resta y multiplicación, dependiendo del *bitstream* parcial que se haya descargado. Este periférico se encontrará unido al bus *opb* a través del *opb_socket_brigde*, un *wrapper* que conecta a través de *buses macro* las señales del bus *opb* con las señales que van al periférico.

La placa utilizada fue la *Virtex-4 FX Card* de *Nu Horizonst*, *FPGA xc4vfx12-*10sf363 de Xilinx, con el software *ISE 8.2 Service Pack 1, EDK 8.2, PlanAhead 9.1.1* y *EA_PR5* para *ISE 8.2.01i.*

El diseño auto-reconfigurable realizado es el siguiente:

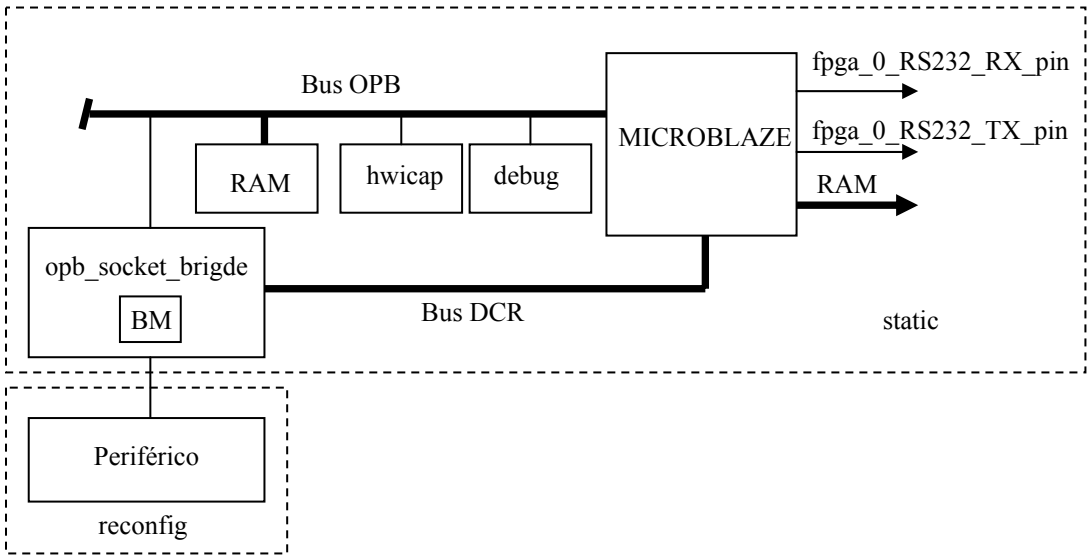


Figura 38. Diagrama de bloques de Auto-Reconfiguración de periféricos con *Virtex-4*

Es imprescindible realizar los cambios necesarios en el archivo ***.mhs*** del proyecto *EDK* que se indicaron en el apartado 4.2, para un correcto funcionamiento de la memoria *RAM* en dicha placa.

6.2. Estructura del Proyecto Reconfigurable:

En primer lugar se procede a la estructuración en carpetas para los distintos pasos de la realización del proyecto:

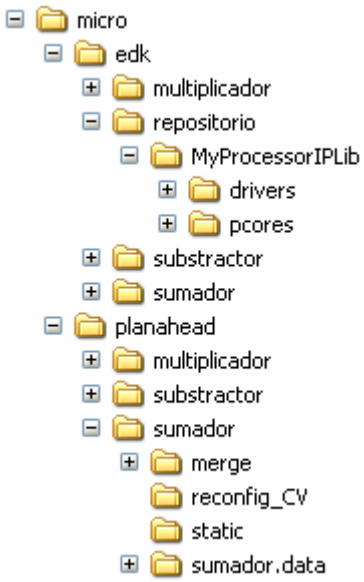


Figura 39. Estructura de directorios para el proyecto reconfigurable con PlanAhead

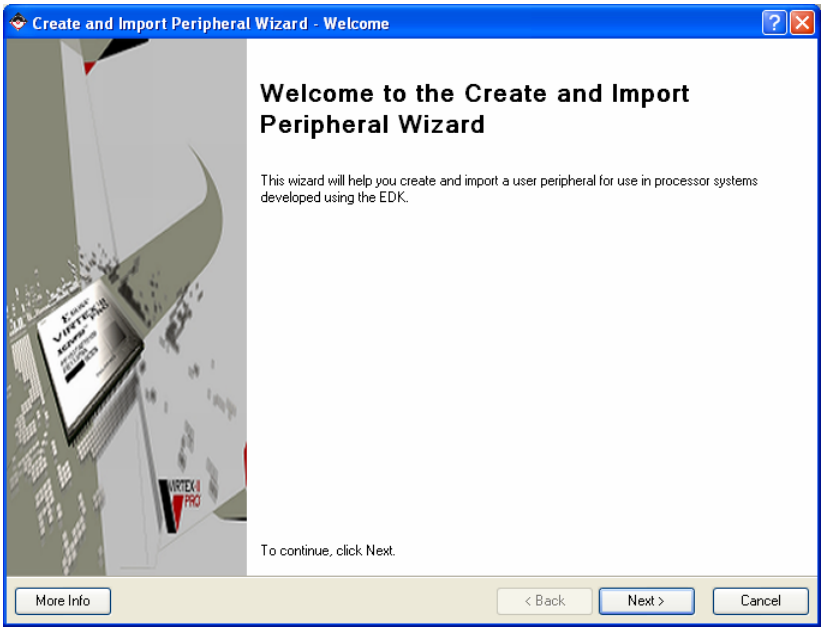
Para ello se crea bajo el directorio principal la carpeta *edk*, que contiene un subdirectorio por cada módulo reconfigurable, en el que se crearán los respectivos proyectos *EDK* para obtener la síntesis de los módulos. También se situará en esta carpeta el repositorio de periféricos.

Además se crea la carpeta *planahead*, que también contiene una carpeta por cada uno de los módulos reconfigurables, que contendrá los respectivos proyectos de *PlanAhead* en los que se hará la implementación y la generación final de los *bitstreams*, que estarán situados en la carpeta *merge*.

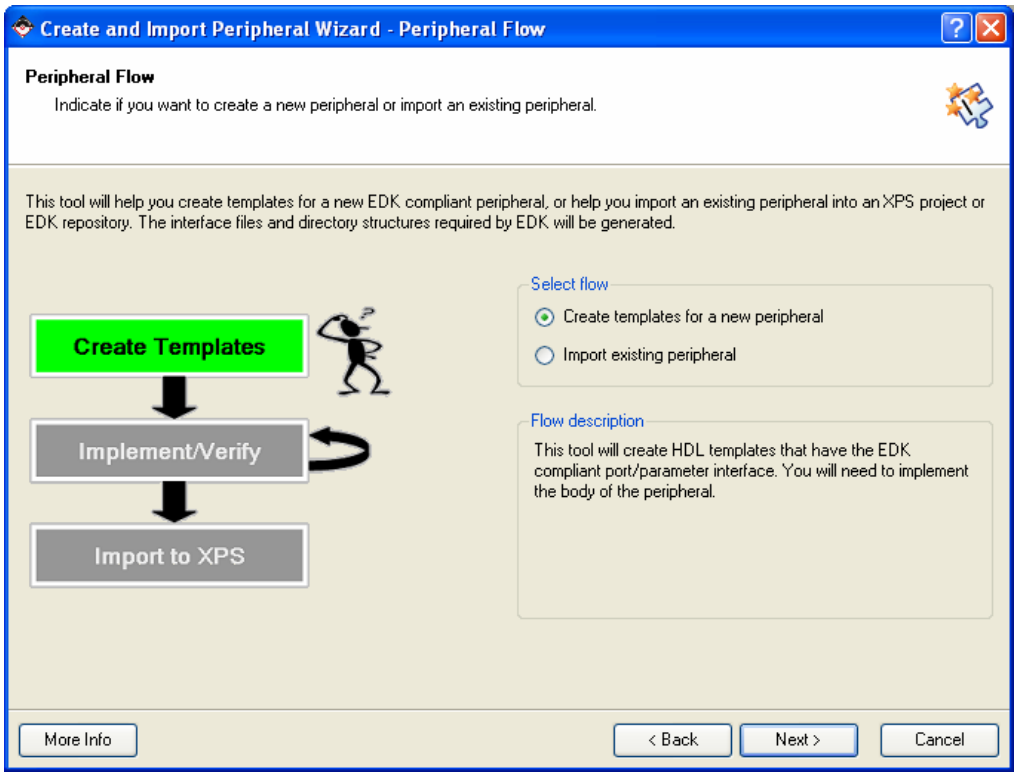
6.3. El proyecto en EDK

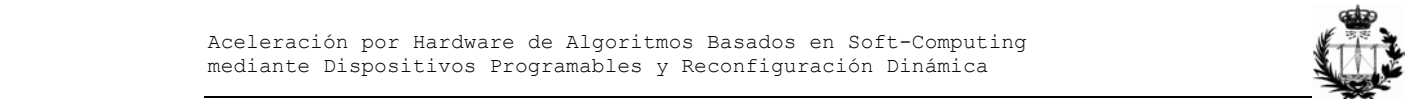
6.3.1. Creación de un Periférico con EDK

En primer lugar se hace clic en el menú Hardware */ Create or Import Peripheral*, de tal forma que se abrirá el *Wizard* de periféricos, donde se pincha en *Next*:

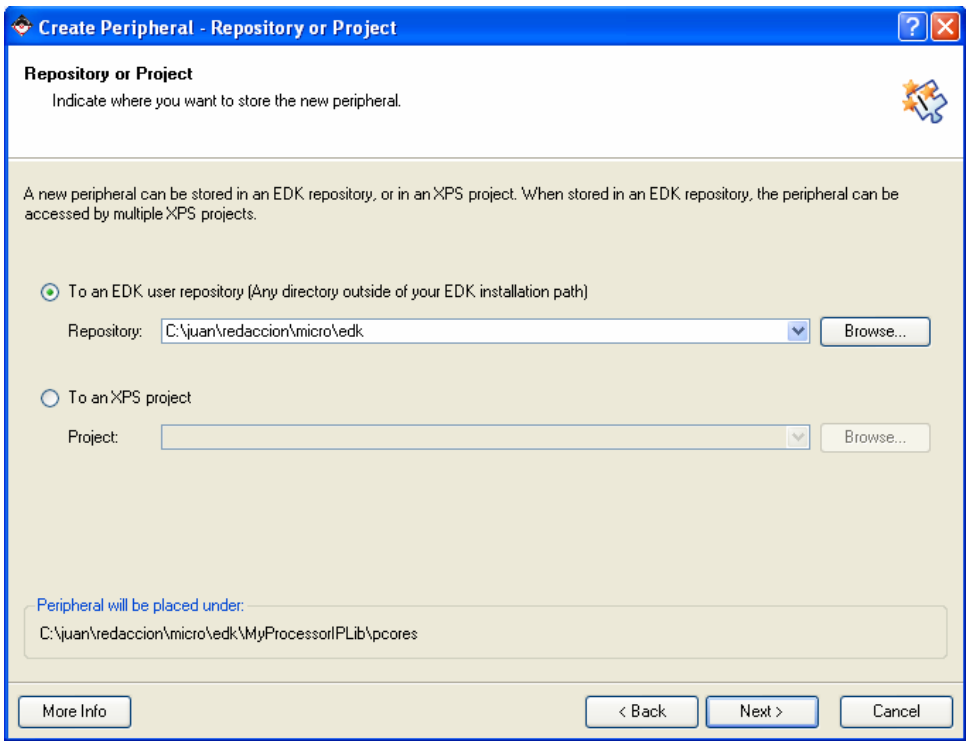


A continuación se selecciona el botón *Create templates for a new peripheral* y se hace clic en *Next:*

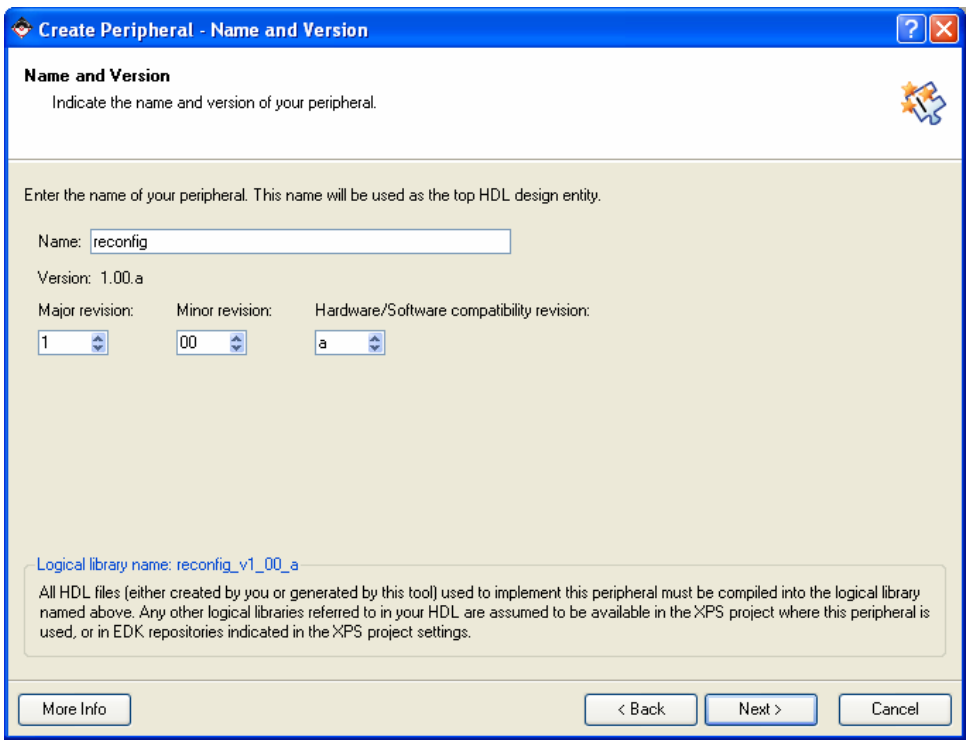


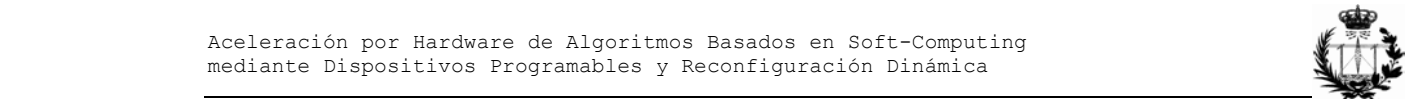


A continuación se puede situar el periférico tanto dentro de un repositorio (lugar donde se almacenan varios periféricos, que puede ser accedido por varios proyectos), como dentro de un proyecto. En este caso se selecciona la primera opción *To an EDK user repository*, e indicamos que el repositorio se guarde dentro del directorio *edk*, haciendo finalmente clic en *Next*:

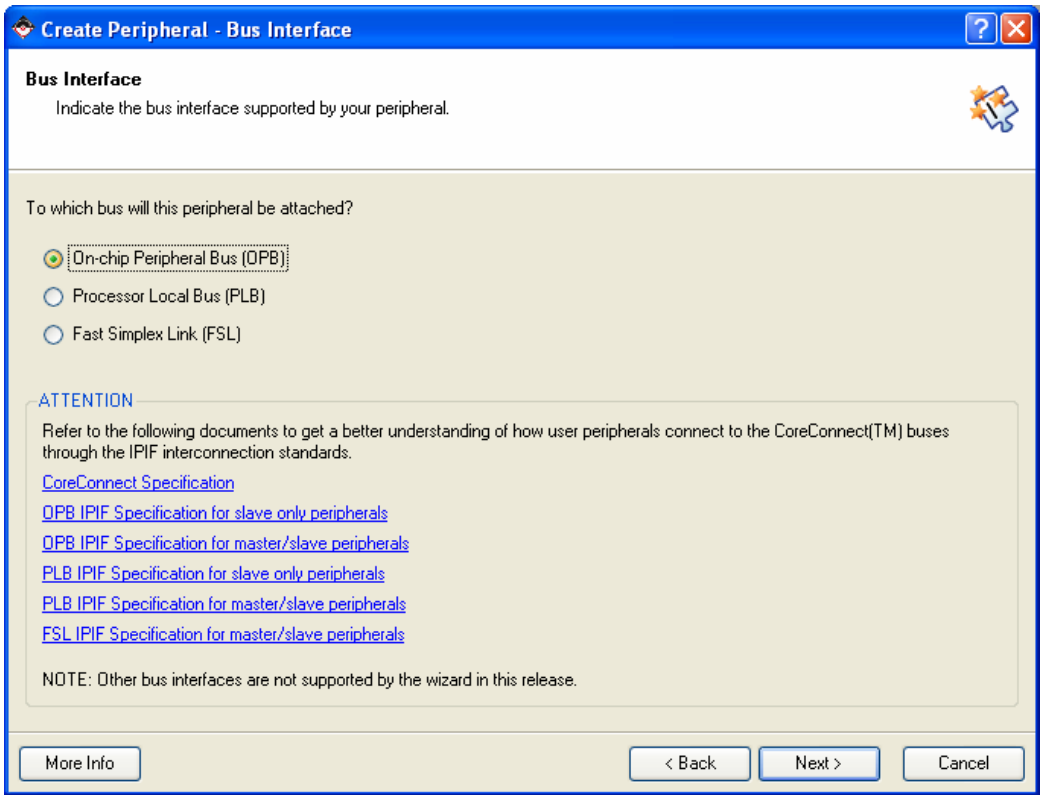


A continuación se le pone nombre al periférico, *reconfig* en este caso y se hace clic en *Next*:

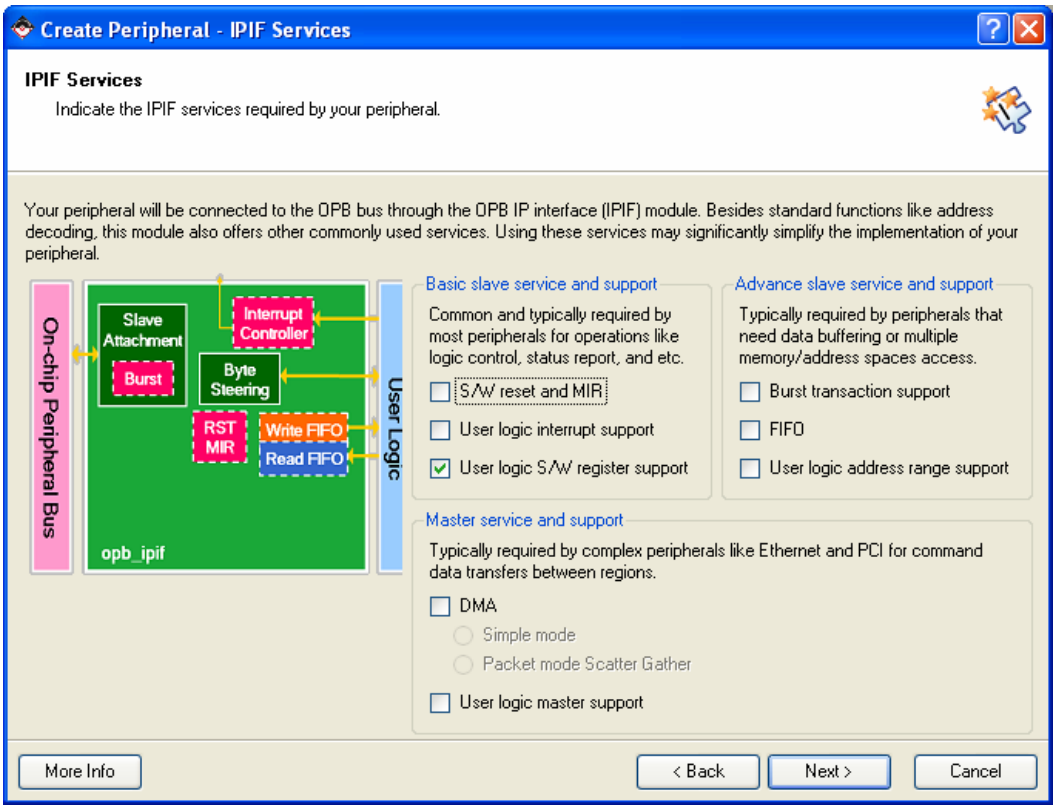




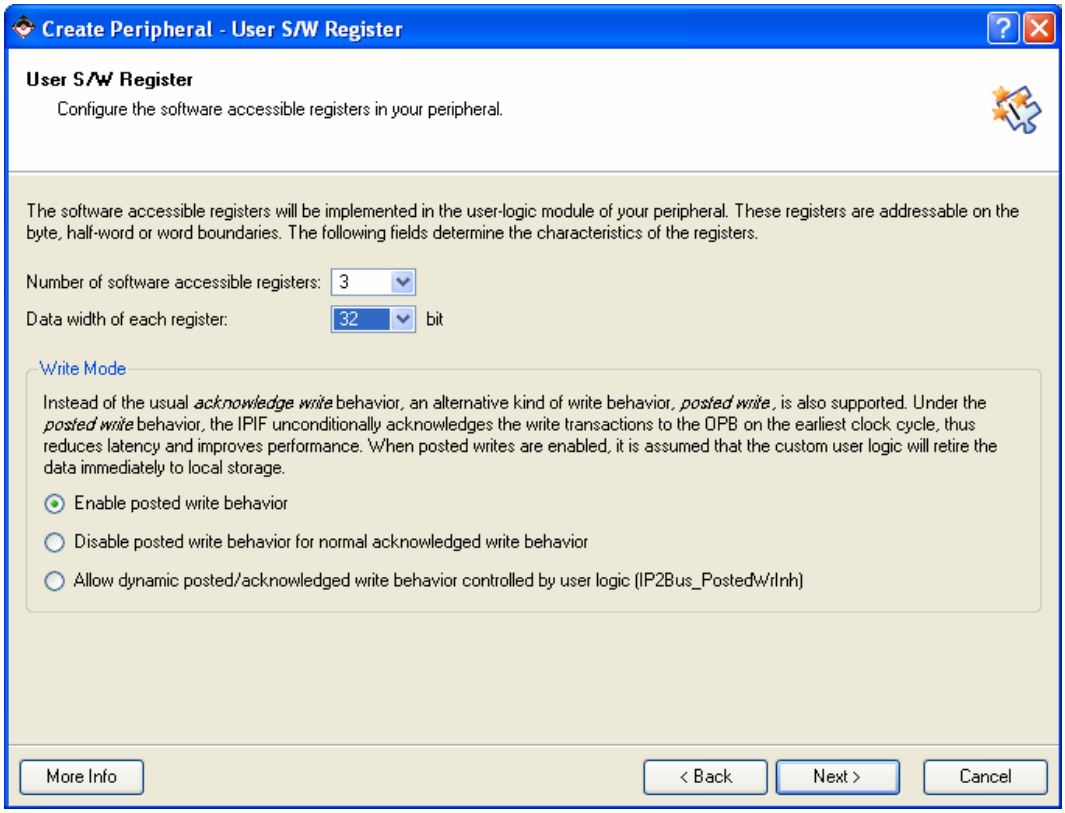
Se selecciona el bus al que va a estar unido el periférico (en este caso al bus *OPB*) y se hace clic en *Next*:



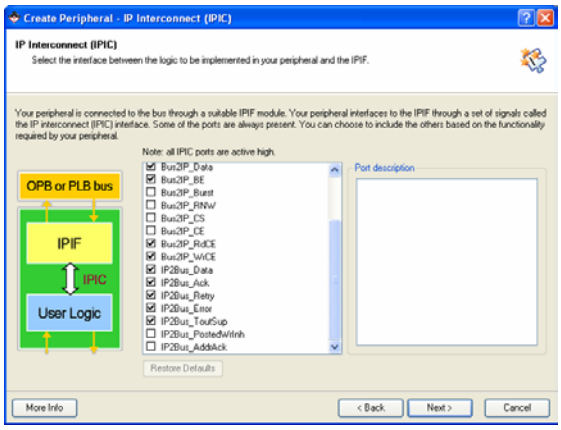
Se deseleccionan las casillas *S/W reset and MIR* y *User logic Interrupt support*, dejando solamente la casilla *User logic S/W register support*. Y se hace clic en *Next.*

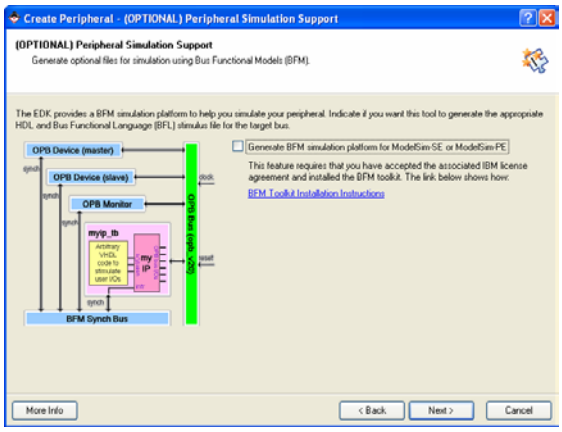


Se seleccionan 3 registros (*Number of software accesible registers*) de 32 bits de tamaño de palabra (*Data width of each register*) y se hace clic en *Next*.

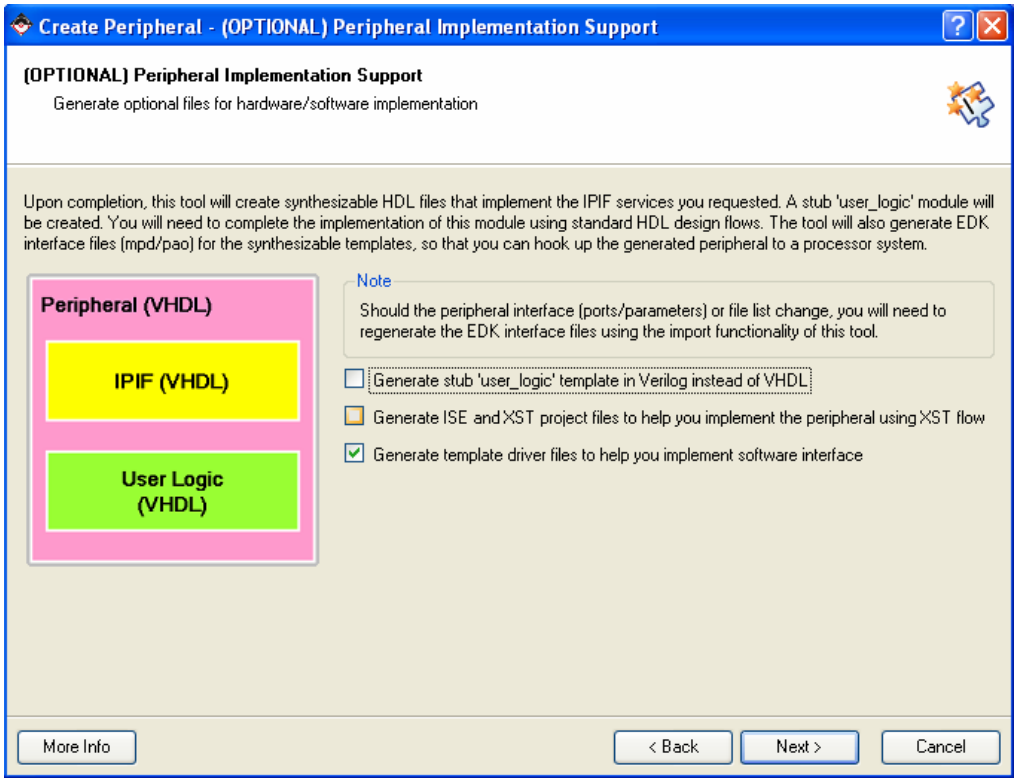


Las siguientes dos ventanas se dejan tal cual están, haciendo clic en ambas en *Next*:

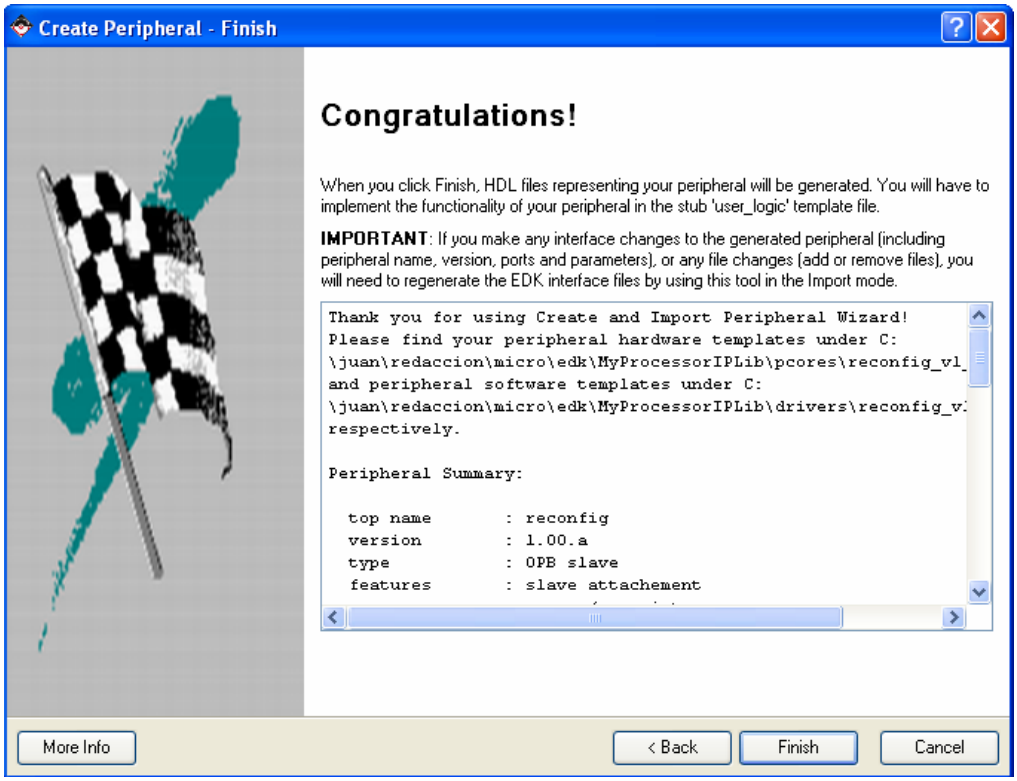


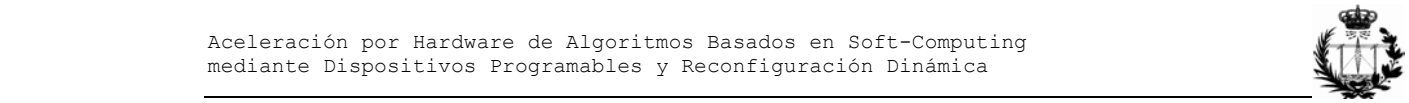


Se deselecciona *Generate ISE and XST project files to help you implement the peripheral using XST flow*, ya que posteriormente añadiremos el periférico a un proyecto *EDK*, y se hace clic en *Next*.



Para concluir se hace clic en *Finish*:





Dentro del directorio *edk* se encuentra el repositorio de periféricos *MyProcessorIPLib*. Accediendo a él hay dos carpetas: *drivers* (que contiene los archivos de código *C++* de los periféricos y sus librerías) y *pcores* (que contiene el código *VHDL*).

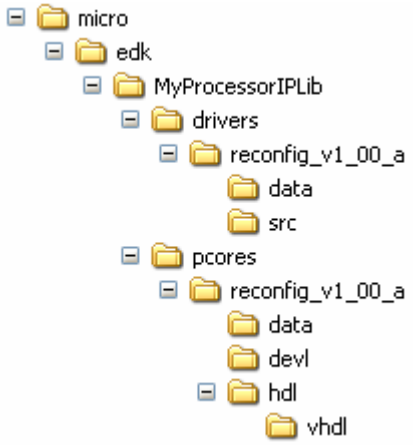


Figura 40. Estructura de directorios del repositorio de periféricos

Entrando en *pcores/reconfig_v1_00_a/hdl/vhdl* se encuentran los dos archivos *VHDL* del periférico:

- *reconfig.vhd*: *wrapper* en el que está instanciado el *IPIF*, un módulo que se utiliza para el correcto direccionamiento entre el bus *OPB* y el periférico.
- *userlogic.vhd*: archivo en el que se escribe el código correspondiente a la lógica que deseamos implementar.

En la carpeta *drivers/reconfig_v1_00_a/src* se encuentran los *drivers* del periférico, y en él hay que modificar el archivo *reconfig.c,* para adaptarlo a las necesidades del proyecto reconfigurable.

6.3.2. Adaptación de un Periférico Genérico al opb_socket_brigde:

Los periféricos estándar que se generan en el entorno *EDK*, se crean con las señales de entrada y salida necesarias para unirse al bus correspondiente, en este caso al bus *OPB*. En el archivo ***.mpd*** del periférico aparece indicado qué puertos pueden unirse a qué bus, para que, al agregar una instancia del periférico al proyecto *EDK*, se puedan conectar.

Pero en este caso, el periférico irá conectado al bus *OPB* a través del módulo *opb_socket_brigde*, en el que irán los *buses macro*, por lo tanto no deberá aparecer en el entorno *EDK* una posible conexión entre el módulo reconfigurable y el bus *OPB*.

Para ello modificar esta cuestión se debe editar el archivo ***.mpd***, situado en el directorio *pcores\reconfig_v1_00_a\data\reconfig_v2_1_0.mpd*

```
##############################################################
##
## Name      : reconfig
## Desc      : Microprocessor Peripheral Description
##           : Automatically generated by PsfUtility
##
```

```
##############################################################

BEGIN reconfig

## Peripheral Options
OPTION IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
OPTION HDL = VHDL
OPTION IP_GROUP = MICROBLAZE:PPC:USER
OPTION CORE_STATE = DEVELOPMENT

## Bus Interfaces
BUS_INTERFACE BUS = SOPB, BUS_TYPE = SLAVE, BUS_STD = OPB

## Generics for VHDL or Parameters for Verilog
PARAMETER C_BASEADDR = 0xffffffff, DT = std_logic_vector, MIN_SIZE = 0x100,
BUS = SOPB, ADDRESS = BASE, PAIR = C_HIGHADDR
PARAMETER C_HIGHADDR = 0x00000000, DT = std_logic_vector, BUS = SOPB, ADDRESS
= HIGH, PAIR = C_BASEADDR
PARAMETER C_OPB_AWIDTH = 32, DT = INTEGER, BUS = SOPB
PARAMETER C_OPB_DWIDTH = 32, DT = INTEGER, BUS = SOPB
PARAMETER C_FAMILY = virtex2p, DT = STRING

## Ports
PORT OPB_Clk = "", DIR = I, SIGIS = Clk, BUS = SOPB
PORT OPB_Rst = OPB_Rst, DIR = I, SIGIS = Rst, BUS = SOPB
PORT Sl_DBus = Sl_DBus, DIR = O, VEC = [0:(C_OPB_DWIDTH-1)], BUS = SOPB
PORT Sl_errAck = Sl_errAck, DIR = O, BUS = SOPB
PORT Sl_retry = Sl_retry, DIR = O, BUS = SOPB
PORT Sl_toutSup = Sl_toutSup, DIR = O, BUS = SOPB
PORT Sl_xferAck = Sl_xferAck, DIR = O, BUS = SOPB
PORT OPB_ABus = OPB_ABus, DIR = I, VEC = [0:(C_OPB_AWIDTH-1)], BUS = SOPB
PORT OPB_BE = OPB_BE, DIR = I, VEC = [0:((C_OPB_DWIDTH/8)-1)], BUS = SOPB
PORT OPB_DBus = OPB_DBus, DIR = I, VEC = [0:(C_OPB_DWIDTH-1)], BUS = SOPB
PORT OPB_RNW = OPB_RNW, DIR = I, BUS = SOPB
PORT OPB_select = OPB_select, DIR = I, BUS = SOPB
PORT OPB_seqAddr = OPB_seqAddr, DIR = I, BUS = SOPB

END
```

Como se puede observar, aparece una instancia al bus *opb*, y en todos los puertos se indica la conexión mediante *BUS = SOPB*. Para evitarlo habrá que comentarlas, quedando de la siguiente forma (modificaciones en rojo):

```
##############################################################################
##
## Name      : reconfig
## Desc      : Microprocessor Peripheral Description
##           : Automatically generated by PsfUtility
##
##############################################################################

BEGIN reconfig

## Peripheral Options
OPTION IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
OPTION HDL = VHDL
OPTION IP_GROUP = MICROBLAZE:PPC:USER
OPTION CORE_STATE = DEVELOPMENT

## Bus Interfaces
#BUS_INTERFACE BUS = SOPB, BUS_TYPE = SLAVE, BUS_STD = OPB
```

```
## Generics for VHDL or Parameters for Verilog
PARAMETER C_BASEADDR = 0xffffffff, DT = std_logic_vector, MIN_SIZE = 0x100#,
BUS = SOPB, ADDRESS = BASE, PAIR = C_HIGHADDR
PARAMETER C_HIGHADDR = 0x00000000, DT = std_logic_vector#, BUS = SOPB, ADDRESS
= HIGH, PAIR = C_BASEADDR
PARAMETER C_OPB_AWIDTH = 32, DT = INTEGER#, BUS = SOPB
PARAMETER C_OPB_DWIDTH = 32, DT = INTEGER#, BUS = SOPB
PARAMETER C_FAMILY = virtex2p, DT = STRING

## Ports
PORT OPB_Clk = "", DIR = I, SIGIS = Clk#, BUS = SOPB
PORT OPB_Rst = OPB_Rst, DIR = I, SIGIS = Rst#, BUS = SOPB
PORT Sl_DBus = Sl_DBus, DIR = O, VEC = [0:(C_OPB_DWIDTH-1)]#, BUS = SOPB
PORT Sl_errAck = Sl_errAck, DIR = O#, BUS = SOPB
PORT Sl_retry = Sl_retry, DIR = O#, BUS = SOPB
PORT Sl_toutSup = Sl_toutSup, DIR = O#, BUS = SOPB
PORT Sl_xferAck = Sl_xferAck, DIR = O#, BUS = SOPB
PORT OPB_ABus = OPB_ABus, DIR = I, VEC = [0:(C_OPB_AWIDTH-1)]#, BUS = SOPB
PORT OPB_BE = OPB_BE, DIR = I, VEC = [0:((C_OPB_DWIDTH/8)-1)]#, BUS = SOPB
PORT OPB_DBus = OPB_DBus, DIR = I, VEC = [0:(C_OPB_DWIDTH-1)]#, BUS = SOPB
PORT OPB_RNW = OPB_RNW, DIR = I#, BUS = SOPB
PORT OPB_select = OPB_select, DIR = I#, BUS = SOPB
PORT OPB_seqAddr = OPB_seqAddr, DIR = I#, BUS = SOPB

END
```

6.3.3. Adaptación del opb_socket_brigde para un Periférico Creado con el Wizard

Debido a problemas de direccionamiento, es necesario adaptar el *opb_socket_brigde* aportado por *Xilinx*, para un correcto funcionamiento con periféricos creados con el *wizard* del *EDK*. Estos periféricos disponen de *IPIF*, un decodificador del bus *opb*, que hace que se decodifiquen mal las señales una vez que han sido previamente decodificadas en el *opb_socket_brigde* proporcionado por *Xilinx*.

El código vhdl del *opb_socket_brigde* proporcionado por *Xilinx* en las herramientas *EA_PR* es el siguiente (archivo *pcores\opb_socket_bridge\hdl\vhdl\opb_socket.vhd*):

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

library common_v1_00_a;
use common_v1_00_a.pselect;

entity opb_socket_bridge is
  generic (
    C_OPB_AWIDTH      : integer                   := 32;
    C_OPB_DWIDTH      : integer                   := 32;
    C_BASEADDR        : std_logic_vector(0 to 31) := X"FFFF_0000";
    C_HIGHADDR        : std_logic_vector          := X"FFFF_FFFF";
       C_MASKADDR               : std_logic_vector              :=
X"0000_FFFF";
    C_DCR_BASEADDR    : std_logic_vector(0 to 9)  := B"00_0000_0000");
  port (
    status      : out std_logic_vector(0 to 3);

    DCR_ABus    : in  std_logic_vector(0 to 9);
    DCR_Sl_DBus : in  std_logic_vector(0 to 31);
    DCR_Read    : in  std_logic;
    DCR_Write   : in  std_logic;
```

```
    Sl_dcrAck   : out std_logic;
    Sl_dcrDBus  : out std_logic_vector(0 to 31);

    OPB_ABus     : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE       : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    OPB_Clk      : in  std_logic;
    OPB_DBus     : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW      : in  std_logic;
    OPB_Rst      : in  std_logic;
    OPB_select   : in  std_logic;
    OPB_seqAddr  : in  std_logic;

    Sln_DBus     : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sln_errAck   : out std_logic;
    Sln_retry    : out std_logic;
    Sln_toutSup  : out std_logic;
    Sln_xferAck  : out std_logic;

    -- Interrupt signals
    Interrupt    : out std_logic;

    ROPB_ABus     : out  std_logic_vector(0 to C_OPB_AWIDTH-1);
    ROPB_BE       : out  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    ROPB_Clk      : out  std_logic;
    ROPB_DBus     : out  std_logic_vector(0 to C_OPB_DWIDTH-1);
    ROPB_RNW      : out  std_logic;
    ROPB_Rst      : out  std_logic;
    ROPB_select   : out  std_logic;
    ROPB_seqAddr  : out  std_logic;

    RSln_DBus     : in std_logic_vector(0 to C_OPB_DWIDTH-1);
    RSln_errAck   : in std_logic;
    RSln_retry    : in std_logic;
    RSln_toutSup  : in std_logic;
    RSln_xferAck  : in std_logic;

    -- Interrupt signals
    RInterrupt    : in std_logic
    );


  attribute MIN_SIZE : string;
  attribute MIN_SIZE of C_BASEADDR : constant is "0x100";

  attribute SIGIS : string;
  attribute SIGIS of OPB_Clk : signal is "Clk";
  attribute SIGIS of OPB_Rst : signal is "Rst";

end entity opb_socket_bridge;

--library unisim;
--use unisim.all;

architecture IMP of opb_socket_bridge is

  component dcr_if is
  generic (
    C_DCR_BASEADDR : std_logic_vector(0 to 9) := B"00_0000_0000";
    C_ON_INIT : std_logic := '0');
  port (
    clk         : in  std_logic;
    rst         : in  std_logic;
    DCR_ABus    : in  std_logic_vector(0 to 9);
    DCR_Sl_DBus : in  std_logic_vector(0 to 31);
    DCR_Read    : in  std_logic;
    DCR_Write   : in  std_logic;
    Sl_dcrAck   : out std_logic;
    Sl_dcrDBus  : out std_logic_vector(0 to 31);
```

```
    ctrl_reg    : out std_logic_vector(0 to 31));
  end component;

  component pselect is
    generic (
      C_AB  : integer;
      C_AW  : integer;
      C_BAR : std_logic_vector);
    port (
      A      : in  std_logic_vector(0 to C_AW-1);
      AValid : in  std_logic;
      ps     : out std_logic);
  end component pselect;

  function Addr_Bits (x, y : std_logic_vector(0 to C_OPB_AWIDTH-1)) return
integer is
    variable addr_nor : std_logic_vector(0 to C_OPB_AWIDTH-1);
  begin
    addr_nor := x xor y;
    for i in 0 to C_OPB_AWIDTH-1 loop
      if addr_nor(i) = '1' then return i;
      end if;
    end loop;
    return(C_OPB_AWIDTH);
  end function Addr_Bits;

  constant C_AB : integer := Addr_Bits(C_HIGHADDR, C_BASEADDR);

       component busmacro_vector8_xc4v_r2l_async_narrow is
       port (
             macro_in  : in  std_logic_vector(7 downto 0);
             macro_out : out std_logic_vector(7 downto 0)
       );
       end component;


       component busmacro_vector8_xc4v_l2r_async_narrow is
       port (
             macro_in  : in  std_logic_vector(7 downto 0);
             macro_out : out std_logic_vector(7 downto 0)
       );
       end component;

       component busmacro_vector8_xc4v_r2l_async_enable_narrow is
       port (
             macro_in  : in  std_logic_vector(7 downto 0);
             enable : in  std_logic_vector(7 downto 0);
             macro_out : out std_logic_vector(7 downto 0)
       );
       end component;


       component busmacro_vector8_xc4v_l2r_async_enable_narrow is
       port (
             macro_in  : in  std_logic_vector(7 downto 0);
             enable : in  std_logic_vector(7 downto 0);
             macro_out : out std_logic_vector(7 downto 0)
       );
       end component;

  signal ctrl_reg        : std_logic_vector(0 to 31);
  signal busmacro_enable : std_logic;
  signal socket_reset    : std_logic;

  signal OPB_ABus_pr     :   std_logic_vector(0 to C_OPB_AWIDTH-1);
  signal OPB_BE_pr       :   std_logic_vector(0 to C_OPB_DWIDTH/8-1);
  signal OPB_DBus_pr     :   std_logic_vector(0 to C_OPB_DWIDTH-1);
  signal OPB_RNW_pr      :   std_logic;
```

```
  signal OPB_Rst_pr      :   std_logic;
  signal OPB_select_pr   :   std_logic;
  signal OPB_seqAddr_pr  :   std_logic;

  signal Sln_DBus_pr     :  std_logic_vector(0 to C_OPB_DWIDTH-1);
  signal Sln_errAck_pr   :  std_logic;
  signal Sln_retry_pr    :  std_logic;
  signal Sln_toutSup_pr  :  std_logic;
  signal Sln_xferAck_pr  :  std_logic;

  signal Interrupt_pr    : std_logic;
  signal External_IO     : std_logic_vector(0 to 63);
  signal External_I      : std_logic_vector(0 to 63);
  signal External_O      : std_logic_vector(0 to 63);

  signal busmacro_enable_vector   : std_logic_vector(7 downto 0);
  signal dummy1          :  std_logic_vector(3 downto 0);
  signal dummy2          :  std_logic_vector(5 downto 0);
  signal dummy3          :  std_logic_vector(3 downto 0);
  signal dummy4          :  std_logic_vector(6 downto 0);

  signal Socket2PRR_ABus :  std_logic_vector(0 to C_OPB_AWIDTH-1);
  signal MY_ADDR_SPACE     :      std_logic;

begin

  -- Do the OPB address decoding
  pselect_I : pselect
    generic map (
      C_AB  => C_AB,                    -- [integer]
      C_AW  => C_OPB_AWIDTH,            -- [integer]
      C_BAR => C_BASEADDR)              -- [std_logic_vector]
    port map (
      A      => OPB_ABus,               -- [in  std_logic_vector(0 to C_AW-1)]
      AValid => OPB_select,             -- [in  std_logic]
      ps     => MY_ADDR_SPACE);         -- [out std_logic] to PRR as P_SELECT

  -- Do the address translating for the OPB Peripheral
  -- For PRR region mask out the most n-bits according to C_MASKADDR and then
send to PRR
  Socket2PRR_ABus <= OPB_ABUS & C_MASKADDR;

  -- dcr interface instantiation
  dcr_control: dcr_if
    generic map (
      C_DCR_BASEADDR => C_DCR_BASEADDR)
    port map (
        clk         => OPB_clk,
        rst         => OPB_Rst,
        DCR_ABus    => DCR_ABus,
        DCR_Sl_DBus => DCR_Sl_DBus,
        DCR_Read    => DCR_Read,
        DCR_Write   => DCR_Write,
        Sl_dcrAck   => Sl_dcrAck,
        Sl_dcrDBus  => Sl_dcrDBus,
        ctrl_reg    => ctrl_reg);

  -- bus macro related signals and instantiations
  status(0) <= not(busmacro_enable);
  status(1) <= not(socket_reset);
  status(2) <= '1';
  status(3) <= '1';

  busmacro_enable <= ctrl_reg(31);
  socket_reset <= ctrl_reg(30) or OPB_Rst;

  busmacro_enable_vector <= (busmacro_enable,
                    busmacro_enable,
```

```
                    busmacro_enable,
                    busmacro_enable,
                    busmacro_enable,
                    busmacro_enable,
                    busmacro_enable,
                    busmacro_enable);



  ROPB_ABus <= OPB_ABus_pr;
  ROPB_BE <= OPB_BE_pr;
  ROPB_Clk <= OPB_Clk;
  ROPB_DBus <= OPB_DBus_pr;
  ROPB_RNW <= OPB_RNW_pr;
  ROPB_Rst <= OPB_Rst_pr;
  ROPB_select <= OPB_select_pr;
  ROPB_seqAddr <= OPB_seqAddr_pr;

  Sln_DBus_pr <= RSln_DBus;
  Sln_errAck_pr <= RSln_errAck;
  Sln_retry_pr <= RSln_retry;
  Sln_toutSup_pr <= RSln_toutSup;
  Sln_xferAck_pr <= RSln_xferAck;

  -- Interrupt signals
  Interrupt <= '0';

  Control1_BM : busmacro_vector8_xc4v_l2r_async_narrow
    port map (
                    macro_in(7)  => MY_ADDR_SPACE,
                    macro_in(6)  => OPB_RNW,
                    macro_in(5)  => socket_reset,
                    macro_in(4)  => OPB_seqAddr,
                    macro_in(3 downto 0)  => OPB_BE(0 to 3),
                    macro_out(7) => OPB_select_pr,
                    macro_out(6) => OPB_RNW_pr,
                    macro_out(5) => OPB_Rst_pr,
                    macro_out(4) => OPB_seqAddr_pr,
                    macro_out(3 downto 0) => OPB_BE_pr(0 to 3)
                    );

  Control2_BM : busmacro_vector8_xc4v_r2l_async_enable_narrow
    port map (
      enable => busmacro_enable_vector,
      macro_in(7)  => '0',
      macro_in(6)  => '0',
      macro_in(5)  => '0',
      macro_in(4)  => '0',
      macro_in(3)  => Sln_xferAck_pr,
      macro_in(2)  => Sln_toutSup_pr,
                    macro_in(1)  => Sln_retry_pr,
      macro_in(0)  => Sln_errAck_pr,
                    macro_out(7 downto 4) => dummy1,
      macro_out(3) => Sln_xferAck,
                    macro_out(2) => Sln_toutSup,
      macro_out(1) => Sln_retry,
                    macro_out(0) => Sln_errAck
                    );

  ABUS_BM_GENERATE: for i in 0 to C_OPB_AWIDTH/8-1 generate
    ABUS_BM : busmacro_vector8_xc4v_l2r_async_narrow
      port map (
        macro_in(7)  => Socket2PRR_ABus(2*i+0),
        macro_in(6)  => Socket2PRR_ABus(2*i+1),
        macro_in(5)  => Socket2PRR_ABus(2*i+8),
        macro_in(4)  => Socket2PRR_ABus(2*i+9),
        macro_in(3)  => Socket2PRR_ABus(2*i+16),
        macro_in(2)  => Socket2PRR_ABus(2*i+17),
```

```
        macro_in(1)  => Socket2PRR_ABus(2*i+24),
        macro_in(0)  => Socket2PRR_ABus(2*i+25),
        macro_out(7)  => OPB_ABus_pr(2*i+0),
        macro_out(6)  => OPB_ABus_pr(2*i+1),
        macro_out(5)  => OPB_ABus_pr(2*i+8),
        macro_out(4)  => OPB_ABus_pr(2*i+9),
        macro_out(3)  => OPB_ABus_pr(2*i+16),
        macro_out(2)  => OPB_ABus_pr(2*i+17),
        macro_out(1)  => OPB_ABus_pr(2*i+24),
        macro_out(0)  => OPB_ABus_pr(2*i+25)
        );
  end generate ABUS_BM_GENERATE;

  DBUS_BM_GENERATE: for i in 0 to C_OPB_DWIDTH/8-1 generate
    DBUS_BM : busmacro_vector8_xc4v_r2l_async_enable_narrow
      port map (
        enable => busmacro_enable_vector,
        macro_in(7 downto 0)  => Sln_DBus_pr(i*8 to i*8+7),
        macro_out(7 downto 0) => Sln_DBus(i*8 to i*8+7)
        );
  end generate DBUS_BM_GENERATE;

  SDBUS_BM_GENERATE: for i in 0 to C_OPB_DWIDTH/8-1 generate
    SDBUS_BM : busmacro_vector8_xc4v_l2r_async_narrow
      port map (
        macro_in(7)  => OPB_DBus(2*i+0),
        macro_in(6)  => OPB_DBus(2*i+1),
        macro_in(5)  => OPB_DBus(2*i+8),
        macro_in(4)  => OPB_DBus(2*i+9),
        macro_in(3)  => OPB_DBus(2*i+16),
        macro_in(2)  => OPB_DBus(2*i+17),
        macro_in(1)  => OPB_DBus(2*i+24),
        macro_in(0)  => OPB_DBus(2*i+25),
        macro_out(7)  => OPB_DBus_pr(2*i+0),
        macro_out(6)  => OPB_DBus_pr(2*i+1),
        macro_out(5)  => OPB_DBus_pr(2*i+8),
        macro_out(4)  => OPB_DBus_pr(2*i+9),
        macro_out(3)  => OPB_DBus_pr(2*i+16),
        macro_out(2)  => OPB_DBus_pr(2*i+17),
        macro_out(1)  => OPB_DBus_pr(2*i+24),
        macro_out(0)  => OPB_DBus_pr(2*i+25)
        );
  end generate SDBUS_BM_GENERATE;

end architecture IMP;
```

Este código se debe editar para eliminar toda la parte de direccionamiento que originalmente realiza, y dejarlo como un módulo que hace pasar las señales de entrada por los *buses macro* (controlados por el bus *dcr*) y las envía por la salida. El código resultante es el siguiente, marcando en rojo las líneas añadidas:

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

library common_v1_00_a;
use common_v1_00_a.pselect;

entity opb_socket_bridge is
  generic (
    C_OPB_AWIDTH      : integer                     := 32;
    C_OPB_DWIDTH      : integer                     := 32;
    C_BASEADDR        : std_logic_vector(0 to 31)  := X"FFFF_0000";
```

```
    C_HIGHADDR        : std_logic_vector          := X"FFFF_FFFF";
       C_MASKADDR               : std_logic_vector               :=
X"0000_FFFF";
    C_DCR_BASEADDR    : std_logic_vector(0 to 9)  := B"00_0000_0000");
  port (
    status      : out std_logic_vector(0 to 3);

    DCR_ABus    : in  std_logic_vector(0 to 9);
    DCR_Sl_DBus : in  std_logic_vector(0 to 31);
    DCR_Read    : in  std_logic;
    DCR_Write   : in  std_logic;
    Sl_dcrAck   : out std_logic;
    Sl_dcrDBus  : out std_logic_vector(0 to 31);

    OPB_ABus     : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE       : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    OPB_Clk      : in  std_logic;
    OPB_DBus     : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW      : in  std_logic;
    OPB_Rst      : in  std_logic;
    OPB_select   : in  std_logic;
    OPB_seqAddr  : in  std_logic;

    Sln_DBus     : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sln_errAck   : out std_logic;
    Sln_retry    : out std_logic;
    Sln_toutSup  : out std_logic;
    Sln_xferAck  : out std_logic;

    -- Interrupt signals
    Interrupt    : out std_logic;

    ROPB_ABus     : out  std_logic_vector(0 to C_OPB_AWIDTH-1);
    ROPB_BE       : out  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    ROPB_Clk      : out  std_logic;
    ROPB_DBus     : out  std_logic_vector(0 to C_OPB_DWIDTH-1);
    ROPB_RNW      : out  std_logic;
    ROPB_Rst      : out  std_logic;
    ROPB_select   : out  std_logic;
    ROPB_seqAddr  : out  std_logic;

    RSln_DBus     : in std_logic_vector(0 to C_OPB_DWIDTH-1);
    RSln_errAck   : in std_logic;
    RSln_retry    : in std_logic;
    RSln_toutSup  : in std_logic;
    RSln_xferAck  : in std_logic;

    -- Interrupt signals
    RInterrupt    : in std_logic
    );


  attribute MIN_SIZE : string;
  attribute MIN_SIZE of C_BASEADDR : constant is "0x100";

  attribute SIGIS : string;
  attribute SIGIS of OPB_Clk : signal is "Clk";
  attribute SIGIS of OPB_Rst : signal is "Rst";

end entity opb_socket_bridge;

--library unisim;
--use unisim.all;

architecture IMP of opb_socket_bridge is

  component dcr_if is
  generic (
```

```
    C_DCR_BASEADDR : std_logic_vector(0 to 9) := B"00_0000_0000";
    C_ON_INIT : std_logic := '0');
  port (
    clk         : in  std_logic;
    rst         : in  std_logic;
    DCR_ABus    : in  std_logic_vector(0 to 9);
    DCR_Sl_DBus : in  std_logic_vector(0 to 31);
    DCR_Read    : in  std_logic;
    DCR_Write   : in  std_logic;
    Sl_dcrAck   : out std_logic;
    Sl_dcrDBus  : out std_logic_vector(0 to 31);
    ctrl_reg    : out std_logic_vector(0 to 31));
  end component;

      component busmacro_vector8_xc4v_r2l_async_narrow is
      port (
            macro_in  : in  std_logic_vector(7 downto 0);
            macro_out : out std_logic_vector(7 downto 0)
      );
      end component;


      component busmacro_vector8_xc4v_l2r_async_narrow is
      port (
            macro_in  : in  std_logic_vector(7 downto 0);
            macro_out : out std_logic_vector(7 downto 0)
      );
      end component;

      component busmacro_vector8_xc4v_r2l_async_enable_narrow is
      port (
            macro_in  : in  std_logic_vector(7 downto 0);
            enable : in  std_logic_vector(7 downto 0);
            macro_out : out std_logic_vector(7 downto 0)
      );
      end component;

      component busmacro_vector8_xc4v_l2r_async_enable_narrow is
      port (
            macro_in  : in  std_logic_vector(7 downto 0);
            enable : in  std_logic_vector(7 downto 0);
            macro_out : out std_logic_vector(7 downto 0)
      );
      end component;

  signal ctrl_reg        : std_logic_vector(0 to 31);
  signal busmacro_enable : std_logic;
  signal socket_reset    : std_logic;

  signal OPB_ABus_pr     :   std_logic_vector(0 to C_OPB_AWIDTH-1);
  signal OPB_BE_pr       :   std_logic_vector(0 to C_OPB_DWIDTH/8-1);
  signal OPB_DBus_pr     :   std_logic_vector(0 to C_OPB_DWIDTH-1);
  signal OPB_RNW_pr      :   std_logic;
  signal OPB_Rst_pr      :   std_logic;
  signal OPB_select_pr   :   std_logic;
  signal OPB_seqAddr_pr  :   std_logic;

  signal Sln_DBus_pr     :  std_logic_vector(0 to C_OPB_DWIDTH-1);
  signal Sln_errAck_pr   :  std_logic;
  signal Sln_retry_pr    :  std_logic;
  signal Sln_toutSup_pr  :  std_logic;
  signal Sln_xferAck_pr  :  std_logic;

  signal Interrupt_pr    : std_logic;
  signal External_IO     : std_logic_vector(0 to 63);
  signal External_I      : std_logic_vector(0 to 63);
  signal External_O      : std_logic_vector(0 to 63);
```

```
  signal busmacro_enable_vector   : std_logic_vector(7 downto 0);
  signal dummy1           :  std_logic_vector(2 downto 0);
  signal dummy2           :  std_logic_vector(5 downto 0);
  signal dummy3           :  std_logic_vector(3 downto 0);
  signal dummy4           :  std_logic_vector(6 downto 0);

begin

  -- dcr interface instantiation
  dcr_control: dcr_if
    generic map (
      C_DCR_BASEADDR => C_DCR_BASEADDR)
    port map (
        clk         => OPB_clk,
        rst         => OPB_Rst,
        DCR_ABus    => DCR_ABus,
        DCR_Sl_DBus => DCR_Sl_DBus,
        DCR_Read    => DCR_Read,
        DCR_Write   => DCR_Write,
        Sl_dcrAck   => Sl_dcrAck,
        Sl_dcrDBus  => Sl_dcrDBus,
        ctrl_reg    => ctrl_reg);

  -- bus macro related signals and instantiations
  status(0) <= not(busmacro_enable);
  status(1) <= not(socket_reset);
  status(2) <= '1';
  status(3) <= '1';

  busmacro_enable <= ctrl_reg(31);
  socket_reset <= ctrl_reg(30) or OPB_Rst;

  busmacro_enable_vector <= (busmacro_enable,
                    busmacro_enable,
                    busmacro_enable,
                    busmacro_enable,
                    busmacro_enable,
                    busmacro_enable,
                    busmacro_enable);

  ROPB_ABus <= OPB_ABus_pr;
  ROPB_BE <= OPB_BE_pr;
  ROPB_Clk <= OPB_Clk;
  ROPB_DBus <= OPB_DBus_pr;
  ROPB_RNW <= OPB_RNW_pr;
  ROPB_Rst <= OPB_Rst_pr;
  ROPB_select <= OPB_select_pr;
  ROPB_seqAddr <= OPB_seqAddr_pr;

  Sln_DBus_pr <= RSln_DBus;
  Sln_errAck_pr <= RSln_errAck;
  Sln_retry_pr <= RSln_retry;
  Sln_toutSup_pr <= RSln_toutSup;
  Sln_xferAck_pr <= RSln_xferAck;

  -- Interrupt signals
  Interrupt_pr <= RInterrupt;

  Control1_BM : busmacro_vector8_xc4v_l2r_async_narrow
    port map (
                    macro_in(7)  => OPB_select,
                    macro_in(6)  => OPB_RNW,
                    macro_in(5)  => socket_reset,
                    macro_in(4)  => OPB_seqAddr,
                    macro_in(3 downto 0)  => OPB_BE(0 to 3),
                    macro_out(7) => OPB_select_pr,
                    macro_out(6) => OPB_RNW_pr,
```

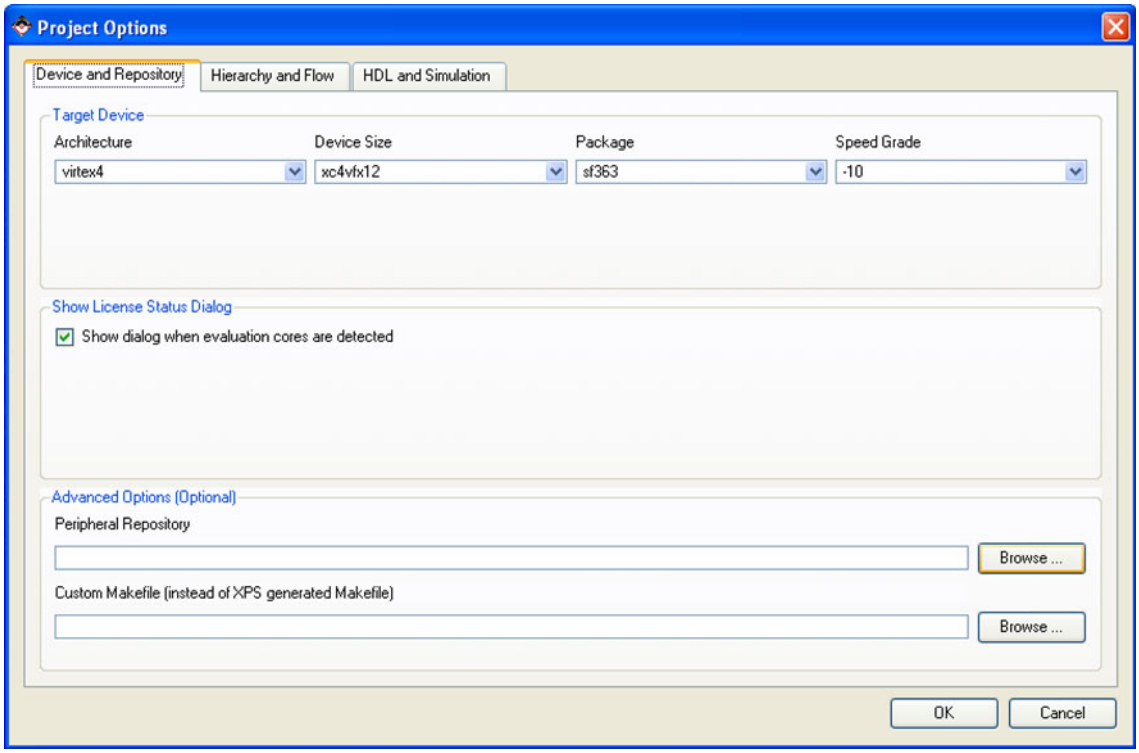
```
                    macro_out(5) => OPB_Rst_pr,
                    macro_out(4) => OPB_seqAddr_pr,
                    macro_out(3 downto 0) => OPB_BE_pr(0 to 3)
                    );

  Control2_BM : busmacro_vector8_xc4v_r2l_async_enable_narrow
    port map (
      enable => busmacro_enable_vector,
      macro_in(7)  => '0',
      macro_in(6)  => '0',
      macro_in(5)  => '0',
      macro_in(4)  => Interrupt_pr,
      macro_in(3)  => Sln_xferAck_pr,
      macro_in(2)  => Sln_toutSup_pr,
                      macro_in(1)  => Sln_retry_pr,
      macro_in(0)  => Sln_errAck_pr,
                      macro_out(7 downto 5) => dummy1,
      macro_out(4) => Interrupt,
      macro_out(3) => Sln_xferAck,
                      macro_out(2) => Sln_toutSup,
      macro_out(1) => Sln_retry,
                      macro_out(0) => Sln_errAck
                      );

  ABUS_BM_GENERATE: for i in 0 to C_OPB_AWIDTH/8-1 generate
    ABUS_BM : busmacro_vector8_xc4v_l2r_async_narrow
      port map (
        macro_in(7)  => OPB_ABus(2*i+0),
        macro_in(6)  => OPB_ABus(2*i+1),
        macro_in(5)  => OPB_ABus(2*i+8),
        macro_in(4)  => OPB_ABus(2*i+9),
        macro_in(3)  => OPB_ABus(2*i+16),
        macro_in(2)  => OPB_ABus(2*i+17),
        macro_in(1)  => OPB_ABus(2*i+24),
        macro_in(0)  => OPB_ABus(2*i+25),
        macro_out(7)  => OPB_ABus_pr(2*i+0),
        macro_out(6)  => OPB_ABus_pr(2*i+1),
        macro_out(5)  => OPB_ABus_pr(2*i+8),
        macro_out(4)  => OPB_ABus_pr(2*i+9),
        macro_out(3)  => OPB_ABus_pr(2*i+16),
        macro_out(2)  => OPB_ABus_pr(2*i+17),
        macro_out(1)  => OPB_ABus_pr(2*i+24),
        macro_out(0)  => OPB_ABus_pr(2*i+25)
        );
  end generate ABUS_BM_GENERATE;

  DBUS_BM_GENERATE: for i in 0 to C_OPB_DWIDTH/8-1 generate
    DBUS_BM : busmacro_vector8_xc4v_r2l_async_enable_narrow
      port map (
        enable => busmacro_enable_vector,
        macro_in(7 downto 0)  => Sln_DBus_pr(i*8 to i*8+7),
        macro_out(7 downto 0) => Sln_DBus(i*8 to i*8+7)
        );
  end generate DBUS_BM_GENERATE;

  SDBUS_BM_GENERATE: for i in 0 to C_OPB_DWIDTH/8-1 generate
    SDBUS_BM : busmacro_vector8_xc4v_l2r_async_narrow
      port map (
        macro_in(7)  => OPB_DBus(2*i+0),
        macro_in(6)  => OPB_DBus(2*i+1),
        macro_in(5)  => OPB_DBus(2*i+8),
        macro_in(4)  => OPB_DBus(2*i+9),
        macro_in(3)  => OPB_DBus(2*i+16),
        macro_in(2)  => OPB_DBus(2*i+17),
        macro_in(1)  => OPB_DBus(2*i+24),
        macro_in(0)  => OPB_DBus(2*i+25),
        macro_out(7)  => OPB_DBus_pr(2*i+0),
        macro_out(6)  => OPB_DBus_pr(2*i+1),
```

```
        macro_out(5)   => OPB_DBus_pr(2*i+8),
        macro_out(4)   => OPB_DBus_pr(2*i+9),
        macro_out(3)   => OPB_DBus_pr(2*i+16),
        macro_out(2)   => OPB_DBus_pr(2*i+17),
        macro_out(1)   => OPB_DBus_pr(2*i+24),
        macro_out(0)   => OPB_DBus_pr(2*i+25)
        );
  end generate SDBUS_BM_GENERATE;

end architecture IMP;
```

6.3.4. Creación del Proyecto Reconfigurable en EDK

Una vez se ha creado el proyecto *EDK* según los pasos descritos en el Anexo 2, y modificado el ***.mhs*** para adaptarlo a la placa tal y como se vio en el apartado 4.2, se procede a añadir los elementos necesarios para la auto-reconfiguración parcial de un periférico del *MicroBlaze*.

En primer lugar se añade el repositorio de periféricos al proyecto, si no se ha hecho al crearlo, haciendo clic en *Project/*  *Project Options*, de tal forma que aparecerá la siguiente ventana:



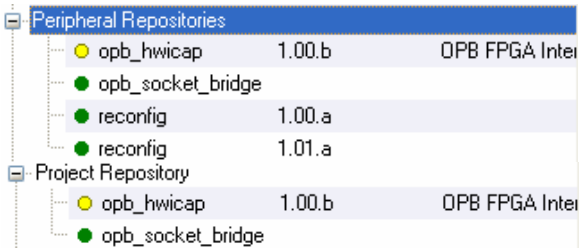
Dentro de la pestaña *Device and Repository*, en el apartado *Advanced Options*, existe la posibilidad de indicar donde se encuentra el repositorio de periféricos. Se hace clic en *Browse* y se selecciona la carpeta que contiene la carpeta del repositorio, y finalmente se hace clic en *OK*.

A continuación se ha de añadir al *IP Catalog* el que será el periférico real del *MicroBlaze*, el *opb_socket_brigde* modificado, copiando la carpeta *opb_socket_brigde* dentro de la carpeta *pcores* del proyecto *EDK*.

También se añade el *hwicap*, teniendo en cuenta lo especificado en el apartado 4.4.1, para obtener un *hwicap* válido para ***Virtex-4***.

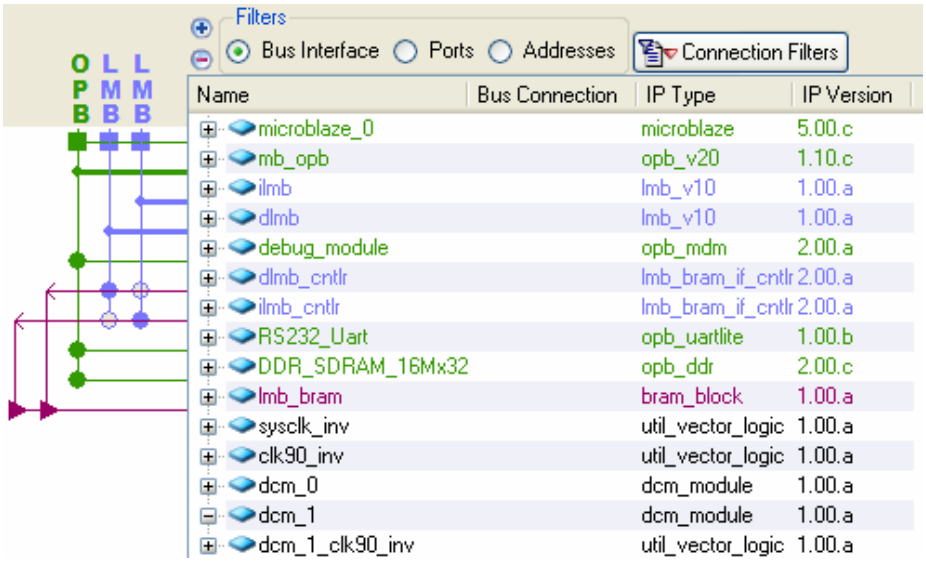
Para que aparezcan tanto los periféricos como el *opb_socket_brigde* en la pestaña *IP Catalog* de la ventana *Project Information Area* (y que, por lo tanto, se puedan añadir al proyecto), es necesario hacer clic en el botón *Rescan User Repository Directories*.

Una vez hecho esto aparecerán de la siguiente forma:



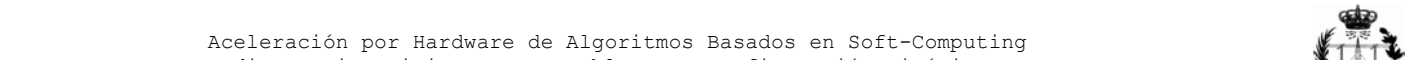
Se puede observar que en el apartado *Peripheral Repositiores* se muestran todos los elementos, tanto los periféricos del repositorio (*reconfig*) como los añadidos a la carpeta *pcores* del proyecto (*opb_socket_brigde* y *opb_hwicap*).

Actualmente, tras haber realizado los cambios para la placa especificados en el apartado 4.2, el *Bus Interface* del proyecto es el siguiente:

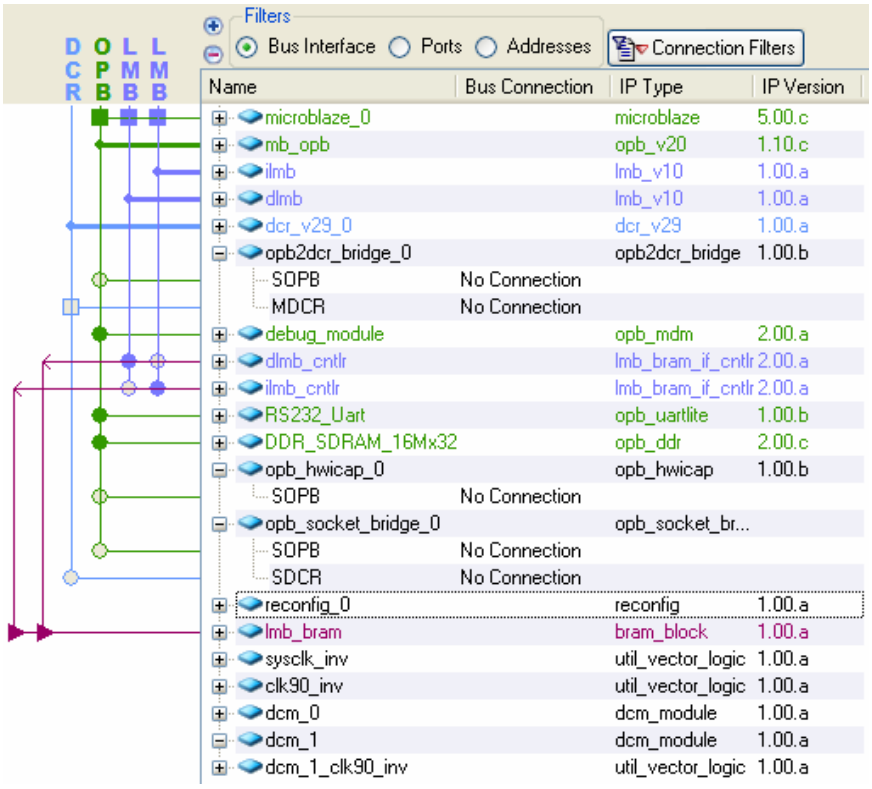


Se añaden los componentes necesarios para la reconfiguración del periférico, buscándolos en el Catálogo de IPs y haciendo doble clic sobre ellos:

- *FPGA Reconfiguration* (*EDK8.2 SP2*) o *Peripheral Repositories:* ***opb_hwicap***
- *Peripheral Repositories:* ***opb_socket_brigde***
- *Bus:* ***dcr_v29***
- *Bus Brigde:* ***opb2dcr_brigde***

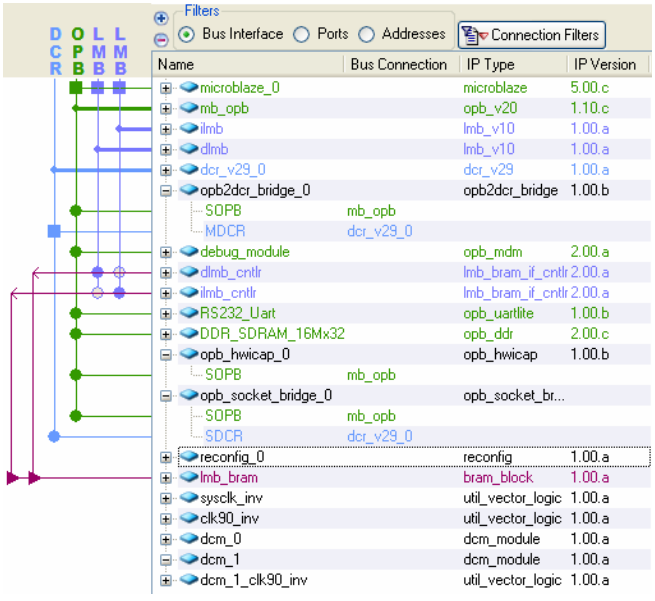


Se añade uno de los dos periféricos reconfigurables (*Peripheral Repositories:* ***reconfig***), aquel que va a ser utilizado en primer lugar por el *MicroBlaze.* De esta forma el *Bus Interface* quedará de la siguiente forma:



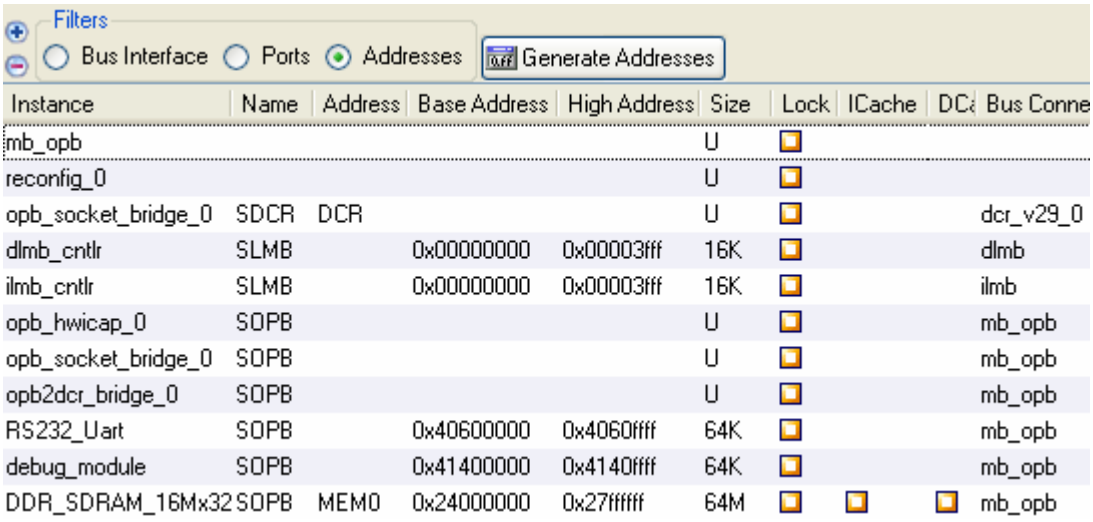
Se puede observar que aparece un nuevo bus *dcr*, un puente entre el bus *dcr* y el bus *opb*, y el *socket* que contiene los *buses macro* y se conectará al módulo reconfigurable.

A continuación se han de conectar todos los elementos añadidos: los puertos SOPB y MDCR del *opb2dcr_brigde_0*, los puertos SOPB y SDCR del *opb_socket_brigde_0*, y el SOPB del *opb_hwicap_0*, para lo que se ha de hacer clic en los círculos y cuadrados de conexión con los buses, quedando de la siguiente forma:



Como se puede ver, el módulo reconfigurable (*reconfig_0*) no tiene ningún puerto de conexión al OPB, ya que irá conectado a él a través del *opb_socket_brigde*, que sí que lo está.

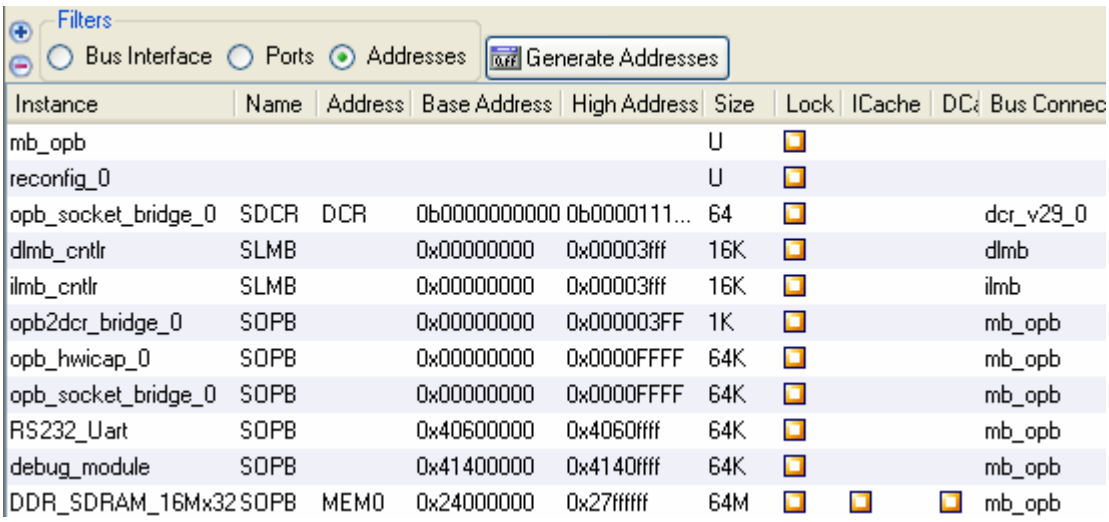
Si se accede a la pestaña *Addresses*, dentro del *System Assembly View*, aparece:



Instance	Name	Address	Base Address	High Address	Size	Lock	ICache	DC	Bus Conne
mb_opb					U				
reconfig_0					U				
opb_socket_bridge_0	SDCR	DCR			U				dcr_v29_0
dlmb_cntlr	SLMB		0x00000000	0x00003fff	16K				dlmb
ilmb_cntlr	SLMB		0x00000000	0x00003fff	16K				ilmb
opb_hwicap_0	SOPB				U				mb_opb
opb_socket_bridge_0	SOPB				U				mb_opb
opb2dcr_bridge_0	SOPB				U				mb_opb
RS232_Uart	SOPB		0x40600000	0x4060ffff	64K				mb_opb
debug_module	SOPB		0x41400000	0x4140ffff	64K				mb_opb
DDR_SDRAM_16Mx32	SOPB	MEM0	0x24000000	0x27ffffff	64M				mb_opb

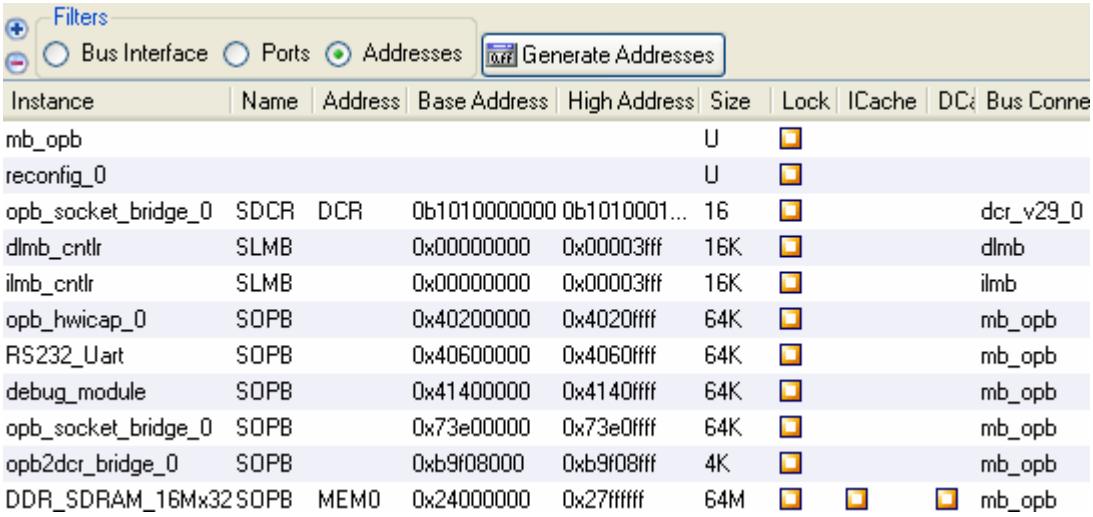
Se deben asignar los correspondientes espacios de direcciones a cada uno de los periféricos añadidos. Para ello se selecciona la *U* bajo la columna *Size*, y se introducen los siguientes valores:

- Puerto *SOPB* del opb*_socket_brigde_0*: 64 KB.
- Puerto *SDCR* del *opb_socket_brigde_0*: 64 bytes.
- Puerto *SOPB* del *opb2dcr_brigde_0*: 1 KB.
- Puerto *SOPB* del *opb_hwicap_0*: 64 KB



Instance	Name	Address	Base Address	High Address	Size	Lock	ICache	DC	Bus Connec
mb_opb					U				
reconfig_0					U				
opb_socket_bridge_0	SDCR	DCR	0b0000000000	0b0000111...	64				dcr_v29_0
dlmb_cntlr	SLMB		0x00000000	0x00003fff	16K				dlmb
ilmb_cntlr	SLMB		0x00000000	0x00003fff	16K				ilmb
opb2dcr_bridge_0	SOPB		0x00000000	0x000003FF	1K				mb_opb
opb_hwicap_0	SOPB		0x00000000	0x0000FFFF	64K				mb_opb
opb_socket_bridge_0	SOPB		0x00000000	0x0000FFFF	64K				mb_opb
RS232_Uart	SOPB		0x40600000	0x4060ffff	64K				mb_opb
debug_module	SOPB		0x41400000	0x4140ffff	64K				mb_opb
DDR_SDRAM_16Mx32	SOPB	MEM0	0x24000000	0x27ffffff	64M				mb_opb

Obsérvese que no se introduce valor alguno para *reconfig_0*, al no ir conectado directamente al bus *OPB*. A continuación se pincha en *Generate Addresses*, para generar el espacio de direcciones correspondiente al tamaño asignado, quedando de la siguiente manera:



Instance	Name	Address	Base Address	High Address	Size	Lock	ICache	DC	Bus Conne
mb_opb					U				
reconfig_0					U				
opb_socket_bridge_0	SDCR	DCR	0b1010000000	0b1010001...	16				dcr_v29_0
dlmb_cntlr	SLMB		0x00000000	0x00003fff	16K				dlmb
ilmb_cntlr	SLMB		0x00000000	0x00003fff	16K				ilmb
opb_hwicap_0	SOPB		0x40200000	0x4020ffff	64K				mb_opb
RS232_Uart	SOPB		0x40600000	0x4060ffff	64K				mb_opb
debug_module	SOPB		0x41400000	0x4140ffff	64K				mb_opb
opb_socket_bridge_0	SOPB		0x73e00000	0x73e0ffff	64K				mb_opb
opb2dcr_bridge_0	SOPB		0xb9f08000	0xb9f08fff	4K				mb_opb
DDR_SDRAM_16Mx32	SOPB	MEM0	0x24000000	0x27ffffff	64M				mb_opb

Como se ha comentado con anterioridad, el periférico *opb_socket_brigde* irá conectado al módulo reconfigurable (*reconfig*). Para ello se habrán de añadir las siguientes líneas en el ***.mhs.***

- Bajo la instancia del módulo *opb_socket_brigde* y antes del end:

```
PORT ROPB_Rst = ROPB_Rst_0
PORT ROPB_ABus = ROPB_ABus_0
PORT ROPB_BE = ROPB_BE_0
PORT ROPB_RNW = ROPB_RNW_0
PORT ROPB_select = ROPB_select_0
PORT ROPB_seqAddr = ROPB_seqAddr_0
PORT ROPB_DBus = ROPB_DBus_0
PORT RSln_DBus = RSl_DBus_0
PORT RSln_errAck = RSl_errAck_0
PORT RSln_retry = RSl_retry_0
PORT RSln_toutSup = RSl_toutSup_0
PORT RSln_xferAck = RSl_xferAck_0
PORT ROPB_Clk = ROPB_Clk_0
```

- Bajo la instancia del módulo *reconfig* y antes del end:

```
PORT OPB_Clk = ROPB_Clk_0
PORT OPB_Rst = ROPB_Rst_0
PORT OPB_ABus = ROPB_ABus_0
PORT OPB_BE = ROPB_BE_0
PORT OPB_RNW = ROPB_RNW_0
PORT OPB_select = ROPB_select_0
PORT OPB_seqAddr = ROPB_seqAddr_0
PORT OPB_DBus = ROPB_DBus_0
PORT Sl_DBus = RSl_DBus_0
PORT Sl_errAck = RSl_errAck_0
PORT Sl_retry = RSl_retry_0
PORT Sl_toutSup = RSl_toutSup_0
PORT Sl_xferAck = RSl_xferAck_0
```

En el caso en el que el periférico sea producto del *wizard* del *EDK* (es decir, que contenga el *IPIF*), también es necesario añadir manualmente dos líneas de direccionamiento bajo la instancia del módulo *reconfig* y antes del *end*. Estas líneas deberán de copiarse de los parámetros C_*BASEADDR* y *C_HIGHADDR* del *opb_socket_brigde*:

```
PARAMETER C_BASEADDR = 0x********
PARAMETER C_HIGHADDR = 0x********
```

En donde *0x********* son las direcciones de los parámetros anteriormente citados, una vez se ha ejecutado *Generate Addresses*.

Para editar el ***.mhs*** se abre la pestaña *Project* del *Project Information Area* y se hace doble clic en *MHS File: system.mhs*, de tal forma que se abrirá a la derecha. Al añadir el código (en rojo), las instancias de los módulos *opb_socket_brigde_0* y *reconfig_0* quedarán de la siguiente forma:

```
BEGIN opb_socket_bridge
 PARAMETER INSTANCE = opb_socket_bridge_0
 PARAMETER C_BASEADDR = 0x73e00000
 PARAMETER C_HIGHADDR = 0x73e0ffff
 PARAMETER C_DCR_BASEADDR = 0b1010000000
 PARAMETER C_DCR_HIGHADDR = 0b1010001111
 BUS_INTERFACE SDCR = dcr_v29_0
 BUS_INTERFACE SOPB = mb_opb
 PORT ROPB_Rst = ROPB_Rst_0
 PORT ROPB_ABus = ROPB_ABus_0
 PORT ROPB_BE = ROPB_BE_0
 PORT ROPB_RNW = ROPB_RNW_0
 PORT ROPB_select = ROPB_select_0
 PORT ROPB_seqAddr = ROPB_seqAddr_0
 PORT ROPB_DBus = ROPB_DBus_0
 PORT RSln_DBus = RSl_DBus_0
 PORT RSln_errAck = RSl_errAck_0
 PORT RSln_retry = RSl_retry_0
 PORT RSln_toutSup = RSl_toutSup_0
 PORT RSln_xferAck = RSl_xferAck_0
 PORT ROPB_Clk = ROPB_Clk_0
END

BEGIN reconfig
 PARAMETER INSTANCE = reconfig_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x73e00000
 PARAMETER C_HIGHADDR = 0x73e0ffff
 PORT OPB_Clk = ROPB_Clk_0
 PORT OPB_Rst = ROPB_Rst_0
 PORT OPB_ABus = ROPB_ABus_0
 PORT OPB_BE = ROPB_BE_0
 PORT OPB_RNW = ROPB_RNW_0
 PORT OPB_select = ROPB_select_0
 PORT OPB_seqAddr = ROPB_seqAddr_0
 PORT OPB_DBus = ROPB_DBus_0
 PORT Sl_DBus = RSl_DBus_0
 PORT Sl_errAck = RSl_errAck_0
 PORT Sl_retry = RSl_retry_0
 PORT Sl_toutSup = RSl_toutSup_0
 PORT Sl_xferAck = RSl_xferAck_0
END
```

Al guardar el ***.mhs*** se actualizará el proyecto y quedarán conectados y direccionados ambos módulos.

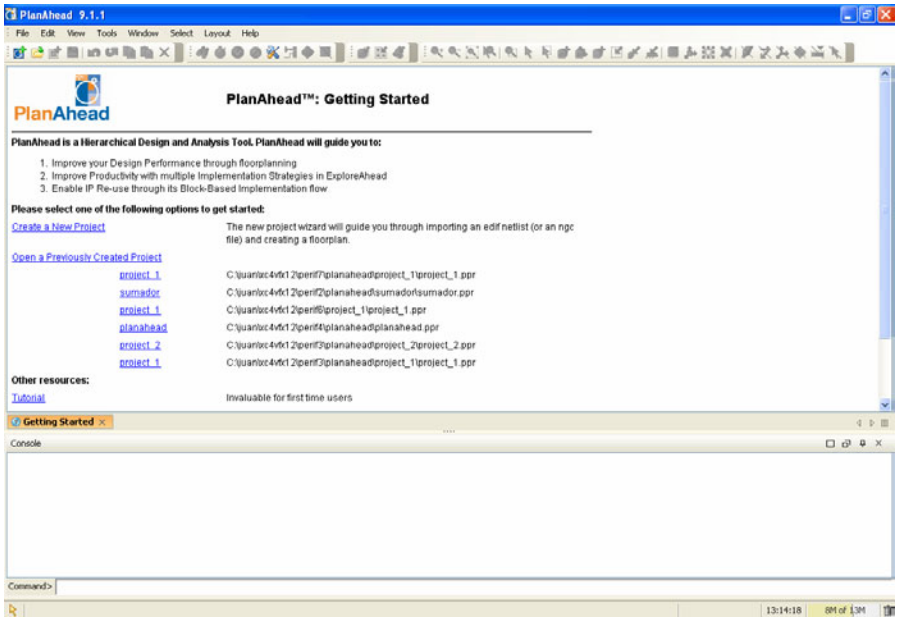
Para finalizar se habrá de crear el *netlist* sintetizando el proyecto, para lo que se hace clic en *Hardware/* *Generate Netlist.* Con esto habrá concluido por ahora el trabajo del primer periférico con el *EDK*.

6.4. Floorplanning con PlanAhead

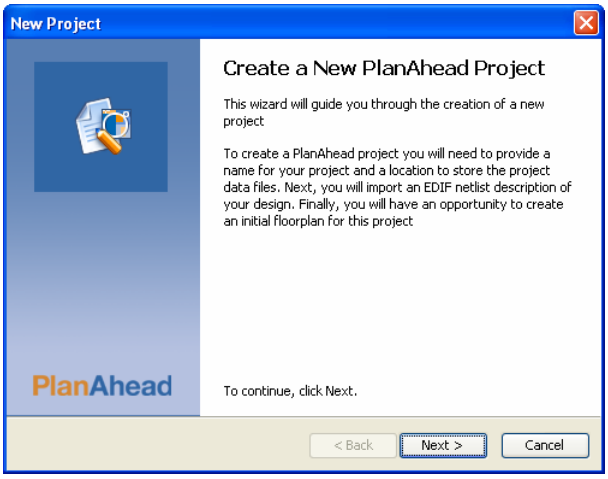
6.4.1. Creación del Proyecto Reconfigurable con PlanAhead

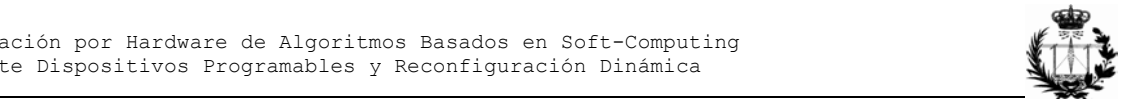
Para trabajar con *PlanAhead* se crea una carpeta en el proyecto reconfigurable llamada *planahead*, en la que se crearán los distintos proyectos.

Al abrir el entorno PlanAhead 9.1.1 aparece la siguiente ventana, en la que se debe hacer clic en *Create a New Project*

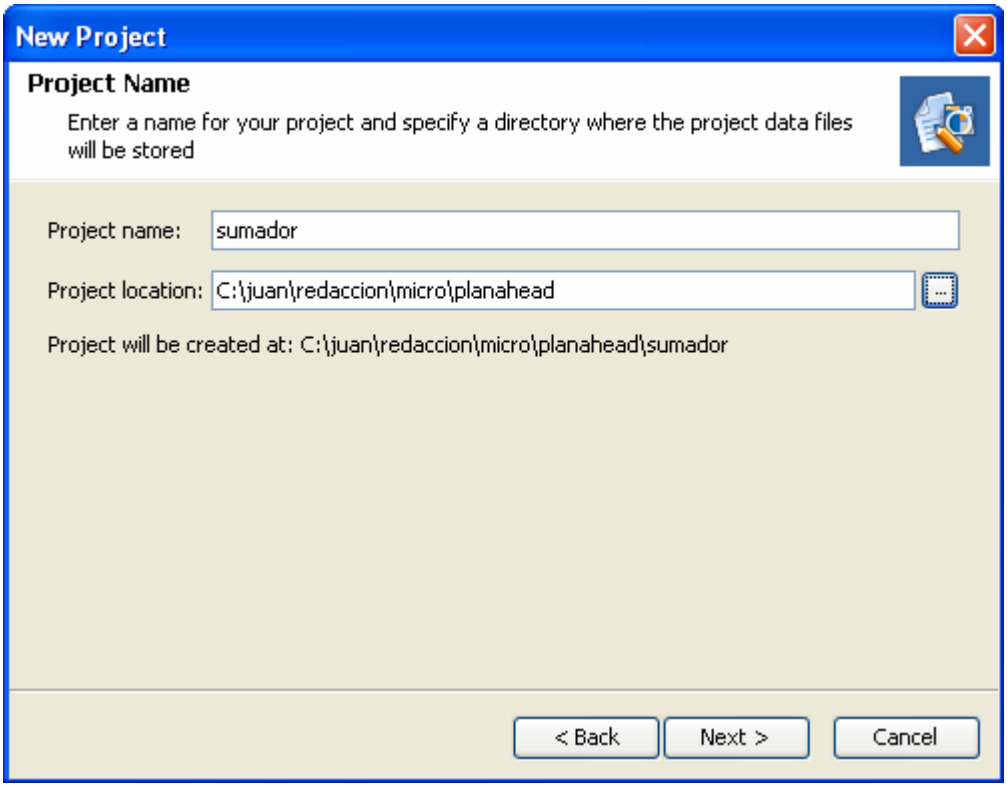


En ese momento aparecerá una ventana del *wizard*, en la que se hace clic en *Next*.

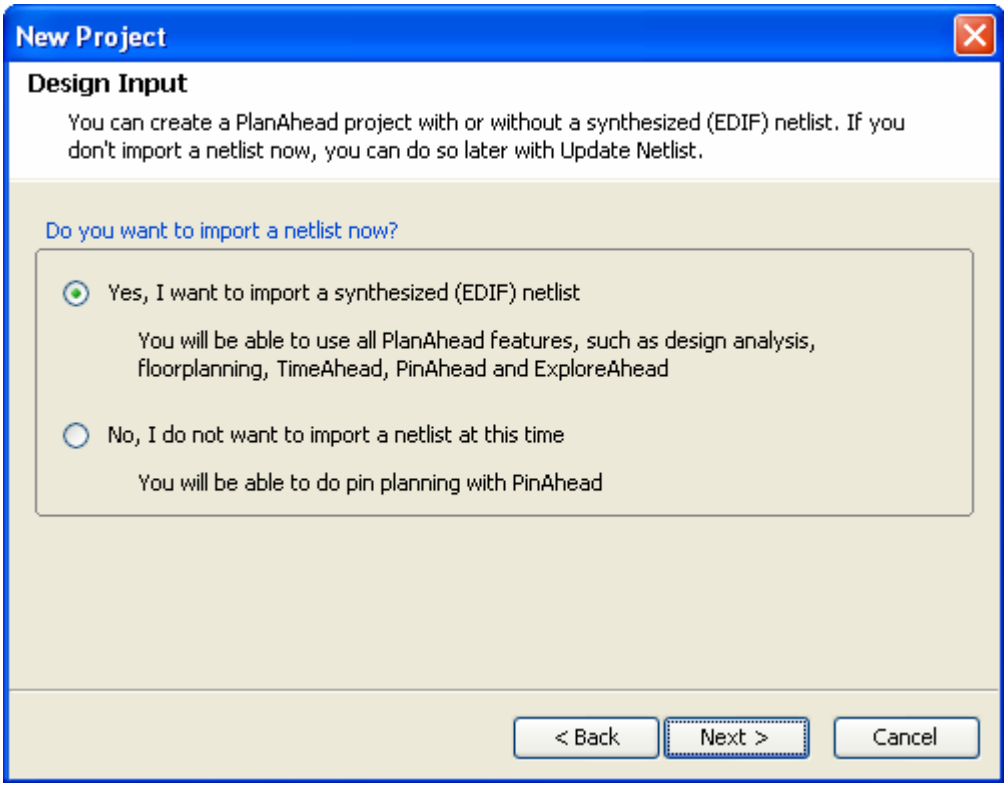




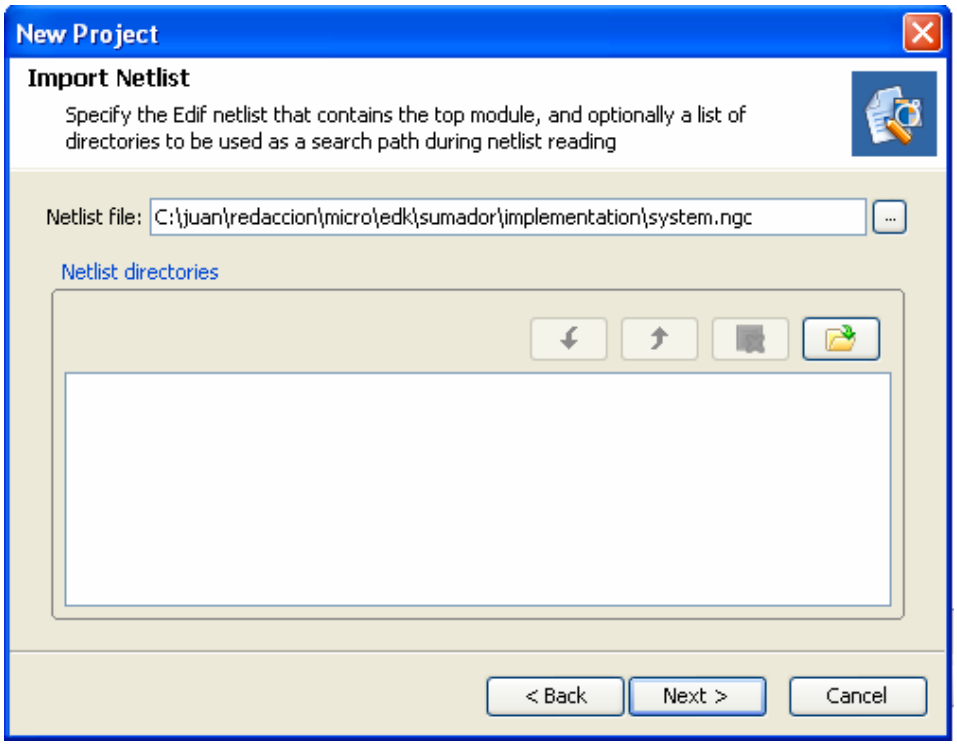
Se indica el nombre del proyecto para el primero de los módulos reconfigurables, en este caso, sumador, así como el directorio en el que será guardado, la carpeta *planahead* del proyecto reconfigurable. Se hace clic en *Next.*



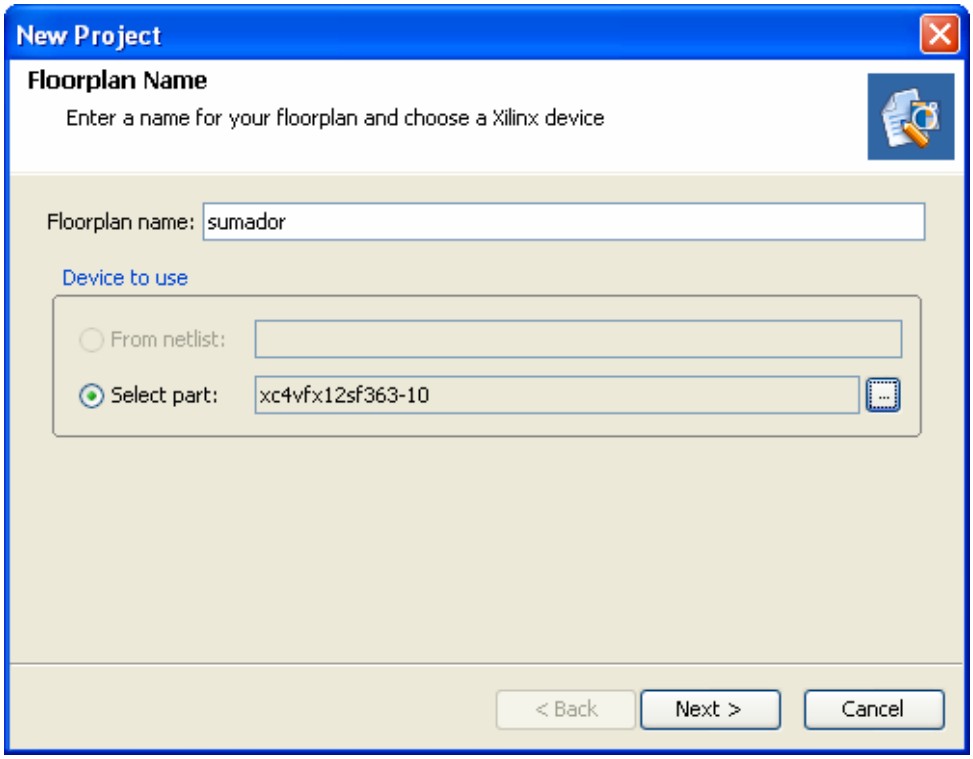
Se elige *Yes, I want to import a synthesized (EDIF) netlist*, para importar el fichero sintetizado del proyecto *EDK* al *PlanAhead*, y se hace clic en *Next*.



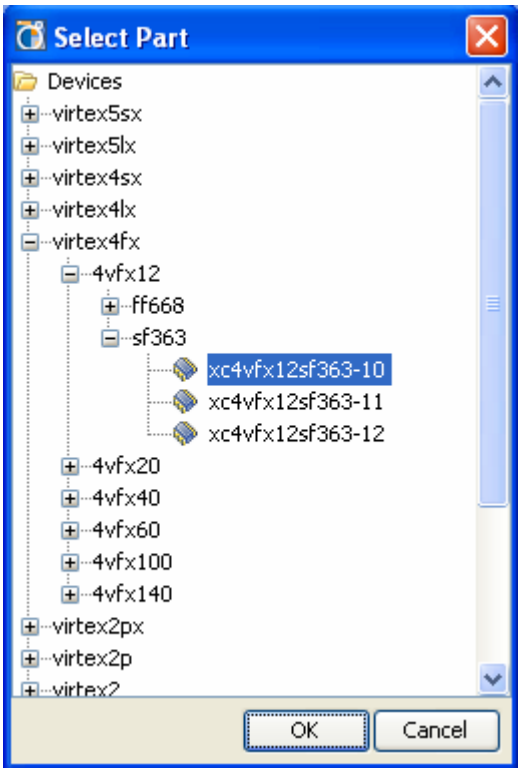
Dentro de la ventana *Import Netlist*, en el cuadro *Netlist file* se busca el archivo *system.ngc*, dentro de la carpeta *edk\sumador\implementation*, es decir, se selecciona el archivo de síntesis del primer proyecto *EDK* y se hace clic en *Next,* de esta forma se cargarán todos los archivos ***.ngc*** del proyecto.



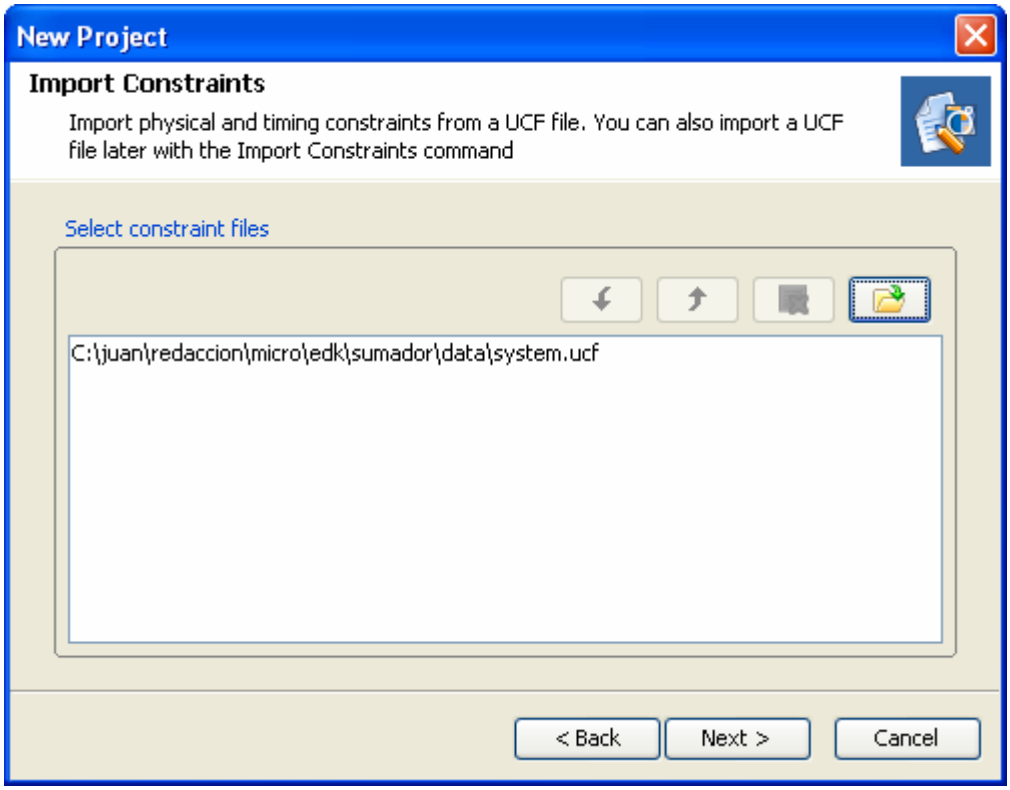
A continuación se selecciona un nombre para el *Floorplan*, en este caso sumador, y se hace clic en el botón "..." de *Select part*.



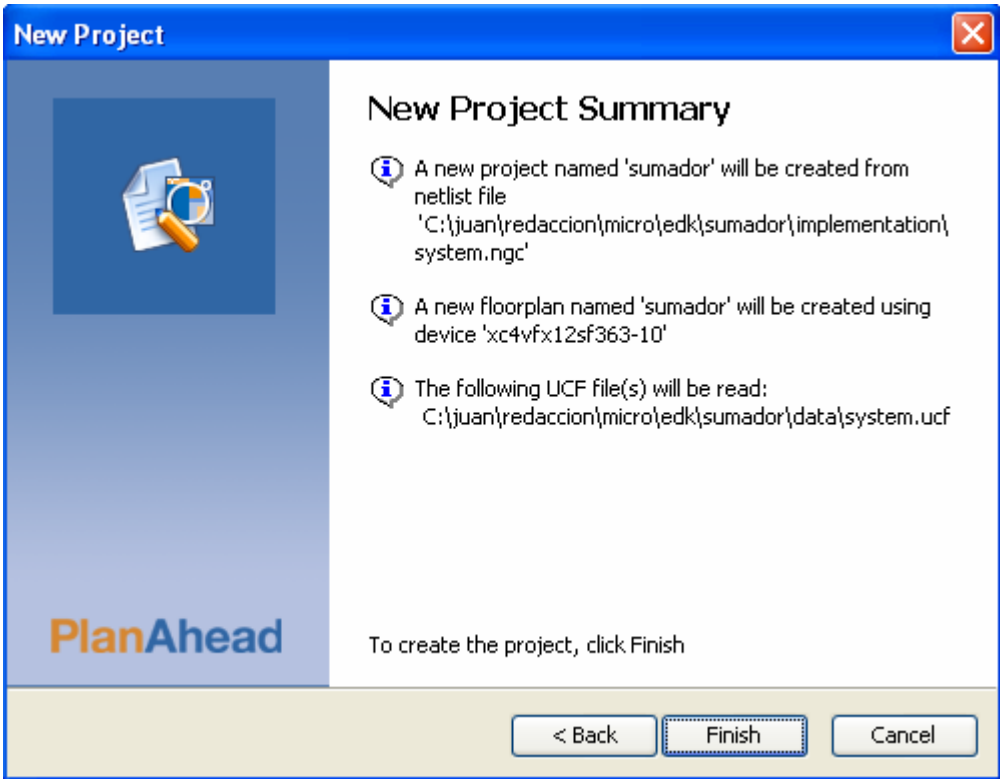
Entonces aparece la siguiente ventana, en la que se elige la *FPGA*, en este caso la *Virtex 4 4vfx12-sf363-10* y se hace clic en *OK* y a continuación en *Next.*



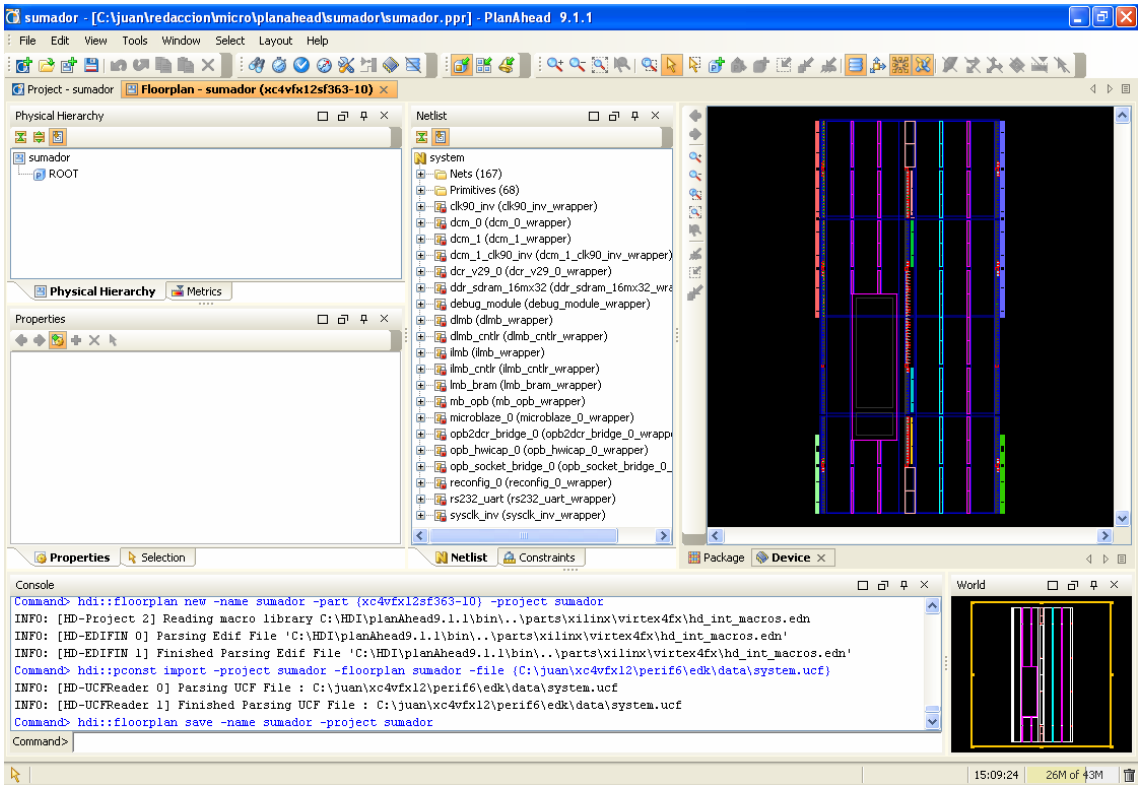
Haciendo clic en la carpeta, se selecciona el fichero de restricciones ***.ucf*** del proyecto *EDK* del primer periférico reconfigurable, que se encuentra en la carpeta *edk\sumador\data,* y finalmente se pincha en *Next.*



Para concluir con la creación del proyecto de *PlanAhead*, se pulsa en *Finish*.



Aparecerá entonces la ventana de trabajo de la herramienta *PlanAhead.*



El árbol de carpetas del proyecto reconfigurable quedará de la siguiente forma:

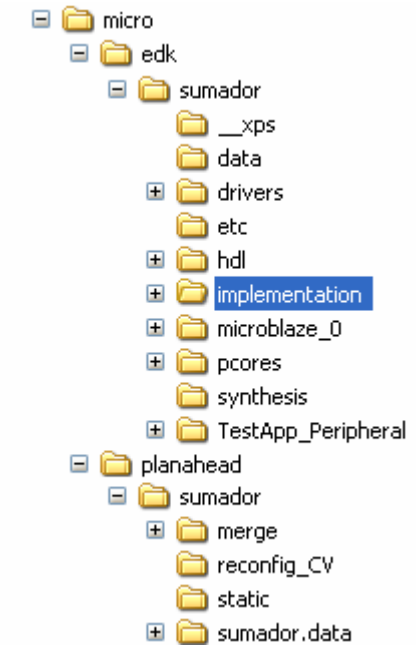
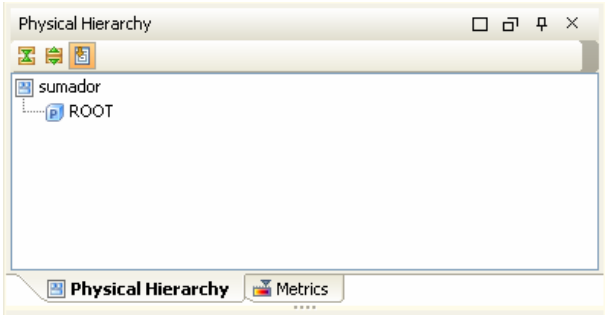


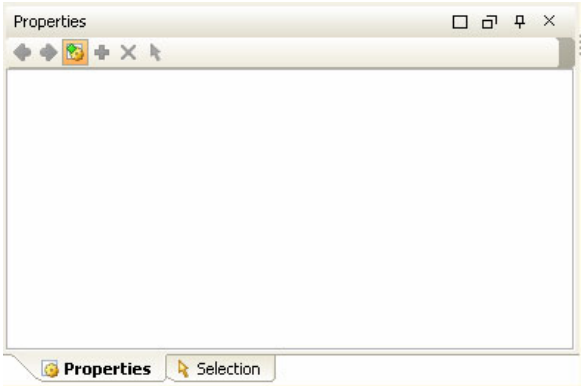
Figura 41. Estructura de directorios con *PlanAhead*

La herramienta de *floorplaning PlanAhead* está formada por varias ventanas que se explican a continuación.

En primer lugar la ventana *Phisical Hierarchy*, en la que aparecerá la jerarquía de módulos del diseño.

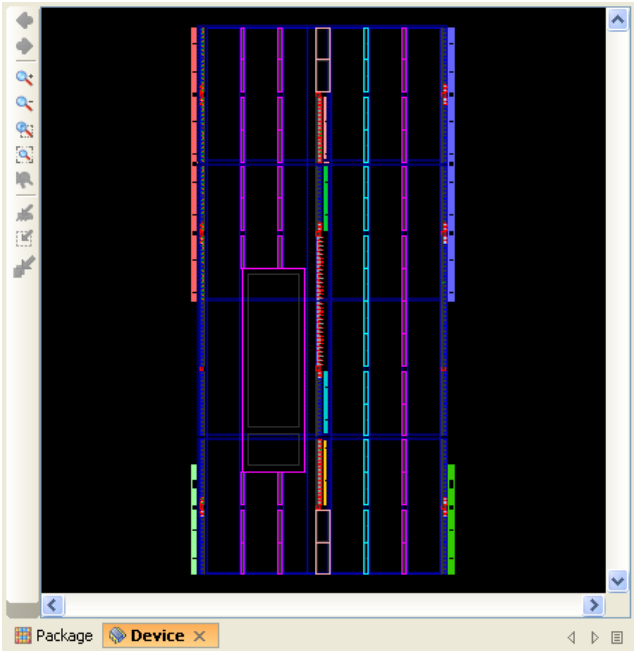


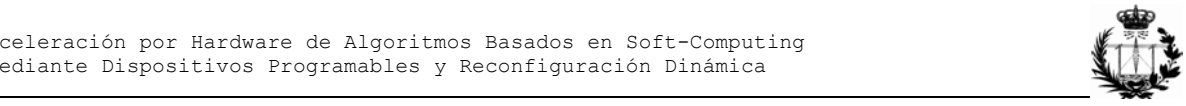
La ventana *Properties* permitirá modificar las propiedades de los distintos elementos del proyecto.



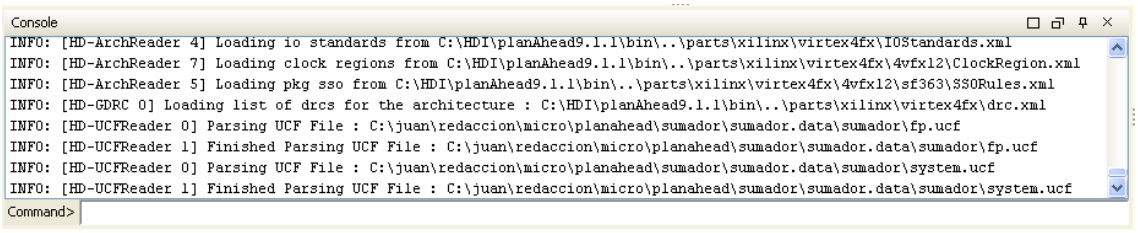
La ventana *Netlist* en la que aparecen los distintos elementos (buses, periféricos, módulos...) que se sintetizaron con el *EDK*, y que se han añadido a partir del *system.ngc* que se ha seleccionado en el *wizard*.

En la ventana *Device*, podemos observar los elementos físicos de la *FPGA*, y será el lugar donde se deberán localizar los elementos del proyecto reconfigurable.





Finalmente está la consola, donde irán apareciendo los distintos comandos que se ejecutan al utilizar *PlanAhead*, así como toda la información resultante de ellos.

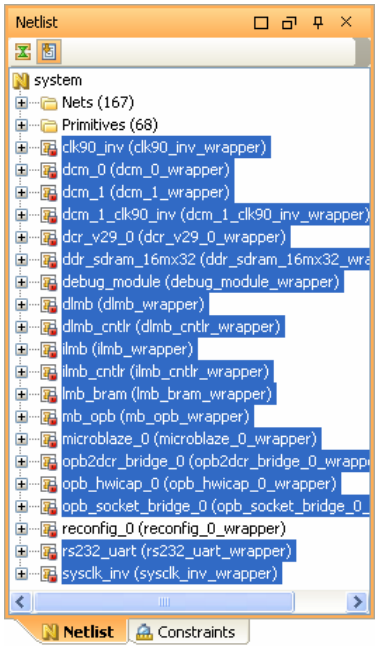


Para comenzar a trabajar con Reconfiguración Parcial en *PlanAhead 9.1.1*, se ha de ejecutar el siguiente comando en la consola:

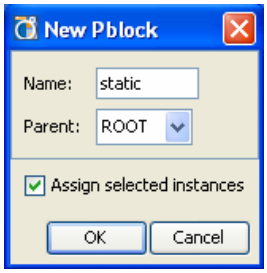
```
hdi::param set -name project.enablePR -bvalue yes
```

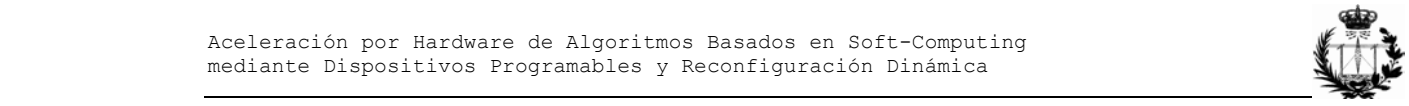
6.4.2. Creación de la Parte Estática:

En la ventan *Netlist*, se seleccionan todos los elementos de la parte estática del proyecto reconfigurable (es decir, todos los elementos, menos el módulo *reconfig*):

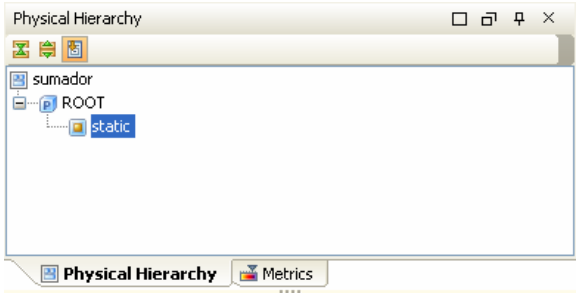


Se pincha encima con el botón derecho y se selecciona *New Pblock*, de tal forma que aparece la siguiente ventana. En el aparado *Name* se le pone el nombre a la parte estática: *static*. Se hace clic en *OK*.



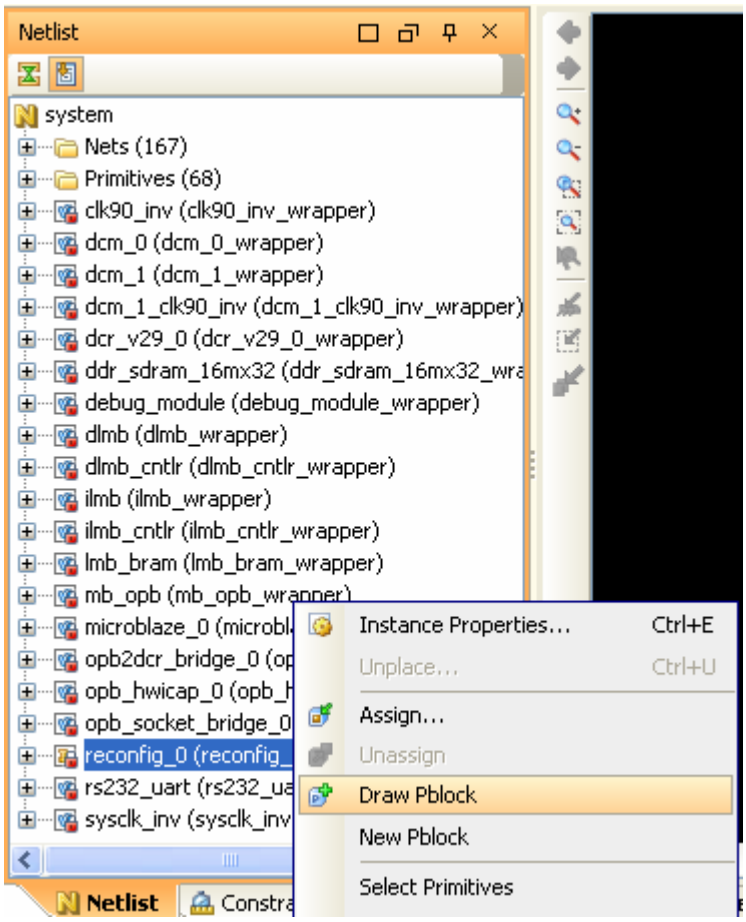


De esta forma deberá aparecer un bloque *static* en la ventana *Physical Hierarchy*:

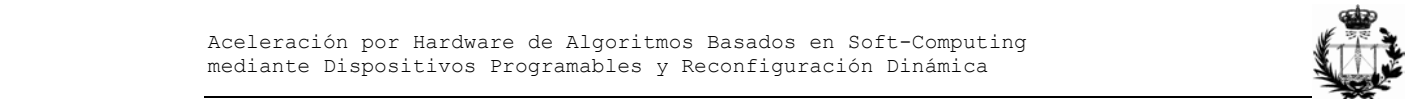


6.4.3. Creación de la Zona Reconfigurable:

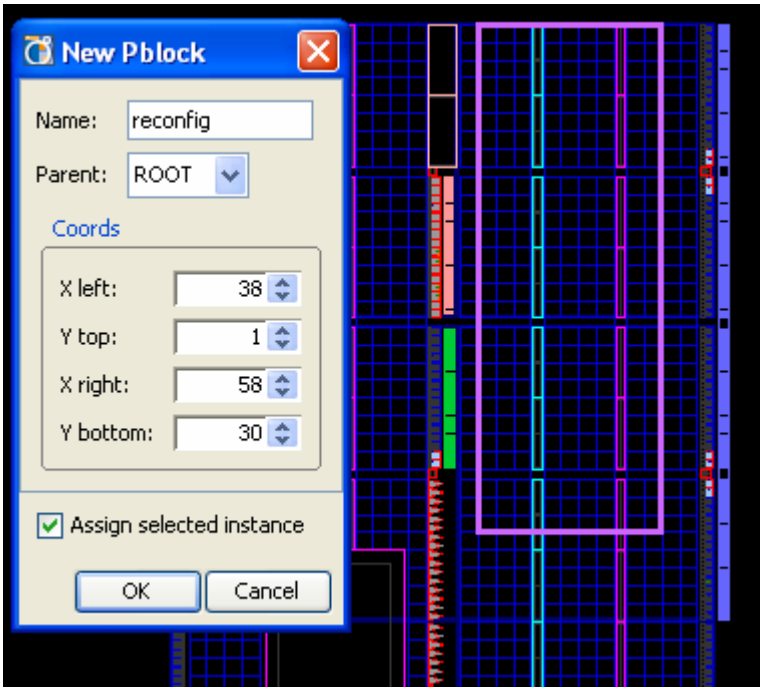
Se hace clic con el botón derecho encima del módulo reconfigurable *reconfig*, dentro de la ventana *Netlist*, y se hace clic en *Draw Pblock.*



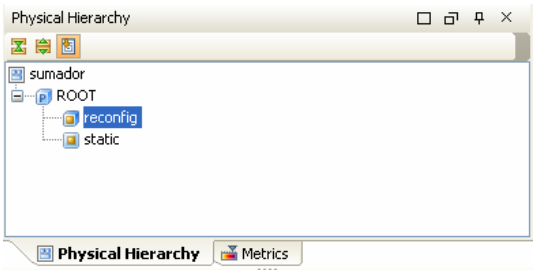
De esta forma se podrá seleccionar dentro de la ventana *Device*, el área que ocupará la parte reconfigurable (seleccionando un rectángulo con el ratón, de color lila en la imagen inferior). Este rectángulo deberá englobar regiones de *DSP* así como de *BRAM* (azul claro y fucsias respectivamente).



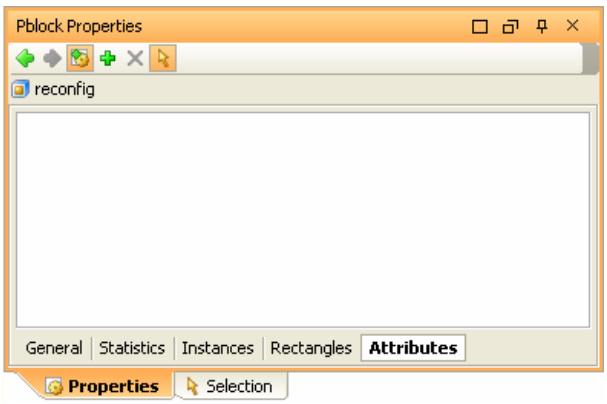
En ese momento aparecerá una ventana de *New Pblock*, en la que cambiamos el nombre a *reconfig*, y podemos modificar las coordenadas del área reconfigurable seleccionada. Finalmente se hace clic en OK.



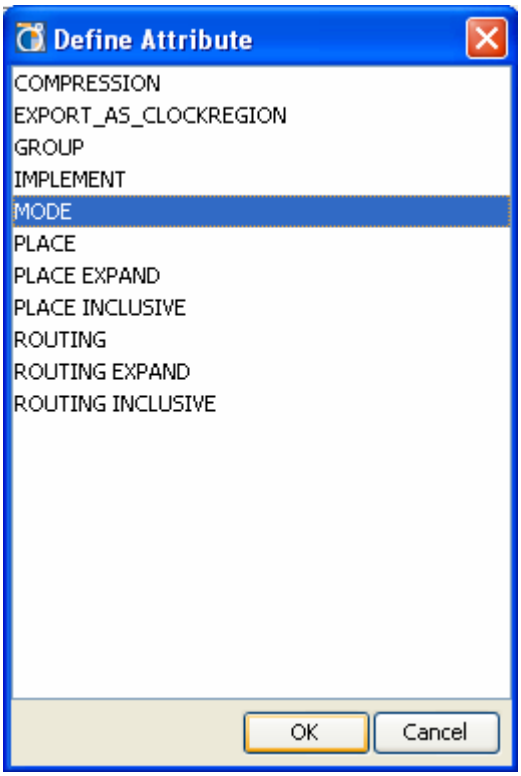
Habrá aparecido el módulo *reconfig* dentro de la ventana *Physical Hierarchy*, se selecciona:



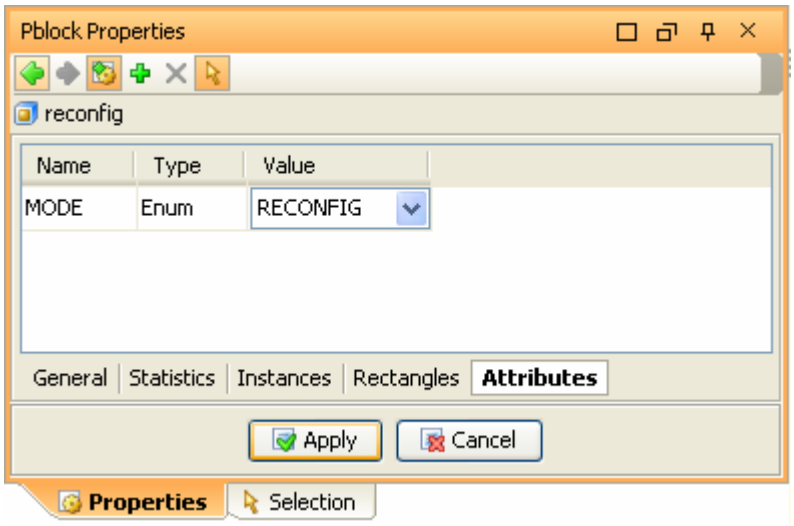
De esta forma, en la ventana *Properties*, aparecerán las propiedades del módulo reconfigurable *reconfig*. Se selecciona la pestaña *Attributes*.



En esta ventana se hace clic en +, de tal forma que aparece la ventana *Define Attribute*, donde se selecciona *MODE* y se hace clic en OK.

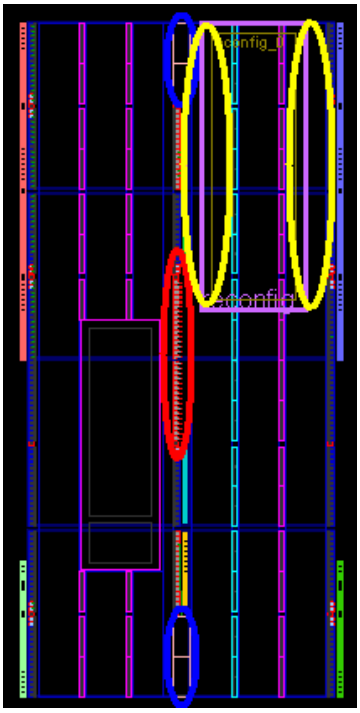


Se habrá creado un nuevo atributo *MODE*, al que se le da el valor *RECONFIG*, y se pincha en *Apply* para confirmarlo.



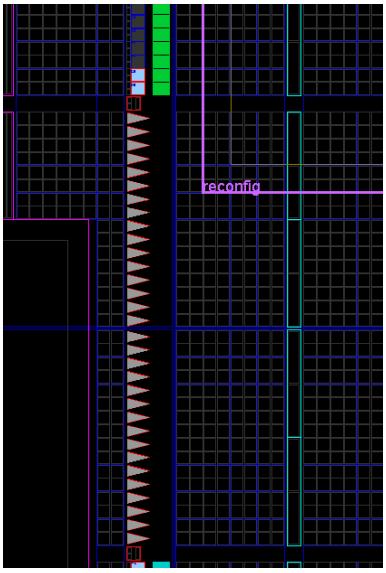
6.4.4. Localización de BUFG, DCM y Buses Macro

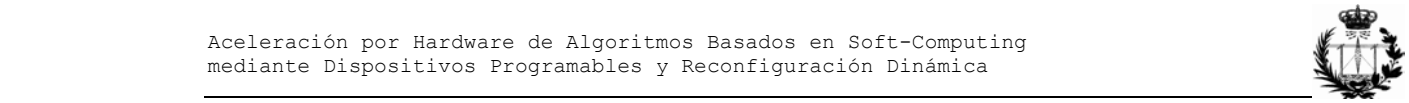
Para el funcionamiento de la reconfiguración parcial es necesario que se sitúen sobre la placa todos los *BUFG* (marcado en rojo), *DCM* (marcado en azul) y *Buses macro* (marcado en amarillo) utilizados.



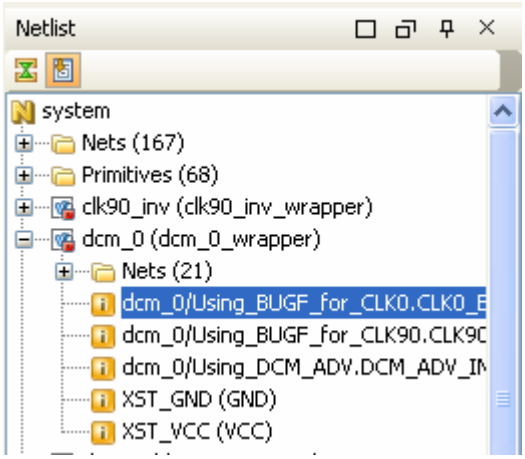
Para ello se tiene que hacer clic en el botón *Create Site Constraint Mode*

Para la localización de los *BUFG* se hace zoom pinchando en el botón  *Zoom Area* y seleccionando el área marcada en rojo en la figura anterior. De esta manera quedará:





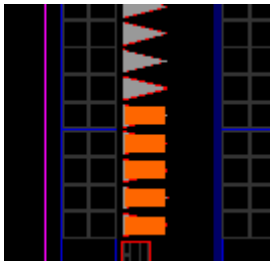
En la ventana *Netlist*, se abren los items que contienen *BUFG* (en este caso *dcm_0, dcm_1*, y *debug_module*), se seleccionan los *BUFG* y se arrastran hasta los triángulos en los que se deben localizar los *BUFG* en la ventana *Device.*



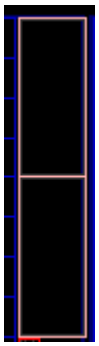
Este paso se repite para los siguientes *BUFG*:

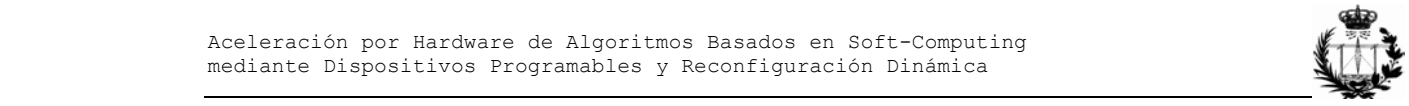
dcm_0: dcm_0/Using_BUFG_for_CLK0.CLK0_BUFG_INST (BUFG)
dcm_0: dcm_0/Using_BUFG_for_CLK90.CLK90_BUFG_INST (BUFG)
dcm_1: dcm_1/Using_BUFG_for_CLK90.CLK90_BUFG_INST (BUFG)
debug_module: debug_module/BUFG_DRCK1 (BUFG)
debug_module: debug_module/BUFG_DRCK2 (BUFG)

Quedando en la ventana *Device*:

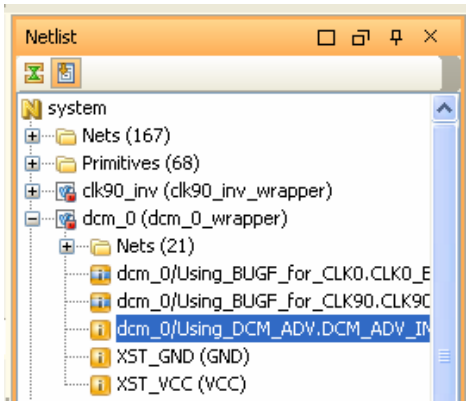


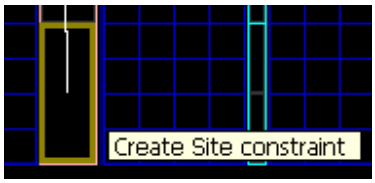
Para la localización de los *DCM* se hace zoom en el área marcada en azul, quedando de la siguiente forma:





En la ventana *Netlist*, se abren los items que contienen *DCM* (en este caso *dcm_0* y *dcm_1*), se seleccionan los *DCM* y se arrastran hasta los rectángulos en los que se deben localizar los *DCM* en la ventana *Device*.

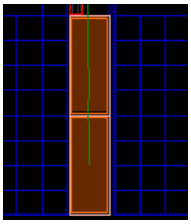




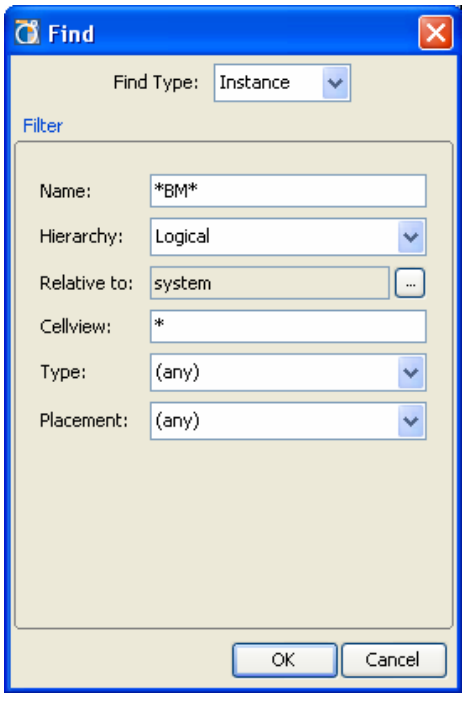
Este paso se repite para los *DCM*:

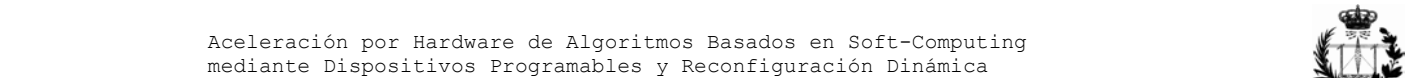
dcm_0: dcm_0/Using_DCM_ADV.DCM_ADV_INST (DCM_ADV)
dcm_1: dcm_1/Using_DCM_ADV.DCM_ADV_INST (DCM_ADV)

Quedando en la ventana *Device*:

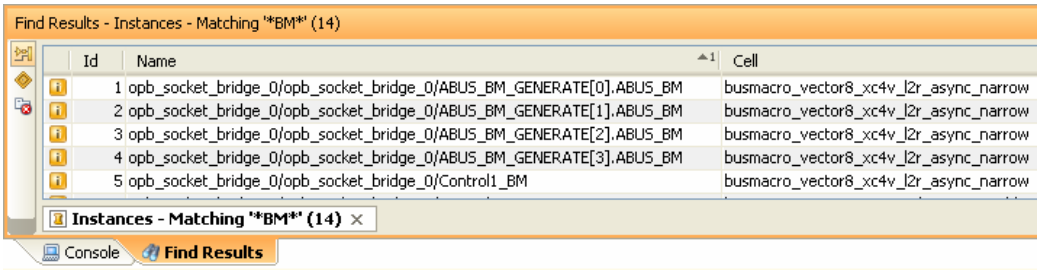


Para la localización de los *buses macro* se hace clic en *Edit* / *Find*. Se escribe **BM** en el apartado *Name*, y se hace clic en *OK*:





De esta forma aparece un lista de los Buses Macro, en la ventana de la consola.



La colocación de los *Buses Macro* en la frontera derecha o izquierda de la región reconfigurable deberá de realizarse dependiendo del tipo del *Bus Macro* (*r2l* o *l2r*) y de si las señales son de entrada o de salida, tal y como se vio en el apartado 2.3.1.

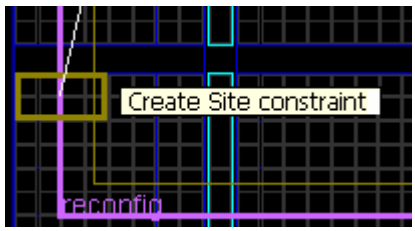
Colocación en la frontera izquierda:

- Todas las señales del b*us macro* son entradas, y es del tipo *r2l.*
- Todas las señales del b*us macro* son salidas, y es del tipo *l2r.*

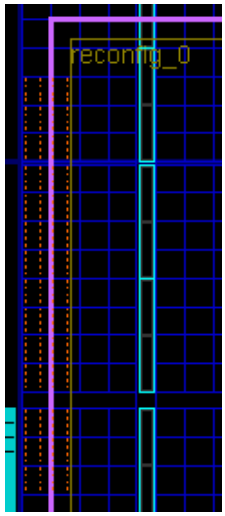
Colocación en la frontera derecha:

- Todas las señales del b*us macro* son entradas, y es del tipo *l2r.*
- Todas las señales del b*us macro* son salidas, y es del tipo *r2l.*

Todos los *buses macro* del módulo *opb_socket_brigde* están configurados para situarlos en la frontera izquierda de la región reconfigurable. Por lo tanto, se seleccionan y, uno por uno, se arrastran hasta la frontera izquierda de la región reconfigurable en la ventana *Device*, situándolos con la línea rosa de la frontera en medio del *bus macro*:

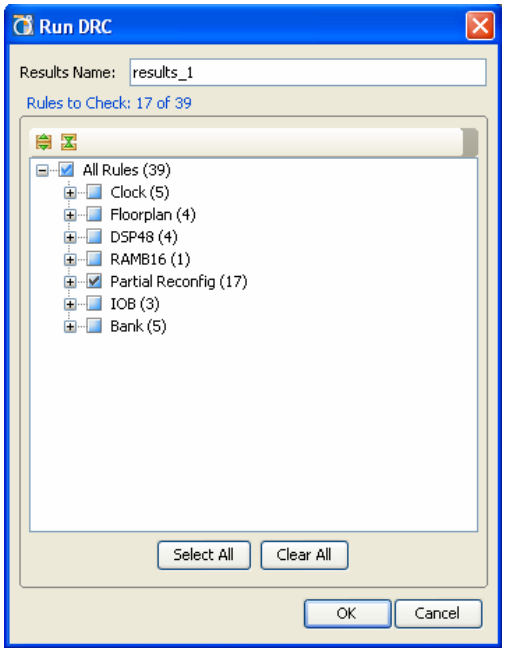


Al colocar todos los b*uses macro*, la región reconfigurable queda:

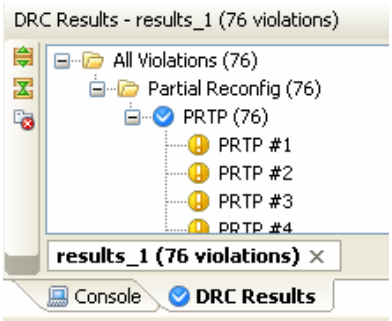


La herramienta *DRC Checking* detectará cualquier error que se haya producido en el diseño reconfigurable. Para activarla se hace clic en *Tools/ Run DRC*:

Aparecerá una ventana en la que se ha de dejar seleccionado únicamente *Partial Reconfig*, y se hace clic en *OK*.

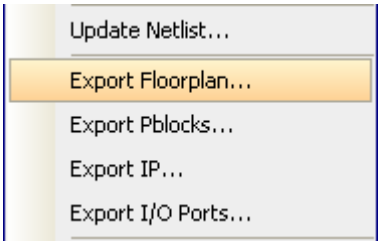


De esta forma aparecerán en la consola los resultados *del DCR Checking*. Mostrará muchas violaciones, pero mientras no haya ninguna en color rojo, la colocación de elementos y el diseño reconfigurable será correcto:

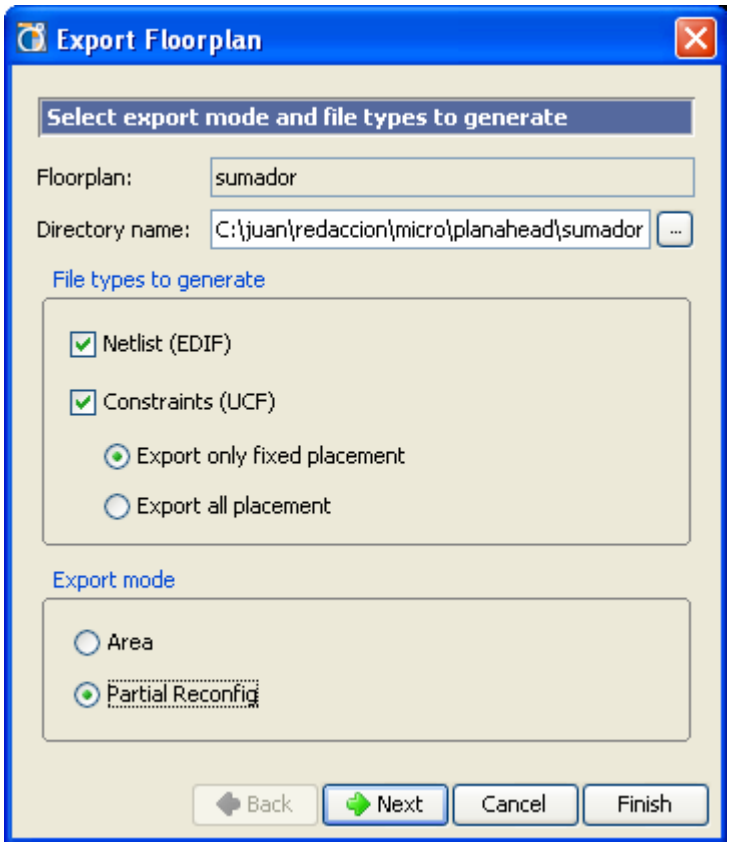


6.4.5. Implementación y Generación de los Bitstreams:

A continuación, tras guardar los cambios realizados, se debe exportar el *floorplanning*. Para ello se hace clic en *File* / *Export Floorplan*:



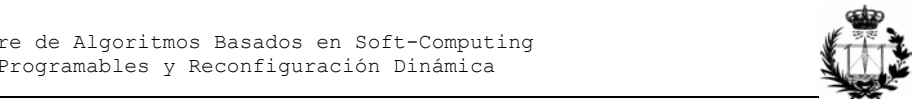
Se selecciona *Partial reconfig* dentro del apartado *Export mode* y se hace clic en *Finish*:



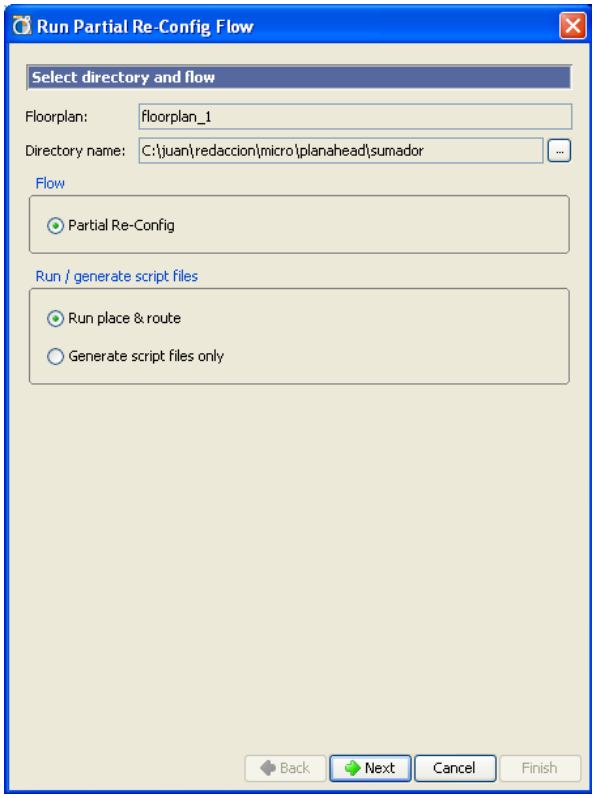
Debido a cambios en el flujo de diseño entre el *EDK 8.2* y el *PlanAhead 9.1.1*, es necesario realizar algunas acciones fuera del programa *PlanAhead*:

- Se accede a la carpeta *planahead/sumador/static* del proyecto y se copian los archivos ***.ngc*** que contienen algún elemento que deba estar en el *top level* (b*uses macro, DCM* y *BUFG*) y el ***.ucf*** en la carpeta *planahead/sumador*. En este caso:
  - *dcm_0_wrapper.ngc*
  - *dcm_1_wrapper.ngc*
  - *debug_module_wrapper.ngc*
  - *opb_socket_bridge_0_wrapper.ngc*
  - *top.ucf*

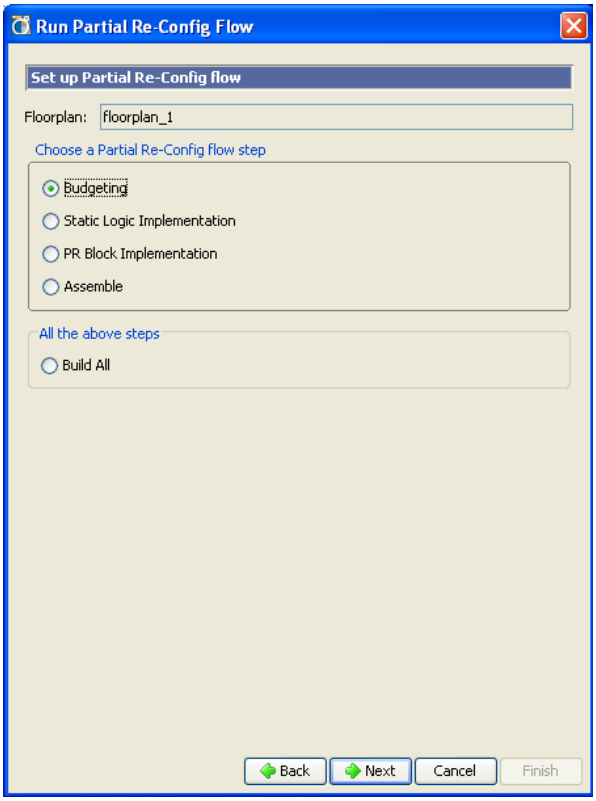
- Se copia el ***.ucf*** de la *carpeta planahead/sumador/static* en la carpeta *planahead/sumador/reconfig_CV*.

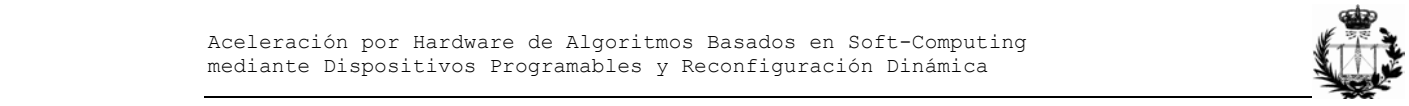


Una vez realizado esto se procede a la implementación del proyecto reconfigurable, haciendo clic en *Tools /Run Partial Reconfig*...Aparece la siguiente ventana, en la que se pincha en *Next*:

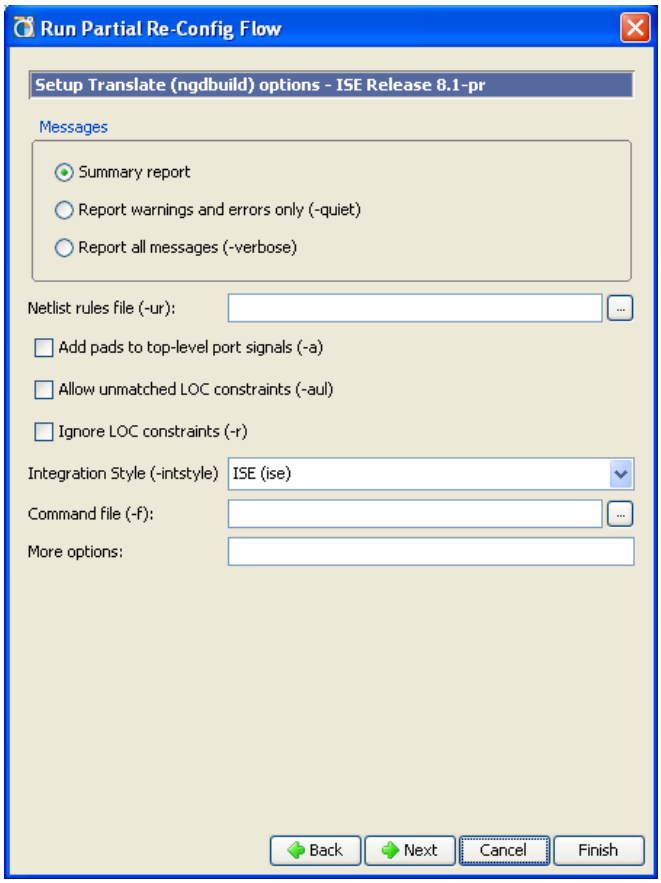


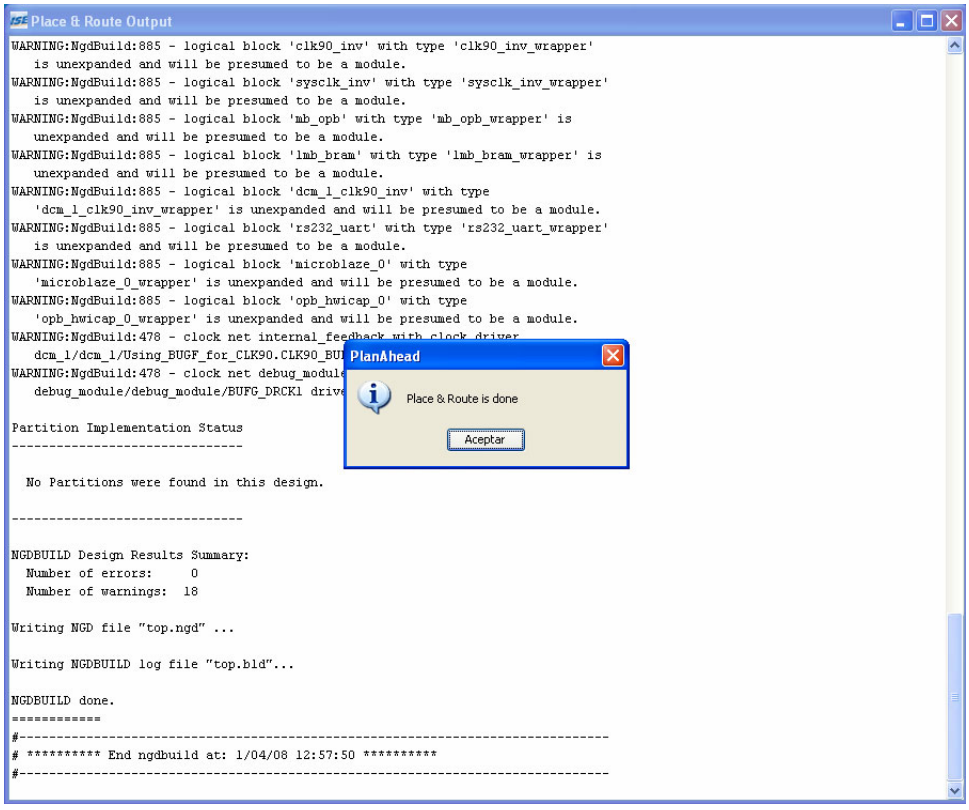
A continuación se selecciona *Budgeting* y se hace clic en *Next*:



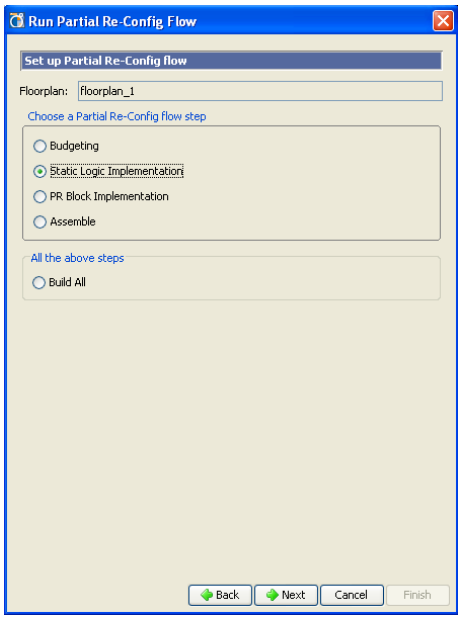


Se hace clic en *Finish* y se espera a que concluya la primera fase de la implementación:

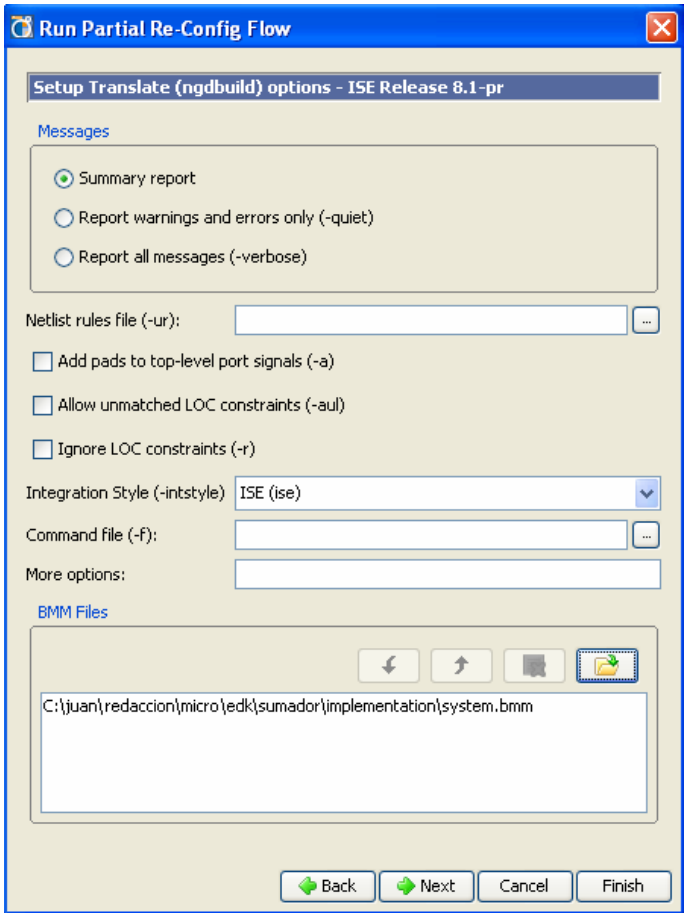


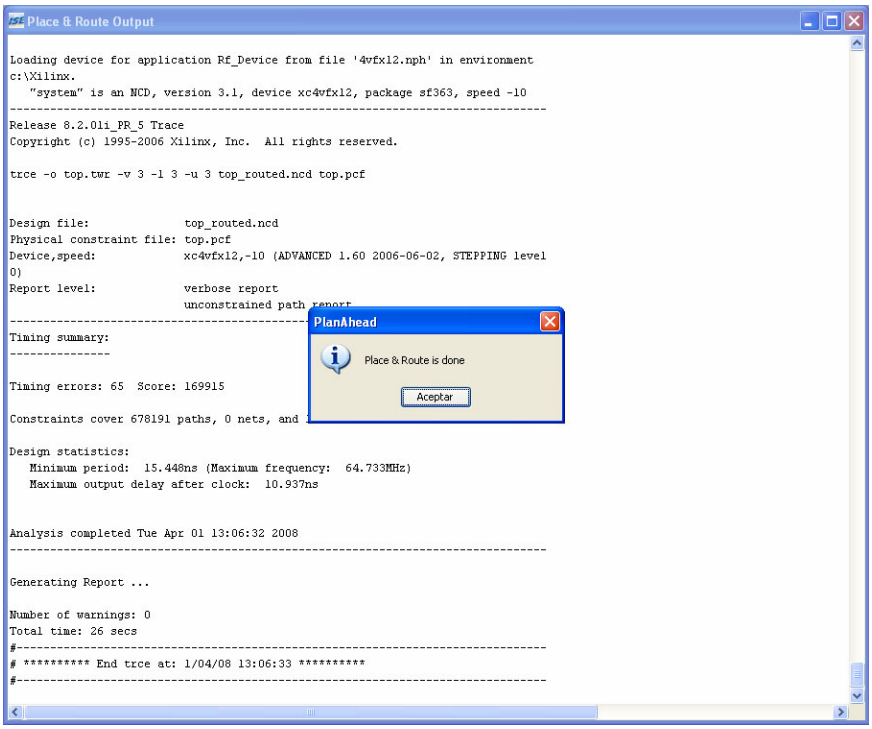


Se hace clic de nuevo en el menú *Tools /Run Partial Reconfig*...y en *Next*, y se selecciona *Implement Static Logic*, haciendo clic una vez más en *Next*:

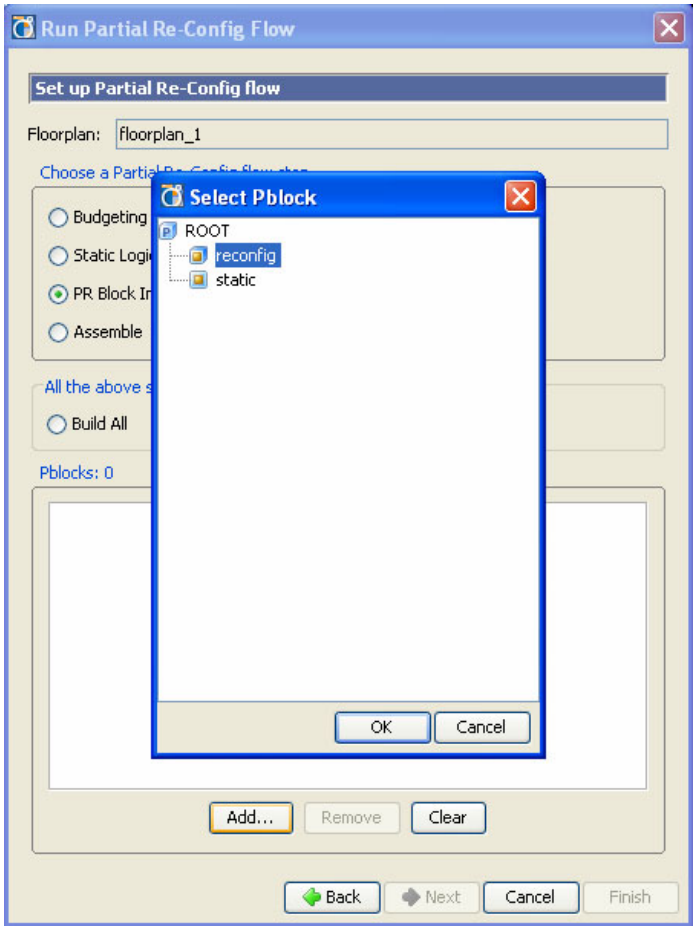


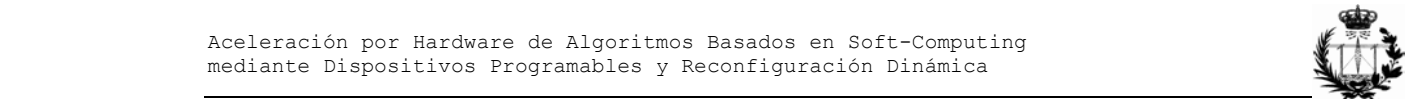
En la siguiente ventana se dejan las opciones tal cual están, y se vuelve a hacer clic en *Next*. A continuación se hace clic en el icono de la carpeta en *BMM Files*, para buscar el archivo *system.bmm* situado en la carpeta *implementation* del proyecto *EDK*. Para concluir se pincha en *Finish* y se espera a que se realice la implementación de la parte estática.



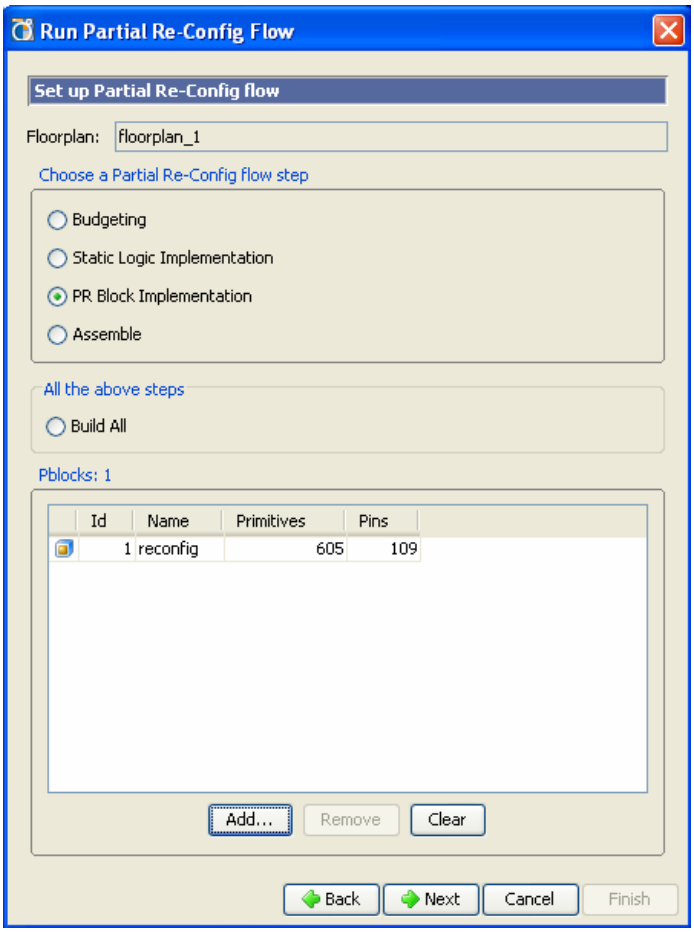


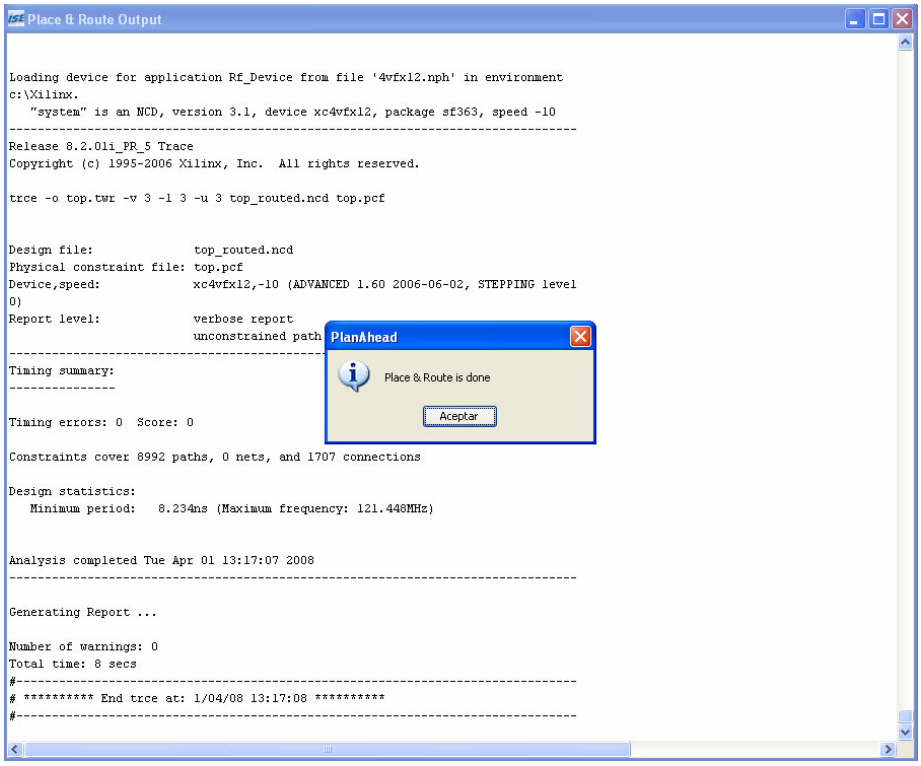
Se hace clic de nuevo en *Tools /Run Partial Reconfig*...y en *Next*, y se selecciona *PR Block Implementation.* A continuación se hace clic en el botón *add*, se selecciona el módulo reconfigurable *reconfig* y se hace clic en *OK.*



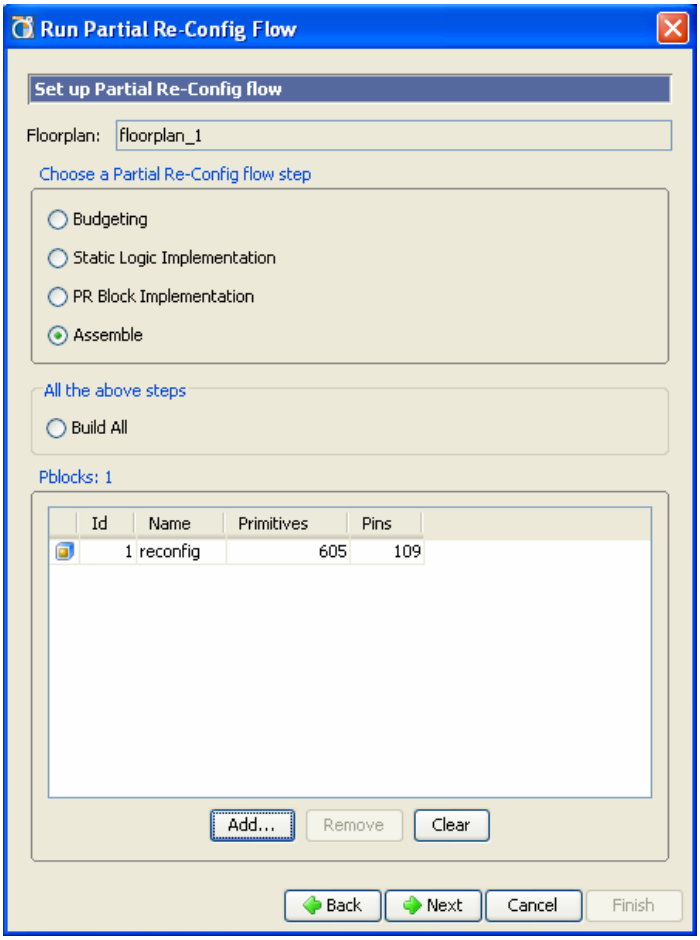


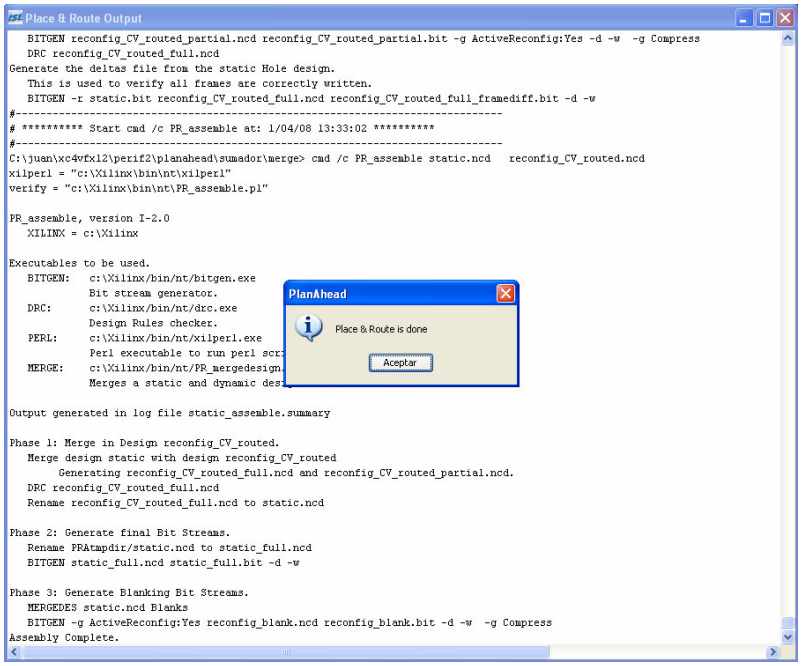
Se hace clic en *Next* y a continuación en *Finish*, de tal forma que se realizará la implementación del módulo reconfigurable:





Para concluir con la fase de implementación del proyecto reconfigurable en *PlanAhead* se hace clic de nuevo en *Tools /Run Partial Reconfig…*y en *Next*, y se selecciona *Assemble.* A continuación se hace clic en el botón *add*, se selecciona el módulo reconfigurable *reconfig* y se hace clic en *OK.* Se pincha en *Next* y finalmente en *Finish*, de tal manera que se generan los *bitstreams* del proyecto.





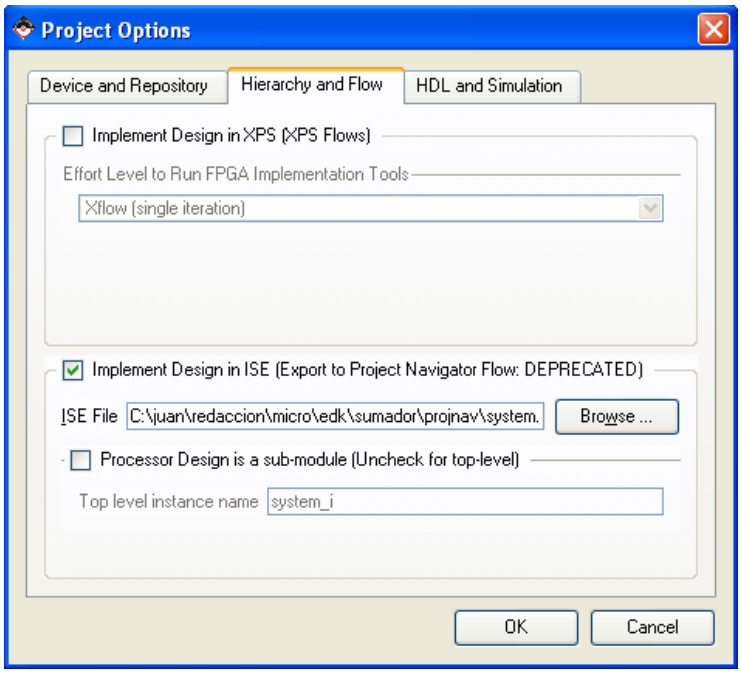
De esta forma se han generado una serie de archivos ***.bit*** y ***.bmm*** en la carpeta *planahead/sumador/merge*:

- *static_full.bit*: *bistream* completo del proyecto reconfigurable.
- *reconfig_cv_routed_partial.bit: bitstream* del modulo reconfigurable.
- *system_bd.bmm*: archivo de mapeado del programa en el bitstream.
- *reconfig_blank.bit*: *bitstream* vacío del modulo reconfigurable.

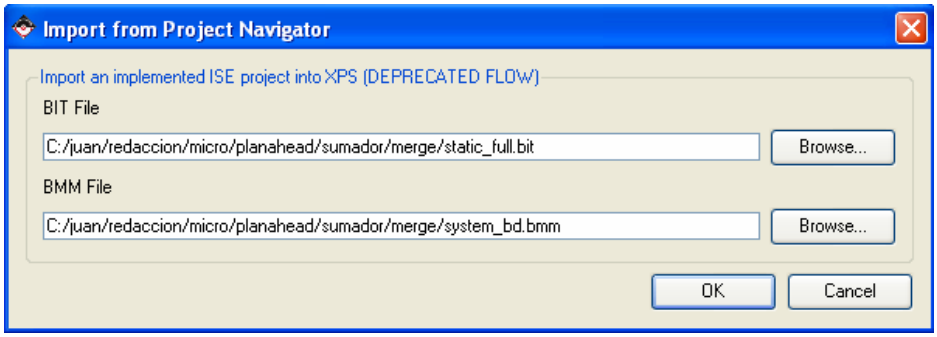
Ya es posible descargar en la *FPGA* el diseño. Pero en primer lugar se han de importar los ficheros obtenidos en *PlanAhead* al entorno *EDK*. **Es importante no cerrar el proyecto de *PlanAhead,* ya que si no se tendrán que repetir todos los pasos de nuevo.**

6.5. Carga del Programa de Reconfiguración en el Bitstream

Para cargar el programa de reconfiguración en el *bitstream* se abre el proyecto *EDK* del sumador y se pincha en el menú *Project/* *Project Options,* entrando en la pestaña *Hierachy and Flow*. Se selecciona la casilla *Implement Design in ISE* y se hace clic en *OK.*



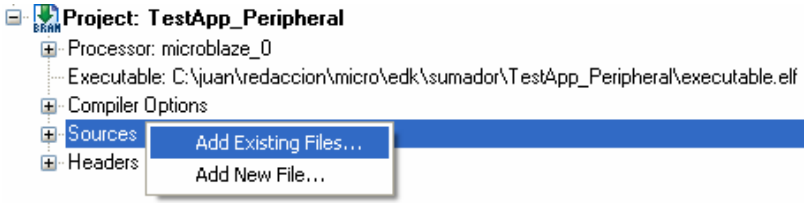
A continuación se hace clic en el menú *Options/* *Import from ISE* y se buscan en la carpeta *planahead/sumador/merge* los archivos *static_full.bit* y *system_bd.bmm.* Finalmente se hace clic en *OK*.



De esta manera se han importado ambos ficheros a la carpeta *implementation* del proyecto *EDK*, es decir, el programa supone que se ha realizado la implementación y la generación del *bitstream*, en este caso en *PlanAhead*.

A continuación se accede a la pestaña *Applications*, dentro de la ventana *Project Information Area*, donde se tiene la aplicación *TestApp_Peripheral*. Dentro del apartado *Sources* se han de incluir todos los archivos de código *C++* y librerías que afectan al proyecto (en este caso los del sumador y los del *hwicap*).

Para ello se hace clic con el botón derecho encima de *Sources* y se selecciona *Add Existing Files*.



En este caso se añaden los archivos de código:

- *edk\repositorio\MyProcessorIPLib\drivers\reconfig_v1_00_a\src\ reconfig.c*
- *edk\sumador\drivers\hwicap_v1_00_a\src\xhwicap_g.c*
- *edk\sumador\drivers\hwicap_v1_00_a\src\xhwicap_set_configuration.c*

Se realiza el mismo proceso para añadir las siguientes librerías:

- *edk\repositorio\MyProcessorIPLib\drivers\reconfig_v1_00_a\src\ reconfig.h*
- *edk\sumador\drivers\hwicap_v1_00_a\src\xhwicap_i.h*
- *edk\sumador\drivers\hwicap_v1_00_a\src\xhwicap.h*

En el archivo de código principal (*TestApp_Peripheral.c*) se escribe el siguiente código:

```
/*PARTIAL RECONFIGURATION OF FPGAs*/
/*Peripheral reconfiguration*/
/*Juan Quero Llor*/

#include <xuartlite_l.h>
#include "xparameters.h"

#include <xHwIcap.h>
#include "xstatus.h"
#include "xhwicap_i.h"

#define MEM_REG (*(unsigned*)MEM_ADDR)
unsigned MEM_ADDR;
XHwIcap InstIcap;

unsigned RAM_start_addr=0x24000000;
unsigned RAM_end_addr=0x24100000;
//get number from uart
int getNumber (){
       Xuint8 byte;
       Xuint8 uartBuffer[16];
       Xboolean validNumber;
       int digitIndex;
       int digit, number, sign;
       int c;
```

```
	while(1){

		byte = 0x00;
		digit = 0;
		digitIndex = 0;
		number = 0;
		validNumber = XTRUE;

		//get bytes from uart until RETURN is entered

		while(byte != 0x0d){

			byte = XUartLite_RecvByte(XPAR_RS232_UART_BASEADDR);
			uartBuffer[digitIndex] = byte;
			digitIndex++;
		}

		//calculate number from string of digits

		for(c = 0; c < (digitIndex - 1); c++){

			if(c == 0){

				//check if first byte is a "-"
				if(uartBuffer[c] == 0x2D){
					sign = -1;
					digit = 0;
				}
				//check if first byte is a digit
				else if((uartBuffer[c] >> 4) == 0x03){
					sign = 1;
					digit = (uartBuffer[c] & 0x0F);
				}
				else
					validNumber = XFALSE;
			}
			else{
				//check byte is a digit
				if((uartBuffer[c] >> 4) == 0x03){
					digit = (uartBuffer[c] & 0x0F);
				}
				else
					validNumber = XFALSE;
			}
			number = (number * 10) + digit;
		}

		number *= sign;

		if(validNumber == XTRUE){
			return number;
		}
		print("This is not a valid number.\n\r");
	}
}

int main (void) {
	char option;
	XStatus status;
	Xint32  operand1, operand2, result, step, words;
	unsigned start_addr, end_addr;
	step=0;
	operand1 = 0;
	operand2 = 0;
	result = 0;
```

```
      //bus macros disabled
      XIo_Out32(XPAR_OPB_SOCKET_BRIDGE_0_DCR_BASEADDR,0x00000000);

      print("\r\n-- Entering main() --"); //this lines executes but is not seen
   while(1) {
            print("\fFPGAs PARTIAL RECONFIGURATION");
            print("\r\nPeripheral partial self-reconfiguration");
            print("\r\nVirtex 4 (xc4vfx12)");
            print("\r\n\tJuan Quero Llor");

            print("\r\n\nSelect an option [1-4]:");
            if(step>0)
                  print("\r\n\t 1. Initialize hwicap (DONE)");
            else
                  print("\r\n\t 1. Initialize hwicap.");
            print("\r\n\t 2. New partial bitstream downloaded to system RAM");
            print("\r\n\t 3. Perform Partial Reconfiguration");
            print("\r\n\t 4. Microblaze peripheral operations");

            option = (char)inbyte();

            switch(option){

            case '1'://hwicap initialization
                  xil_printf("\r\n\r\nExecuting option %c.\r\n",option);
                  if(step==0){
                        status = XHwIcap_Initialize(&InstIcap,
                              XPAR_OPB_HWICAP_0_DEVICE_ID,
                              XHI_XC4VFX12);

                        if (status != XST_SUCCESS) {
                              xil_printf("\r\nXHwICAP Initialization failed. Device ID read: %x
status %x", InstIcap.DeviceIdCode, status);
                        } else {
                        xil_printf("\r\nXHwICAP Initialization success. Device ID read: %x",
                                          InstIcap.DeviceIdCode);
                        }

                        xil_printf("\r\n Full info received from ICAP: \
                              \r\n\t IsReady: %x \
                              \r\n\t DeviceIdCode: %x \
                              \r\n\t DeviceId: %x \
                              \r\n\t Rows: %x \
                              \r\n\t Cols: %x \
                              \r\n\t BramCols: %x \
                              \r\n\t BytesPerFrame: %x \
                              \r\n\t WordsPerFrame: %x \
                              \r\n\t ClbBlockFrames: %x \
                              \r\n\t BramBlockFrames: %x \
                              \r\n\t BramIntBlockFrames: %x", \
                              InstIcap.IsReady, InstIcap.DeviceIdCode, InstIcap.DeviceId, \
                              InstIcap.Rows, InstIcap.Cols, InstIcap.BramCols,
InstIcap.BytesPerFrame, \
                              InstIcap.WordsPerFrame, InstIcap.ClbBlockFrames, \
                              InstIcap.BramBlockFrames, InstIcap.BramIntBlockFrames);
                        step++;
                  } else {
                        print("\r\nHwicap Initialization already done");
                  }
                  print("\r\n\r\nPlease download partial bitstream with XMD and select
2\r\nPress any key to continue...");    option = (char)inbyte();
                  break;

            case '2':    //Recognize new partial bitstream downloaded to system RAM
                  if(step==0)
                  {
                        print("\r\n\r\nPlease initialize Hwicap before (step 1)");
                        print("\r\nPress any key to continue...");    option = (char)inbyte();
```

```
                        break;
                }

                xil_printf("\r\n\r\nExecuting option %c.\r\n",option);

                start_addr=RAM_start_addr;
                end_addr=RAM_start_addr;
                MEM_ADDR = start_addr;

                while (MEM_REG!=0x0000000D && MEM_ADDR<RAM_end_addr){
                        MEM_ADDR=MEM_ADDR+4;
                        end_addr=MEM_ADDR;
                }
                if(MEM_ADDR>=RAM_end_addr){
                        print("\r\n\r\nThere is not a valid partial bitstream on RAM memory,
please download it with XMD");
                        print("\r\nPress any key to continue...");    option = (char)inbyte();
                        break;
                }
                words=0;
                for (MEM_ADDR = start_addr; MEM_ADDR <= end_addr; MEM_ADDR += 4) {
                        //xil_printf("value(0x%X)=%8X \r\n",MEM_ADDR,MEM_REG);
                        if(MEM_ADDR == start_addr || MEM_ADDR == end_addr)
                                xil_printf("value(0x%X)=%8X \r\n",MEM_ADDR,MEM_REG);
                        words++;
                }
                step=2;
                print("\r\nPartial Bitstream is ready.\r\nPress any key to continue...");
      option = (char)inbyte();
                break;

        case '3': //FPGA Partial reconfiguration

                if(step==0)
                {
                        print("\r\n\r\nPlease initialize Hwicap before(step 1)");
                        print("\r\nPress any key to continue...");    option = (char)inbyte();
                        break;
                }
                if(step==1)
                {
                        print("\r\n\r\nPlease download partial bitstream with XMD (step 2)");
                        print("\r\nPress any key to continue...");    option = (char)inbyte();
                        break;
                }
                xil_printf("\r\n\r\nExecuting option %c.",option);

                status=XHwIcap_SetConfiguration(&InstIcap, start_addr, words);
                if (status != XST_SUCCESS)
                        xil_printf("\r\nFPGA Reconfiguration failed.\r\n\nIMPORTANT: Please
download the initial full bitstream again",
                                                InstIcap.DeviceIdCode, status);
                else
                        xil_printf("\r\nFPGA Reconfiguration success. Device ID read: %x status
%x\r\n",
                                                InstIcap.DeviceIdCode, status);
                step=1;
                print("\r\nPress any key to continue...");    option = (char)inbyte();
        break;

        case '4' :   //Microblaze Operations

                //bus macros enabled
                XIo_Out32(XPAR_OPB_SOCKET_BRIDGE_0_DCR_BASEADDR,0x00000001);

                //get first operand
                print("\r\n\r\nEnter first operand.\r\n");
                operand1 = getNumber();
```

```
                    xil_printf("Operand1 = %d\r\n", operand1);

                    //get second operand
                    print("Enter second operand.\r\n");
                    operand2 = getNumber();
                    xil_printf("Operand2 = %d\r\n", operand2);

                    //send operands to peripheral
                    result=operacion(operand1, operand2);
                    xil_printf("Result = %d\r\n", result);

                    //bus macros disabled
                    XIo_Out32(XPAR_OPB_SOCKET_BRIDGE_0_DCR_BASEADDR,0x00000000);

                    print("\r\nPress any key to continue...");    option = (char)inbyte();

             break;

             default:
                    xil_printf("\r\n The character selected %c (ASCII %d) was not valid
option.",option,option);
                    print("\r\nPress any key to continue...");
                    option = (char)inbyte();

             }
      }

      //this line never executes
      print("\r\n-- Exiting main() --\r\n");
      return 0;
}
```

El método *getNumber()* simplemente recibe un número de la *UART* y permite que éste tenga más de un dígito (todos los dígitos tecleados hasta pulsar *enter*).

Por otro lado el método *main()* tiene cuatro opciones (tecleando números del 1 al 4):

1. Se inicializa el *hwicap*.
2. Una vez cargado el *bitstream* en la *RAM* con el *debugger*, se reconoce el inicio y el final del mismo.
3. Se reconfigura, leyendo el *bitstream* parcial de la memoria *RAM* (es importante haberlo cargado antes con el depurador) y descargándolo en la placa a través del *hwicap* mediante *XHwIcap_SetConfiguration()*.
4. Se realiza la operación aritmética del periférico reconfigurable en cuestión, dependiendo de la reconfiguración el periférico será sumador, substractor o multiplicador. Para ello es importante que se habiliten los *enable* de los *buses macro*.

Para compilar el programa se utiliza el menú *Software /*  *Build all user Applications*. A continuación se carga el ejecutable en el *bitstream*, pinchando en el menú *Device Configuration /*  *Update Bitstream.*

Se abre una ventana de *HyperTerminal* según el Anexo 3 y finalmente se puede descargar el *bitstream* del sumador en la placa con el menú *Device Configuration* / *Download Bitstream*. De esta forma se podrá comprobar el correcto funcionamiento del proyecto que se ha realizado con el primer periférico, sumador, pulsando la tecla "4" e introduciendo los números para la suma:

```
FPGAs PARTIAL RECONFIGURATION
Peripheral partial self-reconfiguration
Virtex 4 (xc4vfx12)
        Juan Quero Llor

Select an option [1-4]:
         1. Initialize hwicap.
         2. New partial bitstream downloaded to system RAM
         3. Perform Partial Reconfiguration
         4. Microblaze peripheral operations
```

6.6. Síntesis de los demás Módulos Reconfigurables

Una vez realizado el primer proyecto (sumador) al completo y comprobado que funciona correctamente en la placa, se puede proceder a realizar la síntesis de los otros periféricos.

Para ello se deberá copiar la carpeta *edk/sumador* cambiándole el nombre, sustituyéndolo por el del otro periférico, en este caso multiplicador (*edk/multiplicador*). Esto se podrá realizar tantas veces como periféricos tengamos en el repositorio:

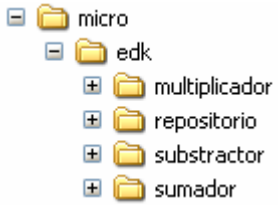
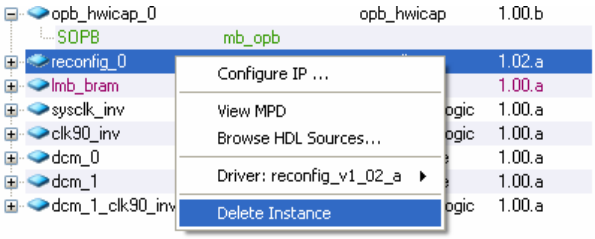


Figura 42. Estructura de directorios de proyectos de *EDK*

Se abre el nuevo proyecto multiplicador, y se sustituye un periférico por otro, para ello se hace clic con el botón derecho en *reconfig_0* en la ventana *System Assembly View* y se selecciona *Delete Instance*:



De esta forma ya se puede añadir un nuevo periférico, el multiplicador (*reconfig 1.02a*), haciendo doble clic sobre él en la ventana *IP Catalog*, dentro de *Peripheral Repositories*:

El *EDK 8.2* comete un fallo al cerrar y abrir proyectos que estaban guardados en carpetas y éstas han sido movidas a otro lugar. Cuando se vuelven a abrir, siguen utilizando el repositorio de periféricos del directorio anterior (exista o no), y si se procede a cambiarlo en el menú correspondiente y se guarda y se cierra el proyecto, al volverlo a abrir el programa lee los periféricos de ambos repositorios (pudiéndose producir entonces errores humanos).

Para que no ocurra esto es necesario editar el fichero *system.xmp* con *WordPad* (ojo: abrir con *WordPad*, pero no seleccionar "Utilizar siempre el programa seleccionado para realizar esta acción"), y eliminar todas las líneas *"ModuleSearchPath"*, dejando sólo la del repositorio actual: *ModuleSearchPath: C:/juan/redaccion/micro/edk/repositorio/*

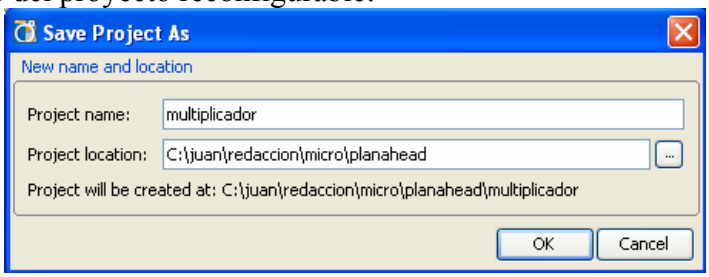
A continuación se debe abrir el archivo ***.mhs*** para añadir las mismas líneas de código que se añadieron al anterior módulo *reconfig*, que han sido borradas por el *EDK* al sustituir el periférico:

```
BEGIN reconfig
 PARAMETER INSTANCE = reconfig_0
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_BASEADDR = 0x73e00000
 PARAMETER C_HIGHADDR = 0x73e0ffff
 PORT OPB_Clk = ROPB_Clk_0
 PORT OPB_Rst = ROPB_Rst_0
 PORT OPB_ABus = ROPB_ABus_0
 PORT OPB_BE = ROPB_BE_0
 PORT OPB_RNW = ROPB_RNW_0
 PORT OPB_select = ROPB_select_0
 PORT OPB_seqAddr = ROPB_seqAddr_0
 PORT OPB_DBus = ROPB_DBus_0
 PORT Sl_DBus = RSl_DBus_0
 PORT Sl_errAck = RSl_errAck_0
 PORT Sl_retry = RSl_retry_0
 PORT Sl_toutSup = RSl_toutSup_0
 PORT Sl_xferAck = RSl_xferAck_0
END
```

Para finalizar se puede generar el nuevo *netlist* sintetizando el proyecto, para lo que se hace clic en *Hardware/* *Generate Netlist.*

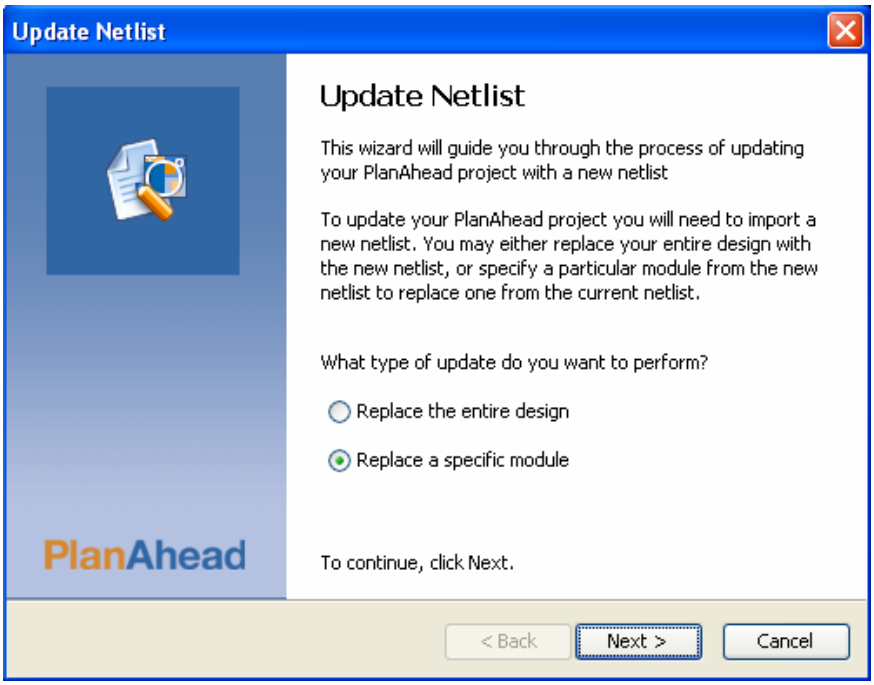
6.7. Implementación de los demás Módulos Reconfigurables

Se ha de obtener el *bitstream parcial* de los otros módulos reconfigurables del diseño, en este caso el periférico multiplicador y el substractor. Se vuelve a la ventana abierta de *PlanAhead* en la que está el proyecto sumador, se hace clic en el menú *File / Save Project As*. Se escribe el nombre del proyecto así como su ubicación dentro de la carpeta *planahead* del proyecto reconfigurable:

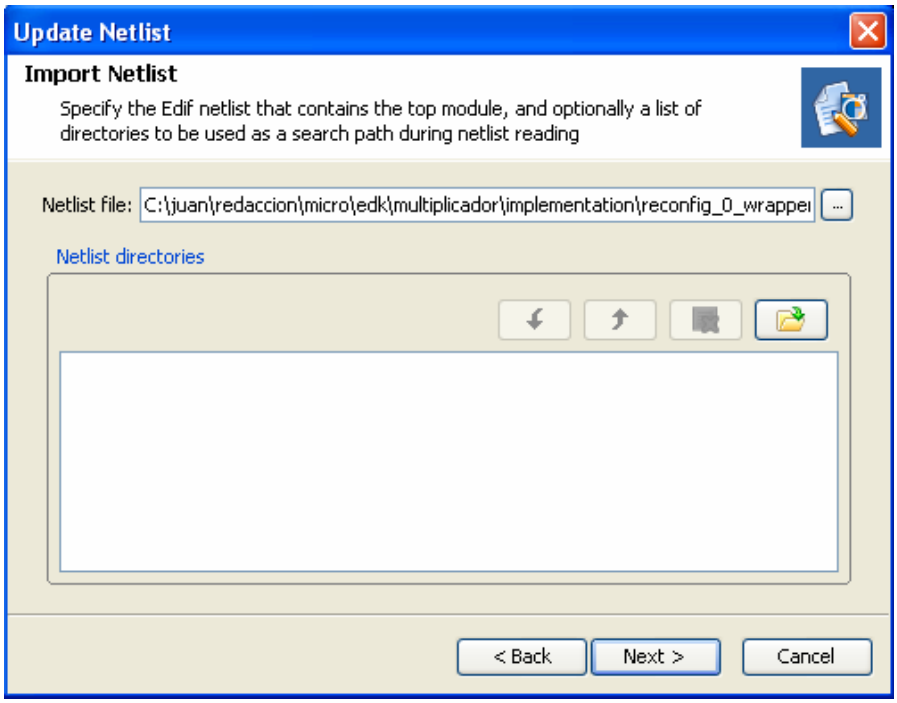


A continuación se han de copiar todos los ficheros de la carpeta *planahead/sumador*, dentro de la carpeta *planahead/multiplicador* (operación externa a *PlanAhead*, pero sin cerrarlo en ningún caso).

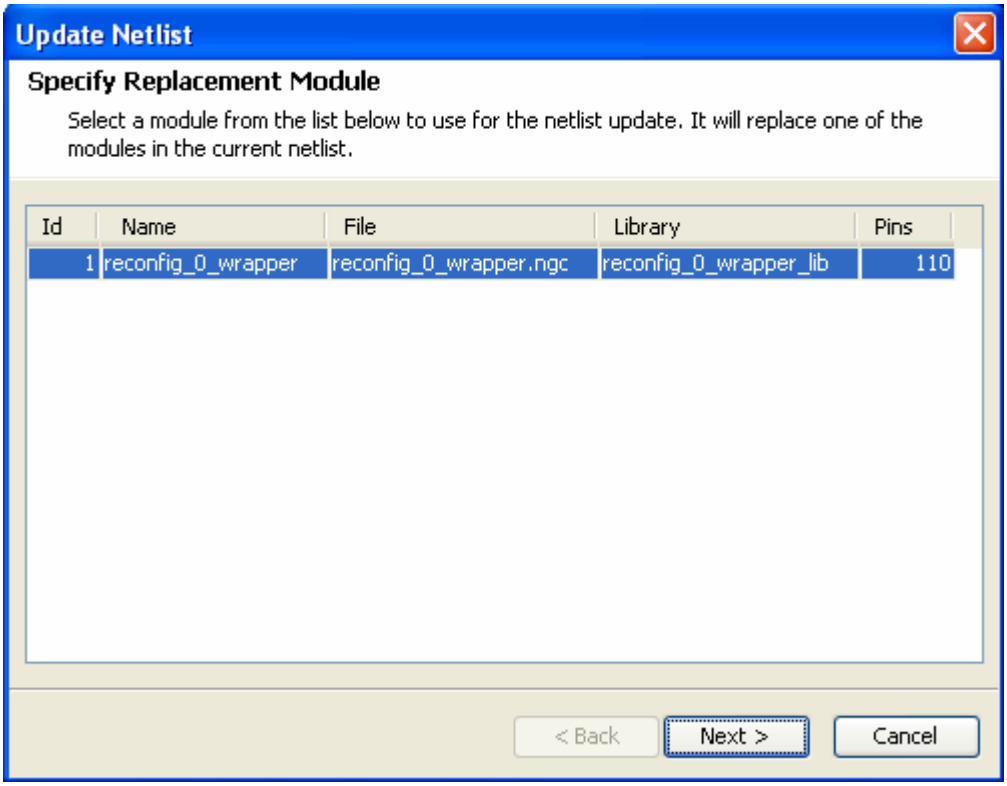
Una vez realizado esto se ha de sustituir el módulo reconfigurable por el nuevo (multiplicador). Para ello se hace clic en el menú *File / Update Netlist*, se selecciona *Replace a specific module* y se pincha en *Next*.



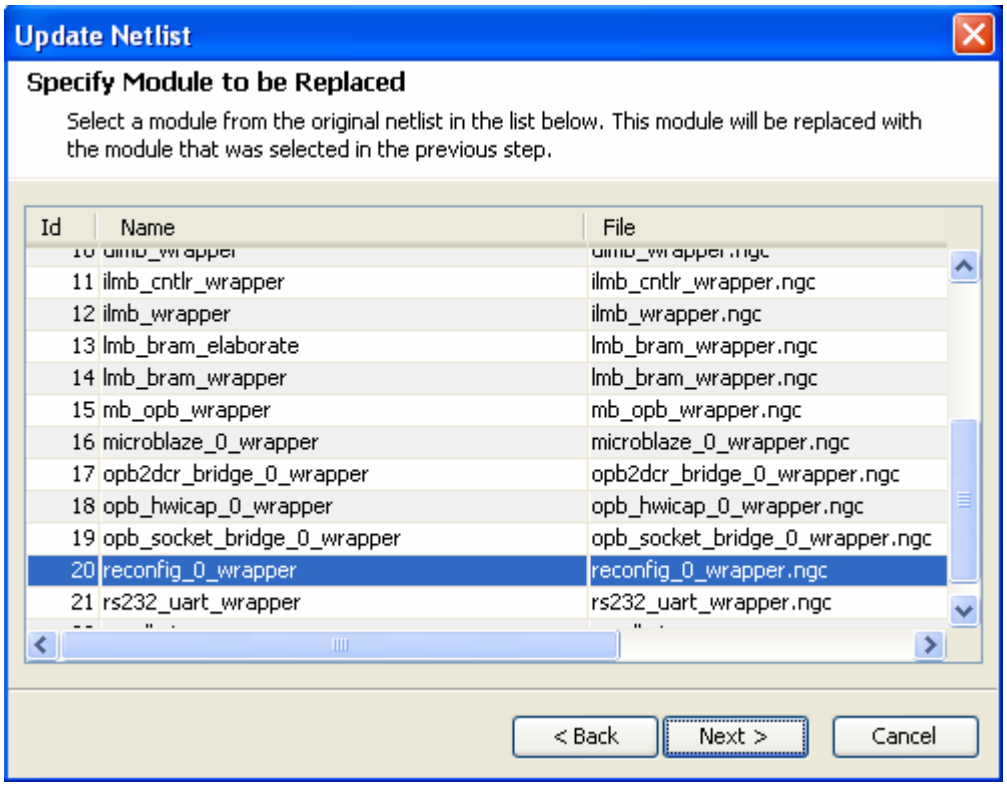
A continuación se busca en la carpeta *edk/multiplicador/implementation* el archivo *netlist* del módulo reconfigurable (*reconfig_0_wrapper.ngc*) y se hace clic en *Next*:



Se selecciona el módulo que contiene dicho archivo y se hace clic en *Next*:



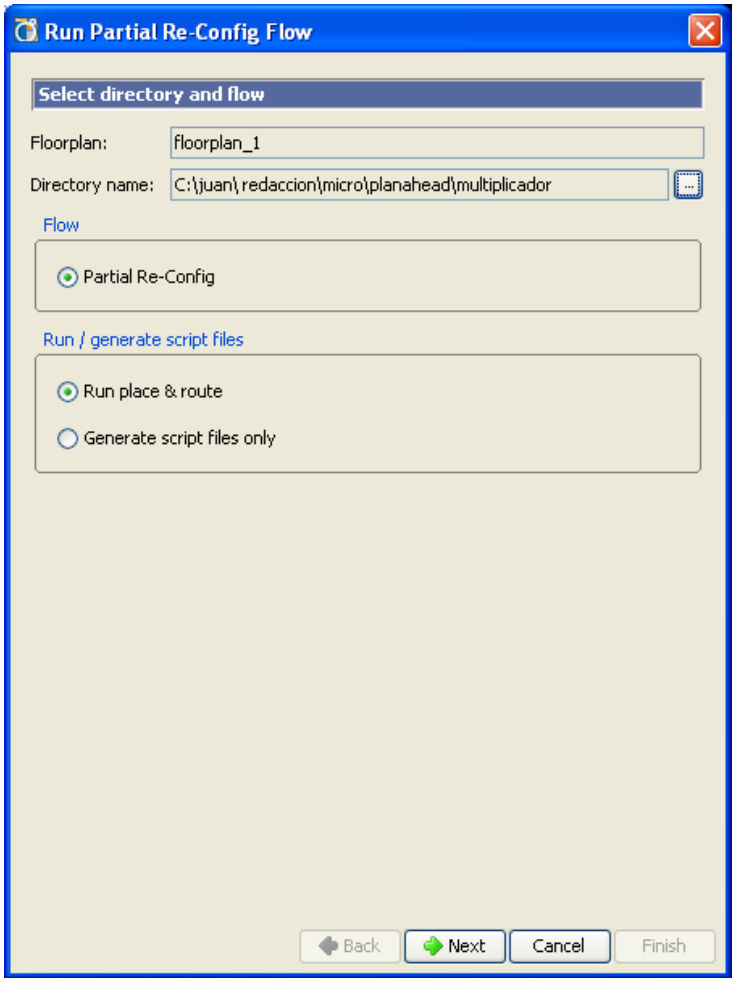
Se selecciona el módulo que va a ser reemplazado (*reconfig_0_wrapper*) y se hace clic en *Next* y finalmente en *Finish*.



Finalmente se ha de hacer clic en *File / Save Floorplan…,* para guardar los cambios. De esta manera se ha sustituido el *netlist* del módulo reconfigurable sumador, por el *netlist* del módulo reconfigurable multiplicador.

Antes de obtener el nuevo *bitstream* parcial hay que copiar el archivo *reconfig_0_wrapper.ngc* situado en la carpeta *edk/multiplicador/implementation* en la carpeta *planahead/multiplicador/reconfig_CV*, ya que este archivo no se actualiza automáticamente.

A continuación ya se podrá implementar el módulo reconfigurable y obtener su *bitstream* parcial. Para ello se hace clic en el menú *Tools / Run Partial Reconfig…*, se busca en el apartado *Directory Name* la carpeta del proyecto actual *planahead/multiplicador* y se hace clic en *Next*.



Ahora sólo se han de repetir los pasos *PR Block Implementation* y *Assemble* tal y como se hizo en el apartado anterior, obteniendo con ello los *bitstreams* del nuevo proyecto multiplicador. A continuación se repiten todos los pasos dados para generar nuevos *bitstreams* parciales con el substractor.

6.8. Ejecución del Programa de Auto-Reconfiguración Parcial

Una vez conectado el *HyperTerminal*, se procede a abrir el proyecto *EDK* del sumador y a descargar en la placa su *bitstream* completo, tal y como se vio en capítulos anteriores (importándolo previamente de la carpeta *PlanAhead/sumador/merges* si no se ha hecho ya).

Aparecerá entonces el menú del programa de reconfiguración, donde se pueden ver las cuatro opciones: inicializar el *hwicap*, reconocer el bitstream parcial, reconfigurar y realizar la operación.

En primer lugar se comprueba mediante la cuarta opción que el programa y el periférico funcionan, realizando la suma correctamente:

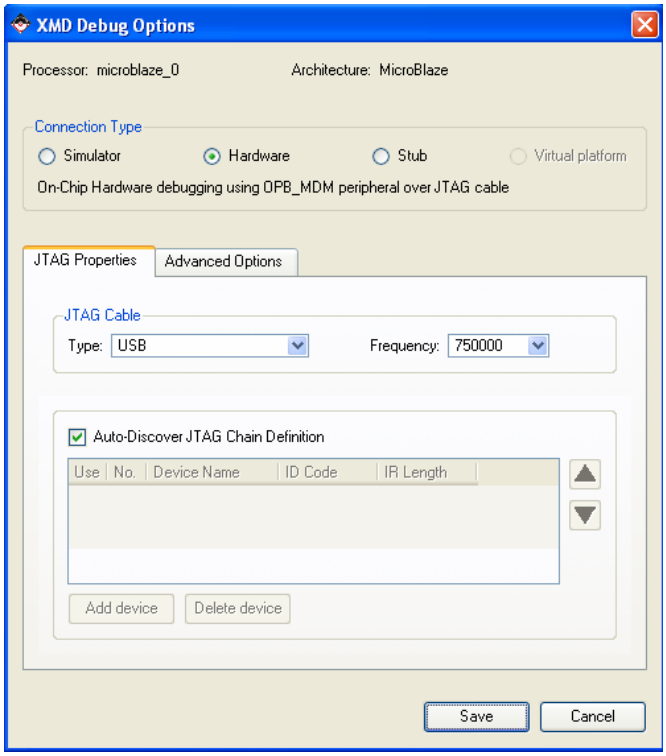
```
FPGAs PARTIAL RECONFIGURATION
Peripheral partial self-reconfiguration
Virtex 4 (xc4vfx12)
        Juan Quero Llor

Select an option [1-4]:
         1. Initialize hwicap.
         2. New partial bitstream downloaded to system RAM
         3. Perform Partial Reconfiguration
         4. Microblaze peripheral operations

Enter first operand.
Operand1 = 25
Enter second operand.
Operand2 = 23
Result = 48

Press any key to continue...
```

A continuación se inicializa el *hwicap*, para poder realizar la reconfiguración:

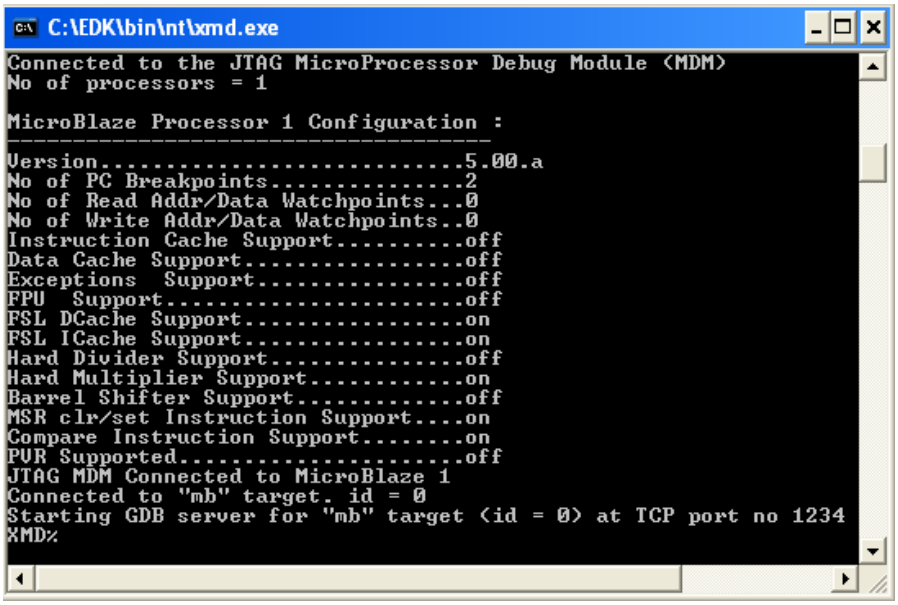
```
Select an option [1-4]:
         1. Initialize hwicap.
         2. New partial bitstream downloaded to system RAM
         3. Perform Partial Reconfiguration
         4. Microblaze peripheral operations

Executing option 1.

XHwICAP Initialization success. Device ID read: 1E58093
 Full info received from ICAP:
         IsReady: 11111111
         DeviceIdCode: 1E58093
         DeviceId: 0
         Rows: 40
         Cols: 18
         BramCols: 3
         BytesPerFrame: A4
         WordsPerFrame: 29
         ClbBlockFrames: 248
         BramBlockFrames: C0
         BramIntBlockFrames: 42

Please download partial bitstream with XMD and select 2
Press any key to continue...
```

Para descargar el *bitstream* parcial en la memoria *RAM*, es necesario ejecutar el depurador hardware en el *EDK*. En primer lugar se ha de copiar el *bitstream* parcial del substractor (*reconfig_cv_routed_partial.bit*) de la carpeta *planahead/multiplicador/merge* a la carpeta *edk/sumador,* para poder descargarlo en la placa. Puede cambiarse el nombre a *substractor.bit*, para que sea más sencilla su utilización.

A continuación se establecen las opciones del depurador, para lo que hace clic en *Debug /* *XMD Debug Options* y se marcan las opciones del cable por el que se va a depurar el microprocesador (en este caso el cable *USB* por el que se programa), pinchando finalmente en *Save*.



Para ejecutar el depurador *XMD* se hace clic en el menú *Debug /* *Launch XMD...*



Se ejecuta la sentencia para descargar un archivo en una dirección del *MicroBlaze*: *dow –data substractor.bit 0x24000000*, donde *substractor.bit* es el *bitstream* parcial del substractor, y 0x24000000 es la dirección donde empieza la memoria *RAM*, lugar en el que el programa reconfigurable empieza a leer el *bitstream* para enviarlo al *ICAP*.

```
XMD% dow -data substractor.bit 0x24000000
XMD% con
```

Se puede observar que el depurador *XMD* tarda unos segundos para la descarga, antes de aparecer la siguiente línea *XMD%*. Si no tarda quiere decir que no se está cargando correctamente el archivo en memoria *RAM*, posiblemente por un fallo de ésta, porque no se ha situado el *bitstream* parcial del multiplicador en la carpeta *edk/sumador* o porque ha habido alguna errata al escribir el nombre del *bitstream.*

A continuación es necesario ejecutar la sentencia *"con",* para que el programa continúe ejecutándose:

```
XMD% con
Processor started. Type "stop" to stop processor
RUNNING> _
```

Una vez se ha realizado la descarga, se vuelve a la ventana de HyperTerminal, y se reconoce el bitstream parcial, pulsando "2".

```
FPGAs PARTIAL RECONFIGURATION
Peripheral partial self-reconfiguration
Virtex 4 (xc4vfx12)
        Juan Quero Llor

Select an option [1-4]:
         1. Initialize hwicap (DONE)
         2. New partial bitstream downloaded to system RAM
         3. Perform Partial Reconfiguration
         4. Microblaze peripheral operations

Executing option 2.
value(0x24000000)=   90FF0
value(0x24000C78)=       D

Partial Bitstream is ready.
Press any key to continue...
```

Para finalizar se ejecuta la opción "3", que realiza la reconfiguración parcial.

```
FPGAs PARTIAL RECONFIGURATION
Peripheral partial self-reconfiguration
Virtex 4 (xc4vfx12)
        Juan Quero Llor

Select an option [1-4]:
         1. Initialize hwicap (DONE)
         2. New partial bitstream downloaded to system RAM
         3. Perform Partial Reconfiguration
         4. Microblaze peripheral operations

Executing option 3.
FPGA Reconfiguration success. Device ID read: 1E58093 status 0

Press any key to continue...
```

En este momento la *FPGA* debería haberse auto-reconfigurado, y el programa debería funcionar como substractor, aunque fallos en el parche provisional de *Xilinx* para la reconfiguración (*EA_PR*) hacen que no se auto-reconfigure correctamente.

Probablemente el fallo del parche se encuentre en la interacción entre el *hwicap*, la memoria *RAM* y los periféricos con *IPIF* (elemento instanciado automáticamente por el *wizard* del *EDK* que permite el direccionamiento y la conexión del periférico con el bus *OPB*) en el momento de la reconfiguración parcial

Para comprobar esta hipótesis se procede a realizar reconfiguración parcial en lugar de auto-reconfiguración parcial con los *bitstreams* obtenidos (es decir, en lugar de ser descargados por el *hwicap*, son descargados por *iMPACT*).

6.9. Reconfiguración Parcial de Periférico

Una vez se ha descargado en la placa el *bitstream* completo del sumador con el *EDK* y comprobado su correcto funcionamiento como sumador, se abre el *iMPACT* y se descarga en la *FPGA* el *bitstream* parcial del substractor. Esto es posible hacerlo así por dos razones:

- Porque el parche *EA_PR* está instalado (no se puede descargar un *bitstream* parcial desde un ordenador que no lo tenga instalado).
- Porque el programa ejecutado en *MicroBlaze* hace que en todo momento los *buses macro* estén desconectados excepto en el momento de realizar las operaciones aritméticas, lo que permite descargar el *bitstream* parcial en cualquier momento sin problemas.

A continuación se procede a confirmar que se ha realizado correctamente la reconfiguración, comprobando si se realiza la resta:

```
FPGAs PARTIAL RECONFIGURATION
Peripheral partial self-reconfiguration
Virtex 4 (xc4vfx12)
        Juan Quero Llor

Select an option [1-4]:
         1. Initialize hwicap (DONE)
         2. New partial bitstream downloaded to system RAM
         3. Perform Partial Reconfiguration
         4. Microblaze peripheral operations

Enter first operand.
Operand1 = 45
Enter second operand.
Operand2 = 32
Result = 13

Press any key to continue...
```

Se pueden repetir los mismos pasos para descargar el *bitstream* parcial del multiplicador:

```
FPGAs PARTIAL RECONFIGURATION
Peripheral partial self-reconfiguration
Virtex 4 (xc4vfx12)
        Juan Quero Llor

Select an option [1-4]:
         1. Initialize hwicap.
         2. New partial bitstream downloaded to system RAM
         3. Perform Partial Reconfiguration
         4. Microblaze peripheral operations

Enter first operand.
Operand1 = 12
Enter second operand.
Operand2 = 3
Result = 36

Press any key to continue...
```

Queda por tanto confirmado que el fallo no se encuentra en el proceso realizado de síntesis, implementación u obtención del *bitstream* parcial ya que éstos funcionan correctamente. Por último, para terminar de confirmar la hipótesis, falta crear un periférico que no esté dotado de *IPIF* y proceder a comprobar si es capaz de realizar la auto-reconfiguración parcial.

Por ello se propone a continuación otra solución alternativa para la auto-reconfiguración de periféricos de *MicroBlaze*, sin utilizar el *IPIF* (y por tanto modificando los periféricos realizados con *EDK)*.

6.10. Auto-Reconfiguración Parcial de Periférico sin IPIF

Debido a los problemas relacionados con el parche de *Xilinx* y la auto-reconfiguración parcial de periféricos con *IPIF*, se puede realizar auto-reconfiguración parcial con un periférico que no esté dotado de *IPIF* (otro código realiza la función de este módulo).

Se parte del mismo proyecto con los mismos elementos, solo que ahora se instancia un periférico distinto que realiza el direccionamiento sin *IPIF*, que en este caso se llama *opb_PRR* y de un *opb_socket_brigde* también distinto.

El periférico en este ejemplo simplemente cambia el valor de un registro cuando es reconfigurado parcialmente. Su principal diferencia con respecto al tratado con anterioridad se encuentra en el archivo *opb_PRR.vhd* del código del periférico, que no realiza el direccionamiento mediante *IPIF,* sino mediante un código de mucho menor peso que realiza una función similar pero de forma más simple.

El archivo *opb_PRR.vhd* se encuentra en el directorio *pcores\opb_PRR\hdl\vhdl* (del proyecto *EDK* o del repositorio, según donde se haya situado el periférico).

El código del *opb_socket_brigde* es el original proporcionado con las herramientas *EA_PR*, ya que en este caso se realiza la decodificación de la dirección del periférico en dos tiempos, primero en el *opb_socket_brigde*, y a continuación en el propio periférico. Se encuentra en el directorio *pcores\opb_socket_bridge\hdl\vhdl\opb_socket.vhd*

Es importante corregir un error de direccionamiento en este archivo proporcionado por *Xilin*x. En la línea 198 donde pone:

```
  Socket2PRR_ABus <= OPB_ABUS & C_MASKADDR;
```

Tiene que poner:

```
  Socket2PRR_ABus <= OPB_ABUS and C_MASKADDR;
```

Esto no provoca ningún error para periféricos de un solo registro como en este ejemplo, pero si que provocaría un error de direccionamiento en caso de crear un periférico con más de un registro.

Una vez que se han añadido el nuevo *opb_socket_brigde* y el periférico *opb_PRR*, se realizan exactamente los mismos pasos en el *EDK* que en el caso anterior, a excepción de que no hay que incluir el código referente al direccionamiento del periférico *opb_PRR* en el archivo ***.mhs***, es decir, **no hay que incluir** las siguientes líneas:

```
PARAMETER C_BASEADDR = 0x********
PARAMETER C_HIGHADDR = 0x********
```

Esto es así porque el nuevo *opb_socket_brigde* y el periférico ya realizan la decodificación completa de la señal del bus de direcciones del *OPB*. Si que se deben añadir en el ***.mhs*** las líneas que conexionan el *opb_socket_brigde* con el periférico.

A continuación también se repiten exactamente los mismos pasos en *PlanAhead*, y posteriormente ya se puede volcar el programa en memoria y descargar en la placa.

En la ventana de *HyperTerminal* se teclea "4", lo que hace que el periférico envíe el valor de su registro interno, en este caso 3.

```
FPGAs PARTIAL RECONFIGURATION
Peripheral partial self-reconfiguration without IPIF
Virtex 4 (xc4vfx12)
        Juan Quero Llor

Select an option [1-4]:
         1. Initialize hwicap.
         2. New partial bitstream downloaded to system RAM
         3. Perform Partial Reconfiguration
         4. Microblaze peripheral register

Executing option 4.

Result = 3

Press any key to continue...
```

A continuación se teclea "1" para inicializar el *hwicap* y se procede a descargar el *bitstream* parcial en la memoria *RAM* de la placa a través del *debugger* con el procedimiento descrito con anterioridad (en este caso *dow –data reconfig.bit 0x2400000*).

```
Select an option [1-4]:
        1. Initialize hwicap.
        2. New partial bitstream downloaded to system RAM
        3. Perform Partial Reconfiguration
        4. Microblaze peripheral register

Executing option 1.

XHwICAP Initialization success. Device ID read: 1E58093
 Full info received from ICAP:
        IsReady: 11111111
        DeviceIdCode: 1E58093
        DeviceId: 0
        Rows: 40
        Cols: 18
        BramCols: 3
        BytesPerFrame: A4
        WordsPerFrame: 29
        ClbBlockFrames: 248
        BramBlockFrames: C0
        BramIntBlockFrames: 42

Please download partial bitstream with XMD and select 2
Press any key to continue...
```

Se teclea "2" para reconocer el bitstream parcial:

```
FPGAs PARTIAL RECONFIGURATION
Peripheral partial self-reconfiguration without IPIF
Virtex 4 (xc4vfx12)
        Juan Quero Llor

Select an option [1-4]:
        1. Initialize hwicap (DONE)
        2. New partial bitstream downloaded to system RAM
        3. Perform Partial Reconfiguration
        4. Microblaze peripheral register

Executing option 2.
value(0x24000000)=   90FF0
value(0x24010178)=       D

Partial Bitstream is ready.
Press any key to continue...
```

"3" para proceder a la auto-reconfiguración parcial:

```
FPGAs PARTIAL RECONFIGURATION
Peripheral partial self-reconfiguration without IPIF
Virtex 4 (xc4vfx12)
        Juan Quero Llor

Select an option [1-4]:
        1. Initialize hwicap (DONE)
        2. New partial bitstream downloaded to system RAM
        3. Perform Partial Reconfiguration
        4. Microblaze peripheral register

Executing option 3.
FPGA Reconfiguration success. Device ID read: 1E58093 status 0

Press any key to continue...
```

Y finalmente de nuevo “4” para observar que la auto-reconfiguración parcial activa del periférico de *MicroBlaze* se ha conseguido, ya que obtiene el valor 63 de su registro:

```
FPGAs PARTIAL RECONFIGURATION
Peripheral partial self-reconfiguration without IPIF
Virtex 4 (xc4vfx12)
        Juan Quero Llor

Select an option [1-4]:
         1. Initialize hwicap (DONE)
         2. New partial bitstream downloaded to system RAM
         3. Perform Partial Reconfiguration
         4. Microblaze peripheral register

Executing option 4.

Result = 63

Press any key to continue...
```

Esto confirma pues la hipótesis del fallo del parche de *Xilinx* cuando la auto-reconfiguración parcial se realiza cuando interactúan un periférico con *IPIF*, el *hwicap* y la memoria *RAM*.

CAPÍTULO 7
Conclusiones

7.1. Conclusiones

Durante el desarrollo del presente proyecto se ha cumplido ampliamente el objetivo principal del mismo, habiéndose experimentado con todos los procedimientos y metodologías de trabajo necesarios para poder reconfigurar y auto-reconfigurar cualquiera de los tres tipos de *FPGAs* de *Xilinx* que admiten Peconfiguración Parcial hasta el momento, las ***Virtex-II***, ***Virtex-II Pro*** y las ***Virtex-4***.

Se ha detectado la conveniencia de desarrollar los diseños para Reconfiguración Parcial en dispositivos de la casa *Xilinx*, ya que son los más avanzados en la materia, y las que mejores características poseen.

Se han estudiado todos los programas y herramientas que posibilitan o facilitan las tareas en la Reconfiguración Parcial, como son *ISE* (*Project Navigator, iMPACT, PACE*), *EDK, PlanAhead, Matlab*, parche *EA_PR*, etc., documentando exhaustivamente su funcionalidad, uso y características a este respecto.

A lo largo del periodo en el que se ha realizado el presente proyecto han ido apareciendo nuevos flujos de trabajo, nuevas actualizaciones, nuevas versiones e incluso nuevos programas que han sido incorporadas al mismo, con lo cual se dispone de información y ejemplos completamente actualizados, con la última tecnología y métodos de trabajo desarrollados en escasas universidades, lo cual sitúa al departamento en la vanguardia de la Reconfiguración Parcial.

En los comienzos del proyecto éramos conscientes de la posibilidad de no conseguir todos los objetivos principales del mismo, aunque finalmente se ha logrado todo lo propuesto: tanto la reconfiguración parcial estática, como la reconfiguración parcial activa y la auto-reconfiguración parcial de elementos externos a *MicroBlaze* y de periféricos, proponiendo diversas soluciones a los problemas que se han ido encontrando, en su mayoría por fallos en el software de *Xilinx*.

Se han desarrollado los flujos de trabajo en diferencias, en módulos y el *Early Access Partial Reconfiguration (EA_PR)*, indicando los pasos a seguir en cada uno de ellos, y las ventajas que poseen, poniendo especial énfasis en este último, ya que se trata del flujo más moderno disponible.

Se ha trabajado con tres tipos de placas: ***Virtex-II, Virtex-II pro*** y ***Virtex-4***, estudiando el funcionamiento de las tres y sus posibilidades de reconfiguración parcial, así como de los distintos elementos de que disponen: memoria *RAM, leds, displays*, puerto *RS-232, jumpers* de programación, pulsadores, *switchs*, etc., solucionando problemas de funcionamiento de las memorias *RAM*, que servirán para futuros diseños con estas placas, reconfigurables o no.

Se han estudiando a fondo los distintos elementos que participan en la reconfiguración parcial en cualquiera de sus flujos: *buses macro, buffers triestado, hwicap, dcm, bufg*…

Se han estudiado los *cores* y *drivers* de distintos procesadores embedidos, especialmente *MicroBlaze*, que ha sido la opción elegida para la auto-reconfiguración parcial, y elementos de éste, como son *hwicap, bus opb, bus dcr*, periféricos genéricos, memoria *RAM, UART, debugger*, direccionamiento, etc. Se han trabajado las técnicas de *floorplanning* en *PACE* y *FPGA Editor* de *ISE* y en *PlanAhead*.

Se han desarrollado programas, ejemplos y manuales para todas las técnicas y todos los flujos de (auto)reconfiguración parcial, permitiendo con ello realizarla de una manera rápida y segura en futuras aplicaciones, en lugar de los meses que se tardaría en obtener cada una de las técnicas sin disponer de esta recopilación.

Se han mantenido contactos con otros investigadores e incluso con los propios diseñadores de la empresa *Xilinx,* lo cual ha favorecido el desarrollo de este proyecto con la última tecnología disponible. Por el mismo motivo, se trata de tecnología y software en desarrollo, lo cual se hace patente en la cantidad de fallos que se provocan, y en la imposibilidad de conseguir ciertos diseños hasta que *Xilinx* depure estos fallos.

Se ha hecho un estudio de posibles aplicaciones futuras, así como de las ventajas que crea la reconfiguración parcial, siendo patente la gran cantidad de salidas que tiene esta tecnología en un futuro cercano.

En definitiva, se ha demostrado la utilidad y el correcto funcionamiento de la (auto)reconfiguración parcial de *FPGAs* en varias de sus facetas y se ha abierto una puerta importante para futuras investigaciones en el departamento.

7.2. Líneas Futuras

Los manuales desarrollados en el presente proyecto servirán para futuras investigaciones en el departamento, sirviéndose de todas las ventajas que proporciona la reconfiguración parcial.

En base a estos manuales se podrán desarrollar diseños de mayor complejidad para la aceleración hardware de algoritmos concretos de tipo adaptativo o evolutivo, el estudio de técnicas de cifrado hardware o comunicaciones reconfigurables, técnicas de actualización hardware a distancia, métodos de auto-reparación de la propia *FPGA*…

Se podrán desarrollar arquitecturas de hardware de cómputo eficientes con diseños ya realizados anteriormente, que permitan un ahorro de tamaño, de velocidad y de consumo, características muy importantes a la hora de crear sistemas independientes de alta tecnología.

La reconfiguración parcial de *FPGAs* es un avance que podrá ser observado con normalidad a medio-largo plazo. Deberán irse actualizando los distintos métodos de trabajo presentados en este proyecto conforme se vaya mejorando esta tecnología.

ANEXO 1
Instalación del Parche EA_PR para Reconfiguración

En primer lugar antes de instalar ningún software para la reconfiguración, es más que conveniente "limpiar" el sistema de cualquier instalación *ISE* o *EDK* anterior o posterior al que se va a instalar.

Las herramientas Early *Access Partial Reconfiguration* (en adelante *EA_PR*) se pueden descargar previo registro desde la web de *Xilinx* en la dirección:

```
http://www.xilinx.com/support/prealounge/protected/index.htm
```

Instalación de ISE y su Service Pack

Cabe recordar que las herramientas *EA_PR* están en desarrollo, y por lo tanto pueden provocar daños en la *FPGA* en caso de mal uso de la Reconfiguración Parcial. También es importante saber que no debe utilizarse ninguna instalación de *ISE* con *EA_PR* para realizar proyectos no reconfigurables.

En la citada web se puede descargar la versión actual del parche para un determinado *ISE,* con un determinado *Service Pack (SP).* Es importante por tanto instalar el *ISE* y el *Service Pack* adecuado para el parche.

Una vez instalado el *ISE*, se puede obtener el *Service Pack* adecuado de la siguiente página web:

```
http://www.xilinx.com/support/answers/10959.htm
```

Instalación de EDK y su Service Pack

Cuando se ha instalado el *ISE* y su *SP*, es necesario instalar la versión adecuada de *EDK*. Habitualmente se instalará el mismo número de versión (*EDK 8.2* para *ISE 8.2*), pero es conveniente revisar qué *SP* de *EDK* es compatible con el *SP* del *ISE* instalado. Esto se puede comprobar en la siguiente web:

```
http://www.xilinx.com/ise/embedded/edk_download.htm
```

Se pueden obtener los *Service Packs de EDK* adecuados desde la siguiente página web:

```
http://www.xilinx.com/support/answers/25399.htm
```

Instalación del Parche EA_PR

Una vez descargado desde la primera web, el parche *EA_PR* adecuado (llamado *PartialFlow_vX.zip*) se descomprime en una carpeta y se accede a ella a través de la una ventana de comandos.

Se teclea *xilperl –help* para comprobar que el intérprete de *Perl* está disponible.

Se teclea *set*, para comprobar que las variables de sistema de *Xilinx* apuntan a la instalación correcta de éste.

Y finalmente, para realizar la instalación, se teclea:

```
xilperl PRinstall.pl PRfiles.txt
```

Si no se tienen instalados en *ISE* todos los tipos de *FPGAs*, puede ser que aparezcan errores del tipo "No se ha encontrado…". Aún así la instalación de EA_PR se habrá realizado correctamente para las *FPGAs* instaladas.

Instalación de PlanAhead

Para concluir con la instalación de herramientas, se procede a instalar el entorno gráfico de *floorplanning PlanAhead*. Sólo soportan reconfiguración parcial las versiones posteriores a la 8.1.

Para versiones inferiores a la 9.2, una vez que se ha creado el proyecto *PlanAhead*, es necesario teclear lo siguiente para activar las opciones de reconfiguración parcial:

```
hdi::param set -name project.enablePR -bvalue yes
```

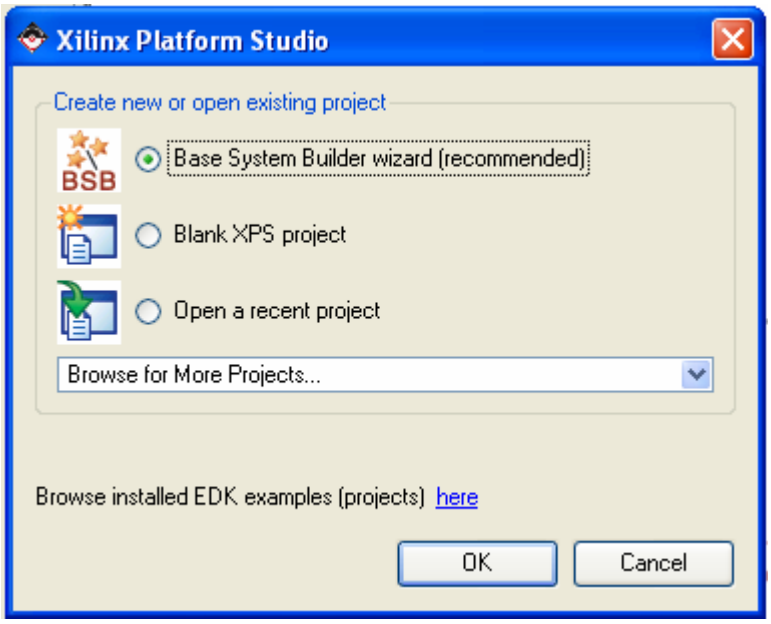
Si no se dispone de *PlanAhead* se puede obtener una versión de evaluación con licencia de un mes, en la web:

```
http://www.xilinx.com/ise/optional_prod/planahead.htm
```

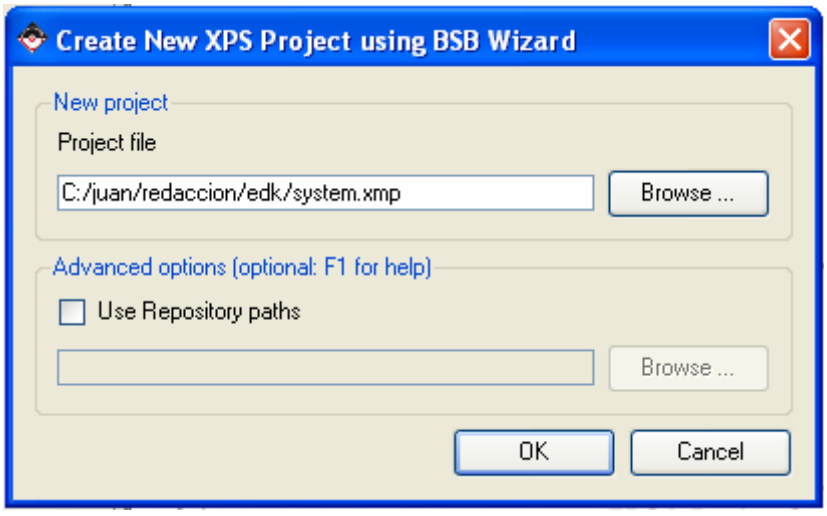
ANEXO 2
Creación de un Proyecto en EDK

Antes de abrir el *EDK* es conveniente guardar en el directorio de instalación de éste los datos de configuración de la placa que vienen en un archivo ***.xbd***, dentro de la carpeta *fabricante/boards* en el cd de soporte de la placa. Esta carpeta debe copiarse en el directorio *c:\edk\board,* de tal forma que el *wizard* de *EDK* leerá estos datos y podrá automáticamente conocer de qué elementos electrónicos se dispone en la placa en cuestión.

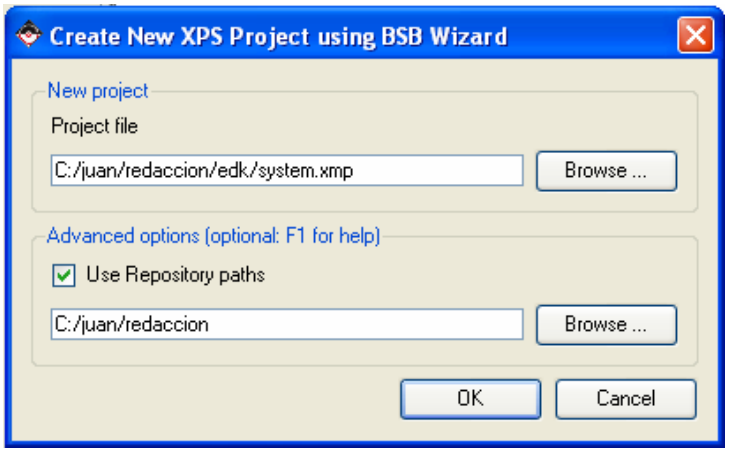
Al abrir el Xilinx *Platform Studio* aparece una ventana en la que indica si se quiere entrar en un *Wizard*, abrir un proyecto en blanco o abrir un proyecto existente. Se selecciona la opción *Base System Builder Wizard* para la creación de un nuevo proyecto en base al *wizard* y se hace clic en *OK*.



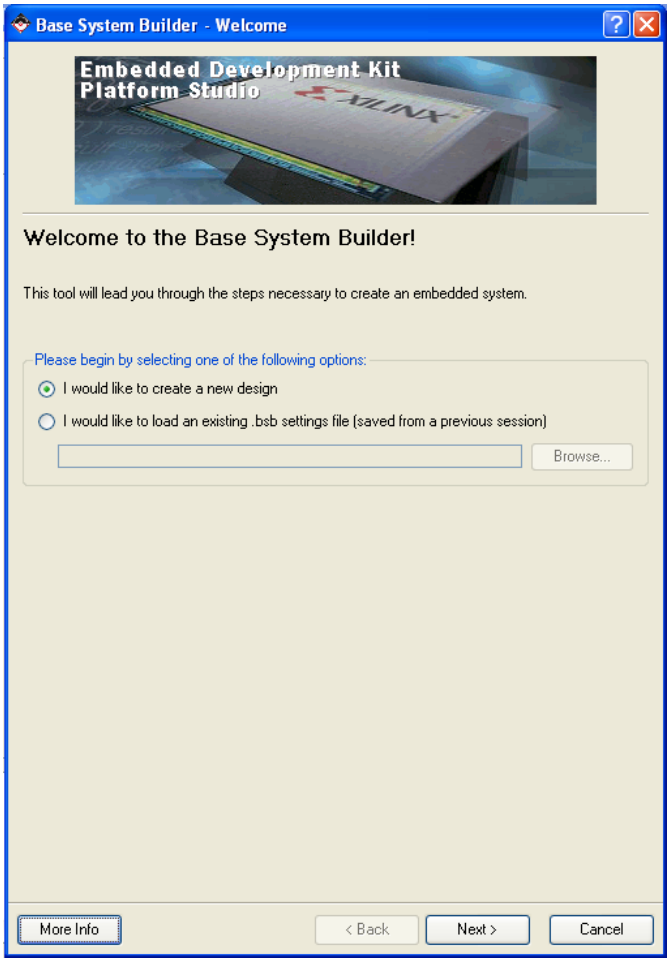
A continuación se escribe el nombre del proyecto (la ruta no puede contener espacios).



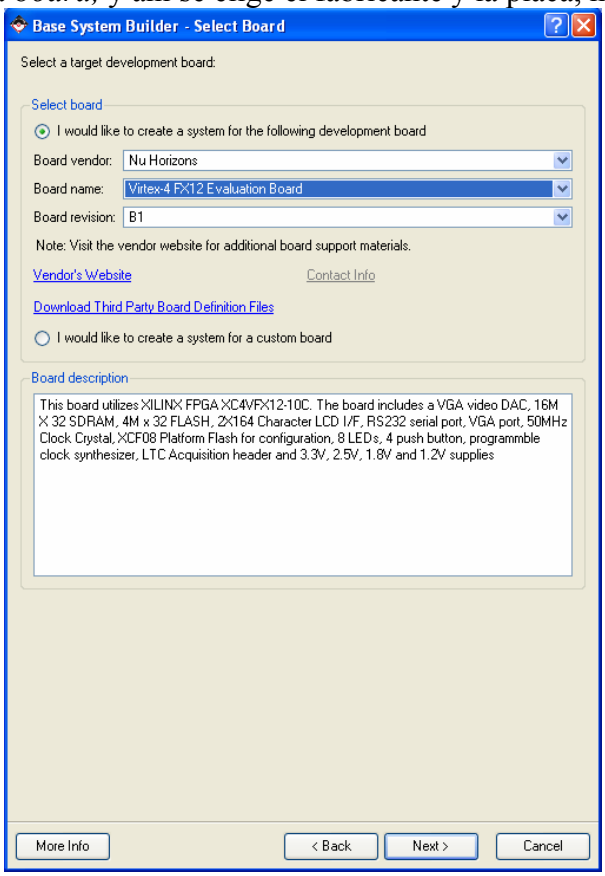
Existe la opción de utilizar un directorio como repositorio de periféricos para el procesador, si se indica la opción *Use Repository paths*. Este repositorio permitirá almacenar los distintos periféricos de manera independiente al proyecto, de tal forma que puedan ser utilizados por diferentes proyectos. Se indica el directorio en el que se desea guardar el repositorio (dada su función, es conveniente que se encuentre fuera del directorio del proyecto *EDK*) y se hace clic en *OK*.



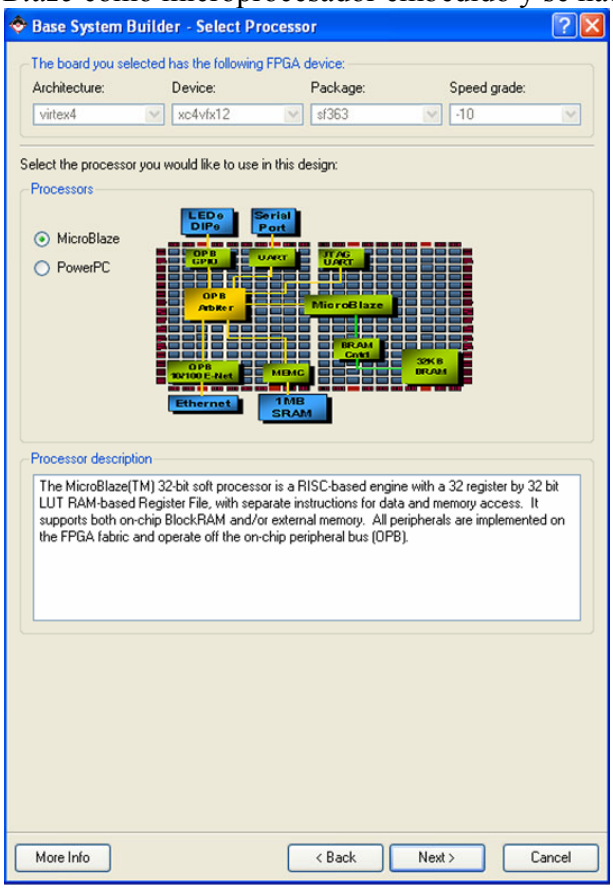
En la primera ventana que aparece del *wizard* se selecciona *I would like to create a new design* y se hace clic en *Next*.

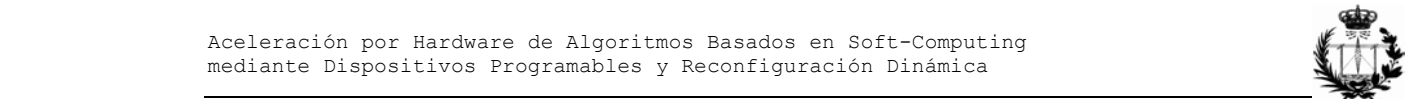


A continuación se selecciona la casilla *I would like to create a system for the following development board,* y ahí se elige el fabricante y la placa, haciendo clic en *Next.*

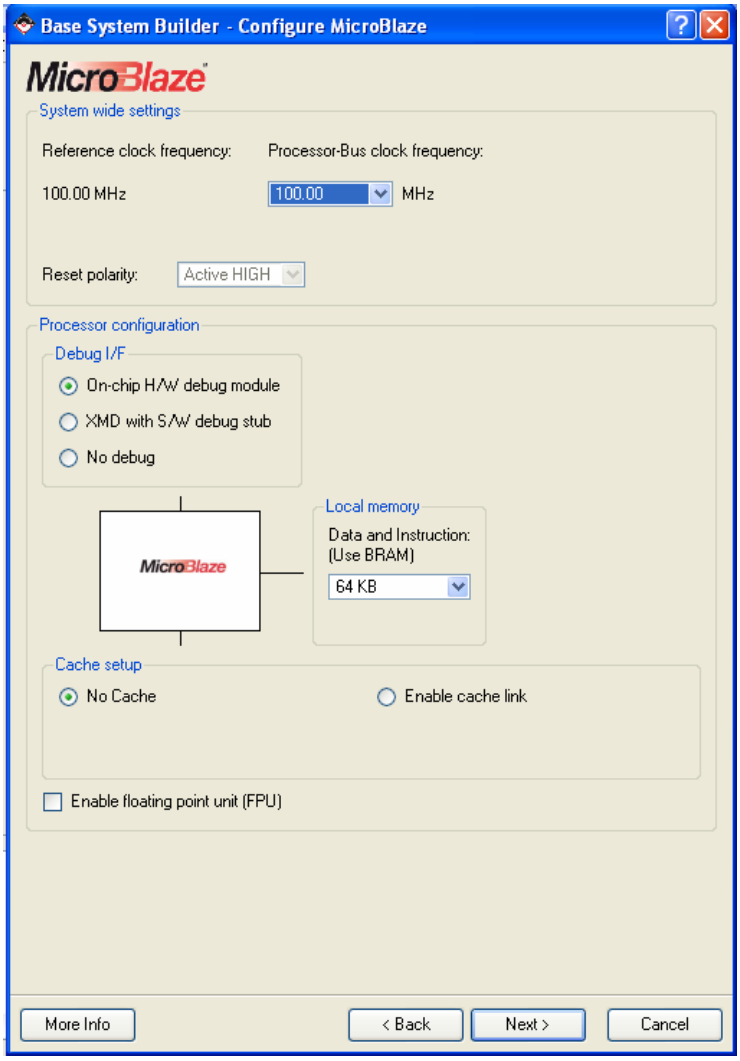


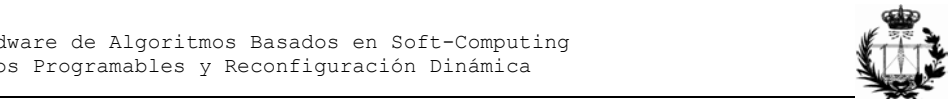
Se elige *MicroBlaze* como microprocesador embedido y se hace clic en *Next.*



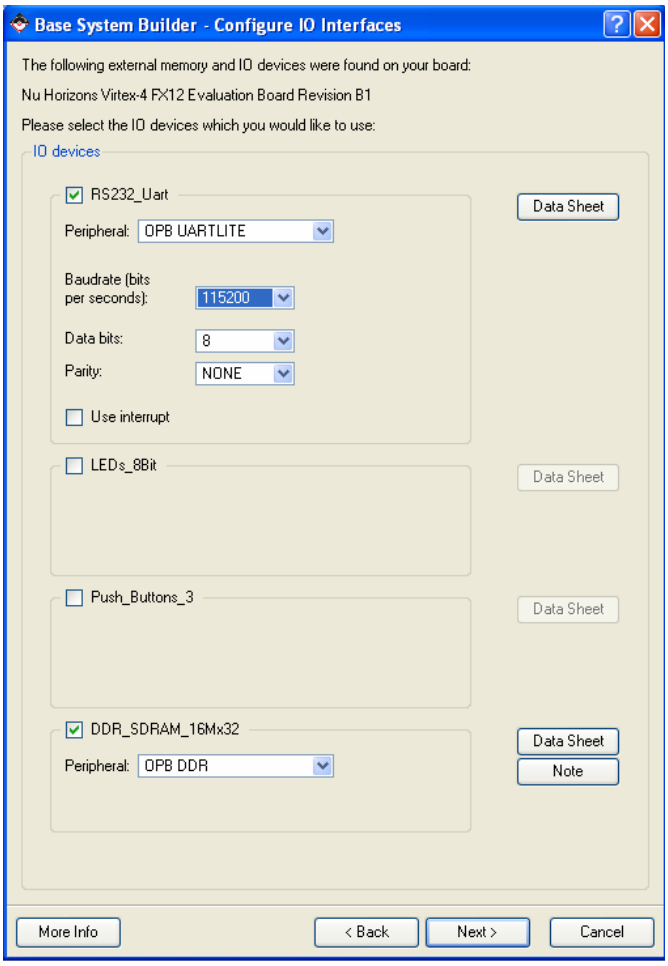


Se selecciona el valor de la velocidad de reloj, en este caso 100 MHz. El tipo de depurado *On-chip H/W debug module* (que instalará un periférico depurador que permitirá descargar en la memoria *RAM* el *bitstream* parcial). En *Data and Instruction (use BRAM)* se selecciona 64KB y en *Cache setup* se selecciona *No Cache*, haciendo finalmente clic en *Next.*

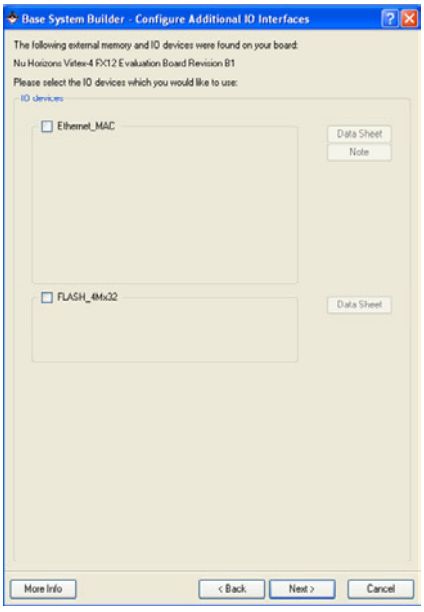




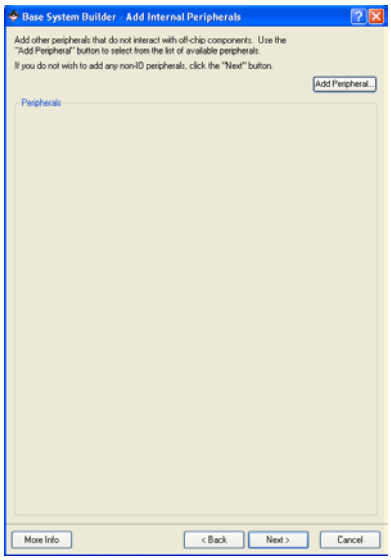
A continuación se eligen los elementos de memoria y de entrada/salida que se desean, en este caso el *RS232_Uart* (configurada con *Baudrate 115200*, *Data bits 8* y *Parity NONE*) para comunicación entre el ordenador y el *MicroBlaze*, así como la memoria *DDR_SDRAM_16Mx32*. Es importante leer los *Data Sheet* y los *Note* de las memorias *RAM* para que éstas funcionen correctamente.



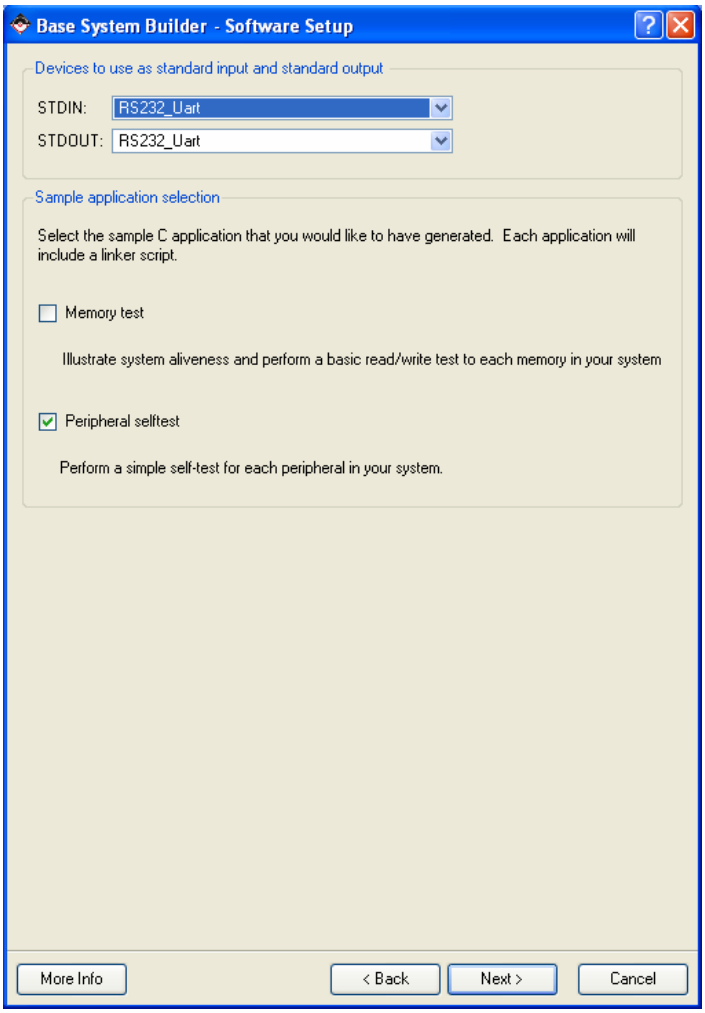
Se hace clic en *Next:*

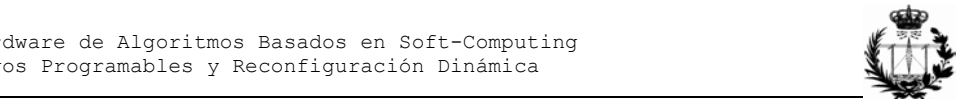


Se hace clic de nuevo en *Next:*

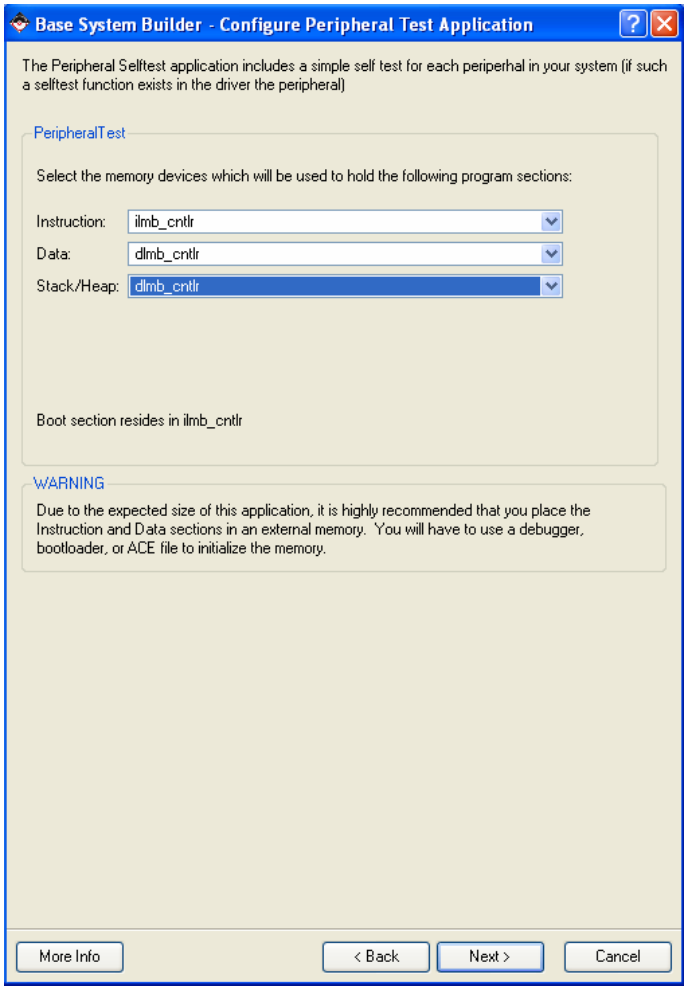


A continuación se selecciona como *STDIN* y como *STDOUT* el *RS232_Uart*, es decir que tanto la entrada como la salida estándar del *MicroBlaze* estarán conectadas a la *UART* y a través de ésta a *HyperTerminal* (Vease Anexo 3). Además se crea un *Peripheral selftest* o un *Memory test* para comprobar el funcionamiento de periféricos o memoria, o para añadir ahí el código fuente de la aplicación.

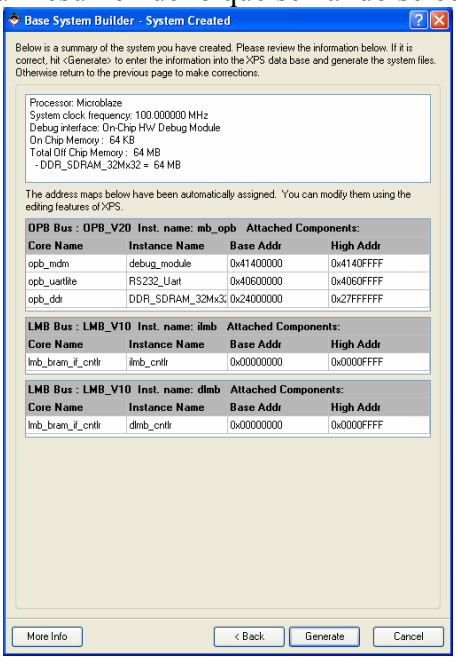




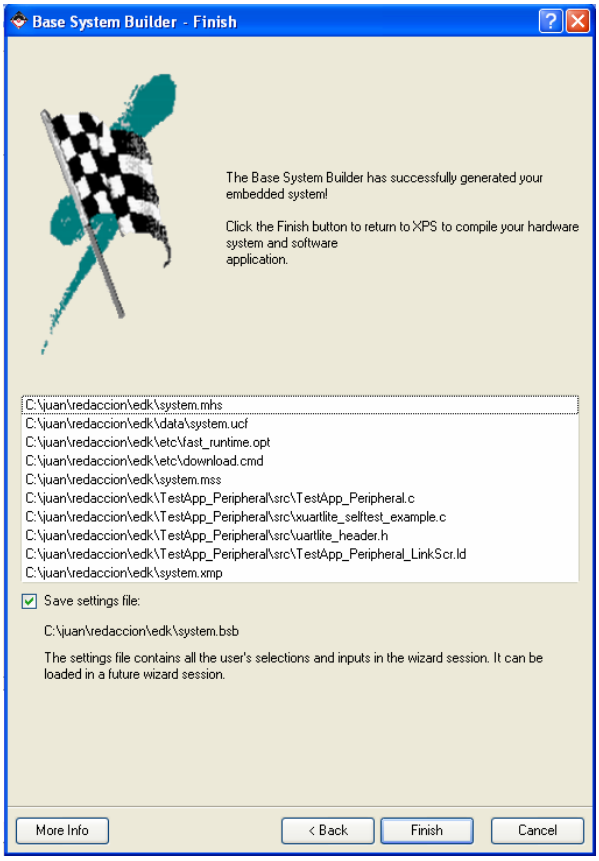
A continuación se elige como memoria de instrucciones *ilmb_cntlr*, como memoria de datos *dlmb_cntlr* y como pila *dlmb_cntlr*, con lo que se estará utilizando la memoria *bram* para cargar y ejecutar el programa.



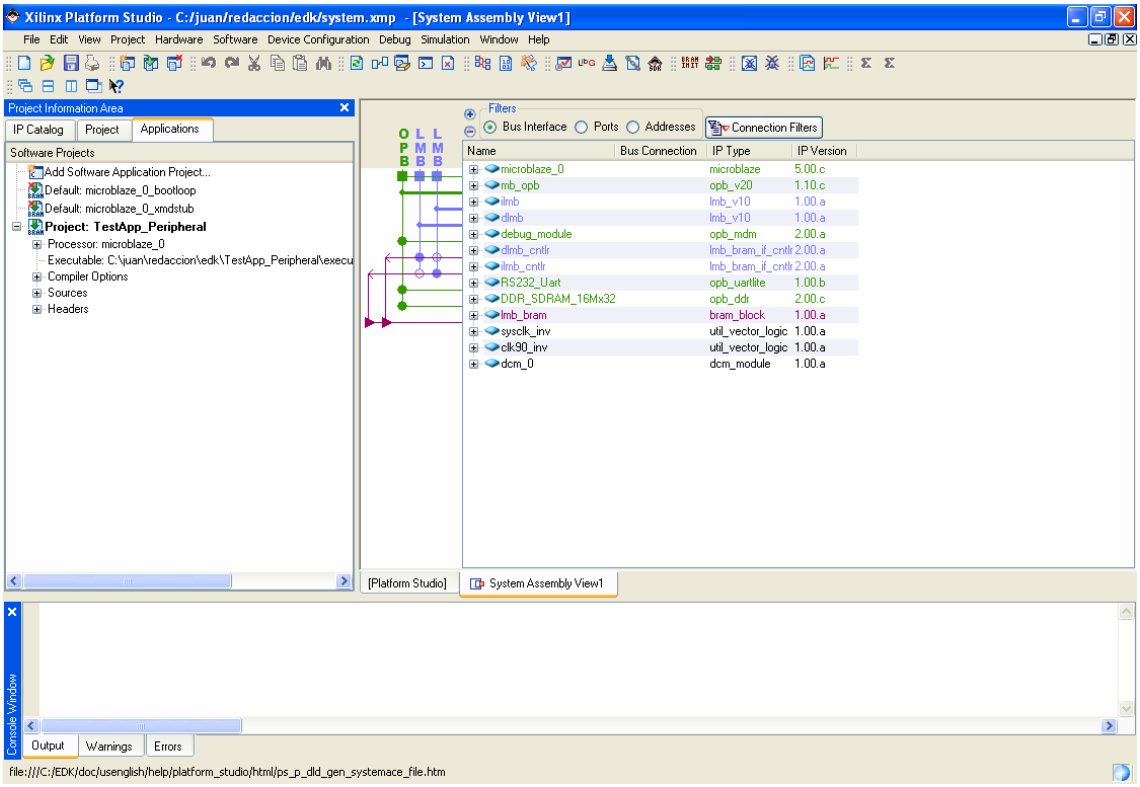
Finalmente aparece un resumen de lo que se ha ido seleccionando en el *wizard*.



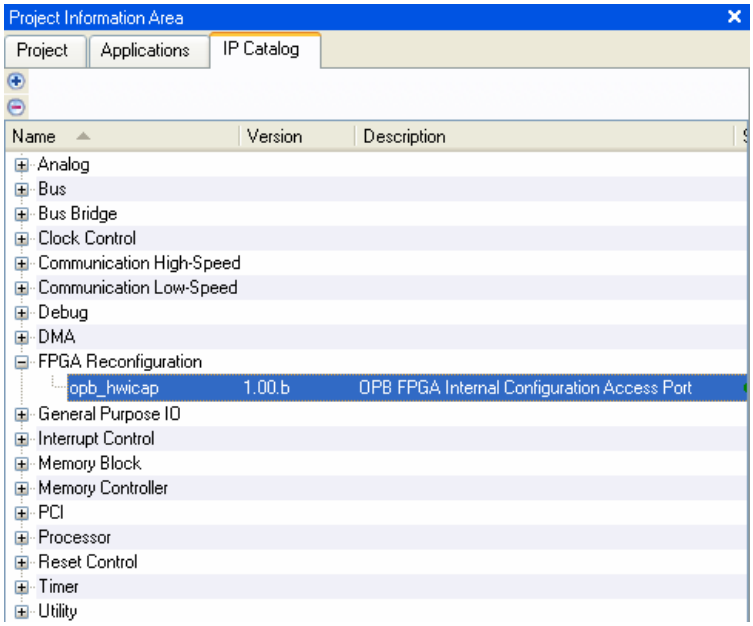
Para concluir el *wizard* se pulsa *Finish.*



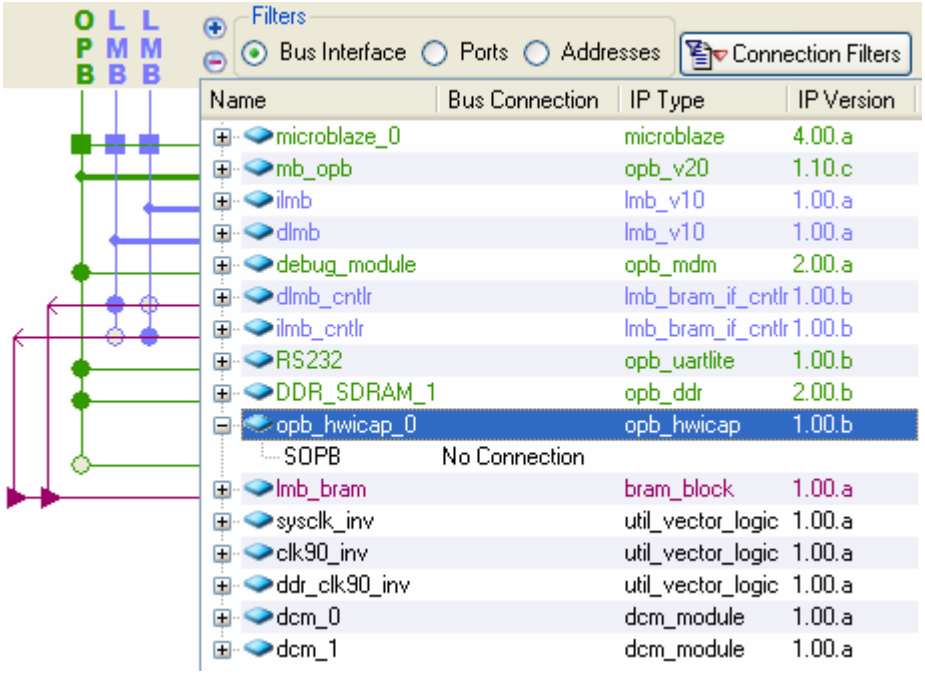
De esta manera se abre la ventana de *EDK* con los elementos requeridos ya instalados, a falta del *hwicap*:



Para añadir el *hwicap,* se hace clic en la pestaña *IP Catalog* de la ventana *Project Information Area,* seleccionando el apartado *FPGA Reconfiguration*, haciendo doble clic en *opb_hwicap,* y seleccionando *Yes* en el mensaje que aparece:



Finalmente, se debe conectar el *hwicap* al bus *opb*, para lo que se hace clic en el único círculo blanco que hay bajo el *OPB* en la ventana *System Assembly View*:



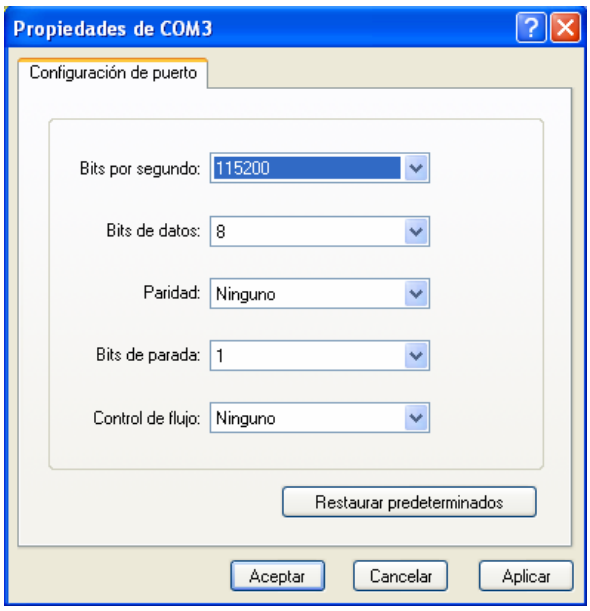
ANEXO 3
Configuración de HyperTerminal

Cuando se vaya a ejecutar un programa en *MicroBlaze*, es necesario tener abierta una ventana de *HyperTerminal* que permita la comunicación con él, a través del puerto *RS-232*.

Este programa se abre en el menú Inicio de *Windows,* dentro de *Accesorios/ Comunicaciones/ HyperTerminal.*

Antes de descargar el *bitstream* en la placa, se abre una ventana de *HyperTerminal* y se guarda la conexión para futuras ocasiones.

Esta conexión se configura siguiendo los parámetros indicados a la *UART* en el *wizard* del *EDK* (Ver Anexo 2), en este caso 115000 bps, 8 bits de datos sin paridad, 1 bit de parada y sin control de flujo. Finalmente se hace clic en aceptar, de tal manera que comienza la conexión.

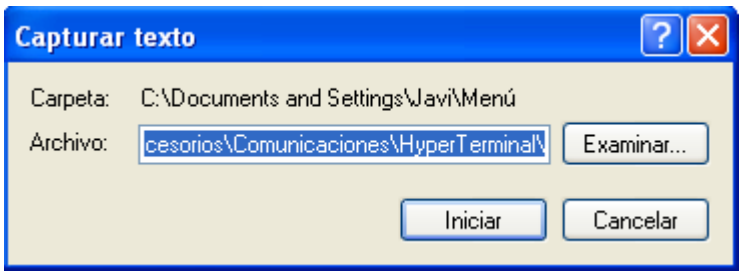


Una función muy útil que permite *HyperTerminal* es la captura del texto recibido en un archivo de texto. Esto es necesario cuando se recibe una cantidad ingente de datos, por ejemplo cuando se lee un *frame* del *ICAP*, o cuando se leen los datos del *bitstream* parcial que se ha cargado en la memoria *RAM*.

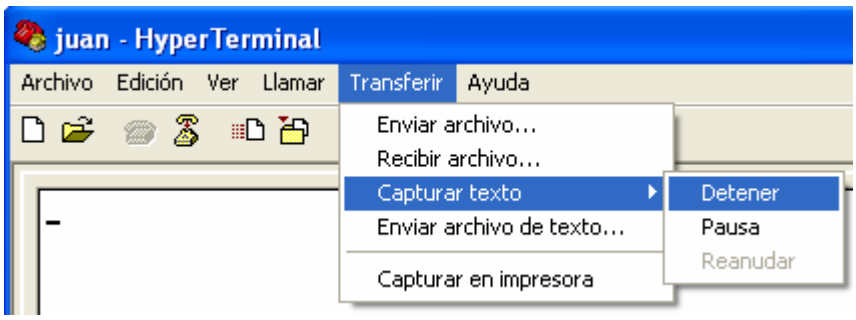
Para ello se hace clic en el menú *Transferir/Capturar Texto*:

En la ventana que aparece se selecciona el nombre del fichero y el lugar en el que se desea guardar y se hace clic en *Iniciar*

.



Una vez que se ha recibido todo el texto deseado, se hace clic en el menú *Transferir/Capturar Texto/Detener*, y el fichero quedará guardado:



ANEXO 4
Modificación de Buses Macro

En ocasiones los *buses macro* de que disponemos, no tienen el nombre de pines de entrada y de salida que se desea, o no tienen los delimitadores de bus que se utilizan normalmente.

Para poder editar la definición de un *bus macro* (el archivo ***.nmc***) es necesario convertirlo primero a fichero *ASCII*, puesto que aquél no lo es. Esto se realiza abriendo una ventana de comandos, accediendo a la carpeta en la que se encuentra el *bus macro* y tecleando:

```
xdl -ncd2xdl busmacro_xc2vp_l2r_async_narrow.nmc
```

De esta forma se obtiene un archivo con el mismo nombre pero de extensión ***.xdl***, que se puede editar como texto con *WordPad*, y cambiar los delimitadores "( )" por "< >" o viceversa.

```
# ============================================================
# XDL NCD CONVERSION MODE $Revision: 1.0$
# time: Tue Nov 28 13:46:37 2006

# ============================================================

# ============================================================
# The syntax for the design statement is:
# design <design_name> <part> <ncd version>;
# or
# design <design_name> <device> <package> <speed> <ncd_version>
# ============================================================
design "__XILINX_NMC_MACRO" xc2vp2ff672-6;

# ============================================================
# The syntax for modules is:
#     module <name> <inst_name> ;
#     port <name> <inst_name> <inst_pin> ;
#     .
#     .
#     instance ... ;
#     .
#     .
#     net ... ;
#     .
#     .
#     endmodule <name> ;
# ============================================================

# ============================================================
# MODULE of "busmacro_xc2vp_l2r_async_narrow"
# ============================================================
module "busmacro_vector8_xc2vp_l2r_async_narrow" "slice3" , cfg
"_SYSTEM_MACRO::FALSE" ;
  port "macro_in(0)" "slice0" "G1";
  port "macro_in(1)" "slice0" "F1";
  port "macro_in(2)" "slice1" "G1";
  port "macro_in(3)" "slice1" "F1";
  port "macro_in(4)" "slice2" "G1";
  port "macro_in(5)" "slice2" "F1";
```

```
  port "macro_in(6)" "slice3" "G1";
  port "macro_in(7)" "slice3" "F1";
  port "macro_out(0)" "slice4" "Y";
  port "macro_out(1)" "slice4" "X";
  port "macro_out(2)" "slice5" "Y";
  port "macro_out(3)" "slice5" "X";
  port "macro_out(4)" "slice6" "Y";
  port "macro_out(5)" "slice6" "X";
  port "macro_out(6)" "slice7" "Y";
  port "macro_out(7)" "slice7" "X";

[……]

endmodule "busmacro_xc2vp_l2r_async_narrow" ;
```

Finalmente se devuelve el fichero a su estado natural tecleando:

```
xdl -xdl2ncd busmacro_xc2vp_l2r_async_narrow.xdl
```

Y renombrando el archivo resultante, de extensión ***.ncd,*** a la extensión ***.nmc***.

ANEXO 5
Proyecto EDK Ejecutándose en la Memoria RAM

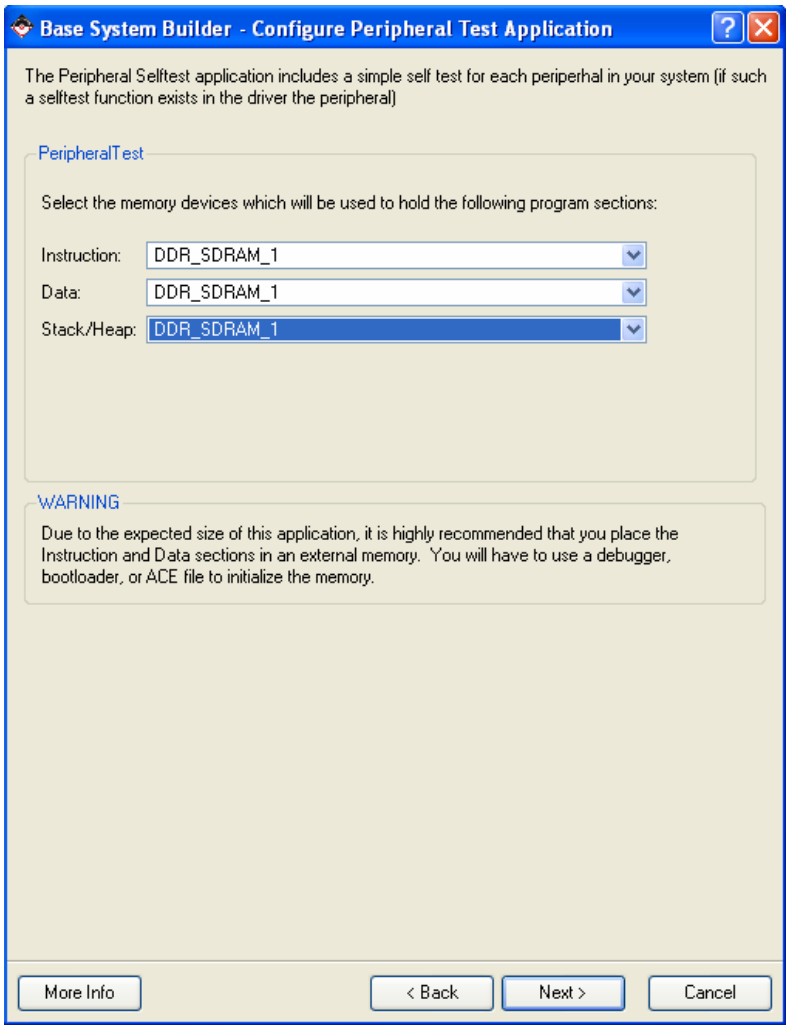
Habitualmente cuando se ejecuta un programa en un microprocesador tipo *MicroBlaze* se hace desde la memoria *bram* interna de la propia *FPGA*.

En ocasiones, cuando el programa tiene un tamaño demasiado grande como para caber en la *bram*, es necesario instalarlo en la memoria *RAM* de la placa, y que se ejecute desde allí.

Para que esto se pueda realizar es necesario hacer una serie de cambios en la configuración habitual de los proyectos de *EDK*.

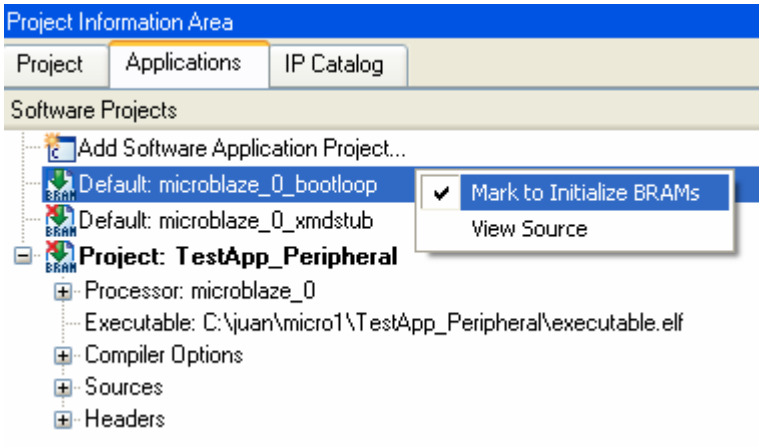
El ejemplo se puede encontrar en el cd adjunto, en la carpeta *anexo/memoriaram*. Está realizado con *EDK 8.1*.

En primer lugar, cuando se ejecuta el *wizard*, se debe de seleccionar la memoria *RAM* como memoria de instrucciones, de datos y de pila:



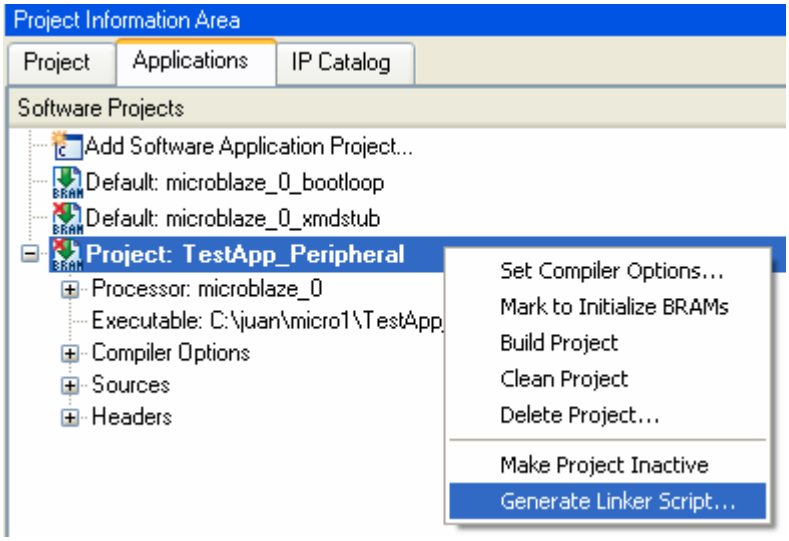
No hay que olvidar realizar los cambios pertinentes en el ***.mhs*** y el ***.ucf*** para que funcionen las memorias *RAM* en ciertas placas.

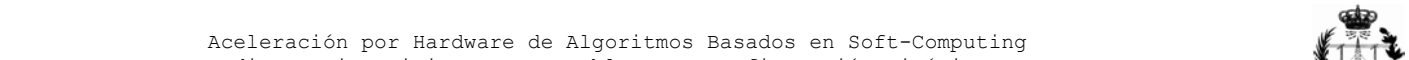
Se accede a la pestaña *Applications* de la ventana *Project Information Area*, y se hace clic con el botón derecho encima de *Default:microblaze_0_bootloop*, seleccionando *Mark to initialize BRAM*:



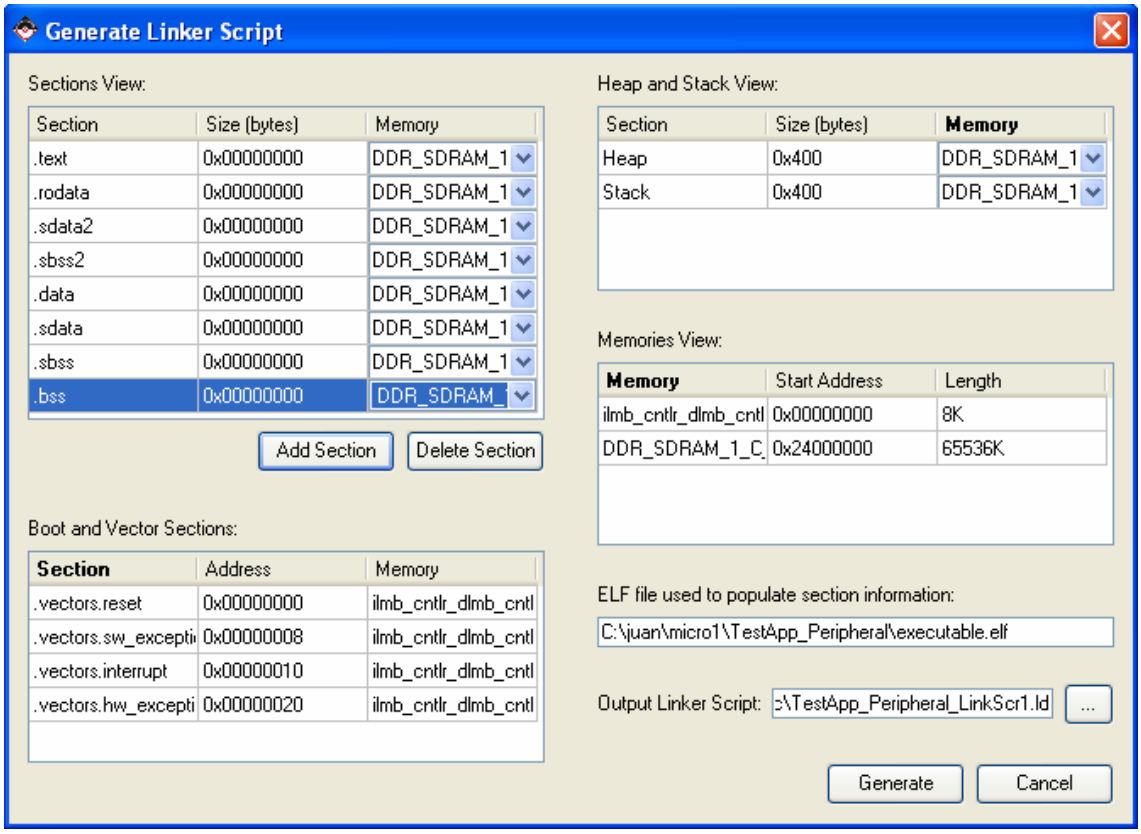
De esta manera se iniciará un *bootloop* en la *bram* de la *FPGA*, que permitirá que se ejecute el programa en la memoria *RAM*.

A continuación hay que indicar donde se desea que se ejecute el programa, para lo que hay que rellenar el *Linker Script*. Se hace clic con el botón derecho encima de *Project: TestApp_Peripheral* y se selecciona *Generate Linker Script*.





En la ventana que aparece se selecciona *DDR_SDRAM_1_C_MEMO_BASEADDR* para los apartados *.text, .rodata, .sdata2, .data, .sdata, .sbss* y *.bss,* así como para *Heap* y *Stack*. Finalmente se hace clic en *Generate.*



A continuación ya se puede escribir el código fuente de *TestApp_Peripheral*, compilarlo y descargarlo en la placa, observando que se ejecuta el código en la memoria *RAM.*

El código utilizado para comprobar la memoria *RAM* es el siguiente:

```
#include <xuartlite_l.h>
#include <xparameters.h>

#define STEP_RAND(x) ((x) = 0x0019660d * (x) + 0x3c6ef35f)
#define MEM_REG (*(unsigned*)MEM_ADDR)
unsigned MEM_ADDR;

int main (void) {
// ----------------------------------------------------------------
// --  Sequential address, pseudo-random data.
// ----------------------------------------------------------------
unsigned start_addr=0x25000000;
unsigned end_addr=0x26000000;

       unsigned x = 0;
       int error = 0;
       unsigned seed = 0xdeadbeef;
       unsigned rand;

       print("Testing : 0x");
       putnum(start_addr);
       print(" - 0x");
```

```
        putnum(end_addr);
        print("\r\n");
        // -- Fill mem with values of "address"
        print("Writing pseudo-random data...0x");
        putnum(start_addr);
        rand = seed;
        STEP_RAND(rand);
        for (MEM_ADDR = start_addr; MEM_ADDR <= end_addr; MEM_ADDR += 4) {
              // write the value of "address" to each address
              MEM_REG = rand;
              STEP_RAND(rand);

              if ((MEM_ADDR & 0x0000ffff) == 0x00008000) {
                    xil_printf("\b\b\b\b\b\b\b\b"); // Write backspace
                    putnum(MEM_ADDR);
              }
        }
        print("\r\n");

        // -- Check memory to see if content matches "address"
        print("Reading : 0x");
        putnum(start_addr);
        rand = seed;
        STEP_RAND(rand);
        for (MEM_ADDR = start_addr; MEM_ADDR <= end_addr; MEM_ADDR += 4) {
              x = MEM_REG;
              if (x != rand) {
                    print("\r\n\033[5mMem error\033[0m\r\n");
                    print("At: 0x");
                    putnum(MEM_ADDR);
                    print("   Expected: 0x");
                    putnum(rand);
                    print("   Got: 0x");
                    putnum(x);
                    print("\r\n");
                    error ++;
                    //return;
              }
              STEP_RAND(rand);

              if ((MEM_ADDR & 0x0000ffff) == 0x00008000) {
                    xil_printf("\b\b\b\b\b\b\b\b"); // Write backspace
                    putnum(MEM_ADDR);
              }
        }
        xil_printf("\r\nTotal Errors = %d\r\n", error);
        if (error > 0)
              print("\r\nMem \033[5mpassed\033[0m\r\n");

        xil_printf("-- exiting main() --\r\n");
    return 0;
}
```

ANEXO 6
Proyecto con Reconfiguración de LUTs

Se puede aprovechar las ventajas que proporcionan las funciones de los *drivers* de *hwicap*, para realizar reconfiguración de ciertos elementos de la *FPGA*, como pueden ser *LUTs* o *frames*, sin la necesidad de seguir el flujo de reconfiguración ni obtener *bitstreams* parciales.

El ejemplo propuesto ha sido realizado con la placa **Virtex-II** *xc2v1000* y el sofware *EDK 8.1* e *ISE 8.1 Service Pack 1*, y se encuentra en la carpeta *Anexos/reconfigLUTvirtex2.*

En el ejemplo se crea un proyecto *EDK* con *hwicap*, que se exporta a *ISE* y en cuyo *top* se implementa una función sencilla de dos puertas *AND* de 4 entradas cada una (2 *LUTs*), cuya entrada es controlada por *switchs* y la salida va conectada a los *LEDs* de la placa. Este proceso se realiza de forma similar al Apartado 3.4.1.

Estas dos *LUTs* son localizadas mediante el *FPGA Editor* en la posición *Slice_X62Y40*. Una vez finalizado el proyecto *ISE* se implementa y se obtiene el *bitstream.*

En *EDK* se compila el siguiente programa, que se carga en el *bitstream* obtenido con *ISE,* siguiendo los pasos de los Apartados 4.7 o 6.5:

```
// Located in: microblaze_0/include/xparameters.h
#include "xparameters.h"

#include "stdio.h"

#include "uartlite_header.h"
#include "xtmrctr.h"
#include "tmrctr_header.h"

#include "xhwicap.h" //necessary tu include, dont know why
#include "xhwicap_clb_lut.h"
#include "xstatus.h" // not necessary to include, declared in xhwicap.h
#include "xhwicap_i.h" //low level defines, ej block types, etc.
//#include "reconf_data.c" //reconf data to load device with.

#define LUT_SIZE        16
#define TEST_COL        62
#define TEST_ROW        40

XHwIcap InstIcap;
XTmrCtr TmrCtrInstancePtr;
//====================================================

int main (void) {


   print("-- Entering main() --\r\n");

   char opcion;

       XStatus status;
       Xint32 Block, MajorFrame, MinorFrame, i;
       Xint32 config_data_read[424]; //xc2v1000 has 106 32bit-words per frame = 424bytes
```

```
      Xuint32 XRow, XCol, Row, Col, slice;
      Xuint8 LUT_Value[16];

      Xuint32 TmrCtrValue1, TmrCtrValue2;

      while(1) {

            print("\f Select an option [1-5]:");
            print("\r\n\t 1. Initialize hwicap.");
            print("\r\n\t 2. Desynchronize hwicap.");
            print("\r\n\t 3. Read frame from device config.");
            print("\r\n\t 4. Read frame from storage buffer.");
            print("\r\n\t 5. Write frame to storage buffer.");
            print("\r\n\t 6. Write frame to device config.");
            print("\r\n\t 7. Write partial bitstream.");
            print("\r\n\t 8. Read CBL/LUT.");
            print("\r\n\t 9. Write CLB/LUT");
            print("\r\n\t a. Initialize Timer");
            print("\r\n\t s. Time to read 100 LUTs");
            print("\r\n\t d. Time to write 100 LUTs");
            print("\r\n\t ?. Any other command?.");

            opcion = (char)inbyte();

            switch(opcion){
            case '1': //Initialize hwicap
                  xil_printf("\r\n\r\n Executing option %c.\r\n",opcion);

                  status = XHwIcap_Initialize(&InstIcap,
                        XPAR_OPB_HWICAP_0_DEVICE_ID,
                        XHI_READ_DEVICEID_FROM_ICAP);

            if (status != XST_SUCCESS) {
                        xil_printf("\r\nXHwICAP Initialization failed. Device ID read: %x
status %x",
                                                InstIcap.DeviceIdCode, status);
                        //exit(1);
            } else {
            xil_printf("\r\nXHwICAP Initialization success. Device ID read: %x",
                                    InstIcap.DeviceIdCode);
                  }

                  xil_printf("\r\n Full info received from ICAP: \
                                    \r\n\t IsReady: %x \
                                    \r\n\t DeviceIdCode: %x \
                                    \r\n\t DeviceId: %x \
                                    \r\n\t Rows: %x \
                                    \r\n\t Cols: %x \
                                    \r\n\t BramCols: %x \
                                    \r\n\t BytesPerFrame: %x \
                                    \r\n\t WordsPerFrame: %x \
                                    \r\n\t ClbBlockFrames: %x \
                                    \r\n\t BramBlockFrames: %x \
                                    \r\n\t BramIntBlockFrames: %x", \
                                    InstIcap.IsReady, InstIcap.DeviceIdCode,
InstIcap.DeviceId, \
                                    InstIcap.Rows, InstIcap.Cols, InstIcap.BramCols,
InstIcap.BytesPerFrame, \
                                    InstIcap.WordsPerFrame, InstIcap.ClbBlockFrames, \
                                    InstIcap.BramBlockFrames,
InstIcap.BramIntBlockFrames);

                  print("\r\nPress any key to continue...");    opcion = (char)inbyte();
                  break;

            case '2'://Desynchronize hwicap
                  xil_printf("\r\n Executing option %c.",opcion);
```

```
                  // Should the ICAP be desynchronized
                  status = XHwIcap_CommandDesync(&InstIcap);

                  if (status != XST_SUCCESS) {
                        xil_printf("\r\nXHwICAP Desynchronization failed. Device ID read:
%x status %x",
                                          InstIcap.DeviceIdCode, status);
            } else {
            xil_printf("\r\nXHwICAP Desynchronization success. Device ID read: %x",
                                    InstIcap.DeviceIdCode);
                  }

                  print("\r\nPress any key to continue...");    opcion = (char)inbyte();
                  break;


            case '3'://Read frame from device config
                  xil_printf("\r\n Executing option %c.",opcion);

                  /* Reads one frame from the device and puts it in the storage buffer.
*/
                  /* XStatus XHwIcap_DeviceReadFrame(XHwIcap *InstancePtr, Xint32 Block,
                              Xint32 MajorFrame, Xint32 MinorFrame); */

                  //Block values: XHI_FAR_CLB_BLOCK, XHI_FAR_BRAM_BLOCK,
XHI_FAR_BRAM_INT_BLOCK
                  Block = XHI_FAR_CLB_BLOCK;
                  MajorFrame = 0;
                  MinorFrame = 1;
                  status = XHwIcap_DeviceReadFrame(&InstIcap, Block, MajorFrame,
MinorFrame);
                  if (status != XST_SUCCESS) {
                        xil_printf("\r\nXHwICAP Frame read failed. Device ID read: %x
status %x",
                                          InstIcap.DeviceIdCode, status);
            } else {
            xil_printf("\r\nXHwICAP Frame read success. Device ID read: %x",
                                    InstIcap.DeviceIdCode);
                  }
                  print("\r\nPress any key to continue...");    opcion = (char)inbyte();
                  break;

            case '4'://Read frame from storage buffer
                  xil_printf("\r\n Executing option %c.",opcion);

                  for (i=0; i<511 ;i++){
                        /* Reads word from the storage buffer. */
                        /*Xuint32 XHwIcap_StorageBufferRead(XHwIcap *InstancePtr, Xuint32
Address);
                        WARNING: Address spans [0-511], as data words are 4-bytes. To
read just 1 frame
                        addres should be [0-106] as xc2v1000 has 106 32bit words*/
                        config_data_read[i] = XHwIcap_StorageBufferRead(&InstIcap, i);
                        xil_printf("\r\n%d: %x", i, config_data_read[i]);
                  }

                  print("\r\nPress any key to continue...");    opcion = (char)inbyte();
                  break;

            case '5'://Write frame to storage buffer
                  xil_printf("\r\n Executing option %c.",opcion);

                  /* Writes word to the storage buffer. */
                  /*void XHwIcap_StorageBufferWrite(XHwIcap *InstancePtr, Xuint32
Address,
                              Xuint32 Data);*/
                  for(i=0;i<511;i++){
```

```
                    XHwIcap_StorageBufferWrite(&InstIcap, i, i);
                    }
                    print("\r\nPress any key to continue...");    opcion = (char)inbyte();
                    break;

             case '6'://Write frame to device config
                    xil_printf("\r\n Executing option %c.",opcion);
                    print("\r\nFunction not yet implemented...");
                    /* Writes one frame from the storage buffer and puts it in the device.
*/
                    /* XStatus XHwIcap_DeviceWriteFrame(XHwIcap *InstancePtr, Xint32 Block,
                                Xint32 MajorFrame, Xint32 MinorFrame); */

                    print("\r\nPress any key to continue...");    opcion = (char)inbyte();
                    break;

             case '7'://Write partial bitstream
                    xil_printf("\r\n Executing option %c.",opcion);
                    print("\r\nNot doing it really (in this program rev)");

                    print("\r\nPress any key to continue...");    opcion = (char)inbyte();
                    break;

             case '8'://Read CLB/LUT
                    xil_printf("\r\n Executing option %c.",opcion);
                    print("\r\nLed(0) LUT_G on Slice_X62Y40, Led(1) on Slice_X62Y6.");
                    print("\r\nXCol: ");      XCol = (Xuint32)inbyte()-48;
                    XCol = XCol*10;           XCol = XCol + (Xuint32)inbyte()-48;
                    xil_printf("%d.",XCol);
                    print("\r\nXRow: ");      XRow = (Xuint32)inbyte()-48;
                    XRow = XRow*10;                 XRow = XRow + (Xuint32)inbyte()-48;
                    xil_printf("%d.",XRow);

                    /* Identify the LUT to change: LUT in SLICE_X[Col]Y[Row]. */
                    Col = XHwIcap_mSliceX2Col(XCol);
                    Row = XHwIcap_mSliceY2Row(&InstIcap, XRow);
                    slice = XHwIcap_mSliceXY2Slice(XCol, XRow);
                    xil_printf("\r\n Col,Row,slice: %d,%d,%d.",Col,Row,slice);
                    status = XHwIcap_GetClbBits(&InstIcap, Row, Col,
          XHI_CLB_LUT.CONTENTS[slice][XHI_CLB_LUT_G], LUT_Value, LUT_SIZE);

                    if (status != XST_SUCCESS) {
                           xil_printf("\r\nXHwICA_GetClbBits function failed. Device ID
read: %x status %x",
                                        InstIcap.DeviceIdCode, status);
             } else {
             xil_printf("\r\nXHwICAP_GetClbBits function success. Device ID read: %x\r\n",
                                  InstIcap.DeviceIdCode);
                    }
                    /* Output read LUT array */
                    for (i = 0; i < LUT_SIZE; i++) {
                           xil_printf("%d",LUT_Value[i]);
                    }
                    print("\r\nPress any key to continue...");    opcion = (char)inbyte();
                    break;

             case '9'://Write CLB/LUT
                    xil_printf("\r\n Executing option %c.",opcion);
                    print("\r\nLed(0) LUT_G on Slice_X62Y40, Led(1) on Slice_X62Y6.");
                    print("\r\nXCol: ");      XCol = (Xuint32)inbyte()-48;
                    XCol = XCol*10;           XCol = XCol + (Xuint32)inbyte()-48;
                    xil_printf("%d.",XCol);
                    print("\r\nXRow: ");      XRow = (Xuint32)inbyte()-48;
                    XRow = XRow*10;                 XRow = XRow + (Xuint32)inbyte()-48;
                    xil_printf("%d.",XRow);

                    /* Identify the LUT to change: LUT in SLICE_X[Col]Y[Row]. */
                    Col = XHwIcap_mSliceX2Col(XCol);
```

```
                    Row = XHwIcap_mSliceY2Row(&InstIcap, XRow);
                    slice = XHwIcap_mSliceXY2Slice(XCol, XRow);
                    xil_printf("\r\n Col,Row,slice %d,%d,%d.",Col,Row,slice);

                    print("\r\nType LUT_G Contents: ");
                    for (i = 0; i < LUT_SIZE; i++){
                          LUT_Value[i]=(Xuint8)inbyte() -48;
                          xil_printf("%d",LUT_Value[i]);
                    }

                    status = XHwIcap_SetClbBits(&InstIcap, Row, Col,
          XHI_CLB_LUT.CONTENTS[slice][XHI_CLB_LUT_G], LUT_Value, LUT_SIZE);

                    if (status != XST_SUCCESS) {
                          xil_printf("\r\nXHwICA_SetClbBits function failed. Device ID
read: %x status %x",
                                                InstIcap.DeviceIdCode, status);
              } else {
              xil_printf("\r\nXHwICAP_SetClbBits function success. Device ID read: %x\r\n",
                                      InstIcap.DeviceIdCode);
                    }
                    print("\r\nPress any key to continue...");    opcion = (char)inbyte();
                    break;

              case 'a'://Time to read 100 LUTs
                    xil_printf("\r\n Executing option %c.",opcion);
                    print("\r\nInitialize Timer.");

                    /* Initialize the timer counter so that it's ready to use,
                     * specify the device ID that is generated in xparameters.h */
                    status = XTmrCtr_Initialize(&TmrCtrInstancePtr,
XPAR_OPB_TIMER_1_DEVICE_ID);
                    if (status != XST_SUCCESS){return XST_FAILURE;}

                    /* Perform a self-test to ensure that the hardware was built
                     * correctly, use the 1st timer in the device (0) */
                    status = XTmrCtr_SelfTest(&TmrCtrInstancePtr, 0);
                    if (status != XST_SUCCESS){return XST_FAILURE;}

                    print("\r\nPress any key to continue...");    opcion = (char)inbyte();
                    break;

              case 's'://Time to read 100 LUTs
                    xil_printf("\r\n Executing option %c.",opcion);
                    print("\r\nReading 100 times LUT_G on Slice_X62Y40.");
                    XCol = 62; XRow = 40;
                    /* Identify the LUT to change: LUT in SLICE_X[Col]Y[Row]. */
                    Col = XHwIcap_mSliceX2Col(XCol);
                    Row = XHwIcap_mSliceY2Row(&InstIcap, XRow);
                    slice = XHwIcap_mSliceXY2Slice(XCol, XRow);
                    xil_printf("\r\n Col,Row,slice %d,%d,%d.",Col,Row,slice);

                    /* Start the timer counter such that it's incrementing by default */
                    XTmrCtr_Start(&TmrCtrInstancePtr, 0);

                    /* Get a snapshot of the timer counter value before it's started
           * to compare against later */
                    TmrCtrValue1 = XTmrCtr_GetValue(&TmrCtrInstancePtr, 0);

                    // Perform 100 LUT reads
                    for (i = 0; i < 100; i++){
                          status = XHwIcap_GetClbBits(&InstIcap, Row, Col,
                                XHI_CLB_LUT.CONTENTS[slice][XHI_CLB_LUT_G], LUT_Value,
LUT_SIZE);
                    }

                    /* Read the value of the timer counter and wait for it to change,
                     * since it's incrementing it should change, if the hardware is not
```

```
                     * working for some reason, this loop could be infinite such that the
                     * function does not return */
          TmrCtrValue2 = XTmrCtr_GetValue(&TmrCtrInstancePtr, 0);
       xil_printf("\r\nRead time: %d.", TmrCtrValue2 - TmrCtrValue1);

                     print("\r\nPress any key to continue...");    opcion = (char)inbyte();
                     break;

              case 'd'://Time to write 100 LUTs
                     xil_printf("\r\n Executing option %c.",opcion);
                     print("\r\nReading 100 times LUT_G on Slice_X62Y40.");
                     XCol = 62; XRow = 40;
                     /* Identify the LUT to change: LUT in SLICE_X[Col]Y[Row]. */
                     Col = XHwIcap_mSliceX2Col(XCol);
                     Row = XHwIcap_mSliceY2Row(&InstIcap, XRow);
                     slice = XHwIcap_mSliceXY2Slice(XCol, XRow);
                     xil_printf("\r\n Col,Row,slice %d,%d,%d.",Col,Row,slice);

                     print("\r\nWriting LUT_G Contents: ");
                     for (i = 0; i < LUT_SIZE; i++){
                            LUT_Value[i]=1;
                            xil_printf("%d",LUT_Value[i]);
                     }
                     /* Start the timer counter such that it's incrementing by default */
                     XTmrCtr_Start(&TmrCtrInstancePtr, 0);

                     /* Get a snapshot of the timer counter value before it's started
           * to compare against later */
                     TmrCtrValue1 = XTmrCtr_GetValue(&TmrCtrInstancePtr, 0);

                     // Perform 100 LUT writes
                     for (i = 0; i < 100; i++){
                            status = XHwIcap_SetClbBits(&InstIcap, Row, Col,
                                   XHI_CLB_LUT.CONTENTS[slice][XHI_CLB_LUT_G], LUT_Value,
LUT_SIZE);
                     }

                     /* Read the value of the timer counter and wait for it to change,
                      * since it's incrementing it should change, if the hardware is not
                      * working for some reason, this loop could be infinite such that the
                      * function does not return */
          TmrCtrValue2 = XTmrCtr_GetValue(&TmrCtrInstancePtr, 0);
       xil_printf("\r\nRead time: %d.", TmrCtrValue2 - TmrCtrValue1);

                     print("\r\nPress any key to continue...");    opcion = (char)inbyte();
                     break;

              default:
                     xil_printf("\r\n The character selected %c (ASCII %d) was not a valid
option.",opcion,opcion);
                     print("\r\nPress any key to continue...");
                     opcion = (char)inbyte();

              }
   }

       //this line never executes
       print("\r\n-- Exiting main() --\r\n");
       return 0;

}
```

Este programa permite probar la mayoría de funciones del *hwicap* (explicadas en el Apartado 2.3.3), de esta forma, si se pulsa:

1- Se inicializa el *hwicap*
2- Se desincroniza el *hwicap*
3- Se lee un *frame* de la memoria de programación
4- Se lee un *frame* del *buffer* de almacenamiento
5- Se escribe un *frame* en la memoria de programación
6- Se escribe un *frame* al *buffer* de almacenamiento
7- Se reconfigura con un *bitstream* parcial (opción no implementada)
8- Se lee un *CLB/LUT*
9- Se escribe en un *CLB/LUT*
10- Se obtiene el tiempo en leer 100 *LUTs*
11- Se obtiene el tiempo en escribir 100 *LUTs.*

Por tanto si una vez inicializado el *hwicap* se pulsa 9 se cambia la función *AND* de las *LUTs*, lo cual se puede comprobar en el resultado que producen los *LEDs* en la placa.

Además se ha obtenido el tiempo que tarda *hwicap* en leer y en escribir una *LUT*:

```
clk: 40MHz (25ns)

Tiempos para 100 LUTs:
      leer: 432654 tics. * 25ns = 0,0108s => 9245 LUTs/s
      escribir: 458359 tics.
```

Por otro lado se ha implementado un ejemplo similar para **Virtex-4,** que se encuentra en la carpeta *Anexos/reconfigLUTvirtex4*, realizado en *EDK 8.2 Service Pack 1,* en el que no se realiza la modificación relativa a las puertas AND, sino que simplemente se realiza el proyecto EDK con un programa similar al explicado anteriormente, obteniendo los mismos resultados positivos en la reconfiguración de *LUTs* y *frames*, y cuyos resultados temporales son los siguientes:

```
clk: 100MHz (10ns)
Tiempos para 100 LUTs:
      leer: 383346 tics. * 10ns = 0,00383346s => 26086 LUTs/s
      escribir: 423035 tics. * 10ns = 0,00423035s => 23638 LUTs/s

clk: 50MHz (20ns)
Tiempos para 100 LUTs:
      leer: 393246 tics. * 20ns = 0,007864920s => 12714 LUTs/s
      escribir: 417751 tics. * 20ns = 0,008355020s => 11968 LUTs/s
```

BIBLIOGRAFÍA

- *Síntesis de Sistemas Digitales con VHDL.* Garrigós, Toledo, Martínez.
- *Página web: www.xilinx.com.* Xilinx Inc.
- *Página web: www.itee.uq.edu.au/~listarch/partial-reconfig.* School of Information Technology & Electrical Engineering
- *Application Note XAPP151: Virtex Series Configuration Architecture User Guide*. Xilinx Inc.
- *Application Note XAPP290: Two Flows for Partial Reconfiguration: Module Based or Difference Based.* Xilinx Inc.
- *Application Note XAPP 255: XAPP 255: Using Partial Reconfiguration to Time-Share Device Resources in Virtex-II and Virtex-II Pro.* Xilinx Inc.
- *Application Note XAMP502: Using a Microprocessor to Configure Xilinx FPGAs via Slave Serial or SelectMAP Mode*. Xilinx Inc.
- *User Guide UG002: Virtex-II Platform FPGA User Guide*. Xilinx Inc.
- *User Guide UG070: Virtex-4 FPGA User Guide.* Xilinx Inc.
- *User Guide UG071: Virtex-4 FPGA Configuration User Guide.* Xilinx Inc.
- *User Guide UG 208: Early Access Partial Reconfiguration User Guide.* Xilinx Inc.
- *Xilinx Device Drivers Documentation.* Xilinx Inc.
- *Partial Reconfiguration Software User's Guide.* Xilinx Inc.
- *EA PR Training. XUP Professor workshop.* Xilinx Inc.
- *Embedded Processing in FPGA Partial Reconfiguration. 2006.* Dan Isaacs, Punit Kalra, and Eric Shiflet.
- *Auto-Reconfiguración sobre FPGAs.* Castillo, Gonzalez, Huerta, Martinez.
- *The designer's guide to VHDL.* Ashenden.
- *Digital signal processing with field programmable gate arrays.* Meyer-Baese.