Sistemas Informáticos
Curso 2003-04

Análisis y construcción de un interfaz JAVA para reprogramación dinámica de dispositivos FPGA

Diego Sánchez Casas
Neftalí Mañes García
Javier Sánchez Jurado

Dirigido por:
Prof. Julio Septién
Dpto. Arquitectura de computadores y Automática

Facultad de Informática
Universidad Complutense de Madrid

Los autores de este proyecto autorizan a la Universidad Complutense de Madrid a difundir y a utilizar con fines académicos no comerciales, y mencionando expresamente a sus autores tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Neftalí Mañes García **Diego Sánchez Casas** **F. Javier Sánchez Jurado**

Resumen del proyecto:

El objetivo de este proyecto es establecer unos mecanismos básicos para lograr multitarea hardware utilizando dispositivos FPGAs y las cualidades de reprogramación dinámica de estos. En él se difunden los distintos problemas que esto implica, mencionándose soluciones a algunos de ellos y explicando las distintas características de cada aproximación. Se ha construido un prototipo en lenguaje Java que, estableciendo comunicación con sistemas de FPGAs, permite realizar las operaciones primitivas necesarias para desarrollar esa tecnología. La arquitectura desarrollada se integra con las herramientas de Xilinx Foundations, utilizando la librería JBits para la reconfiguración parcial de los dispositivos y respetando los estándares de la industria.

Abstract:

The goal of this project is to set the basic ways to achieve hardware multitask using FPGA devices and their dynamic reprogrammable attributes. It attempts to proportionate solutions to the several problems originated during the process, trying different approximations. A Java prototype has been built in order to connect to the FPGA using the basic primitives of this technology. The developed architecture is integrated with the tools provided by Xilinx Foundations, using JBits library for partial reconfiguration of the devices and always according to the standards of the industry.

Palabras Clave:

FPGA, JBits, BoardScope, XHWIF, multitarea, reconfiguración , Virtex, Xilinx, CLB, Core

Indice:

Neftalí Mañes García
Diego Sánchez Casas
F. Javier Sánchez Jurado

1. Introducción

1.1. ¿Qué es una FPGA? Convirtiendo el hardware en software

Desde los albores de la informática, han existido gran cantidad de cambios en la forma de entender y construir los distintos dispositivos que la hacen realidad. Si bien esto es habitual en la evolución de cualquier disciplina, la evolución en el campo de las computadoras no ha sido ni normal ni mucho menos esperable.

Suponga que, actualmente, usted desea construir un circuito independiente, sin tener ni siquiera que integrarlo con otros dispositivos. Es más, usted quiere construir un sencillo ecualizador analógico de tres bandas por el método tradicional. ¿Por donde empieza?

El camino tradicional sería diseñar el circuito. Se imagina que va a necesitar una serie de amplificadores operacionales para filtrar la señal, resistencias variables para la modulación, resistencias fijas para los circuitos, dos jacks para la entrada y la salida de la señal, más amplificadores operacionales para sumar la señal resultante y algunos inversores de fase para que todo llegue adecuadamente. A esto debemos añadir una placa para colocar los componentes, estaño, un soldador, y una mesa de laboratorio donde trabajar. Se dirige a la tienda más cercana y vuelve con una bolsa. Se sienta, saca todos los componentes, diseña las conexiones de la placa con un programa de ordenador y coloca ésta en la insoladora (¿porque tiene insoladora, verdad?). Ahora “sólo” tiene que agujerear la placa y soldar los circuitos...

Ahora suponga que, en lugar de construirlo, usted desea programar una aplicación que se comporte como un ecualizador (digital, para simplificar). Ahora también debe de diseñar el circuito, pero no a nivel de transistores y resistencias sino a nivel de métodos y funciones. El siguiente paso consiste en sentarse delante de su ordenador, que representa todas las herramientas que usted necesita. Escribe el programa en cualquier lenguaje que desee, lo compila, y listo.

Estos dos ejemplos de desarrollo son los dos arquetipos básicos de la informática: el desarrollo HW y el desarrollo SW. Como ve, los dos métodos son perfectamente válidos, pero el primero es mucho más complicado y engorroso. Si bien los dos generan un ecualizador perfectamente válido, el primero requiere un equipamiento mucho mayor y un esfuerzo proporcional a la complejidad del circuito. ¿Con cuál se queda?

Ahora vamos a ampliar nuestro punto de vista. Suponga que un compañero suyo, que vive en Moscú, desea ver su ecualizador. Con la segunda opción, podemos enviarle el programa por correo electrónico junto con unas sencillas instrucciones de uso. Con la primera, “simplemente” debemos repetir todo el trabajo para crear una copia, meterla en una caja, mandarla por correo, y pedir disculpas a su amigo por el retraso.... y esperemos que no se pierda en correos, más vale que lo mande como frágil y certificado.... Es más, si nuestro ecualizador nos gusta y queremos más copias, ¡nos resulta casi imposible si lo hemos construido en HW! Este hecho, que puede parecer quizás anecdótico, representa pocos problemas si lo comparamos con los fallos de refinamiento de nuestro proyecto. Si nuestro programa no funciona, tenemos a nuestra disposición herramientas de debug para

localizar el problema, mientras que buscar un posible fallo en un circuito HW resulta extremadamente complicado. A tal cantidad de problemas debemos sumarle posibilidades reales como que se nos estropee una pieza física de hardware o que queramos modificar el diseño. Como ve, el método de diseño software es mucho más eficiente.

La industria de la electrónica mundial se enfrenta básicamente a los problemas anteriores. Actualmente el sistema de creación de nuevos componentes electrónicos pasa por tres fases: Diseño---> Prueba --> Producción. Mientras que la primera y la última son inevitables y de coste conocido “a priori”, la segunda de ellas representa actualmente más del 70% del coste de cada proyecto y su coste en tiempo y dinero es impredecible. Cada prueba que se realiza en el proyecto obliga a la construcción de un nuevo prototipo con los costes que conlleva y, en caso de encontrarse un error, es necesario retroceder a diseño y construir un prototipo nuevo para la fase siguiente.

Dado que esta opción es a todas luces inaceptable, en la práctica se han desarrollado simuladores para comprobar la corrección de nuestro diseño, no construyéndose un prototipo hasta que los simuladores generan respuestas correctas. Sin embargo, y a pesar de que los simuladores han mejorado muchísimo en un periodo de tiempo relativamente corto, resulta imposible simular la realidad, y además estamos obligados a “diseñar” en un lenguaje aceptable por el simulador. Lógicamente era necesario refinar el procedimiento

Asimilemos este sistema de diseño al sistema que siguen algunas comunidades de “software libre”, por ejemplo. Estas comunidades de diseño se basan en una biblioteca de programas cuyo funcionamiento ya ha sido probado y que van reutilizándose. Cuando detectan algún fallo en la fase de prueba, cogen el programa, lo corrigen, y lo vuelven a introducir en un “recipiente de pruebas” (normalmente una computadora típica) para su prueba. Actualmente este sencillo sistema ha demostrado un excelente rendimiento esfuerzo-coste, propiciando una evolución muy fuerte en el campo de las aplicaciones informáticas.

Si quisiéramos aplicar ese modelo de desarrollo al hardware, utópicamente necesitaríamos una biblioteca de componentes HW (algo parecido a un armario inacabable de donde sacamos componentes ya construidos) y un recipiente donde pudiéramos mezclarlos de una manera cómoda y rápida, obteniendo el circuito resultante con poco esfuerzo y en un tiempo pequeño. Este ideal, que a primera vista provoca escepticismo, es la base de un sistema de desarrollo hardware que, parece, acabará desbancando al método tradicional.

Gracias a la necesidad de simuladores en los años 70 y 80, se crearon lenguajes de descripción hardware para representar circuitos muy parecidos a los actuales lenguajes de programación de propósito general. La diferencia es que, mientras que los segundos deben ser compilados para generar un ejecutable, los primeros son interpretados por un (perdónenme por la expresión) “compilador” que genera las salidas y estados del circuito en función de las entradas. Tomando como herramienta estos lenguajes, es perfectamente posible obtener la biblioteca de componentes que necesitamos, así como crear un compilador que nos permita unir varias de estas piezas para formar un todo. Aunque este proceso sirve de gran ayuda, todavía estamos hablando de simulación, no de electrónica. Precisamente esta diferencia suscito la creación de dispositivos que, para la mayor parte de la gente, son a la vez desconocidos y sorprendentes. Estamos hablando de las FPGAs (Functional Programmable Gate Arrays).

Una FPGA es, a grosso modo, una colección de transistores y conexiones que podemos ajustar como deseemos. Partiendo de la base de que todo circuito digital consta de una serie de dispositivos (formados por transistores) conectados entre sí, podemos hacer realidad dentro de la placa FPGA cualquier diseño simplemente conectando los transistores entre sí para formar los dispositivos y más tarde uniendo estos. Aunque esta descripción suele ser complicada de entender en principio, vamos a poner un sencillo ejemplo: un diagrama de las vías del tren. Este diagrama está dado por bloques (estaciones) conectado por aristas (vías) formando un grafo. El circuito (camino del tren) viene dado por la posición de las agujas de los cruces de las vías, lo que condiciona qué estaciones se encuentran interconectadas. Dependiendo de la función que demos a los bloques podemos tener un contador (un tren de pasajeros si las estaciones son de pasajeros) o un registro (un tren de mercancías). Básicamente, una FPGA es con lo que sueñan los afinados a un Mecano.

El problema de cómo configurar los dispositivos y conectar las líneas se reduce nuevamente a contar con las herramientas adecuadas. En la actualidad, y partiendo de módulos HW programados en los lenguajes de descripción hardware previamente nombrados, existen herramientas de sinterización que “compilan” el diseño generando un archivo de configuración que podemos cargar directamente en la FPGA, convirtiéndola en el circuito que nosotros deseemos. La E/S de dichos circuitos puede ser también programada, ya que en la actualidad las FPGAs vienen montadas sobre una placa que contiene diversos componentes de E/S ya conectados a la misma en determinados puntos. Si queremos que una determinada línea del circuito use un determinado dispositivo no tenemos mas que conectar esa línea al punto correspondiente

Los escépticos a estos componentes pueden argumentar que las FPGAs requieren de una cantidad desorbitada de transistores en su arquitectura, siendo entonces tremendamente caras y grandes. Esta idea no puede ser más opuesta a la realidad. Debido al enorme avance en la capacidad de integración, el tamaño de la lógica digital se ha reducido enormemente. Los índices de integración actuales usados para crear mares de puertas consiguen integrar millones de transistores en zonas 10 veces inferiores que hace 5 años. Y lo que puede ser incluso más importante: a un coste incluso menor. Por ello las FPGAs, aunque en la actualidad resulten un poco caras para el usuario normal, so una alternativa muy económica que con un corto periodo de amortización

De esta manera el coste de desarrollar un nuevo prototipo es prácticamente cero, como pasa en el modelo de SW. Ha cambio, es necesario un nivel de conocimiento al menos básico de estas nuevas herramientas, lo que pone de manifiesto la realidad actual de sobra conocida en el mundo de la tecnología: renovarse o morir.

1.2. Multitarea Hardware. Posibilidades reales y aproximaciones teóricas

Buscando de nuevo un reflejo óptimo, es necesario un breve resumen de la evolución de los sistemas operativos en el tratamiento de la ejecución de programas. Inicialmente, los grandes mainframes de hace cincuenta años recibían los programas mediante la manipulación de conmutadores y medidores. Aun con el posterior uso de tarjetas perforadas la pérdida de tiempo que suponía la introducción de los programas por parte de los ingenieros no era admisible en máquinas de un coste tan elevado. Por ello, se creó un tipo especial de programa (batch), el antecesor de los actuales sistemas operativos

multitarea. Dicho programa, que residía permanentemente en el ordenador, se encargaba de gestionar automáticamente la carga de los programas y de optimizar esta, ahorrando una enorme cantidad de tiempo que redundaba en un mayor rendimiento de la máquina. Este sistema, denominado carga por lotes, seguía teniendo dos graves problemas, generados por la linealidad de su manera de operar. El primero de ellos era que unos pocos programas muy largos podían bloquear durante una gran cantidad de tiempo a muchos programas más pequeños. El segundo era que, mientras el programa en ejecución esperaba una E/S, la CPU no realizaba ningún trabajo, con la correspondiente pérdida de rendimiento.

Esto fue resuelto en posteriores años mediante la modificación del modo de entender la ejecución de los programas. Desapareció la linealidad de operación cargándose varios programas en la, ahora, mayor memoria de las computadoras. A pesar de que en cada momento únicamente puede haber en ejecución un programa por cada procesador que posee la máquina, al estar los procesos en memoria el procesador puede ir dividiendo su atención entre todos ellos, de forma que todos avancen en igual medida. Aunque esto conlleva una sobrecarga en el sistema debido al coste de intercambiar el programa activo, consigue eliminar el problema de la inanición. Además, cuando un programa se encuentra esperando una E/S, el procesador puede atender al siguiente sin necesidad de gastar su tiempo. Este procedimiento, por el cual un procesador simula atender a la vez a varios procesos, se denomina multitarea simulada o multitarea software.

Como el lector podrá suponer, la siguiente evolución de este sistema viene por supuesto orientada a conseguir una verdadera multitarea, es decir, que el HW ejecute realmente dos tareas al mismo tiempo. La manera más normal de conseguir esto es instalar más de un procesador, compartiendo la carga de trabajo entre los dos y encargándose cada uno de una tarea. Este proceso es en realidad enormemente complejo de llevar a la práctica, debido a problemas de compenetración y sincronización. Además, tiene el problema de que aunque dupliquemos el procesador no podemos hacer lo mismo con el resto de componentes de la máquina, por lo que procesos que necesiten el mismo recurso deben forzosamente esperar.

Este enfoque tradicional se ha usado durante los últimos años para construir ordenadores de alta gama prácticamente por encargo. Si bien existen organizaciones que requieren máquinas con estas capacidades, el alto coste de diseño y de fabricación de estas plataformas hace inviable su masificación, siendo reemplazadas por otro tipo de computadoras de menor cuantía.

Sin embargo, en el concurrido espacio tecnológico actual la multitarea HW representa una verdaderamente provechosa línea de investigación. Un equipo capaz de ejecutar diversas tareas a la vez con una complejidad razonable elevaría muchísimo la productividad global, sin contar con la posibilidad de empezar a tratar problemas que, hasta ahora, se consideran como no computables. Además, tendríamos la ventaja añadida de trabajar en alta disponibilidad de componentes, ya que si un procesador se estropea, el resto podrían seguir trabajando. Lamentablemente y aunque se han realizado notables avances en computación paralela, el problema recurrente de duplicar todos los dispositivos externos para cada micro convierten la multitarea HW en algo más parecido a poner muchas máquinas juntas a trabajar en tareas relacionadas que a una máquina que ejecute todas ellas.

Especulando, animamos al lector a imaginarse (y, por qué no, a diseñar) las características que pudiéramos pedirle a una máquina que funcione con multitarea HW.

Dicha máquina debe ejecutar a la vez varias tareas, sincronizándose entre sí. Además, cada tarea debe tener acceso a todos los recursos internos que necesite, como por ejemplo la memoria, sin tener que esperar a sus compañeras (salvo quizás casos puntuales). Igualmente cada tarea debe ser capaz de usar dispositivos de E/S, de nuevo con capacidad para hacerlo independientemente. Por último, dichas tareas deben poder ser cargadas y retiradas dinámicamente, sin afectar a la ejecución de las ya presentes. A priori la tercera condición parece imposible de cumplir, ya que si ponemos x unidades de un recurso y z tareas, donde z>y, forzosamente una tarea debe esperar... eso es claro, pero ¿está usted seguro?

La realidad es que resulta imposible intentar esto mediante las técnicas actuales, ya que el HW limita la posibilidad de ejecución de las tareas. Sin embargo, la aparición de las FPGAs ha abierto un camino de investigación hasta ahora inexistente. Tradicionalmente los programas se escriben teniendo en cuenta restricciones de HW, pero con ellas es posible construir el HW teniendo en cuenta restricciones de SW, invirtiendo el modelo y propiciando la primera posibilidad real. Partiendo de una placa que puede implementar cualquier circuito en cualquier momento, podemos usarla para analizar un programa, considerar los elementos de HW necesarios para ejecutarla, crearlos y, por último, ejecutarla sea cual sea. Usando diversas placas (o partiendo una sola) podemos ejecutar varias tareas a la vez sin preocuparnos por los dispositivos que estas requieran, simplemente creándolos según los necesitemos.

Este proyecto ha intentado demostrar que este planteamiento, aunque novedoso, es posible.

1.3. Características principales de un proyecto de investigación

Antes de continuar, nos gustaría explicar brevemente lo que han sido para nosotros estos diez meses de trabajo, argumentando cuestiones necesarias para entender el resto de este texto.

Cuando alguien se plantea embarcarse en cualquier trabajo o tarea, lo primero que se pregunta es si alguien lo ha hecho antes y si podemos aprender de ellos. La idea de no reinventar la rueda, ampliamente extendida en todo lo referente a la programación, es siempre primaria, intentando no repetir errores pasados. Esto desemboca en que, en general, una nueva tarea suele tener una pequeña (pero clave) parte de originalidad y una gran parte de reutilización de procesos o herramientas ya usadas anteriormente. Más aún, con el auge de las comunicaciones, es bastante sencillo encontrar a alguien que ha resuelto problemas parecidos a aquellos a los que nos enfrentamos o que al menos pueden orientarnos

Sin embargo, nuestra situación frente a este proyecto ha sido toda la contraria. Partiendo de un objetivo en mente y de unas herramientas, hemos tenido que pensar el camino para llegar a algo, avanzando paso a paso, y frecuentemente llegando a callejones sin salida. En este texto vamos a tratar de explicar los distintos caminos que hemos estudiado y las distintas aproximaciones a problemas que nos han ido surgiendo. Destacando que los resultados negativos son también resultados, queremos hacer hincapié en los enfoques erróneos que hemos tenido y en las soluciones imposibles que hemos intentado, esperando que nadie vuelva a sentirse atraído por ellos y explicando qué fallaba

en ellas. Además dejamos gran cantidad de incógnitas sin resolver y de caminos sin empezar, animando al lector a que estudie la viabilidad de estos y, si lo desea, que avance por ellos, agradeciendo cualquier aportación. En ningún caso queremos afirmar que nuestras elecciones sean las óptimas, simplemente las que hemos encontrado con mayor viabilidad en este momento

Cualquier lector puede quedarse sorprendido viendo el código de nuestros prototipos o el acabado de los mismos por la extrema sencillez de su programación y de su interfaz, pero no debe pensar por ello que son sistemas sencillos. El ajuste de los componentes que tenemos sin una base clara, la falta total de documentación y de información ha hecho necesario muchísimo tiempo para entender el manejo de cada herramienta y para pensar como adaptarlas a nuestros propósitos. Nos hemos esforzado para que el código sea inteligible y sencillo, adaptándose a lo que queríamos probar e intentando como fin último que demuestre que nuestras conclusiones eran correctas.

También queríamos agradecer a nuestros tutor, Julio Setién, por estar abierto a todas nuestras sugerencias y aceptar los cambios de metodología y aproximación que continuamente hemos necesitado. Igualmente, también queríamos agradecer a la profesora Hortensia Mecha el hecho de habernos introducido en el mundo de las FPGAs, con todo lo que eso conlleva.

2. Objetivos y situación Actual

2.1. Objetivos del proyecto

En consonancia con lo explicado en el capítulo anterior, podemos fijar como objetivo principal de este proyecto construir un interfaz (usando Java) que nos permita gestionar reconfiguración dinámica HW en una FPGA. Obviamente resulta un propósito enormemente ambiguo, pero la idea subyacente es intentar buscar una manera de realizar multitarea HW y desarrollar un sistema que facilite su uso.

Idealmente deberíamos avanzar hacia la consecución de un gestor de tareas que oculte al usuario si se está usando multitarea software o hardware (o una mezcla de ambas). Sin embargo, como se verá más adelante, este tipo de multitarea es fuertemente dependiente del HW subyacente, por lo que es mucho más eficiente acotarlo a una arquitectura determinada.

La idea básica de la que partimos es muy simple: vamos a buscar una FPGA que admita reconfiguración dinámica y vamos a tratar de, imaginariamente, dividirla en columnas de tamaño fijo que van a representar zonas disponibles para ejecución. Esta división, puramente geométrica, nos proporcionará zonas en cada una de las cuales crearemos los componentes necesarios para la ejecución de una tarea, de manera parecida al mecanismo de carga de un proceso en memoria por parte de un sistema operativo. Aunque la idea final sea generalizar la posible ejecución, en la práctica vamos a diseñar una serie de tareas muy sencillas que demuestren que nuestro sistema funcione, intentando que su sistema de creación siga un estándar fácilmente abordable en tareas más complejas

Con esos dos componentes fijos, tendremos que desarrollar o encontrar alguna manera de cargar estas tareas en la placa, de una manera dinámica y preferiblemente parcial. Por dinámica entendemos que podamos realizarlo en cualquier momento que

deseemos, y por parcial que no sea necesario parar el resto de las tareas para la inserción de una nueva.

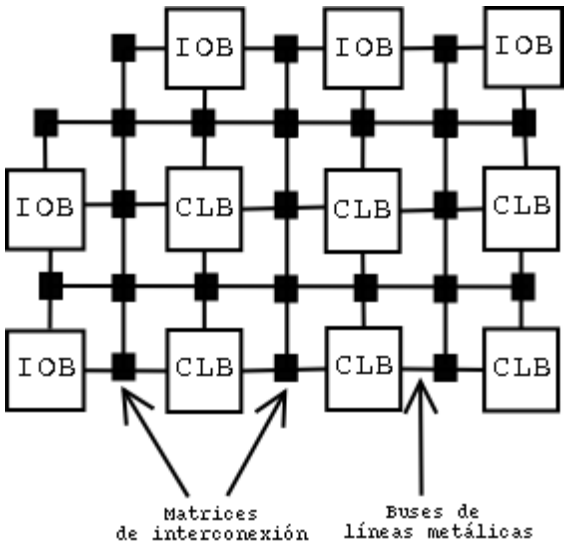
Obviamente el siguiente paso será diseñar un interfaz que permita realizar todo el proceso de una manera sencilla e intuitiva. Este interfaz debe ser fácilmente exportable para poder ser usado en cualquier tipo de terminal, y debe poseer una estructura que facilite la integración del sistema en red.

El lector avezado se habrá dado cuenta de que en ningún momento hemos mencionado la forma en que esas tareas se van a comunicar con el exterior. Ese proceso depende en gran medida de la elección de la placa y de la partición de ésta. Por ello en un principio decidimos no preocuparnos por ello hasta que tuviéramos más información al respecto.

2.2. Herramientas disponibles

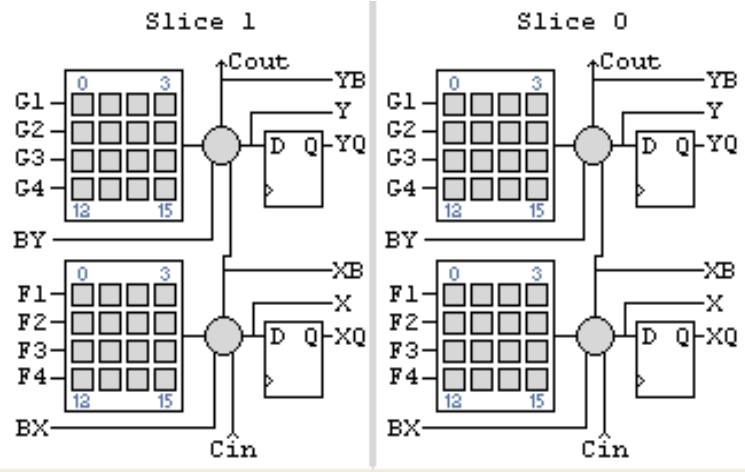
2.2.1. Virtex XCV1000: FPGA de alto rendimiento

De entre todas las placas FPGA del mercado, vamos a usar un ejemplar del modelo Virtex XCV1000 de Xilinx, usando una encapsulación BG560. Una FPGA general, la XCV1000 incluída, posee 3 elementos principales: los CLBs, los IOBs y la red de interconexión.En la siguiente figura puede observarse el esquema general.



Los CLBs son bloques idénticos de funcionalidad lógica, que contienen diversos componentes electrónicos (como multiplexores o biestables) entre los que destacan una serie de tablas de momoria (LookUp tables, LUT). Tanto las conexiones lógicas internas como el contenido de las LUT es configurable, realizando el CLB la función que nosotros queramos. La construcción de nuestro circuito puede realizarse configurando correctamente cada uno de estos bloques, necesitando un mayor número de estos según se amplie el circuito.

Nuestra placa en particular posee 6144 CLBs, colocados en 64 filas de 96 celdas. Cada uno de ellos está dividido en dos Slice que contienen dos biestables, dos LUTs y la lógica necesaria para establecer cualquier tipo de conexión interna. Su diagrama puede verse en el siguiente gráfico



La matriz de interconexión está compuesta por "switches" de conmutación que funcionan mediante un mecanismo parecido a los desvíos de las vías de un tren. Cada uno de las salidas o entradas de un CLBs puede conectarse directamente con cualquiera de los adyacentes, mientras que se pueden usar determinados puertos del switch para avanzar grupos de 6 CLBs. Asimismo cada CLB posee dos conexiones de entrada y una de salida a una serie de triestados configurables que permiten crear estructuras parecidas a un bus (ver capítulo 5.3)

Finalmente, para la entrada salida posee 414 bloques denominados IOBs, que no son sino pines conectados a distintos dispositivos de la placa, como memorias DRAM o LEDs. Si queremos incorporar uno de estos elementos a nuestro diseño, simplemente programamos la matriz de interconexión para que conecte el IOB correspondiente al punto del circuito donde queremos usar el elemento. Durante la práctica totalidad del proyecto vamos a usar los LEDs, cableando a los pines correspondientes, esto es: (AN28, AK25, AL26, AJ24, AM27, AM26, AK24 y AL25) como prueba del correcto funcionamiento de los sucesivos prototipos.

2.2.2. Jbits: Clases básicas de interacción

Jbits es un conjunto de métodos y rutinas, englobados en un paquete Java, que permiten realizar acciones sobre la configuración de una FPGA.

Lo primero que hay que tener en cuenta es que no se trata de una aplicación comercial, sino de una librería realizada sin ánimo de lucro por programadores aficionados a la materia. Por ello, carece por completo de soporte y la documentación existente es, cuanto menos, limitada.

Actualmente existen varias versiones de Jbits, siendo las dos últimas (y por tanto las más usadas) la 2.28 y la 3.0. Para nuestro estudio hemos escogido la 2.28, ya que soporta perfectamente nuestra placa y, en general, es mucho más estable que la siguiente. La restricción a la que nos vemos obligados por esto es que únicamente podemos usar la versión 1.22 de la máquina virtual Java. La razón de esto es que la compilación de algunas de las librerias de esta versión se ha hecho estáticamente usando ésta versión de la máquina virtual. Por ello, el uso de cualquier otra máquina vitual produce errores en las llamadas a ciertos métodos.

En el apéndice D se pueden encontrar los métodos más frecuentes de esta clase, junto con una breve descripción. Su forma exacta de uso y sus peculiaridades serán explicados coincidiendo con su utilización

Mediante el uso de esta librería, podemos acceder y manipular la FPGA directamente desde un programa Java, abstrayénonos incluso de su realidad física si es necesario. Esto último provoca que podamos usar una FPGA simulada, denominada en Jbits VirtexDS, para realizar las pruebas. La razón principal de usar este simulador en vez de la placa real está precisamente en el origen de la herramienta. Al ser JBITS una clase no oficial, y como bien se aseguran de explicar los autores, no existe seguridad en el código de que las modificaciones en el contenido de la placa no puedan dañar la integridad de la misma. Por ello es mucho más prudente realizar las pruebas en simuladores antes de probarlas en el dispositivo real.

2.2.3. Xilinx Foundation: Gestor de Xilinx y VHDL

Igual que ocurre con la mayor parte de los recursos informáticos, de nada sirve tener algo tan potente y flexible como las FPGAs si no disponemos de herramientas adecuadas para utilizarlo. Xilinx Foundations es un programa creado por la marca Xilinx para automatizar las tareas de programación y rutado de todos los dispositivos de la familia Virtex.

Dicho programa actúa de una manera parecida a como lo haría un compilador en el caso de los lenguajes típicos de programación: recibe una entrada en un lenguaje de alto nivel y devuelve el mismo programa en un lenguaje equivalente, pero de muy bajo nivel y comprensible para el hardware. Asimismo dispone de herramientas de debug para comprobar la generación de este código, permitiendo parametrizaciones de la manera de compilación y restricciones a aplicar sobre los recursos de la placa.

Como lenguaje de alto nivel, hemos optado por usar VHDL. Este lenguaje, actualmente uno de los lenguajes de descripción hardware más utilizados, es lo suficientemente genérico para implementar casi todos los dispositivos que pudieran interesarnos. Si bien es cierto que durante el desarrollo del proyecto vamos a usar un sólo circuito muy simple, concretamente un contador de cuatro bits, la idea subyacente es que sea posible la utilización de cualquier circuito programable en VHDL en lugar del nuestro.

La salida de dicho compilador es un archivo denominado bitstream, con extensión .bit. En él se encuentran los valores de todos los recursos de control de la fpga, en formato binario, formando una imagen que puede ser transmitida a la placa por diversos canales utilizando las herramientas de carga adecuadas, por ejemplo, JBits

2.2.4. Jbuilder 9: Compilador de Java

Creado por Borland Enterprises, Jbuilder es un compilador del lenguaje Java ya afianzado en la industria de la programación. Su versión nueve, la más reciente, es la que vamos a utilizar para compilar todo el código de nuestro prototipos.

La elección de este compilador viene determinada por su facilidad de uso y por las herramientas de depuración que nos proporciona. Además nos permite crear librerias sin problemas, modelo de encapsulación que pretendemos utilizar para la fase final de nuestro trabajo. Además presenta gran cantidad de ayudas a la programación, como precompilación o autocompletitud, facilitándonos el manejo de las clases que componen Jbits.

Para la ejecución del programa se ha asociado el compilador a la máquina virtual 1.2 de Sun, por las raones anteriormente explicada. Por esa razón se ha usado métodos obsoletos en las versiones actuales de la jdk, pero totalmente imprescindibles en esa versión.

3. Planteamiento inicial: Interfaz de modificación

3.1. Creando un entorno de trabajo

Para la realización del primer prototipo, hemos elegido montar una Virtex XCV1000 con encapsulación BG560 en un ordenador de sobremesa Pentium IV con 512 MB de RAM. La comunicación con dicha placa se va a realizar a través del bus PCI.

La comunicación con la placa es posible gracias XHWIF, un API proporcionado por XILINX para interactuar con sus placas desde programas Java. Este API, incluido con Jbits, facilita métodos para manejar la placa en su totalidad: escribir y leer datos de la placa, controlar su temporización y establecer la configuración de algunos de sus parámetros concretos, todo ello de manera transparente. En esencia, nos proporciona un nivel de abstracción de la FPGA física permitiéndonos tratarla como un recurso genérico sin preocuparnos de trabajos a bajo nivel.

Para poder usar la clase XHWIF con una determinada placa, es necesario crear una instancia de la clase que se adecue a la misma. Desde un punto de vista puramente de programación, la clase XHWIF es un interfaz Java cuyos métodos de acceso necesitan forzosamente ser creados antes de que puedan ser usados. Esto último se realiza automáticamente mediante el método get, el cual a partir del modelo y del tipo de placa objetivo carga los procedimientos correctos de acceso.

```
fpga = XHWIF.Get("rc1000pp:xcv1000");
```

El API de XHWIF está diseñado para poder conectar tanto como a un hardware local como a uno remoto. Para hacer las aplicaciones portables debemos ejecutar una pequeña cantidad de métodos antes de conectarnos a la placa. Estos métodos sirven para obtener el nombre del host y el puerto en el cual se encuentra el servidor. En el caso concreto de conectarse al hardware local no es estrictamente necesario aportar dichos datos, aunque pudieran ser introducidos igualmente, ignorándolos el interfaz al tratarse de una conexión local. Estos valores van a ser usados en el método connect para establecer el vínculo entre el programa y el hardware real.

```
dispositivo = fpga.connect();
```

En ausencia de hardware físico, esta clase también puede ser usada para interactuar con el simulador de FPGA VirtexDS, también incluido en el paquete de Jbits. La comunicación con este simulador se realizará mediante el BoardScope, una herramienta interactiva para depurar diseños, tanto en FPGAs físicas como en simuladores VirtexDS.

En primer lugar, y por considerar más fácil de testear, optamos por realizar las pruebas iniciales en el simulador VirtexDS, para una vez que funcionase de forma óptima portar los resultados a la FPGA física. Hay una restricción que impone JBits y que hace que el código de ambas versiones (física y simulada) difiera en algunos puntos. Para poder analizar los resultados en la herramienta gráfica Boardscope, es necesario el uso de una nueva clase, XHWIFConnection, que es la que le se pasa al BoardScope para iniciarlo. A partir de XHWIFConnection es posible obtener un nuevo objeto XHWIFWithEvents, que posee los mismos métodos que su equivalente XWHIF.

Por las razones anteriormente mencionadas, vamos a crear una conexión estable con el simulador, para después empezar a modificar la estructura interna de este.

Las sentencias necesarias para iniciar el simulador y conectarse a él son las siguientes:

```
//Creamos la conexión con el simulador, obteniendo la Virtex virtual
con = new XHWIFConnection();

//Arrancamos el Boardscope para poder observar los resultados
BoardScope bs = new BoardScope(con);
bs.show();

//Conectamos al simulador
con.connectToBoard("VirtexDS:XCV1000");

//reseteamos la fpga
con.resetBoard();
```

3.2. Cargar/Descargar Bitstreams: Encargando tareas

Una vez tenemos la placa funcionando y una manera de controlarla, nuestro siguiente paso es configurar la misma con un circuito de prueba, comenzando ahora a utilizar Jbits.

Los objetos Jbits se cren en memoria como un array bidimensional de enteros que representan la configuración de todos los elementos de la placa en un instante dado. Dependiendo de la FPGA que vaya a ser usada, la longitud de este array (número de CLBs) y las características de cada porción varían enormemente. Por ello, es necesario proporcionar al objeto estos datos. La manera más sencilla de realizar esto es utilizar la tabla de índices que viene incorporada con dicho paquete. Su uso es sencillo: simplemente debemos indicarle en la creación del objeto un número entero de referencia al modelo concreto de placa y durante la creación del objeto se consultarán los datos necesarios, creándose correctamente la instancia. Para obtener el número de referencia de nuestro hardware podemos utilizar la clase abstracta Devices, la cual nos devuelve diho número a

partir del nombre de nuestra FPGA. En nuestro caso concreto, la creación del objeto JBits se realiza mediante la siguiente llamada:

```
//creamos el objeto JBits
 miJbit = new JBits(Devices.XCV1000);
```

La modificación del objeto se puede realizar de manera global o específica. Globalmente hablando, podemos copiar dentro del objeto el contenido de una configuración válida mediante el método read. Esta configuración suele venir dada por un fichero .bit, bitstream generado por las herramientas automáticas de Xilinx. Cuando llamamos a dicho método, el objeto JBits lee secuencialmente la configuración de todos los componentes en el fichero traduciéndola y copiándola en el sitio adecuado de su array interno. Asimismo tenemos disponible el método write que realiza la operación inversa, esto es, nos genera un archivo .bit con la configuración actual de la fpga. Por jemplo, la llamada correcta para cargar en el objeto JBits creado en el párrafo anterior el bitstream localizado en el archivo cout.bit sería:

```
// Cargamos el bitstream de un contador cout.bit
Mijbit.read(cout.bit)
```

Una vez cargado el bitstream en el objeto JBits, la propia clase JBits proporciona unos métodos básicos de acceso a cada uno de los recursos de la FPGA. El valor actual de la configuración de cada recurso se representa internamente por un array de enteros. Mediante el método get, y especificando las coordenadas de la clb y el nombre del recurso concreto, podemos extraer el valor de ese dato en el bitstream. Asimismo podemos realizar la operación contraria, es decir, modificar el valor de la configuración de un elemento mediante el comando set, que requiere los mismo parámetros, así como el nuevo valor. Debido a la dificultad de recordar los códigos asociados a cada recurso y los asociados a los valores posibles de cada uno de estos, la propia clase JBits define unas constantes asociadas a estos códigos para poder desenvolvernos en un nivel más cercano. Dichas constantes se hayan encapsuladas en la clase java com.xilinx.JBits.Virtex.Bits. Un ejemplo del uso de estos métodos es el siguiente:

```
Int [] valor = mijbits.get( clbRow, clbCol, S0F1.S0F1)
```

Este caso concreto se utilizaría para obtener a qué está conectada la entrada 1 de la Lut F del Slice 0 del CLB [clbRow,clbCol].

```
mijbits.set( clbRow, clbCol, S0F1.S0F1, S0F1.S1_X)
```

Igualmente, con esta llamada modificaríamos el valor de esa misma entrada conectándolo a la salida del biestable X del Slice 1 de esa misma CLB.

3.3. Procedimiento de modificación dinámica. Lectura del estado actual y modificación de este

Faliarizados ya con el paquete JBits y sus clases de interacción con la FPGA, bien sea física o simulada, llega el momento de aprovechar toda la funcionalidad de dicho paquete. Hasta el momento hemos obtenido una manera de cargar configuraciones en memoria utilizando instancias de JBits. Sin embargo, estas configuraciones son estáticas y

completas. Es decir, únicamente podemos cargar en la placa todo el contenido de la configuración, y en ningún momento podemos leer el estado de la misma en la placa. Por ello carecen de la funcionalidad necesaria para llevar a buen puerto nuestras investigaciones

Sin embargo, la clase JBits proporciona también un repertorio de instrucciones denominado instrucciones parciales. Dichos métodos son muy parecidos en funcionalidad a los explicados anteriormente pero nos permiten realizar acciones únicamente en las partes de la placa que nos interesan.

Realmente, la idea que subyace en la reconfiguración parcial es hacer solo los cambios necesarios a un dispositivo para obtener la configuración deseada. La reconfiguración parcial realiza dicha función mediante la determinación de los cambios hechos desde la última configuración mandada al dispositivo y la presente configuración en memoria.

Desde un punto de vista funcional, la mayor diferencia es que usando métodos de reconfiguración parcial, estos métodos la clase JBits utiliza un sistema de bloques sucios (dirty blocks) para representar los recursos de la placa cuya configuración han sido modificada; de una manera parecida al sistema empleado por las caches de memoria de un sistema operativo. Gracias a esto, podemos configurar únicamente la parte de la placa que necesitamos y leer el estado actual de la placa.

Ahora la posibilidad de leer y parsear un bitstream creado con la herramienta de Xilinx recae en el método readPartial. Este método se comporta igual que su equivalente completo read. Sin embargo, inicializa todos los recursos de la Virtex como dirty en la matriz interna, indicando que se encuentran modificados en memoria pero no en la Virtex. Una vez tengamos en memoria la configuración que queramos cargar definitivamente en la placa, después de los cambios que hayamos introducido en ella, podemos realizar esta tarea usando el método setConfiguration de la clase XHWIF. Con la llamada a dicho método, se produce una serie de llamadas internas que modifican los valores de los recursos de la placa únicamente en el caso de que dichos recursos estén calificados como dirty en la estructura interna de la instancia de JBits; es decir, en el caso de que hayan sido modificados desde la última lectura de la placa

A continuación adjuntamos parte del código que permite realizar la tarea anteriormente mencionada:

```
 //creamos el objeto JBits
 miJbit = new JBits(Devices.XCV1000);
//leemos el bitstream y lo cargamos en un objeto JBits
miJbit.readPartial(nombreBitStream);
// Cargamos en el simulador el contenido del Jbits
fpgaExt.setConfiguration(0,miJbit.getPartial());
```

De esta manera, probamos a cargar una tarea básica (en nuestro caso y durante todo el desarrollo del proyecto optaremos por un contador), en el objeto JBits creado expresamente para la Virtex 1000. Luego, cargarmos en el simulador el bitstream almacenado y comprobamos en el BoardScope que realiza la función adecuada (realiza una cuenta ascendente). Hay que reseñar que hasta ahora no se ha intentado modificar el bitstream, de hecho, ese será el siguiente paso a realizar. Hasta este punto, hemos conseguido interactuar con el simulador de la FPGA, acceder a él mediante un API en java

diseñado para dicha función y posteriormente cargar una tarea en el simulador a través de un objeto JBits.

Una vez cargado el bitstream en la placa y ejecutado un cierto número de ciclos de reloj, es necesario desarrollar un mecanismo para obtener de nuevo la configuración de la misma. Pudiera pensarse que, al igual que hicimos anteriormente, podríamos simplemente usar el método JBits.readPartial. Sin embargo, esto sólo es válido para la primera llamada, puesto que dicho método carga la configuración desde un bitstream, y no desde el contenido de la placa. Sin embargo, la configuración de la placa ya no es la misma que la especificada por la herramienta de diseño de Xilinx, ya que se han ejecutado ciertos ciclos de reloj y por tanto ha cambiado el estado de nuestro circuito. La solución la proporciona también el API de reconfiguración parcial de JBits.

La utilidad ReadBackCommand , incluida en com.xilinx.JBits,Virtex , se usa para recuperar la configuración de la placa en el momento actual y cargarla de nuevo dentro del objeto JBits, para posteriores operaciones en el comon el caso anterior (mediante el método JBits.set, etc.). Esta clase presenta una estructura muy parecida a la reseñada anteriormente como propia de la clase XHWIF. Puesto que el bitstream generado varía lógicamente entre los distintos dispositivos usados, es necesario indicarle a ReadBackCommand el dispositivo particular que vamos a usar (en nuestro caso la Virtex 1000), así como la longitud de los bits que vamos a leer. Para ello configuramos el dispositivo de lectura mediante el método get ClbConfig antes de usarlo, recibiendo dicho método como parámetro el entero asociado a nuestro hardware.

Como orden de lectura, usamos un objeto axiliar XHWIF al que le aplicamos una configuración mediante su método setConfiguration, estableciendo como fuente la clase ReadBackCommand configurada para nuetro dispositivo. A continuacion extraemos dicho contenido a una variable auxiliar mediante un getConfiguration, lo que nos permite obtener un array de enteros que representa el estado actual de la placa. Es importante danos cuenta de uqe cuando leemos el contenido del objeto XHWIF solamente debemos quedarnos con la parte leida de la placa. Por ello le especificamos que lea únicamente un número de byets especificado por la propia clase ReadBackCommand, mediante su métodoo getReadLength() (Es necesario multiplicarlo por cuatro, ya que trabajan en unidades distintas)

Una vez obtenida dicha configuración, la clase JBits permite parsearla dentro de una de sus instancias, dejándola lista para poder accederse a sus componentes individuales (cada CLB particular y sus variables y constantes).

El bloque de código total para realizar un ejemplo de esta operación puede verse a continuación:

```
ordendelectura = ReadbackCommand.getClbConfig(Devices.XCV1000);
//numero de bits q debemos leer
longitud = ReadbackCommand.getReadLength() * 4;
//mandamos a la fpga la orden de lectura
fpgaExt.setConfiguration(0,ordendelectura);
//leemos la configuracion actual
contenido = fpgaExt.getConfiguration(0, longitud);
//parseamos el contenido dentro del jbtis
miJbit.parsePartial(ordendelectura, contenido);
```

Es importante reseñar que el primer paso que realiza la clase XHWIF cuando le ordenamos leer de la placa es para esta completamente, rearrancándola cuando escribamos en ella las modificaciones. Por ello, el tiempo de modificación del objeto Jbits y de carga/descarga de la configuración resulta crítico para el rendimiento del sistema.

Una vez encontrada una vía para recuperar el estado actual de la placa, podemos realizar modificaciones de la configuración de la placa en tiempo real. Para ello, nos basta con leer el contenido de la placa, parseándolo dentro de un objeto Jbits. A continuación realizaremos las modificaciones que consideremos oportunas dentro de éste mediante llamadas a la función set explicada anteriormente. Es muy importante pensar que cada llamada a set que hagamos marca como dirty la clb en que se encuentra dicho recurso, especificando que debe modificarse en la placa. El paso final consiste en cargar en la placa la nueva configuración. Para ello extraemos los cambios que debemos hacer en la configuración mediante el método getPartial y los aplicamos a la FPGA usando el método setConfiguration.

Un ejemplo de la escritura de una nueva configuración en el dispositivo es el siguiente fragmento de código:

```
      byte modificaciones[];
      modificaciones= miJbit.getPartial();
      if (modificaciones!=null) {
        try{
          fpgaExt.setConfiguration(0, modificaciones);
        }
        catch (XHWIFException e){
          System.out.println("No podemos conectar com la placa para
escribir la reconfiguracion");
        }
      }
  fpgaExt.setConfiguration(0, modificaciones);
```

En el primer prototipo realizado se permite modificar una entrada de una LUT, el propio contenido la la LUT, el modo de un biestable, así como también se permite resetearlo. Como conseguir este propósito se explicará en el manual de uso de dicho prototipo en el siguiente punto.

Una vez conseguido modificar una o varias componentes individuales de una CLB concreta, nos interesaría extender esta acción y poder realizar una copia íntegra de todas las componentes de una CLB concreta y recolocarlas en otra posición dada de la FPGA. Esto es equivalente a mover una CLB de posición. Para ello, debemos identificar todas las posibles constantes asignadas a una CLB y luego realizar una iteración de copia de todas la componentes variando las coordenadas que definen la localización de la CLB.

```
    // Copiar todos los recursos de una clb a otra
    for(i = 0; i < recs.length; i++)    {
      try      {
        miJbit.set(clbydest, clbxdest, recs[i],
miJbit.get(clby,clbx,recs[i]));
      }
      catch(Exception e)      {
        System.out.println ("Imposible copiar recurso " + i);
     }
```

Un listado con las constantes asociadas a todos los elementos funcionales de una CLB se puede encontrar en el Apéndice E.

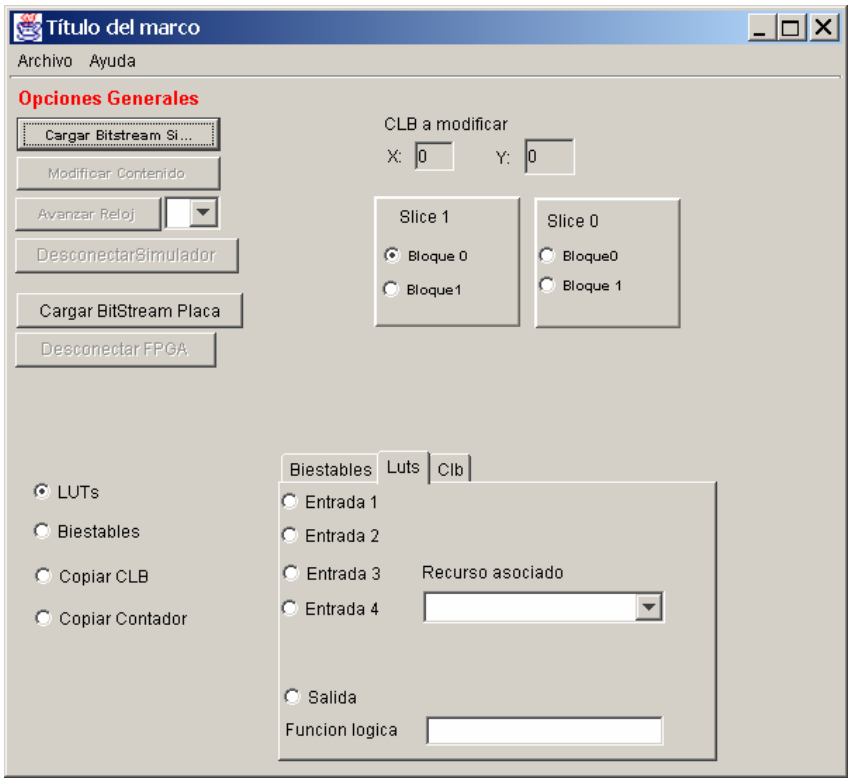
Por último, y una vez que hemos conseguido copiar una CLB entera de una posición a otra de la FPGA, nada nos impide copiar varias CLBs de lugar. Esto es tremendamente útil cuando sabemos el número de clbs de una tarea determinada y la posición en la que están localizadas en la FPGA, ya que nos permitaría mover una tarea de un lado a otro.

Hay que reseñar que al crear un bitstream con la herramienta de Xilinx, si no se le da ningún tipo de restricción de localización, se crea dicho bitstream de manera que luego al cargarlo en la FPGA la tarea simulada se coloque en la parte inferior izquierda de la placa.

En nuestro caso concreto, moveremos el contador de su sitio original a otra posición de la FPGA. Sabiendo que el contador que hemos diseñado ocupa 6 CLBs, lo movemos de su situación inicial (esquina inferior izquierda de la FPGA, clbs: (0,0), (0,1), (0,2), (1,0), (1,1), (1,2)) tres filas más arriba. Esto se consigue simplemente llamando varias veces al método que copia la CLB concreta, previa lectura de la configuración parcial de la FPGA con ReadBackCommand y posterior escritura con setConfiguration.

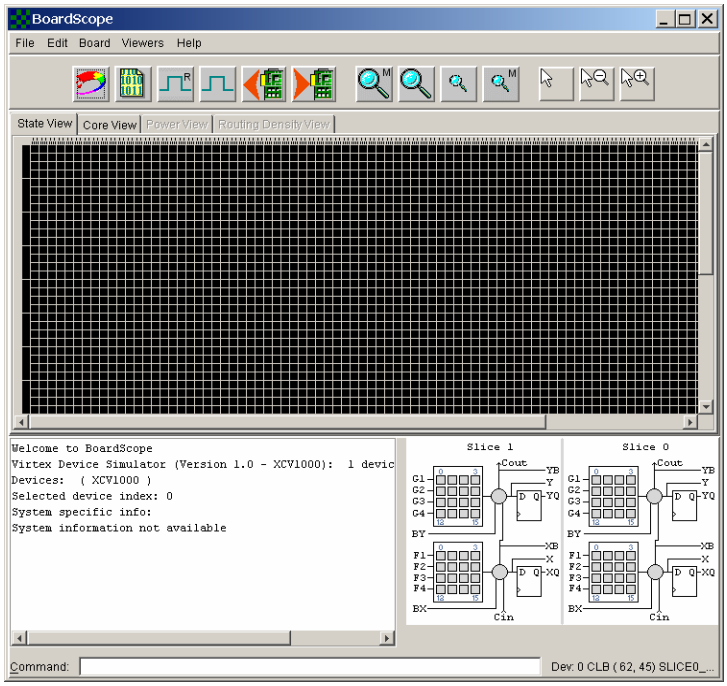
3.4. Prototipo 1 : Interfaz de carga y modificación de bitstreams

Esta es la apariencia de la ventana principal de nuestra aplicación:



Como se puede observar, se distinguen 2 tipos de botones según se esté accediendo a la placa física o al simulador. La forma de uso es la siguiente: se presiona el botón Cargar BitStream Simulador/Cargar BitStream Placa para escoger el bistream que deseemos cargar en el simulador/FPGA física. Si ejecutamos CargarBitStream Simulador, primero habrá que

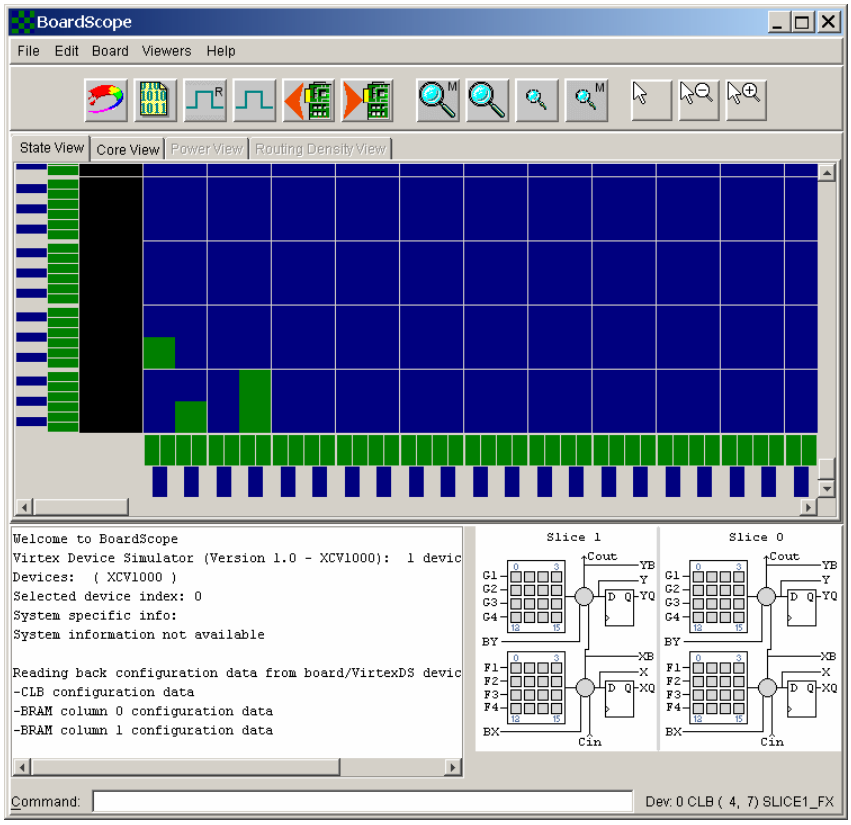
elegir el bitstream a cargar (archivo .bit). En el código de la aplicación se incluye el bitstream correspondiente a un contador que es el que hemos usado de prueba (cout.bit). Posteriormente a la elección del bitstream adecuado, arrancará automáticamente una ventana de debug correspondiente al BoardScope, que simula la estructura de CLBs de una FPGA. El interfaz del BoardScope se observa en la siguiente imagen:



Para poder ver en el simulador el funcionamiento del contador, es necesario en primer lugar pulsar el quinto botón empezando por la izquierda, que es el responsable de que el BoardScope lea los datos de configuración desde el dispositivo (en este caso, nuestro simulador VirtexDS). Una vez hecho esto, se le pueden ir dando ciclos de reloj (botón que representa un pulso de reloj con una R), y se va observando como en las 6 CLBs de mas abajo a la izquierda van apareciendo unos cuadross verdes que representan si la salida del biestable correspondiente al bloque dentro del slice está a 1 .

Hay que decir que el botón AvanzarReloj de nuestra interfaz realiza el mismo proceso que el botón del BoardScope, con la característica adicional de permitir avanzar un número de ciclos de reloj determinado por el usuario de una sóla vez. Solo hay que recordar, una vez que se pulse al botón de AvanzarReloj, volver a pulsar el botón del boardscope que lee la configuración de la FPGA, para refrescar la pantalla.

Esta es una captura de pantalla del contador una vez ejecutados 10 ciclos de reloj. Los cuadrados en verde que se observan corresponden a la salida a 1 del biestable correspondiente (cada CLB se divide en 2 slices y éstas a su vez en 2 bloques cada una).



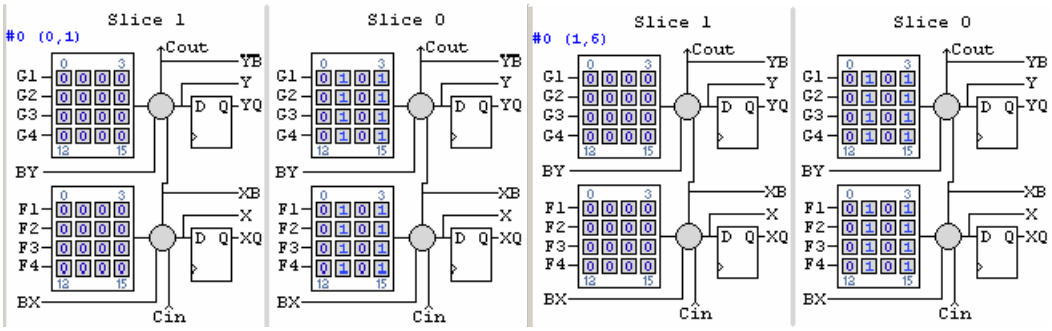
Ahora toca el turno de explicar el cometido del botón Modificar Contenido. Dicho botón es el que nos va a permitir modificar el contenido de un componente concreto de una CLB, bien copiar una CLB entera o incluso copiar la tarea entera (nuestro contador) de un lugar a otro de la FPGA.

Vamos a empezar con lo más sencillo, modificar una parte de una CLB. Para ello, introduciremos las coordenadas X e Y de la CLB a modificar en la parte superior derecha de nuestra aplicación, así como el Slice, bien sea el 0 o el 1 que va a contener la constante a modificar y el bloque correspondiente dentro del Slice. Una vez hecho esto, nos situaremos en el menu de paneles de la parte inferior, donde, según marquemos los botones de la parte izquierda, se seleccionará el panel adecuado en cada caso. Así por ejemplo, si pulsamos el botón LUTs, el panel de LUTs se marcará, dándonos a elegir entre modificar cualquiera de las 4 entradas de las que dispone la LUT correspondiente al Slice de la CLB que hemos decidido modificar y pudiendo cambiar su recurso asociado, es decir, es posible conectar el recurso que elijamos (por ejemplo la constante SINGLE_SOUTH14) a la entrada 2 de la LUT. Una vez elegido lo que queremos modificar, se pulsará el botón Modificar Contenido. Un mensaje aparecerá por pantalla indicando lo que se ha modificado. Hay que reseñar que el BoardScope no nos permite comprobar gráficamente y directamente dichos cambios, ya que sólo refleja el contenido de las LUTs y de los Biestables, y no el estado de las conexiones de los mismos. Pese a todo, una prueba válida es modificar una entrada crítica para la tarea (contador) y comprobar que a partir de entonces la tarea no se ejecuta correctamente.

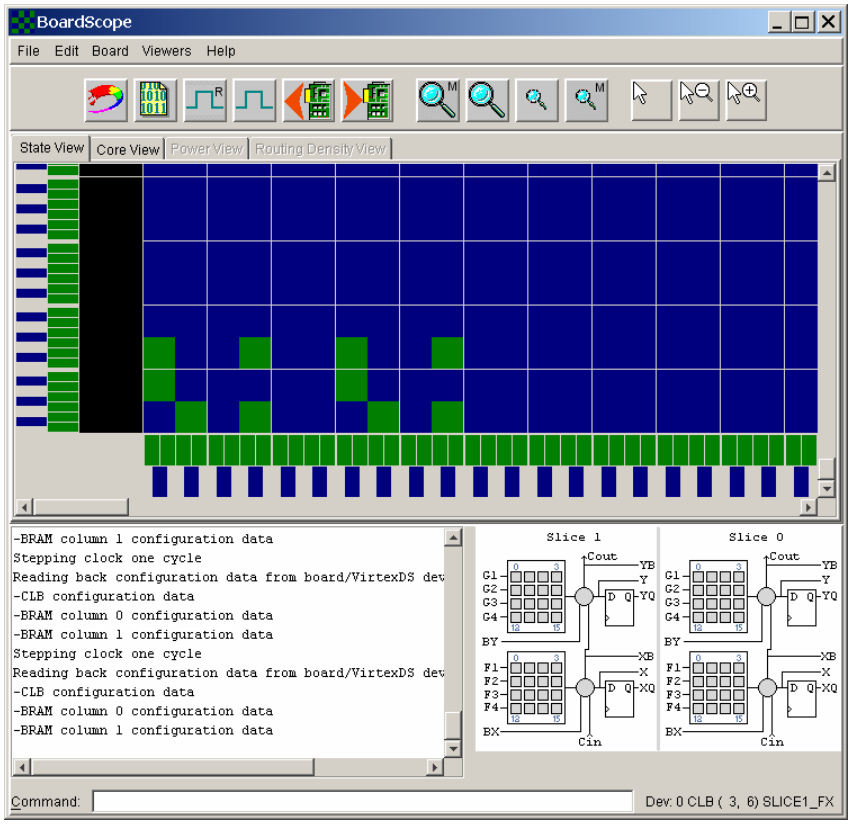
Las acciones permitidas sobre los recursos de una CLB son los siguientes:

- modificar el recurso asociado a una de las 4 entradas de la LUT.
- asociar a la salida una función lógica, con la sintaxis apropiada indicada en el manual de JBits (p.ej: f1 & f2 & f3)
- modificar la entrada de un biestable
- variar el modo del biestable (entre latch y flip flop)
- resetear el biestable, tanto de modo asíncrono como síncrono

Como ya hemos dicho anteriormente, es posible copiar el contenido íntegro de una CLB y trasladarlo a otra posición cualquiera de la FPGA. Eso es lo que vamos a hacer con la CLB (0,1) en el momento en que hemos ejecutado 10 ciclos de reloj, copiándola a la posición (1,6) por ejemplo. Para ello, sólo hay que pulsar el botón Copiar CLB del lado inferior izquierdo e introducir las coordenadas de la CLB destino en el panel que se selecciona. Un mensaje diciendo que la copia ha sido correcta se muestra en la ventana de la aplicación. Podemos comprobar parte de esta afirmación pinchando con el ratón y examinando el contenido de la CLB (0,1) (imagen de la izquierda) y comparándola con la CLB (1,6) (imagen de la derecha) después de la copia. Observamos que el contenido de las LUTs son idénticos, y que dicho contenido ha cambiado en el caso de la CLB(1,6), puesto que antes de la copia todos los valores estaban a 0 (dicha CLB no estaba siendo usada por la tarea).



Por último, el fin real de este primer prototipo es permitirnos copiar una tarea íntegra de una zona de la FPGA a otra, mediante el uso de una interfaz Java y de JBits. Esto es lo que vamos a hacer con el botón Copiar Contador. Pulsándolo y haciendo click en modificar contenido se copiarán las 6 CLBs situadas en la parte inferior izquierda de la FPGA 3 columnas más a la derecha. Si después de la copia empezamos a ejecutar ciclos de reloj en la FPGA, observamos que ahora tenemos un contador duplicado en lugar de el único que teníamos antes, y que se sigue contando adecuadamente con los dos contadores, siendo simétrica su forma de actuar. Esto lo ilustramos con la siguiente captura de pantalla, que corresponde al contador ya duplicado y habiéndose ejecutado unos cuantos ciclos de reloj. Se puede observar la simetría entre los dos contadores, contando con 3 columnas de diferencia:



Por último, para desconectar del simulador y cerrar la aplicación bastará con pulsar el botón Desconectar FPGA/Desconectar Simulador, según se trate del dispositivo físico o del simulador. La ventana del BoardScope registrará la desconexión y tendrá que ser cerrada de forma independiente a la del programa de java.

Todo el proceso que hemos contado se remite a la conexión con un simulador. Como hemos reseñado en el primer párrafo del manual, la dualidad de botones en la aplicación permiten conectar tanto a un dispositivo físico o al simulador. Los botones que hemos comentado que modifican o bien copian CLBs están disponibles también en el caso de la FPGA física, con la salvedad de que no dispondremos de la aplicación BoardScope para observar estos cambios, sino que nos tendremos que valer de los LEDs de que dispone la FPGA como único método de comprobación de los procesos. De aquí que todo el grueso de la aplicación la hayamos orientado al simulador VirtexDS, haciendo siempre las comprobaciones adecuadas para que el código fuera portable y ejecutable en la Virtex 1000.

3.5. Del simulador a la realidad

Como dijimos anteriormente, todos los pasos realizados hasta ahora se han ejecutado en el simulador VirtexDS, por poseer una herramienta de debug (BoardScope) efectiva y por cuestiones de seguridad de la integridad de la placa. Ahora es el momento de ejecutar nuestro contador de prueba en el dispositivo físico.

Para ello, y como ya hemos comentado brevemente en apartados anteriores, debemos modificar la instancia de XHWIFWithEvents para adecuarla al dispositivo físico en lugar de al simulador. Sin embargo no es posible realizar esta tarea mediante el uso de dicha clase; hay que volver a utilizar la clase original XHWIF.

Pero al intentar enlazar esta clase con nuestra Virtex, nos encontramos con un problema que no se producía en el caso del simulador: la clase no reconocía nuestro dispositivo. Para conseguir que funcionase correctamente, fue necesario añadir una nueva libreria, la correspondiente al dispositivo, al directorio de compilación de nuestras clases Java. Asimismo, hubo que crear una nueva clase que representase el conjunto de nuestra Virtex con la encapsulación BG560, para que a través de la libreria pudieramos generar los métodos correctos que necesitaba la clase XHWIF. De nuevo hemos usado la clase abstracta Devices para obtener los valores asociados que necesitábamos

A continuación adjuntamos el código que representa dicha clase:

```
public class xcv1000 extends rc1000pp{
    private final static int boardrc1000pp[] = {Devices.XCV1000};
    private final static int boardrc1000ppkg[] = {Devices.BG560};


  public xcv1000() {
    devType = boardrc1000pp;
    pkgType = boardrc1000ppkg;
  }
}
```

Como reloj real optamos por usar el reloj interno del sistema. A pesar de que hubiera sido más eficiente tracear el reloj interno de forma que pudiéramos controlarlo desde la aplicación Java como en el caso del simulador, esto no fue posible debido a problemas que describiremos más en profundidad en el apartado dedicado a la entrada/salida.

Para comprobar el funcionamiento del contador, conectamos la salida de éste a los LEDS disponibles en la placa y cargamos el bitstream dentro de nuestra Virtex mediante JBits, comprobando como se producían las salidas de manera correcta

4. Abriendo caminos de investigación

4.1. Multitarea: El problema de la entrada/salida

Para establecer una arquitectura de multitarea dentro de la FPGA, es necesario establecer una estructura interna que vamos a seguir en las conexiones y en el comportamiento de las distintas tareas en ejecución.

Como primera aproximación, vamos a dividir la FPGA en cuatro columnas de igual ancho que abarquen toda la FPGA de largo. Esta idea, la más intuitiva, nos permite ejecutar sin problemas una tarea en cada columna, permitiendo multitarea. Así, cuando queremos lanzar una tarea a ejecución, simplemente leemos el contenido de la FPGA, buscamos una columna libre, y cargamos en ella el contenido de la tarea.

Para cargar la tarea, podemos valernos de un objeto JBits auxiliar que carga primeramente la tarea desde un bitstream. Posteriormente, copiaremos la tarea en el objeto Jbits que contiene la imagen de la placa dentro de la columna que nos interesa. Por ello, debemos asegurarnos que la tarea que hemos leído tiene como ancho máximo el ancho de

una columna, ya que no podemos usar recursos pertenecientes al espacio de otra tarea. Finalmente, cuando una tarea acabe su ejecución, simplemente eliminaremos la configuración de todos los recursos pertenecientes al área de su columna.

Partiendo del planteamiento anterior, aparentemente destaca la sencillez del proceso. Sin embargo, estoy seguro de que todo el mundo se habrá dado cuenta de un pequeño detalle: ¿Cómo se realiza la entrada/salida con cada una de las tareas? Existen también otros inconvenientes como la temporización o la imposibilidad de comprobar en qué momento una tarea ha acabado su ejecución; pero todos ellos no son sino variantes concretas de nuestro problema principal.

Intentando solucionar el problema, parece claro que las tareas deben competir por los recursos de la misma manera que esto ocurre en una arquitectura de propósito general. Debido a que todas las tareas pueden necesitar cualquier recurso y que a priori cualquier tarea puede cargarse en cualquier columna, es necesario establecer conexiones posibles entre cada recurso y todas las columnas. Por ello, la mejor manera de organizar la entrada/salida parece que es una estructura de bus., conectado a todas las columnas. Ahora las preguntas directas son: ¿Qué anchura debe tener ese bus? ¿Qué protocolo debemos usar para ordenar la competición de los recursos por parte de las tareas?

Un examen pormenorizado de la situación revela además algunos detalles problemáticos de esta implementación. Las tareas diseñadas deben disponer de una entrada de reloj, siendo la solución más directa conectarlas al reloj principal de la placa. Esto obliga a que todas las tareas se ejecuten a la misma velocidad. Esto aunque en principio no parece problemático ya que las tecnologías de los componentes es la misma para todas las tareas, sí puede devenir en serios problemas cuando se compliquen los circuitos, ya que dos tareas no tienen por qué tener los mismoscaminos críticos y podemos detectar fallos de ejecución. Aunque la parte de temporización se detallará más adelante, basta comentar ahora que la solución a la distinta frecuencia de reloj es colocar en cada tarea un multiplicador/divisor de frecuencia de reloj, alimentando cada tarea por separado.

Volviendo al problema de la entrada/salida, la mejor opción parece ser un bus paralelo de, digamos, 8 bits gobernado por algún tipo de árbitro. Sin embargo, el coste en superficie de semejante bus condicionaría enormemente el tamaño de cada una de las columnas de nuestro sistema, debido a la gran cantidad de CLBs que necesitaríamos únicamente para el sistema de comunicación. A esto deberíamos añadir el tamaño de las conexiones y de la lógica, lo que provoca un desperdicio de casi el 50% de la placa. Así, hemos preferido trabajar con un bus serie, reduciendo drásticamente la velocidad de la E/S pero manteniendo un consumo de recursos aceptable. Esto conlleva lógicamente que nuestro sistema de arbitraje por el control del bus sea algo más complicado, pero diseñando una línea de datos y dos de control creemos que puede funcionar perfectamente.

Partiendo de estas premisas, nos embarcamos en encontrar una solución a este problema. A pesar de que se nos ocurrieron gran cantidad de teorías o formas de arreglarlo, invirtiendo en esto casi 5 meses completos, todas las opciones presentaban gran cantidad de problemas, algunos de ellos totalmente insalvables. A continuación vamos a presentar dos de las ideas que más lejos han llegado en su planteamiento, así como sus ventajas e inconvenientes. Hay que hacer constar que el desarrollo de las líneas que vamos a explicar se hizo en paralelo, sin tener en ningún momento una idea clara de cual de ellas sería o es mejor. En las dos plantearemos la sucesión de problemas a los que nos enfrentamos, y

animamos al lector a que busque soluciones a ellos, ya que en ningún momento queremos afirmar que no existan decisiones mejores a las que nosotros tomamos en su momento.

4.2. Comunicando tareas. VHDL y JRoute

El primer modelo que consideramos fue dividir la fpga en cuatro columnas de 24 CLBs de ancho por 40 CLBs de alto, subdividiendo a su vez el espacio superior entre un pequeña parte para el árbitro y tres líneas horizontales que abarcasen toda la placa. Estas tres líneas, dos de control y una de datos, debían estar interconectadas a cada una de las columnas de forma que al cargar una tarea pudieramos conectarlas sin problemas.

Además, cada tarea dispondría de un pequeño controlador que guiase la entrada salida, mediante un protocolo de comunicación con el árbitro. Así, cuando una tarea quisiera leer (obtener una entrada) se lo diría al controlador, el cual cesaría su ejecución inhabilitando el reloj y mandaría al árbitro una señal de petición por el bus de control. Cuando el bus esté libre el árbitro mandará los datos pedidos al controlador, el cual los colocará en las señales correspondientes y habilitará el reloj de la tarea, ahora con los datos disponibles.

Desde el punto de vista de la implementación, esta idea presenta dos nuevos e interesantes problemas: por un lado, el árbitro necesita acceso a todos los recursos de la placa, para poder obtener los datos de éstos; por otro, necesitamos poder conectar las entradas y la salida de una tarea a los pines correctos del controlador cuando cargamos ésta, eliminando convenientemente las conexiones al descargarla. En cuanto al primer escollo, esto podría resolverse creando una zona envoltorio que contuviera todas las conexiones a los IOBs. No obstante, vamos a centrarnos en el segundo de los subproblemas dejando para el punto 4.3 la solución al enigma.

Con nuestra utilización de bitstreams como formato para las tareas, no podemos saber a priori donde están localizadas las entradas y las salidas de un circuito. Dado un bitstream que representa una tarea en el momento de meterla dentro de la placa, una posible solución para establecer la corresponencia sería mirar la estructura del circuito, localizar estas señales inacabadas y conectarlas manualmente a los pines correctos del controlador.

Sin embargo, resulta imposible mirando simplemente un bitstream determinar la posición de las entradas o las salidas del circuito. Si bien podría crearse un analizador que comprobase las conexiones de dicho bitstream a los IOBs, la tracease y nos diera un posible punto de intersección, resulta en la práctica muy complejo hacer esto ya que llevaría mucho tiempo y no siempre sería posible, ya que una misma señal puede estar conectada a dos elementos y tendríamos ambiguedades. Una posible opción, que además parece sencilla y funcional, sería compilar los bitstreams enlazando las entradas y las salidas a puntos prefijados de antemano, de manera que simplemente tuvieramos que unir esos puntos, ya conocidos, a los pines correspondientes del controlador.

Intentando continuar con esta idea debiamos encontrar una manera de compilar bitstreams fijando determinadas salidas a puntos concretos de la fpga, y luego la manera de unir dichos puntos con el controlador. Para ello, es necesario usar la herramienta Jroute, otra de las funcionalidades del paquete Jbits. Dicha herramienta no es sino un modificador de la estructura de un objeto Jbits que permite realizar en serie todas las modificaciones set en el mapa que conecten dos puntos dados de la placa. Jroute realiza su trabajo en función a una

lista de conexiones usadas que mantiene en memoria, en la cual se especifica si una determinada conexión esta siendo usada o si por el contrario está libre, pudiendo alterarse su estado. Apartir de esa lista y de los dos puntos a unir, realiza una búsqueda heurística de un posible camino, conectando los puntos por el medio resultante y actualizando en consecuencia la lista de enlaces dedicados anteriormente mencionada. El algoritmo heurístico usado es configurable por el programador, aceptando desde simples metodos voraces hasta una variante del conocido algoritmo A*.

Si queremos utilizar JRoute para manejar el proceso, debemos asegurarnos de que tenemos en todo momento una tabla de conexiones lista y actualizada, evitando así modificar conexiones pertenecientes a tareas en ejecución. Ya que el proceso de carga de un bitstream no crea una tabla asociada, la mejor forma es crear un objeto JBits y recorrer el bitstream en su totalidad, marcando cuales son los recursos que no podemos modificar. Aun así, y debido a que esto puede hacerse en local antes de lanzar la tarea a ejecución, esto no ralentiza demasiado el proceso.

Como restricciones de conectividad, debemos evitar que las rutas generadas salgan del espacio asignado a la tarea, en este caso una columna. Afortunadamente el código de la herramienta está diseñado de forma que admite un área de restricción para crear los caminos, por lo que esto es fácilmente solucionable. Lógicamente también puede ocurrir que el algoritmo no sea capaz de encontrar ninguna conexión entre los dos puntos, por existir una barrera de conexiones ya establecidas que lo impida. Para un uso futuro, vamos a denominar estos puntos como puntos aislados.

El proceso de uso de la herramienta es muy sencillo, y hemos constatado que funciona tremendamente bien sin tener que hacer ninguna modificación con las funciones básicas. Por ello, ahora sólo tenemos que preocuparnos por fijar las salidas de las tareas a puntos determinados

Para ello, vamos a valernos de un tipo especial de fichero de configuración de Xilinx Foundations. Los User Constraints File son ficheros especiales de condiciones que sirven para dirigir el proceso de creación de los ficheros .bit, asegurándonos de que los bitstreams resultantes cumplen una serie de condiciones. La restricción más clara que tenemos que implementar es que cualquier bitstream que creemos debe forzosamente caber en el espacio asignado a una tarea. Esto puede realizarse añadiendo al archivo una directiva de localización, como por ejemplo la siguiente:

```
INST Contador LOC=CLB_R1C1:CLB_R5C7;
```

Esta directiva, al ser leída por el compilador, fuerza a que todos los componentes menos las rutas de entrada salida estén situados en un área rectangular determinada. Las interconexiones entre componentes también se ven afectadas por esta restricción. Hay que tener en cuenta que puede ser imposible instanciar el circuito en esas dimensiones, lo que en teoría generaría un error. Sin embargo, en determinadas pruebas se ha comprobado que si no puede calcular una posición válida para el circuito existe un riesgo pequeño pero real de que el proceso siga intentándolo durante más tiempo del debido, llegando incluso a no acabar entrando en un bucle sin salida.

La segunda restricción que debemos introducir en el fichero es la relativa a las rutas de entrada y salida. Recordemos que queríamos conectarlas a un punto fijo para luego

tracearlo. Esto puede realizarse mediante la sentencia LOC, de la cual se muestra el siguiente ejemplo:

```
INST Contador/Acabado LOC = CLB_R17C36
```

Tanto esta directiva como la anterior necesitan ser usadas con los nombres que Xilinx ha proporcionado internamente a los objetos al compilar. Para conocerlos en el caso de que tengamos problemas, podemos usar la herramienta FloorPlanner proporcionada en el paquete de Xilinx.

Teóricamente ya estaría todo solucionado por lo que vamos a llevarlo a la práctica para comprobarlo Para ello construimos un sencillo contador de cuatro bits con reset asíncrono en VHDL, con una salida llamada acabado que representa el momento en que el contador contiene el número 1111 (15 en binario). Lo que queremos es enlazar dicha salida con uno de los leds de la placa dinámicamente.

```
-- Contador


library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;


entity Counter is
    port (
        CLK: in STD_LOGIC;
        DOUT: out STD_LOGIC
    );
end Counter;

architecture Counter_arch of Counter is
    signal Q, NQ : STD_LOGIC_VECTOR(3 downto 0);
begin
    -- Actualizar el estado
    SYNC_PROC: process(CLK)
    begin
        if CLK'event and CLK = '1' then
            Q <= NQ;
        end if;
    end process;

    COMB_PROC: process(Q)
    begin
        -- Estado siguiente
        NQ(0) <= not Q(0);
        NQ(1) <= Q(1) xor Q(0);
        NQ(2) <= Q(2) xor (Q(1) and Q(0));
        NQ(3) <= Q(3) xor (Q(2) and Q(1) and Q(0));

        -- Salida
        if(Q = "0011") then
           DOUT <= '1';
        else
           DOUT <= '0';
        end if;
    end process;
```

```
end Counter_arch;



-- Arquitectura con los componentes

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Task is
    port (
        CLK: in std_logic;
        DIN: out std_logic
    );
end Task;

architecture Task_arch of Task is
    component Counter
        port (
            CLK: in STD_LOGIC;
            DOUT: out STD_LOGIC
        );
    end component;
begin
    -- Contador
    U1: counter port map (CLK => CLK, DOUT => DIN);
end Task_arch;
```

Sin embargo, el primer problema nos aparece en el momento de compilación del circuito, no siendo Xilinx capaz de “parsear” correctamente nuestro fichero. Restringiendo el problema, comprobamos que la directiva de compilación que asocia la salida acabado con un CLB no es aceptada por el intérprete, a pesar de que la documentación afirma que esto es posible. Examinando ejemplos distintos a los que tenemos acceso y consultando la base de conocimiento de Xilinx (disponible en su página web) probamos a modificar nuestras ordenes comprobando que podemos enlazar sin problemas las salidas a cualquier IOB que queramos. Después de investigar durante un tiempo, parece que esta restricción es real y que ya ha sido corroborada por distintas persona; deducimos pues que el software de Xilinx solo permite tracear salidas a IOBs, no a CLBs concretos. Esto quizás puede deberse a que el objetivo del compilador es precisamente aislar al diseñador de la implementación física, por lo que no está previsto que se use a un nivel de precisión tan bajo.

Una vez descubierto esto, tenemos que derivar levemente nuestro objetivo. La primera opción que se nos ocurrió fue la creación de una serie de pines virtuales que conectasen las salidas de la tarea con las entradas del bus. La idea, explicada con profundidad en el capítulo dedicado al VHDL, consiste en desarrollar una serie de entidades conectoras en formato estructural al que se conecte la tarea real, usando la restricción de zona del fichero UCF para que se instancien en la parte superior de cada columna justo en el borde de la misma y a la altura del pin destino. De esta manera, y de una manera parecida a un mecano, los conectores encajarían y se conectarían, ya que la salida de dicho pin virtual coincidiría con la entrada de la CLB del bus.

Diseñando con cuidado todos los componentes, al llevarlo a la práctica aparecen problemas que en absolutos nos aviamos imaginado. El primero es que resulta bastante costoso en tamaño instanciar una cantidad grande de conectores, ya que la unidad mínima de asignación es el CLB. El mayor problema que tenemos es que, aunque enlacemos

correctamente un CLB del camino al CLB destino, no siempre tenemos comunicación. ¿A qué se debe esto?

De nuevo es necesario considerar los niveles de abstracción en los que estamos trabajando. Nuestro objetivo es que encajen la salida de una CLB con la entrada de otra, la cual forma el pin de acceso a nuestro controlador. Esto es cierto, y resulta exasperante comprobar que no funciona correctamente. Si bien es cierto que en ocasiones sí obtenemos el resultado correcto, en la mayor parte de las ocasiones esto no es así. Es más, resulta todavía más preocupante el hecho de que su funcionalidad es errática; funciona "a veces". Pero, si lo que estamos haciendo es siempre lo mismo, ¿cómo es posible que funcione a veces sí a veces no?

El lector supondrá (y está en lo cierto) que algo aleatorio está ocurriendo pero que no somos capaces de percibirlo al nivel que trabajamos. Esta deducción desemboca en otro largo camino de búsqueda de información en la base de conocimiento de Xilinx que nos llevó días, hasta encontrar la razón. Hasta ahora asumíamos que al colocar convenientemente mediante una directiva de área el controlador, se iba a conectar transparentemente con la CLB. El problema es que no nos hemos percatado de que, aunque en todos los diagramas las CLBs está juntas, en realidad esto no es así: existe una matriz de interconexión parecida a un switch. El problema viene porque al instanciar el pin virtual, su salida se conecta a un puerto del switch, digamos el A, mientras que el bus puede estar conectado a cualquiera de los pines del mismo, llamémoslo B. Únicamente cuando el switch comunica A y B el resultado es el correcto. Desgraciadamente no podemos controla resto simplemente con directivas de compilación.

Lamentando el tiempo perdido decidimos volver a la teoría anterior de enlace a una IOB determinada. El razonamiento es que si bien no podemos enlazar la señal a un CLB dado, si podemos enlazarla a un IOB conocido y luego, durante la instanciación de la tarea, "seguir" el cable que acaba en ese IOB hasta una CLB dentro del área de ejecución. Después cortamos la conexión por esta CLB y eliminamos el cable que llega hasta el IOB, enlazando el punto de corte con el objetivo tal y como hacíamos antes. Vamos a ilustrarlo con un ejemplo: dado el caso del contador anterior, podemos definir que la señal acabado se conecte con el IOB 1. Al instanciar la tarea, copiamos dicho circuito en un objeto Jbits temporal. A continuación miramos qué CLB perteneciente al área de la tarea está conectada con la IOB1, simplemente empezando desde ésta y siguiendo las conexiones en sentido inverso; resultando el proceso la CLB X. Ahora usamos JRoute para "borrar" la conexión entre IOB1 y X y para crear una conexión entre X y el puerto del controlador de entrada/salida que deseamos.

El principal inconveniente de esta aproximación es que el proceso de seguir el cable es muy costoso computacionalmente hablando, ya que los enlaces están definidos de origen a destino y no al revés, por lo que hay que probar todas las fuentes posibles para cada uno de los enlaces con el consiguiente gasto de tiempo. Una posible mejora sería realizar esto antes de parar la FPGA, dejando la tarea lista en el JBits auxiliar para ser copiada en el instante siguiente a obtener el estado actual de nuestra placa. Sin embargo, este tiempo, aunque no se multiplicaría en el resto de las tareas, sí debería añadirse al coste de la tarea, obviamente aumentado de manera proporcional por el número de puertos de E/S que debamos enlazar. Otra posible opción sería que el compilador del bitstream ya proporcionase las coordenadas de corte de los cables al igual que debería hacer con las entradas y salidas, bien ejecutando nuestro procedimiento de traceo o bien mirando el

desarrollo del cable a través de una herramienta de visualización como el Floorplanner anteriormente nombrado.

Salvando estos problemas de degradación del rendimiento, de nuevo parece teóricamente posible realizar esto. Probando en la práctica la configuración, decidimos enlazar la señal de acabado al IOB AL30, situado en la esquina inferior izquierda. Para simplificar, vamos a imaginar que el pin correcto del conector al que queremos enlazar la señal se encuentra en las coordenadas (40,1), es decir en la primera columna y en la fila 40, donde se termina la tarea y empezaría el bus. Utilizando la directiva NET y modificando un bitstream en abstracto, vemos que funciona perfectamente. No obstante, es sencillo percatarse de un problema que no habíamos evaluado hasta ahora: sólo puede haber un número de señales por tarea igual o menor al de IOBs conectados al hueco de esa tarea. De lo contrario, no podremos compilar un bitstream que cumpla los requisitos para luego tratarlo.

Para solventar esto podemos volver a nuestra idea de pines virtuales. De la misma manera que la sentencia NET enlaza con un IOB real, podemos instanciar un pin virtual creando la conexión en VHDL y luego cortando la conexión como si fuera un IOB. Obviamente no sería necesario copiar el contenido del pin virtual en el Jbits real que representa la placa.

Después de mucha investigación, tenemos un procedimiento que funciona: partimos de un bus que posee unas conexiones definidas por pines virtuales; es decir, sabemos que la línea del bus que conecta con el LED tiene una conexión con la entrada de la CLB X. Al recibir una tarea con un pin virtual localizado en la posición Y, la introducimos en un objeto JBits y traceamos esa conexión hasta llegar a la primera CLB del circuito. Acto seguido invocamos a JRoute para conectar la salida de dicha CLB con la entrada de la CLB X. Leemos el contenido de la placa colocándolo en un Jbits maestro e introducimos la nueva tarea copiándola. Por último, y muy importante, tenemos que usar JRoute para conectar la frontera, ya que el contenido del switch frontera no ha sido copiado al introducir la tarea. Esto debe hacerse con JRoute, ya que no tenemos la seguridad de que la configuración del switch frontera en el Jbits temporal sea la correcta en el principal.

Esta última tarea, tan simple en apariencia como realizar dos pequeñas conexiones de media CLB, es crítica y encierra una grave problemática: no podemos tener seguridad total de que JRoute no vaya a modificar una conexión que ya existiese de manera errónea. A pesar de que lleva la cuenta de las modificaciones que hace, esto no sucede con los bitstreams copiados o cargados desde Jbits. Por ello, para la herramienta ¡la FPGA está vacía! (exceptuando la franja frontera). Este problema carece totalmente de solución, y puede ocasionar en situaciones de alta densidad de uso que se tracee inadecuadamente las conexiones, disparándose esa posibilidad según aumenta el número de entradas y volviéndose crítica para un número normal de conexiones. Podemos tratar de minimizarlo aumentando la anchura del bloque frontera, con el consiguiente desperdicio de lógica. Sin embargo, aunque esto atenúa el problema no nos sirve como solución.

En la actualidad no disponemos de ninguna solución válida para este problema. Una posibilidad sería contar con un objeto JRoute actualizado, pero resulta imposible calcularlo a partir simplemente de un bitstream, necesitándose un esquemático de rutas para ello, por ejemplo de tipo EDIF. Es de suponer que Xilinx trabaja con archivos de ese tipo durante el proceso de compilación. No obstante, al tratarse de un programa comercial su

funcionamiento interno y el código de sus archivos intermedios no es accesible al público en general, en el que nos incluimos.

Resumiendo, esta línea de investigación proporciona un método teóricamente válido pero imposible de llevar a la práctica, debido a la imposibilidad de encontrar una solución al trazado de las conexiones de las rutas fronterizas. Por ello, y por los buenos resultados de la segunda línea de investigación, decidimos abandonar este camino y centrarnos en el desarrollo de las estructuras que se explican en el siguiente capítulo.

4.3. RunTime Cores: Nucleos para reprogramación dinámica

Al igual que en arquitecturas software existen estructuras de datos fijas que podemos usar para reutilizar código realizado, en el diseño electrónico actual se han desarrollado diseños estándar que nos permiten reutilizar modelos. Estas estructuras, denominadas Cores, son soportadas por los compiladores normales de VHDL, y son parcialmente soportados por el paquete JBits, hasta el punto de desarrollar un sistema de núcleos distintos. En esencia, un Core es un algoritmo que explica como, a partir de una serie de parámetros, se puede crear correctamente un circuito. El ejemplo más claro es un Core de un contador, el cual especifica cómo construirlo a partir del número de bits del dispositivo.

Para usar un Core tradicional se incluye en el modelo VHDL. Generándose el circuito mediante un sistema de producción como por ejemplo CoreGenerator. Para utilizar uno de estos cores, el diseñador invoca la herramienta y selecciona la arquitectura y el tipo de Core, introduciendo gracias a un interfaz gráfico los parámetros necesarios para elaboración del código.

CoreGenerator produce dos archivos necesarios para la integración del core tradicional con el resto de la lógica del diseño. El primer archivo es una lista de Nets (EDIF) construida a partir de los valores pasados al interfaz gráfico como parámetros. El segundo es un archivo de extensión vho que consiste en una serie de fragmentos VHDL para instanciar la lista de señales en un modelo existente. También se suele proporcionar un modelo basado en comportamiento que palía las ineficiencias producidas al simular la lista de señales a bajo nivel.

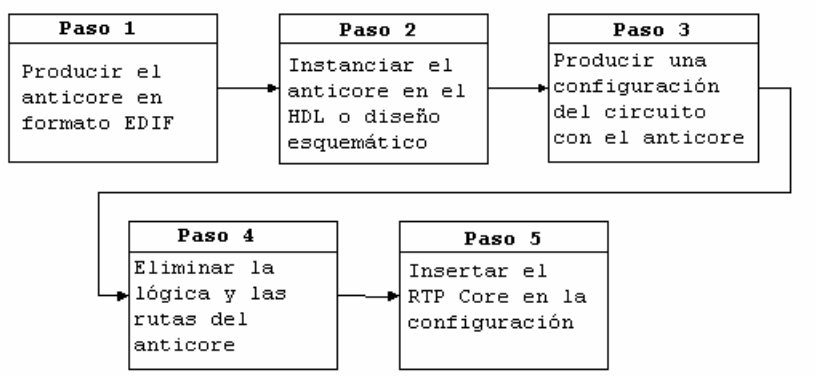
Sin embargo, no podemos usar estos diseños si queremos usar reprogramación dinámica; ya que no podemos modificar la estructura una vez que hemos instanciado el Core dentro de nuestra estructura VHDL. No obstante, JBits nos proporciona una variación de los Cores tradicionales que admite una cierta cantidad de modificaciones en tiempo de ejecución, denominados RTP Cores.

Los RTP Cores (cores parametrizables en tiempo de ejecucion), difieren como su propio nombre indica de los tradicionales en que pueden ser construidos en tiempo de ejecucion. La estructura de estos cores ha sido cuidadosamente construida de forma que soporta una eficiente y rápida implementacion en bitstreams. Adicionalmente se pueden obtener otras salidas como un edif, .vho y un xdl.

Por tanto, un RTP Core produce todas las salidas de un core tradicional (aunque el modelo de comportamiento no es obligatorio), pero la razón de su gran relevancia es su capacidad para ser reconfigurados en tiempo real.

Neftalí Mañes García
Diego Sánchez Casas
F. Javier Sánchez Jurado

Para poder usar la configuración generada por un RTP Core dentro de un modelo VHDL es necesario hacer uso de los anti-cores, no siendo posible mezclar la salida de un RTP Core con un bitstream ya existente. Este anti-core es en realidad una declaración del hueco y pines de entrada/salida que utiliza un determinado RTP Core. El proceso de construcción de los anti-cores es el siguiente:



Para permitir que la herramienta de integración construya adecuadamente el interface entre el resto del circuito y el RTP Core, el anti-core debe duplicar exactamente el interface. Teniendo en cuenta el proceso de implementación de un RTP Core es muy sencillo determinar qué pines intervienen en los puertos de entrada/salida de un core. De esta forma es posible duplicar dichos pines para su posterior uso en la construcción del bitstream.Otro problema que resuelve el anti-core es asegurar que la implementación definitiva no elimina el hueco y lo utiliza para la lógica del resto del circuito. Para ello es necesario desactivar la opción trim del proceso map.

Una vez obtenido el anti-core, hay que instanciarlo en el código VHDL mediante la declaración presente en el archivo vho. De la configuración inicial que se obtiene al generar el bistream hay que vaciar el hueco del anticore para poder implementar en tiempo real el RTP Core. Para ello se utiliza el fragmento de código que aparece en el archivo removeAntiCore.java que se crea automáticamente junto con el EDIF. De hecho, cuando creamos el bitstream el hueco delimitado por el anti-core no se crea vacío, tal y como cabría esperar; sino que se crea totalmente lleno con una densidad de circuitería muy alta para evitar que las herramientas de rutado del resto del circuito puedan usar parte del área. De ahí que sea necesario eliminar el contenido de esta zona como paso previo a la inserción del Core

La inserción del RTP Core se hace llamando al método implement de la clase. JRoute examinará la configuración inicial de las rutas existentes antes de la inserción marcando los recursos que estan en uso. Jroute puede ahora rutar la estructura interna del core sin interferir con el resto del diseño. El bitstream producido constará del RTP Core junto con el resto de la circuitería del diseño.

Esta metodología sirve exclusivamente para integrar un RTP Core dentro de un bitstream perteneciente a un diseño VHDL. El uso de los RTP Cores puede hacerse independientemente de los anti-cores implementando sobre una configuración de la FPGA conocida (posiblemente vacía). En nuestro trabajo final hemos utilizado un modelo de bus construido enteramente sobre RTP Cores programados en Java y hemos dado soporte a la compilación de tareas VHDL para que puedan adaptarse a ella.

4.4. Proceso de construcción de un RTP Core

Un RTP Core es una clase abstracta que define métodos y campos para cualquier tipo de core, siendo necesario que todos los cores tengan un nombre de instancia, que se pasa al constructor de la clase padre. Todos los subcores de un core deben tener nombres de instancia distintos. El constructor de la clase RTP Core define los puertos, alto, ancho y granularidad (en alto y ancho). El método implement() sólo usara la función addchild() en caso de que se trate de un core no primitivo, estableciendo el padre el desplazamiento en horizontal y vertical de un core. La aplicación será la encargada de establecer el desplazamiento del core del nivel superior.

Existen dos formas de crear la lógica interna de un core. La primera de ellas consiste en programar directamente las CLBs o Slices que componen el circuito y establecer las conexiones entre las distintas partes mediante pines. Este tipo de cores se denominan cores primitivos.

La segunda forma es utilizar subcores ya definidos y establecer las lineas de comunicación entre ellos. A estos cores se les llama no primitivos.

La principal diferencia entre ambas formas de construcción radica en el método implement() del core. En el primer caso se asocian pines a los puertos de entrada y de salida y se hacen llamadas a la función set de JBits. En el segundo se crean instancias de los subcores y se añaden mediante el métod addchild(). Posteriormente hay que definir el offset relativo de cada uno de ellos para situarlos dentro del área de la lógica. A los puertos de entrada y salida se les asocia una netlist en lugar de pines. asociando las señales internas del core utilizando JRoute.

Un detalle importante es que la utilización de cores sólo garantiza que la lógica está dentro de los intervalos que se especifican en getWidth() y getHeight(). Un cable de una señal interna puede salirse del área del core si JRoute lo considera necesario (y si encuentra espacio disponible).

4.5. Prototipo 2: Interfaz de gestión de multitarea

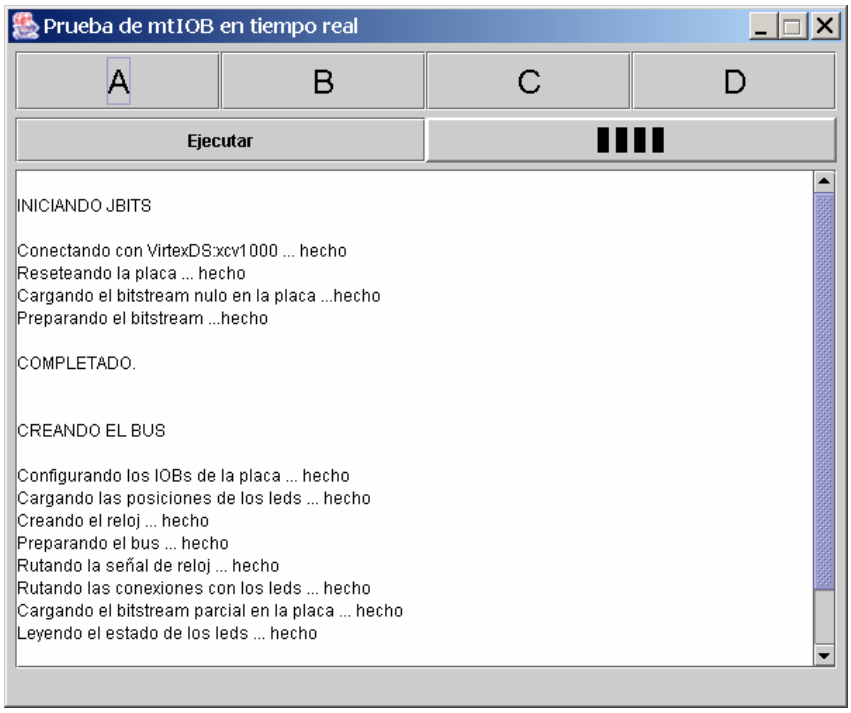
Para este prototipo hemos optado por implementar un mecanismo de comunicación sencillo en el simulador que enlaza las señales de terminado de cada tarea con un led de la placa, de forma que se puede observar el estado de ejecución en cada momento. Este bus tendrá por tanto cuatro entradas (una por cada tarea) que irán por cables independientes hasta el LED que les corresponda y una señal de reloj única que se asocia a GLK1 (reloj utilizado por el simulador BoardScope). Al implementar el circuito, tenemos que asegurarnos que podemos crear las conexiones entre los LEDs y las líneas de las tareas. La mejor forma de hacer esto es utilizar buffers con registro en la entrada del bus en las lineas de las tareas. De esta manera, siempre nos aseguramos que existe siempre un camino posible entre la tarea y el LED.

Por tanto, las tareas tienen una única entrada para el reloj y una salida que se activa en función de la implementación de la tarea. Este prototipo pretende poner de manifiesto la sencillez de uso de los cores una vez que se entienden una serie de consideraciones lógicas. Hemos diseñado una serie de tareas contador a partir de llamadas al método set(), estableciendo directamente a bajo nivel el contenido de los biestables y la forma de las

interconexiones. En esta etapa vamos a usar este código en lugar de cargar la configuración de un bitstreams, para intentar explicar mejor el manejo interno de los Cores.

Todas las tareas derivan de una clase padre llamada Task (que es una implementación abstracta del interfaz para los RTP Cores de JBits). Se han construido tareas que tienen siempre activa la señal de acabado, siempre desactivada y siempre devuelve el último valor del reloj.

Las clases que hemos diseñado se ha agrupado en una única librería de cores llamada mtiob. Para comprobar la funcionalidad de la librería construimos un interfaz gráfico compuesto por cuatro botones (uno por cada tarea a cargar) y cuatro leds (que muestran el estado en que se encuentran las distintas señales de acabado). Además incluimos un botón ejecutar para activar el reloj cada 3 segundos, de manera que se pueda observar en todo momento la evolución de la FPGA simulada. Un cuadro de texto en la parte inferior sirve para mostrar el progreso de la ejecución y los errores que se puedan producir durante la misma.



5. Buses de comunicación

5.1. ¿Por qué es necesario un bus de comunicación?

Todas las arquitecturas de computadores del mundo presentan características en común, pero quizás la más representativa de su sistema de entrada/salida sea la utilización de distintos tipos de buses. El objetivo de este modelo de interconexión es simplificar la cantidad de cables y de terminaciones, además de facilitar la temporización de los distintos dispostivos. De lo contrario, deberiamos utilizar una gran cantidad de multiplexores lógicos para distinguir fuentes y destinos; y lo que es peor, la arquitectura distaría mucho de ser escalable o modificable.

Esa misma idea subyace en la estructura que proponemos en nuestra arquitectura de tareas en la FPGA. Nuestro modelo de ejemplo va a estar, de nuevo, formado por dos contadores de 4 bits cuyas señales de salida se conectan a un LED de salida, obviamente distinto. La primera idea consiste simplementeen cablear los pines de salida de las tareas al IOB de la placa, de una manera simple y sencilla. Ahora bien, ¿qué ocurre cuando ambos contadores son idénticos, es decir, su salida se corresponde con el mismo led?

Obviamente ahora no podemos realizar la maniobra anterior, ya que cortocircuitariamos el circuito. Cuando la señal de acabado del contador A se pusiera a 1, este valor coexistiría en la misma línea con el cero lógico proveniente del contador B. Provocando en el mejor de los casos un valor indeterminado, pero llegando a crear errores en circuntancias puntuales; no especificando la placa que ocurriría en dichos casos. Además debido a que el pin del IOB es único deberíamos crear multiplexores para establecer todas las conexiones necesarias, uno por cada uno de los recursos de la placa. Para una cantidad modesta de tareas y recursos el número de estos dispositivos, así como su complejidad, se vuelve intratable y demasiado costoso en términos de área de placa.

La solución a este problema pasa por implementar un bus tradicional, aunque adaptado a nuestras necesidades. Es necesario buscar un diseño que permita la comunicación de todas las tareas con todos los dispositivos de una manera rápida, pero también que repete ciertas reglas. Al igual que pasa con todos los computadores, debemos evitar en lo posible estados de inanición o de interbloqueo, máxime cuando estos no solo nos consume tiempo, sino también una gran cantidad de espacio. No obstante, la diferencia sustancial es que en cualquier computadora tradicional el espacio que ocupen dicho bus y su lógica asociada es casi siempre un tema irrelevante, mientras que en este caso es muy importante intentar desperdiciar la mínima cantidad de área posible.

5.2. Primer intento: Comunicación total

Como primera aproximación vamos a intentar un modelo de bus en lo que llamamos comunicación total, utilizando comunicación serie por motivos antes comentados. Para ello vamos a trazar un cable que discurra horizontalmente por la parte superior del dispositivo por cada uno los posibles dispositivos de entrada o de salida, y estableciendo una conexión de cada cable con cada columna en una posición específica dependiendo del elemento.

Obviamente ahora no podemos realizar la maniobra anterior, ya que cortocircuitariamos el circuita. Cuando la señal de acabado del contador A se pusiera a 1, este valor coexistiría en la misma línea con el cero lógico proveniente del contador B. Provocando en el mejor de los casos un valor indeterminado, pero llegando a crear errores en circuntancias puntuales; no especificando la placa que ocurriría en dichos casos. Además debido a que el pin del IOB es único deberíamos crear multiplexores para establecer todas las conexiones necesarias, uno por cada uno de los recursos de la placa. Para una cantidad modesta de tareas y recursos el número de estos dispositivos, así como su complejidad, se vuelve intratable y demasiado costoso en términos de área de placa.

La solución a este problema pasa por implementar un bus tradicional, aunque adaptado a nuestras necesidades. Es necesario buscar un diseño que permita la comunicación de todas las tareas con todos los dispositivos de una manera rápida, pero también que repete ciertas reglas. Al igual que pasa con todos los computadores, debemos evitar en lo posible estados de inanición o de interbloqueo, máxime cuando estos no solo nos consume tiempo, sino también una gran cantidad de espacio. No obstante, la diferencia

sustancial es que en cualquier computadora tradicional el espacio que ocupen dicho bus y su lógica asociada es casi siempre un tema irrelevante, mientras que en este caso es muy importante intentar desperdiciar la mínima cantidad de área posible.

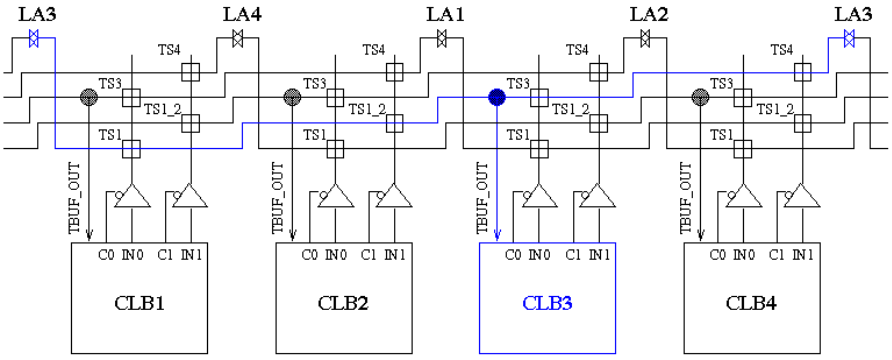
5.3. Segundo intento: Sistema de triestados

Los buses son conexiones compartidas entre muchos dispositivos caracterizados por la imposibilidad de que más de uno de ellos pueda escribir a la vez en ellos, mientras que todos pueden leer su contenido sin problemas. Para realizar esto se usan una serie de puertas de paso controladas por HW que permiten o no a un dispostuivo variar la configuración del bus. Esta serie de puertas, generalmente implementadas usando triestados, pueden adaptarse satisfactoriamente a nuestro sistema.

Nuestra placa en cuestión posee una serie de triestados en posiciones fijas en la placa y conectados a una serie de líneas globales, que discurren en horizontal por toda la placa. La opción más interesante sería poder aprovechar esos mismos triestados para crear las líneas de comunicación, ya que ahorraríamos espacio de la placa. Nuestro objetivo entonces es crear unos puntos de anclaje parecidos a los pines virtuales descritos anteriormente en nuestras tareas, forzando en el archivo ucf la instanciación de estos en la posición de los triestados.

Este proceso se realizó sin problemas, sin embargo las señales no parecían poder comunicarse correctamente. Para asegurarnos de que no se debía a un problema de control, probamos el sistema con las líneas de activación de los triestados a uno, de esa forma los triestrados debían comportarse como simples conexiones. A pesar de esto, el sistema seguía sin funcionar.

Como ya es costumbre en este proyecto, la solución la encontramos después de arduos días de investigación en la documentación de nuestra FPGA. A pesar de que no existe un modelo gráfico de la conexión de estos triestados en el manual de la XCV1000, gracias a otras fuentes pudimos conseguirlo; comprobando que la conexión de estos no era la que nosotros esperábamos. En el siguiente gráfico puede observarse la disposición real:



Como puede verse en el dibujo, las líneas de conexión no son completas, sino que presentan discontinuidades cada 4 CLBs. Además cada CLB presenta una sola conexión a las líneas de triestado de entrada.

Por ello, no es posible utilizar los triestados de la forma en que lo habiamos pensado.

5.4. Tercer intento: Reconstruir un bitstream

En vista de lo anterior, parece que ninguna de las facilidades del hardware nos sirve para solucionar nuestros problemas, ya que aunque dispongamos de un simple bus serie tenemos serios problemas para interconectar la tarea con el bus. Por ello, nos inclinamos por desarrollar nosotros mismos alguna manera de salvar dicho escollo.

Por ello, decidimos usar un core general que configura los accesos a los dispositivos físicos. Este core general recibe la funcionalidad de cada IOB en forma de un archivo ucf y crea las netlists necesarias para el proceso. Cada una de las tareas tiene puntos de acceso a los dispositivos en pines marcados, tal y como explicábamos en el capítulo anterior. Obviamente sería necesario establecer un sistema de arbitraje para evitar problemas, pero esto se discutirá en el siguiente capítulo. Ahora bien, resulta complicado conectar las tareas, ya que no podemos tracear correctamente las salidas. Esa es la causa de que desarrollasemos un sistema alternativo de cargar y tratar las tareas, que nos permita insertarlas sin problemas.

En primer lugar aprovechamos la definición de bus y tarea abstracta que se describió en el prototipo 2. Para que una tarea se cargue con la configuración específica de un bitstream hay que asegurarse que tóda la lógica del circuito se encuentra en un área determinada. Además es preciso que las entradas y salidas de dicha tarea estén localizadas en el bitstream de alguna forma. Por ello se ha utilizado el siguiente archivo UCF con el que se tienen que compilar todas las tareas VHDL para obtener los bitstreams.

```
# Delimitar la zona a usar por la tarea
CONFIG PROHIBIT = clb_r64c1:clb_r64c96;
CONFIG PROHIBIT = clb_r1c25:clb_r63c96;

# Entrada/salida
NET "CLK" LOC = "A17";  # GCK3
NET "DIN" LOC = "AN28";
```

Las directivas PROHIBIT sirven para impedir que el compilador de Xilinx situe la lógica en una extensión superior a la de una tarea (recordar que la FPGA ha sido dividida en cuatro particiones de dimensiones 24 de ancho y 63 de alto) . El reloj se conecta a GCK3 y la salida al primer IOB de la parte inferior izquierda de la placa.

En segundo lugar fue necesario crear un nuevo tipo de tarea (core) que que nos permitiera insertar un circuito modelado por un bitstream dentro de la placa conectándolo correctamente. El proceso de configuración de la lógica interna se realiza siguiendo el mecanismo de copia que vimos en el capítulo dos, es decir, replicando la configuración CLB a CLB desde un objeto auxiliar al objeto definitivo; si bien ahora el proceso ha sido refinado para ser más eficiente. Para la creación de las líneas de conexión, hemos utilizado JRoute.

El proceso se realiza trazando las conexiónes que parten de todas las salidas de las distintas CLBs en donde se supone que se encuentra la tarea. Una vez identificados los pines, se crea una netlist interna al core (en caso de que ninguno de los pines sea la IOB de salida) y luego se conectará utilizando JRoute. Para trazar la señal de reloj se sigue un

proceso aparte, en el que se asigna directamente el reloj GCK1 (para el simulador) o GCK3 (para la FPGA física).

En cuanto a las entradas y salidas del circuito, conectadas a un IOB, usamos el mismo esquema pero con destino el punto correspondiente de comunicación con el bus. Así, todos los puntos conectados a un determinado IOB se conectan a su pin asociado del bus de comunicación, creando el sistema de comunicaciones.

Para poder descargar una tarea del sistema es necesario mantener un registro de todas las conexiones que han sido realizadas en el proceso de reconstrucción del bitstream correspondiente. Este registro se obtiene utilizando el soporte del objeto global de Bitstream. Al inicio de la carga del bitstream se hace la llamada a la función startRouteRecording y al finalizar se obtiene la lista de segmentos usados con una llamada a endRouteRecording. Esta lista de segmentos se almacena en una variable hasta que la tarea se usa en la llamada Unroute al descargarse. De esta forma, borrar la tarea se reduce a desconectar las CLBs que la componen de las líneas exteriores, reloj incluido.

5.5. Sistema multitarea en una Virtex XCV1000

Uno de los mayores problemas al que nos enfrentamos en la construcción del sistema fue la adaptación del código del simulador BoardScope a nuestra FPGA física. Al final decidimos que lo mejor sería mantener dos librerías de interfaz idéntica, pero que implementasen los métodos para la Virtex XCV1000 y el simulador por separado.

Por enumerar algunas de las diferencias entre el simulador y la placa física: utilizan relojes distintos, en la FPGA física no se puede obtener el estado de una IOB ni de ningún otro PIN y la clase XHWIF es distinta según sea para la FPGA o el simulador.

A pesar de estas diferencias hemos desarrollado un interfaz único para ambos casos: la clase mtBits. Con ella podemos cargar un bitstream en un hueco libre de la FPGA (función load), eliminar una tarea de la FPGA (método clear), pasar un ciclo de reloj (step) y desconectar de la placa (close). Todos ellos son iguales independientemente de cual sea la implementación real.

La llamada al método load recibe una cadena de caracteres, con la ruta en donde se encuentra el bitstream a cargar, y la posición del hueco a ocupar, de 0 a 3. Con esta información crea un core TaskBitstream que se encarga de reconstruir el bitstream y enlazarlo con el bus.

Eliminar una tarea ya cargada se hace mediante la función clear. Está recibe como parámetro el hueco a eliminar. En la implementación que hemos realizado sólo nos preocupamos de liberar todas las conexiones en uso, de manera que la tarea se desconecta del bus.

Para poder seguir la evolución de las tareas de forma clara, elegimos usar un reloj paso a paso que creamos con la función step. De esta forma podemos controlar cuando parar una ejecución, a costa de tener un reloj irregular (debido a las transacciones del bus PCI que son necesarias).

Para impedir que la configuración de la FPGA se mantenga cuando ya se ha cerrado la aplicación hemos creado un método close que se encarga de resetear la placa y asegurarse que se ha liberado el interfaz XHWIF.

Para probar ambas librerías creamos dos interfaces gráficos iguales al prototipo 2. Sólo destacar que en el caso de la FPGA física no se muestra el estado de los LEDS en la ventana (dado que no se puede utilizar el objeto XHWIFWithEvents con la Virtex física).

6. Usos futuros

6.1. Desarrollos posteriores

A partir de lo expuesto anteriormente, es posible crear una base sólida para el desarrollo de aplicaciones o arquitecturas que presenten una ganancia de rendimiento usando multitarea HW. Debido a lo novedoso de la implementación, existen una gran cantidad de enfoques de aprovechamiento posibles, algunos de los cuales podrán ser abordados satisfactoriamente con una pequeña inversión de tiempo; mientras que la mayor parte de ellos requerirán una considerable entrega ya que no hay disponible una infraestructura amplía.

El primer paso sería desarrollar un bus de comunicación completo, siguiendo las pautas anteriormente descritas. Este bus debería tener conexiones con todos los dispositivos de E/S de la placa, especialmente con los bancos de memoria compartida. Además sería necesario desarrollar algún tipo de árbitro y de protocolo que regulase el acceso al bus y los mecanismos de comunicación. Desgraciadamente la conplejidad de ese árbitro sería bastante alta si queremos preservar unos límites de rendimiento altos, con lo que consumiríamos gran cantidad de placa. Sería necesario establecer un punto medio entre rendimiento y coste, al igual que ocurrirá en la mayor parte de los casos. En cambio, el protocolo podría ser parecido al usado por un bus tradicional, ya que las tareas podrían considerarse como periféricos en el sentido clásico de la palabra.

Otro avance interesante sería esttudiar los nuevos modelos de FPGA que están saliendo al mercado. Con una concentración mayor de transistores y una potencia más elevada, podrían ofrecer un mayor rendimiento, ya que el principal problema de este diseño es la frecuencia máxima a la que puede trabajar un circuito, motivada por la frecuencia de reloj y por la cantidad de calor que la placa puede disipar. Además, sería muy interesante estudiar si las nuevas distribuciones de hardware presentan algún tipo de facilidad estructural que se adecúe a alguno de nuestros diseños.

Mejoras del hardware aparte, es de esperar que la versión 3.0 de Jbits, actualmente utilizable en la placa Virtex II, nos permita nuevas operaciones y mejore las actuales, una vez sea ya estable. Con ella, es probable que se pueda avanzar en este camino y solucionar algunos problemas que actualmente permanecen inamovibles, o hacerlo de una forma más elegante. Además, la posibilidad de usar versiones más modernas de la máquina virtual de Java nos permitirá un mayor rendimiento, ya que la velocidad de interpretación y ejecución mejorarán notablemente.

Por último es necesario remarcar que, aunque todo este proceso de investigación es muy interesante, no sirve de nada si no se diseñan tareas o circuitos que puedan ser

ejecutados en FPGAs preparadas como las de este proyecto. Para que esta arquitectura sea productiva, es necesario que se nutra de nuestros problemas y de nuestro programas.

6.2. Carga distribuida

Como ejemplo de ampliación a nuestro trabajo, hemos preparado nuestro programa para ser tratado como cliente de una estación servidora.

La idea sería implementar una arquitectura cliente servidor donde un número ilimitado de clientes recibieran tareas por encargo del servidor, ejecutándolas y devolviendo a éste los resultados. Para ello, la aplicación debe ser capaz de cargar bitstreams remotamente, mientras que el lado servidos debe tener algún tipo de planificador que reparta el trabajo equitativamente.

En nuestro caso, hemos realizado la integración con un programa servidor denominado Condor, el cuál se encarga de supervisar la temporización y el número de recursos libres, así como de enviar los correos al cliente correcto.

En el lado de la aplicación hemos tenido que desdoblar toda la programación para que, usando hilos, se pudieran cargar varias tareas en la misma FPGA (recordemos que en el módulo de pruebas sólo una tarea era posible) Esto nos ha obligado a cambiar la temporización y a ajustar los relojes de las mismas mediante barreras software, para evitar que algunas tareas adelantasen a otras.

Como se puede observar, el conjunto funciona satisfactoriamente.

Apéndice A: Código del primer prototipo

Clase MarcoPrincipal

```
package proyectofpga;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.borland.jbcl.layout.*;
import java.io.*;
import javax.swing.border.*;
import java.util.*;
import javax.swing.event.*;

/**
 * <p>Título: </p>
 * <p>Descripción: </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Empresa: </p>
 * @author sin atribuir
 * @version 1.0
 */


public class MarcoPrincipal extends JFrame {
  JPanel contentPane;
  JMenuBar jMenuBar1 = new JMenuBar();
  JMenu jMenuFile = new JMenu();
  JMenuItem jMenuFileExit = new JMenuItem();
  JMenu jMenuHelp = new JMenu();
  JMenuItem jMenuHelpAbout = new JMenuItem();
  //Atributos propios de la aplicación
  InterfazJBits proyecto = new InterfazJBits(); // Interfaz que interactúa con la placa
  String enteros [] = {"1","2","3","4","5","6","7","8","9","10"};

  Border border1;
  Border border2;
  JPanel jPanelIzqSup = new JPanel();
  JPanel jPanelIzqInf = new JPanel();
  JPanel jPanelDrSup = new JPanel();
  JLabel jLabel1 = new JLabel();
  JButton jButtonCargarBit = new JButton();
  JButton jButtonModCont = new JButton();
  XYLayout xYLayout1 = new XYLayout();
  JButton jButtonClock = new JButton();

  JComboBox jComboBoxClock = new JComboBox(enteros);
  XYLayout xYLayout2 = new XYLayout();
  JPanel jPanel1 = new JPanel();
  Border border3;
  JPanel jPanel2 = new JPanel();
  XYLayout xYLayout3 = new XYLayout();
  XYLayout xYLayout4 = new XYLayout();
  ButtonGroup buttonGroup1 = new ButtonGroup();
  ButtonGroup buttonGroup2 = new ButtonGroup();
  ButtonGroup buttonGroup3 = new ButtonGroup();
  ButtonGroup buttonGroup5 = new ButtonGroup();
  JRadioButton jRadioButton1 = new JRadioButton();
  JRadioButton jRadioButton2 = new JRadioButton();
  JRadioButton jRadioButton3 = new JRadioButton();
  JRadioButton jRadioButton4 = new JRadioButton();
  JToggleButton jButtonDesconectar = new JToggleButton();
  JTextField jTextCLBX = new JTextField();
  JTextField jTextCLBY = new JTextField();
  JLabel jLabel2 = new JLabel();
  JLabel jLabel3 = new JLabel();
  JLabel jLabel4 = new JLabel();
  JLabel jLabel5 = new JLabel();
  JLabel jLabel6 = new JLabel();
  XYLayout xYLayout5 = new XYLayout();
  JPanel jPanel4 = new JPanel();
  XYLayout xYLayout6 = new XYLayout();
  XYLayout xYLayout8 = new XYLayout();
```

```
  JRadioButton jRadioButtonLUT = new JRadioButton();
  JRadioButton jRadioButtonBiestables = new JRadioButton();
  JRadioButton jRadioButtonEntrada = new JRadioButton();
  JRadioButton jRadioButtonSalida = new JRadioButton();
  JPanel jPanel5 = new JPanel();
  XYLayout xYLayout9 = new XYLayout();
  JTabbedPane jTabbedPane1 = new JTabbedPane();
  JPanel Luts = new JPanel();
  JTextField jTextNC = new JTextField();
  XYLayout xYLayout10 = new XYLayout();
  JLabel jLabel7 = new JLabel();
  JRadioButton jRadioButtonNC = new JRadioButton();
  JRadioButton jRadioButtonEntrada2 = new JRadioButton();
  JRadioButton jRadioButtonEntrada3 = new JRadioButton();
  JRadioButton jRadioButtonEntrada1 = new JRadioButton();
  JRadioButton jRadioButtonEntrada4 = new JRadioButton();
  private JComboBox jComboEntrada = new JComboBox();
  private JLabel jLabel8 = new JLabel();
  private JPanel Biestables = new JPanel();
  private XYLayout xYLayout7 = new XYLayout();
  private ButtonGroup buttonGroup4 = new ButtonGroup();
  private JRadioButton jRadioButtonBEntrada = new JRadioButton();
  private JRadioButton jRadioButtonBModo = new JRadioButton();
  private JRadioButton jRadioButtonBReset = new JRadioButton();
  private JComboBox jComboBoxEntrada = new JComboBox();
  private JComboBox jComboBoxModo = new JComboBox();
  private JComboBox jComboBoxReset = new JComboBox();
  JToggleButton jButtonCargarBitFis = new JToggleButton();
  JToggleButton jButtonDesconectarFis = new JToggleButton();
  JRadioButton jRadioButtonCopiarCLB = new JRadioButton();
  XYLayout xYLayout11 = new XYLayout();
  JPanel Clb = new JPanel();
  JTextField jTextFieldCLBDestX = new JTextField();
  JTextField jTextFieldCLBDestY = new JTextField();
  JLabel jLabel9 = new JLabel();
  JLabel jLabel10 = new JLabel();
  JLabel jLabel11 = new JLabel();

//Constructor del marco principal
  public MarcoPrincipal() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK); //Activamos la captura de eventos

    try {
      jbInit(); //Inicializamos los componentes del marco
    }
    catch(Exception e) { //Si no podemos inicializar, escribimos la pila y salimos
      e.printStackTrace();
      System.out.println ("Error al crear los componentes del marco principal");
      System.exit(-1);
    }
  }

  //Inicialización de componentes
  private void jbInit() throws Exception  {
    contentPane = (JPanel) this.getContentPane();
    border1 =
BorderFactory.createBevelBorder(BevelBorder.LOWERED,Color.white,Color.white,new Color(99,
99, 99),new Color(142, 142, 142));
    border2 =
BorderFactory.createBevelBorder(BevelBorder.RAISED,Color.white,Color.white,new Color(99,
99, 99),new Color(142, 142, 142));
    border3 =
BorderFactory.createBevelBorder(BevelBorder.RAISED,Color.white,Color.white,new Color(103,
101, 98),new Color(148, 145, 140));
    contentPane.setLayout(xYLayout8);
    this.setSize(new Dimension(513, 498));
    this.setTitle("Título del marco");
    jMenuFile.setText("Archivo");
    jMenuFileExit.setText("Salir");
    jMenuFileExit.addActionListener(new MarcoPrincipal_jMenuFileExit_ActionAdapter(this));
    jMenuHelp.setText("Ayuda");
    jMenuHelpAbout.setText("Acerca de");
    jMenuHelpAbout.addActionListener(new
MarcoPrincipal_jMenuHelpAbout_ActionAdapter(this));
    contentPane.setFont(new java.awt.Font("Dialog", 0, 12));
```

```
    contentPane.setOpaque(true);
    contentPane.setPreferredSize(new Dimension(600, 700));
    jPanelIzqSup.setLayout(xYLayout1);
    jLabel1.setFont(new java.awt.Font("SansSerif", 1, 13));
    jLabel1.setForeground(Color.red);
    jLabel1.setAlignmentY((float) 0.5);
    jLabel1.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel1.setHorizontalTextPosition(SwingConstants.CENTER);
    jLabel1.setText("Opciones Generales");
    jLabel1.setVerticalAlignment(SwingConstants.TOP);
    jButtonCargarBit.setFont(new java.awt.Font("SansSerif", 1, 10));
    jButtonCargarBit.setText("Cargar Bitstream Simulador");
    jButtonCargarBit.addActionListener(new
MarcoPrincipal_jButtonCargarBit_actionAdapter(this));
    jButtonModCont.setEnabled(false);
    jButtonModCont.setFont(new java.awt.Font("SansSerif", 1, 10));
    jButtonModCont.setText("Modificar Contenido");
    jButtonModCont.addActionListener(new
MarcoPrincipal_jButtonModCont_actionAdapter(this));
    jButtonClock.setText("Avanzar Reloj");
    jButtonClock.addActionListener(new MarcoPrincipal_jButtonClock_actionAdapter(this));
    jButtonClock.setEnabled(false);
    jButtonClock.setFont(new java.awt.Font("SansSerif", 1, 10));
    jComboBoxClock.setMinimumSize(new Dimension(30, 24));
    jComboBoxClock.setEditor(null);
    jComboBoxClock.setSelectedIndex(-1);
    jPanelDrSup.setLayout(xYLayout2);
    jPanel1.setBorder(border3);
    jPanel1.setLayout(xYLayout3);
    jPanel2.setBackground(SystemColor.control);
    jPanel2.setBorder(border3);
    jPanel2.setLayout(xYLayout4);
    jRadioButton1.setEnabled(true);
    jRadioButton1.setFont(new java.awt.Font("SansSerif", 1, 10));
    jRadioButton1.setFocusPainted(true);
    jRadioButton1.setSelected(true);
    jRadioButton1.setText("Bloque 0");
    jRadioButton1.addItemListener(new MarcoPrincipal_jRadioButton1_itemAdapter(this));
    jRadioButton2.setText("Bloque1");
    jRadioButton2.addItemListener(new MarcoPrincipal_jRadioButton2_itemAdapter(this));
    jRadioButton2.setFont(new java.awt.Font("SansSerif", 1, 10));
    jRadioButton2.setSelected(false);
    jRadioButton3.setText("Bloque0");
    jRadioButton3.addItemListener(new MarcoPrincipal_jRadioButton3_itemAdapter(this));
    jRadioButton3.setFont(new java.awt.Font("SansSerif", 1, 10));
    jRadioButton4.setFont(new java.awt.Font("SansSerif", 1, 10));
    jRadioButton4.setText("Bloque 1");
    jRadioButton4.addItemListener(new MarcoPrincipal_jRadioButton4_itemAdapter(this));
    jButtonDesconectar.setEnabled(false);
    jButtonDesconectar.setText("DesconectarSimulador");
    jButtonDesconectar.addActionListener(new
MarcoPrincipal_jButtonDesconectar_actionAdapter(this));
    jTextCLBX.setBackground(SystemColor.control);
    jTextCLBX.setText("0");
    jTextCLBY.setBackground(SystemColor.control);
    jTextCLBY.setText("0");
    jLabel2.setText("CLB a modificar");
    jLabel3.setText("X:");
    jLabel4.setText("Y:");
    jLabel5.setText("Slice 1");
    jLabel6.setText("Slice 0");
    jPanelIzqInf.setLayout(xYLayout5);
    jPanel4.setBackground(SystemColor.control);
    jPanel4.setLayout(xYLayout6);
    jPanelDrSup.setBackground(SystemColor.control);
    jRadioButtonLUT.setSelected(true);
    jRadioButtonLUT.setText("LUTs");
    jRadioButtonLUT.addItemListener(new MarcoPrincipal_jRadioButtonLUT_itemAdapter(this));
    jRadioButtonBiestables.setText("Biestables");
    jRadioButtonBiestables.addItemListener(new
MarcoPrincipal_jRadioButtonBiestables_itemAdapter(this));
    jRadioButtonEntrada.setText("Conexiones de entrada");
    jRadioButtonSalida.setText("Conexiones de salida");
    jPanel5.setLayout(xYLayout9);
    jPanelIzqInf.setBackground(SystemColor.control);
```

```
    jPanelIzqSup.setBackground(SystemColor.control);
    Luts.setLayout(xYLayout10);
    jLabel7.setText("Funcion logica");
    jRadioButtonNC.setText("Salida");
    jTextNC.setText("");
    jRadioButtonEntrada2.setText("Entrada 2");
    jRadioButtonEntrada2.addItemListener(new
MarcoPrincipal_jRadioButtonEntrada2_itemAdapter(this));
    jRadioButtonEntrada3.setText("Entrada 3");
    jRadioButtonEntrada3.addItemListener(new
MarcoPrincipal_jRadioButtonEntrada3_itemAdapter(this));
    jRadioButtonEntrada1.setText("Entrada 1");
    jRadioButtonEntrada1.addItemListener(new
MarcoPrincipal_jRadioButtonEntrada1_itemAdapter(this));
    jRadioButtonEntrada4.setText("Entrada 4");
    jRadioButtonEntrada4.addItemListener(new
MarcoPrincipal_jRadioButtonEntrada4_itemAdapter(this));
    jComboEntrada.setEditor(null);
    jLabel8.setText("Recurso asociado");
    Biestables.setLayout(xYLayout7);
    jRadioButtonBEntrada.setText("Entrada");
    jRadioButtonBModo.setText("Modo");
    jRadioButtonBReset.setText("Reset");
    jTabbedPane1.addChangeListener(new MarcoPrincipal_jTabbedPane1_changeAdapter(this));
    jButtonCargarBitFis.addActionListener(new
MarcoPrincipal_jButtonCargarBitFis_actionAdapter(this));
    jButtonCargarBitFis.setEnabled(true);
    jButtonCargarBitFis.setText("Cargar BitStream Placa");
    jButtonCargarBitFis.addActionListener(new
MarcoPrincipal_jButtonCargarBitFis_actionAdapter(this));
    jButtonDesconectarFis.setEnabled(false);
    jButtonDesconectarFis.setToolTipText("");
    jButtonDesconectarFis.setText("Desconectar FPGA");
    jButtonDesconectarFis.addActionListener(new
MarcoPrincipal_jButtonDesconectarFis_actionAdapter(this));
    jButtonDesconectarFis.addActionListener(new
MarcoPrincipal_jButtonDesconectarFis_actionAdapter(this));
    jRadioButtonCopiarCLB.setActionCommand("Copiar CLB");
    jRadioButtonCopiarCLB.setText("Copiar Slice");
    jRadioButtonCopiarCLB.addItemListener(new
MarcoPrincipal_jRadioButtonCopiarClb_itemAdapter(this));
    Clb.setLayout(xYLayout11);
    jLabel9.setText("X:");
    jLabel10.setText("Y:");
    jTextFieldCLBDestX.setText("");
    jLabel11.setText("CLB destino");
    jTextFieldCLBDestY.setText("");
    jMenuFile.add(jMenuFileExit);
    jMenuHelp.add(jMenuHelpAbout);
    jMenuBar1.add(jMenuFile);
    jMenuBar1.add(jMenuHelp);
    contentPane.add(jPanelIzqSup,  new XYConstraints(0, 0, 230, 248));
    jPanelIzqSup.add(jLabel1,         new XYConstraints(0, 5, 140, 20));
    jPanelIzqSup.add(jButtonCargarBit,          new XYConstraints(5, 28, 145, 24));
    jPanelIzqSup.add(jButtonModCont,          new XYConstraints(5, 56, 145, 24));
    jPanelIzqSup.add(jComboBoxClock,   new XYConstraints(110, 84, 40, 24));
    jPanelIzqSup.add(jButtonClock,  new XYConstraints(4, 85, 104, 24));
    jPanel1.add(jRadioButton2, new XYConstraints(4, 53, -1, -1));
    jPanel1.add(jRadioButton1,  new XYConstraints(5, 29, -1, -1));
    jPanel1.add(jLabel5,         new XYConstraints(16, 2, 79, 20));
    jPanelDrSup.add(jPanel4, new XYConstraints(5, 19, 206, 65));
    jPanelIzqSup.add(jButtonDesconectar, new XYConstraints(4, 114, -1, -1));
    jPanelIzqSup.add(jButtonCargarBitFis, new XYConstraints(5, 153, -1, -1));
    jPanelIzqSup.add(jButtonDesconectarFis,  new XYConstraints(4, 181, 143, -1));
    jPanel5.add(jRadioButtonLUT, new XYConstraints(1, 5, -1, -1));
    jPanel5.add(jRadioButtonBiestables, new XYConstraints(1, 33, -1, -1));
    jPanel5.add(jRadioButtonSalida,  new XYConstraints(1, 83, -1, 35));
    jPanel5.add(jRadioButtonCopiarCLB,    new XYConstraints(2, 110, -1, 35));
    jPanel5.add(jRadioButtonEntrada,    new XYConstraints(2, 63, 156, -1));
    contentPane.add(jPanelDrSup,    new XYConstraints(256, 0, 231, 213));
    jPanelIzqInf.add(jTabbedPane1,      new XYConstraints(191, 17, 312, 222));
    jPanel2.add(jRadioButton4,  new XYConstraints(1, 49, -1, -1));
    jPanel2.add(jRadioButton3, new XYConstraints(1, 28, -1, -1));
    jPanel2.add(jLabel6,    new XYConstraints(7, 5, 72, 18));
    jPanelDrSup.add(jPanel1, new XYConstraints(3, 84, 104, 93));
```

```
    jPanelDrSup.add(jPanel2, new XYConstraints(117, 85, 104, 93));
    jPanel4.add(jLabel2, new XYConstraints(5, 4, 99, 18));
    jPanel4.add(jLabel3, new XYConstraints(8, 26, 21, 21));
    jPanel4.add(jTextCLBX, new XYConstraints(27, 26, 28, -1));
    jPanel4.add(jTextCLBY, new XYConstraints(105, 24, 36, 26));
    jPanel4.add(jLabel4, new XYConstraints(85, 28, 13, 18));
    contentPane.add(jPanelIzqInf,  new XYConstraints(0, 247, 510, 248));
    jPanelIzqInf.add(jPanel5,  new XYConstraints(16, 30, -1, 202));
    this.setJMenuBar(jMenuBar1);


    //cosas Nuestras

     buttonGroup1.add(jRadioButton1);
     buttonGroup1.add(jRadioButton2);
     buttonGroup1.add(jRadioButton3);
     buttonGroup1.add(jRadioButton4);

     buttonGroup2.add(jRadioButtonLUT);
     buttonGroup2.add(jRadioButtonBiestables);
     buttonGroup2.add(jRadioButtonEntrada);
     buttonGroup2.add(jRadioButtonSalida);
    buttonGroup2.add(jRadioButtonCopiarCLB);

     Luts.add(jRadioButtonEntrada2, new XYConstraints(0, 28, 94, 16));
    Luts.add(jRadioButtonEntrada1, new XYConstraints(0, 4, 94, 16));
     Luts.add(jRadioButtonEntrada4,   new XYConstraints(0, 80, 94, 16));
     Luts.add(jRadioButtonEntrada3, new XYConstraints(1, 55, 94, 16));
     Luts.add(jRadioButtonNC, new XYConstraints(2, 141, 114, 20));
     Luts.add(jLabel7, new XYConstraints(3, 164, 110, 21));
     Luts.add(jTextNC, new XYConstraints(103, 165, 169, -1));
     Luts.add(jComboEntrada, new XYConstraints(101, 76, 172, 23));
     Luts.add(jLabel8, new XYConstraints(100, 54, 109, 16));
    jTabbedPane1.add(Biestables, "Biestables");
    Biestables.add(jRadioButtonBEntrada, new XYConstraints(11, 20, 91, 17));
    Biestables.add(jRadioButtonBReset, new XYConstraints(13, 120, 70, 16));
    Biestables.add(jRadioButtonBModo, new XYConstraints(10, 67, 96, 16));
    Biestables.add(jComboBoxEntrada,        new XYConstraints(29, 38, 163, 20));
    Biestables.add(jComboBoxModo,   new XYConstraints(29, 87, 163, 20));
    Biestables.add(jComboBoxReset,    new XYConstraints(29, 141, 163, 20));
    jTabbedPane1.add(Luts, "Luts");
    jTabbedPane1.setSelectedComponent(Luts);

    buttonGroup3.add(jRadioButtonEntrada3);
    buttonGroup3.add(jRadioButtonEntrada2);
    buttonGroup3.add(jRadioButtonEntrada1);
    buttonGroup3.add(jRadioButtonEntrada4);
    buttonGroup3.add(jRadioButtonNC);

    buttonGroup4.add(jRadioButtonBEntrada);
    buttonGroup4.add(jRadioButtonBModo);
    buttonGroup4.add(jRadioButtonBReset);




    jComboBoxModo.addItem("Flip-Flop");
    jComboBoxModo.addItem("Latch");
    jComboBoxReset.addItem("Asincrono");
    jComboBoxReset.addItem("Sincrono");
    jTabbedPane1.add(Clb,   "Clb");
    Clb.add(jTextFieldCLBDestX,  new XYConstraints(38, 39, 31, 23));
    Clb.add(jTextFieldCLBDestY,    new XYConstraints(113, 39, 34, 26));
    Clb.add(jLabel9, new XYConstraints(14, 42, 21, 17));
    Clb.add(jLabel10, new XYConstraints(86, 42, 18, 16));
    Clb.add(jLabel11,  new XYConstraints(11, 11, 137, 23));
  }

  //Realizar Archivo | Salir
  public void jMenuFileExit_actionPerformed(ActionEvent e) {
    System.exit(0);
  }

  //Realizar Ayuda | Acerca de
  public void jMenuHelpAbout_actionPerformed(ActionEvent e) {
```

```
  }

  //Modificado para poder salir cuando se cierra la ventana
  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      jMenuFileExit_actionPerformed(null);
    }
  }

  void jButtonModCont_actionPerformed(ActionEvent e) {
     int clbx,clby, bloque = 0, entrada = 0, objetivo = 0;

     clbx=Integer.parseInt(jTextCLBX.getText());
     clby=Integer.parseInt(jTextCLBY.getText());

     // Bloque seleccionado
     if (jRadioButton1.isSelected()){bloque=1;}
     else if (jRadioButton2.isSelected()){bloque=2;}
     else if (jRadioButton3.isSelected()){bloque=3;}
     else if (jRadioButton4.isSelected()){bloque=4;}

     if(jRadioButtonLUT.isSelected())
     // Modificamos una LUT, ya sea su contenido o sus conexiones
     {
       if (jRadioButtonNC.isSelected())
       {
         // Modificar el contenido de la LUT
         proyecto.modificarContenidoLUT(clbx,clby,bloque,jTextNC.getText());
       }
       else
       {
         // Entrada seleccionada
         if (jRadioButtonEntrada1.isSelected()){entrada = 1;}
         else if (jRadioButtonEntrada2.isSelected()){entrada = 2;}
         else if (jRadioButtonEntrada3.isSelected()){entrada = 3;}
         else if (jRadioButtonEntrada4.isSelected()){entrada = 4;}

         // Recurso asociado a la entrada
         objetivo = jComboEntrada.getSelectedIndex();

         // Modificar la lut
         proyecto.modificarEntradaLUT(clbx,clby,bloque,entrada,objetivo);
       }
     }
     else if(jRadioButtonBiestables.isSelected())
     {
       if(jRadioButtonBModo.isSelected())
       {
         // Cambiar el modo del flip-flop
         proyecto.modificarModoBiestable(clbx,clby,bloque,
jComboBoxModo.getSelectedIndex());
       }
       else if(jRadioButtonBReset.isSelected())
       {
         // Cambiar el modo del reset

proyecto.modificarResetBiestable(clbx,clby,bloque,jComboBoxReset.getSelectedIndex());
       }
       else if(jRadioButtonBEntrada.isSelected())
       {
         // Cambiar la entrada del biestable
       }
     }
     else if (jRadioButtonCopiarCLB.isSelected())
     {
       //Copiamos una slice completa
       // Leemos los datos
       int clbxdest=Integer.parseInt(jTextFieldCLBDestX.getText());
       int clbydest=Integer.parseInt(jTextFieldCLBDestY.getText());

       //Realizamos la copia
       proyecto.copiarCLB(clbx,clby,clbxdest,clbydest);
     }
     else
```

```
    {
      System.out.println("Error");
    }
}

void jButtonCargarBit_actionPerformed(ActionEvent e) {
  System.out.println("procedemos a cargar el bitstream en el BoardScope");
  JFileChooser cargaArchivo = new JFileChooser();
  if (cargaArchivo.showOpenDialog(null) == JFileChooser.APPROVE_OPTION)
    proyecto.cargarBitStreamSimulador(cargaArchivo.getSelectedFile().getName());

  jButtonClock.setEnabled(true);
  jButtonDesconectar.setEnabled(true);
  jButtonModCont.setEnabled(true);
  jButtonCargarBit.setEnabled(false);
}

void jButtonClock_actionPerformed(ActionEvent e) {

  proyecto.avanzarReloj(jComboBoxClock.getSelectedIndex()+1);
}

void jButtonDesconectar_actionPerformed(ActionEvent e) {
  proyecto.desconectarSimulador();
  jButtonClock.setEnabled(false);
  jButtonDesconectar.setEnabled(false);
  jButtonModCont.setEnabled(false);
  jButtonCargarBit.setEnabled(true);
}

void jRadioButtonEntrada1_itemStateChanged(ItemEvent e) {
  Actualizar_jCombos();
}

void jRadioButtonEntrada2_itemStateChanged(ItemEvent e) {
  Actualizar_jCombos();
}

void jRadioButtonEntrada3_itemStateChanged(ItemEvent e) {
  Actualizar_jCombos();
}

void jRadioButtonEntrada4_itemStateChanged(ItemEvent e) {
  Actualizar_jCombos();
}

void jRadioButton1_itemStateChanged(ItemEvent e) {
  Actualizar_jCombos();
}

void jRadioButton2_itemStateChanged(ItemEvent e) {
  Actualizar_jCombos();
}

void jRadioButton4_itemStateChanged(ItemEvent e) {
  Actualizar_jCombos();
}

void jRadioButton3_itemStateChanged(ItemEvent e) {
  Actualizar_jCombos();
}

// Actualizar el contenido del combobox de constantes para las entradas
void Actualizar_jCombos()
{
  int bloque = 0;
  int entrada = 0;
  int i;
  String[] constantes = null;

  // Ver que bloque y entrada estan seleccionados
  if (jRadioButton1.isSelected()){bloque=1;}
  if (jRadioButton2.isSelected()){bloque=2;}
  if (jRadioButton3.isSelected()){bloque=3;}
  if (jRadioButton4.isSelected()){bloque=4;}
```

```
    if (jRadioButtonEntrada1.isSelected()){entrada = 1;};
    if (jRadioButtonEntrada2.isSelected()){entrada = 2;};
    if (jRadioButtonEntrada3.isSelected()){entrada = 3;};
    if (jRadioButtonEntrada4.isSelected()){entrada = 4;};

    if(bloque != 0)
    {
      // Obtener la lista de constantes aplicables a la entrada del biestable del bloque
seleccionado
      constantes = proyecto.getRecursosEntradaBiestable(bloque);

      // Borrar todas las entradas anteriores
      jComboBoxEntrada.removeAllItems();

      // Insertar la nueva lista
      for(i = 0; i < constantes.length; i++)
        jComboBoxEntrada.addItem(constantes[i]);

      if(entrada != 0)
      {
        // Obtener la lista de constantes aplicables a la entrada de la LUT seleccionada
        constantes = proyecto.getRecursosEntradasLut(entrada,bloque);

        // Borrar todas las entradas anteriores
        jComboEntrada.removeAllItems();

        // Insertar la nueva lista
        for(i = 0; i < constantes.length; i++)
          jComboEntrada.addItem(constantes[i]);
      }
      else
      {
        // Borrar todas las entradas anteriores
        jComboEntrada.removeAllItems();
      }
    }
    else
    {
      // Borrar todas las combos
      jComboEntrada.removeAllItems();
      jComboBoxEntrada.removeAllItems();
    }
  }

  void jRadioButtonLUT_itemStateChanged(ItemEvent e) {
    if(jRadioButtonLUT.isSelected() == true && jTabbedPane1.getSelectedComponent() != Luts)
      jTabbedPane1.setSelectedComponent(Luts);
  }

  void jRadioButtonBiestables_itemStateChanged(ItemEvent e) {
    if(jRadioButtonBiestables.isSelected() == true && jTabbedPane1.getSelectedComponent()
!= Biestables)
      jTabbedPane1.setSelectedComponent(Biestables);
  }

  void jRadioButtonCopiarClb_itemStateChanged(ItemEvent e) {
    if(jRadioButtonCopiarCLB.isSelected() == true && jTabbedPane1.getSelectedComponent() !=
Clb)
      jTabbedPane1.setSelectedComponent(Clb);
  }

  void jTabbedPane1_stateChanged(ChangeEvent e) {
    if(jTabbedPane1.getSelectedComponent() == Luts)
    {
        if(!jRadioButtonLUT.isSelected())
          jRadioButtonLUT.setSelected(true);
    }
    else if (jTabbedPane1.getSelectedComponent() == Biestables)
    {
        if(!jRadioButtonBiestables.isSelected())
          jRadioButtonBiestables.setSelected(true);
    }
    else if (jTabbedPane1.getSelectedComponent() == Clb)
    {
```

```
        if(!jRadioButtonCopiarCLB.isSelected())
          jRadioButtonCopiarCLB.setSelected(true);
    }
  }

  void jToggleButton1_actionPerformed(ActionEvent e) {
    System.out.println("procedemos a cargar el bitstream en el BoardScope");
       JFileChooser cargaArchivo = new JFileChooser();
       if (cargaArchivo.showOpenDialog(null) == JFileChooser.APPROVE_OPTION)
         proyecto.cargarBitStream(cargaArchivo.getSelectedFile().getName());
       jButtonClock.setEnabled(true);
       jButtonDesconectar.setEnabled(true);
       jButtonModCont.setEnabled(true);
       jButtonCargarBit.setEnabled(false);

  }

  void jToggleButton2_actionPerformed(ActionEvent e) {
    proyecto.desconectarFPGA();

  }

  void jButtonCargarBitFis_actionPerformed(ActionEvent e) {
    System.out.println("procedemos a cargar el bitstream en la FPGA física");
       JFileChooser cargaArchivo = new JFileChooser();
       if (cargaArchivo.showOpenDialog(null) == JFileChooser.APPROVE_OPTION)
         proyecto.cargarBitStream(cargaArchivo.getSelectedFile().getName());
       jButtonClock.setEnabled(true);
       jButtonDesconectarFis.setEnabled(true);
       jButtonModCont.setEnabled(true);
       jButtonCargarBitFis.setEnabled(false);
  }

  void jButtonDesconectarFis_actionPerformed(ActionEvent e) {
    proyecto.desconectarFPGA();
  }
}

class MarcoPrincipal_jMenuFileExit_ActionAdapter implements ActionListener {
  MarcoPrincipal adaptee;

  MarcoPrincipal_jMenuFileExit_ActionAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuFileExit_actionPerformed(e);
  }
}

class MarcoPrincipal_jMenuHelpAbout_ActionAdapter implements ActionListener {
  MarcoPrincipal adaptee;

  MarcoPrincipal_jMenuHelpAbout_ActionAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuHelpAbout_actionPerformed(e);
  }
}

class MarcoPrincipal_jButtonModCont_actionAdapter implements java.awt.event.ActionListener
{
  MarcoPrincipal adaptee;

  MarcoPrincipal_jButtonModCont_actionAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jButtonModCont_actionPerformed(e);
  }
}

class MarcoPrincipal_jButtonCargarBit_actionAdapter implements
java.awt.event.ActionListener {
  MarcoPrincipal adaptee;
```

```
  MarcoPrincipal_jButtonCargarBit_actionAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jButtonCargarBit_actionPerformed(e);
  }
}

class MarcoPrincipal_jButtonClock_actionAdapter implements java.awt.event.ActionListener {
  MarcoPrincipal adaptee;

  MarcoPrincipal_jButtonClock_actionAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jButtonClock_actionPerformed(e);
  }
}

class MarcoPrincipal_jButtonDesconectar_actionAdapter implements
java.awt.event.ActionListener {
  MarcoPrincipal adaptee;

  MarcoPrincipal_jButtonDesconectar_actionAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jButtonDesconectar_actionPerformed(e);
  }
}

class MarcoPrincipal_jRadioButtonEntrada1_itemAdapter implements
java.awt.event.ItemListener {
  private MarcoPrincipal adaptee;

  MarcoPrincipal_jRadioButtonEntrada1_itemAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void itemStateChanged(ItemEvent e) {
    adaptee.jRadioButtonEntrada1_itemStateChanged(e);
  }
}

class MarcoPrincipal_jRadioButtonEntrada2_itemAdapter implements
java.awt.event.ItemListener {
  private MarcoPrincipal adaptee;

  MarcoPrincipal_jRadioButtonEntrada2_itemAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void itemStateChanged(ItemEvent e) {
    adaptee.jRadioButtonEntrada2_itemStateChanged(e);
  }
}

class MarcoPrincipal_jRadioButtonEntrada3_itemAdapter implements
java.awt.event.ItemListener {
  private MarcoPrincipal adaptee;

  MarcoPrincipal_jRadioButtonEntrada3_itemAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void itemStateChanged(ItemEvent e) {
    adaptee.jRadioButtonEntrada3_itemStateChanged(e);
  }
}

class MarcoPrincipal_jRadioButtonEntrada4_itemAdapter implements
java.awt.event.ItemListener {
  private MarcoPrincipal adaptee;

  MarcoPrincipal_jRadioButtonEntrada4_itemAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
```

```
  public void itemStateChanged(ItemEvent e) {
    adaptee.jRadioButtonEntrada4_itemStateChanged(e);
  }
}

class MarcoPrincipal_jRadioButton1_itemAdapter implements java.awt.event.ItemListener {
  private MarcoPrincipal adaptee;

  MarcoPrincipal_jRadioButton1_itemAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void itemStateChanged(ItemEvent e) {
    adaptee.jRadioButton1_itemStateChanged(e);
  }
}

class MarcoPrincipal_jRadioButton2_itemAdapter implements java.awt.event.ItemListener {
  private MarcoPrincipal adaptee;

  MarcoPrincipal_jRadioButton2_itemAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void itemStateChanged(ItemEvent e) {
    adaptee.jRadioButton2_itemStateChanged(e);
  }
}

class MarcoPrincipal_jRadioButton4_itemAdapter implements java.awt.event.ItemListener {
  private MarcoPrincipal adaptee;

  MarcoPrincipal_jRadioButton4_itemAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void itemStateChanged(ItemEvent e) {
    adaptee.jRadioButton4_itemStateChanged(e);
  }
}

class MarcoPrincipal_jRadioButton3_itemAdapter implements java.awt.event.ItemListener {
  private MarcoPrincipal adaptee;

  MarcoPrincipal_jRadioButton3_itemAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void itemStateChanged(ItemEvent e) {
    adaptee.jRadioButton3_itemStateChanged(e);
  }
}

class MarcoPrincipal_jRadioButtonLUT_itemAdapter implements java.awt.event.ItemListener {
  private MarcoPrincipal adaptee;

  MarcoPrincipal_jRadioButtonLUT_itemAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void itemStateChanged(ItemEvent e) {
    adaptee.jRadioButtonLUT_itemStateChanged(e);
  }
}

class MarcoPrincipal_jRadioButtonBiestables_itemAdapter implements
java.awt.event.ItemListener {
  private MarcoPrincipal adaptee;

  MarcoPrincipal_jRadioButtonBiestables_itemAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void itemStateChanged(ItemEvent e) {
    adaptee.jRadioButtonBiestables_itemStateChanged(e);
  }
}

class MarcoPrincipal_jTabbedPane1_changeAdapter implements javax.swing.event.ChangeListener
{
  private MarcoPrincipal adaptee;
```

```
  MarcoPrincipal_jTabbedPane1_changeAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void stateChanged(ChangeEvent e) {
    adaptee.jTabbedPane1_stateChanged(e);
  }
}

class MarcoPrincipal_jButtonCargarBitFis_actionAdapter implements
java.awt.event.ActionListener {
  MarcoPrincipal adaptee;

  MarcoPrincipal_jButtonCargarBitFis_actionAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jButtonCargarBitFis_actionPerformed(e);
  }
}

class MarcoPrincipal_jButtonDesconectarFis_actionAdapter implements
java.awt.event.ActionListener {
  MarcoPrincipal adaptee;

  MarcoPrincipal_jButtonDesconectarFis_actionAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jButtonDesconectarFis_actionPerformed(e);
  }
}

class MarcoPrincipal_jRadioButtonCopiarClb_itemAdapter implements
java.awt.event.ItemListener {
  private MarcoPrincipal adaptee;

  MarcoPrincipal_jRadioButtonCopiarClb_itemAdapter(MarcoPrincipal adaptee) {
    this.adaptee = adaptee;
  }
  public void itemStateChanged(ItemEvent e) {
    adaptee.jRadioButtonCopiarClb_itemStateChanged(e);
  }
}
```

Clase InterfazJBits

```
package proyectofpga;

/**
 * <p>Título: </p>
 * <p>Descripción: </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Empresa: </p>
 * @author sin atribuir
 * @version 1.0
 */
import java.lang.Object;
import java.io.IOException;
import java.io.FileNotFoundException;
import com.xilinx.JBits.Virtex.Bitstream;
import com.xilinx.JBits.Virtex.Devices;
import com.xilinx.JBits.Virtex.JBits;
import com.xilinx.JBits.Virtex.Util;
import com.xilinx.JBits.Virtex.Bits.S0F1;
import com.xilinx.JBits.Virtex.Expr;
import com.xilinx.JBits.Virtex.ConfigurationException;
import com.xilinx.XHWIF.*;
import com.xilinx.gui.boardscope.*;
import com.xilinx.DDTScript.*;
import com.xilinx.JBits.Virtex.ReadbackCommand;
import com.xilinx.JBits.Virtex.Bits.*;
```

```
public class InterfazJBits  {

  public JBits miJbit; // Objeto Jbits para la reconfiguracion parcial
  public XHWIF fpga;   // FPGA real
  public XHWIFConnection con;  // Conexion con la FPGA virtual
  public XHWIFWithEvents fpgaExt; // FPGA virtual

  public InterfazJBits() {
  }

  public void cargarBitStream(String nombreBitStream) {

    try {
        miJbit = new JBits(Devices.XCV1000);
      //leemos el bitstream y lo cargamos en un objeto JBits
      miJbit.readPartial(nombreBitStream);
      //Cargamos el interfaz XHWIF que nos permite manejar la placa física
      fpga = XHWIF.Get("rc1000pp:xcv1000");
      int dispositivo;
      dispositivo = fpga.connect();
      if (dispositivo<0) {System.out.println("Dispositivo no encontrado");}
      else {System.out.println("Conectado al dispositivo numero:" + dispositivo);}

      //reseteamos la fpga
      fpga.reset();
      System.out.println("la fpga se ha reseteado correctamente");


      //descargamos el bitstream en la FPGA física
      int resultado = fpga.setConfiguration(0, miJbit.getPartial());
      System.out.println("parece que hemos cargado el bitstream en la fpga fisica");
    }
    catch (FileNotFoundException fe) {
      System.out.println("File " + nombreBitStream + " not found");
      System.out.println(fe);
    }
    catch (IOException io) {
      System.out.println("Error reading the bitstream file");
      System.out.println(io);
    }
    catch (ConfigurationException ce) {
      System.out.println("Internal error detected in the Bitstream");
      System.out.println(ce);
    }
  }

  public void avanzarReloj(int numCiclos) {
    try {
      //con.getXHWIFWithEvents().clockStep(numCiclos);
      System.out.println(numCiclos);
      fpgaExt.clockStep(numCiclos);
    }
    catch (XHWIFException xe) {
      System.out.println ("algo falla al contar ciclos de reloj");
      System.exit(-1);
    }
  }

  public String mostrarVector(int[] vector, int longitud){
    String resultado ="";
    int i;
    for (i=0;i<longitud;i++){
      resultado=resultado + Integer.toString(vector[i],10);
    }
//    System.out.println(resultado);
    return resultado;
  }

  private void leerContenido(){
    byte ordendelectura [] = null;   //variable q contiene la orden de lectura
    byte contenido [] = null;  //variable q contiene el codigo leido
    int longitud;
    try {
      ordendelectura = ReadbackCommand.getClbConfig(Devices.XCV1000);
    }
```

```
    catch (ConfigurationException e) {
      System.out.println("fallo al configurar el array de lectura con ReadbackCommand");
      System.exit(-1);
    }

    longitud = ReadbackCommand.getReadLength() * 4; //numero de bits q debemos leer

    //obtenemos del objeto XHWIFConnection un objeto XHWIFEvents, q casualmente
    //posee los mismos métodos q el objeto XHWIF a secas y que nos permite enviar comandos
    //a la placa (simulador)

    try {
      fpgaExt.setConfiguration(0,ordendelectura); //mandamos a la fpga la orden de lectura
      contenido = fpgaExt.getConfiguration(0, longitud); //leemos la configuracion actual

    }
    catch (XHWIFException xe) {
      System.out.println("Problemas al leer de la Fpga virtual");
      System.exit(-1);
    }
    try {
      miJbit.parsePartial(ordendelectura, contenido); //parseamos el contenido dentro del
jbtis
    }
    catch (ConfigurationException e) {
      System.out.println("el objeto jbits no puede parsear los datos leidos");
      System.exit(-1);
    }
  }


  public void modificarContenidoLUT(int clbx, int clby, int bloque, String nc) {
    this.leerContenido();
    //modificamos el contenido del bloque seleccionado

    // MODIFICAR CONTENIDO LUT: Seleccionamos la LUT a modificar
    int[][] lutmod=null;
    System.out.println("Modificando el contenido del bloque:" + bloque);
    switch (bloque){
      case 1: lutmod= LUT.SLICE1_G;
        break;
      case 2: lutmod= LUT.SLICE1_F;
        break;
      case 3: lutmod= LUT.SLICE0_G;
        break;
      case 4: lutmod= LUT.SLICE0_F;
        break;
    }
    // Modificamos el contenido de la LUT
    int[] lut=null;
    try{
      lut = miJbit.get(clbx,clby,lutmod);
    }
    catch(Exception e){
      System.out.println ("No podemos leer el valor de la LUT leido por Jbits");
    }
    System.out.println("Valor antiguo: " + mostrarVector(lut,16));


    try {
      int[] expr=null;
      if ((bloque==2) || (bloque==4)){
        expr=Expr.F_LUT(nc);
      }
      else{
        expr=Expr.G_LUT(nc);
      }
      miJbit.set( clby, clbx, lutmod,expr);
    }
    catch (ConfigurationException e) {
      System.out.println("jbits no puede modificar el bitstream");
      System.exit(-1);
    }

    try{
```

```
      lut= miJbit.get(clby,clbx,lutmod);
    }
    catch(Exception e){
      System.out.println ("No podemos leer el valor de la LUT modificado por Jbits");
    }
    System.out.println("Valor nuevo: " + mostrarVector(lut,16));

    this.escribirContenido();
  } //ModificarContenidoLUT

  private void escribirContenido(){
    try {
      byte modificaciones[];
      modificaciones= miJbit.getPartial();
      if (modificaciones!=null) {
        try{
          fpgaExt.setConfiguration(0, modificaciones);
        }
        catch (XHWIFException e){
          System.out.println("No podemos conectar com la placa para escribir la
reconfiguracion");
        }
      }
    }
    catch (ConfigurationException e) {
      System.out.println("no podemos cargar de nuevo el contenido en la fpga");
      System.exit(-1);
    }
    catch (NullPointerException e){
      System.out.println("Problemas con un puntero a null");
    }
  }

  public void modificarEntradaLUT(int clbx, int clby, int bloque, int entrada, int nvalor){
    this.leerContenido();

    // MODIFICAR CONTENIDO LUT: Seleccionamos la LUT a modificar
    int[][] lutmod = null;
    int[] nv = null;
    char[] aux = null;
    int i;

    System.out.println("Modificando la entrada "+ entrada +" del bloque" + bloque + ".
Nuevo recurso = " + nvalor);

    switch (bloque)
    {
      case 1: // Cambiamos la LUT G de la slice 1
        switch(entrada)
        {
          case 1:  //Cambiamos la entrada 1
            lutmod= S1G1.S1G1;
            break;

          case 2: //Cambiamos la entrada 2
            lutmod= S1G2.S1G2;
            break;

          case 3: //Cambiamos la entrada 3
            lutmod = S1G3.S1G3;
            break;

          case 4: //Cambiamos la entrada 4
            lutmod = S1G4.S1G4;
            break;
        }
        break;

      case 2:  // Cambiamos la LUT F de la slice 1
        switch(entrada)
        {
          case 1:  //Cambiamos la entrada 1
            lutmod= S1F1.S1F1;
            break;
```

```
        case 2: //Cambiamos la entrada 2
          lutmod= S1F2.S1F2;
          break;

        case 3: //Cambiamos la entrada 3
          lutmod = S1F3.S1F3;
          break;

        case 4: //Cambiamos la entrada 4
          lutmod = S1F4.S1F4;
          break;
      }
      break;

    case 3: // Cambiamos la LUT G de la slice 0
      switch (entrada)
      {
        case 1:  //Cambiamos la entrada 1
          lutmod= S0G1.S0G1;
          break;

        case 2: //Cambiamos la entrada 2
          lutmod= S0G2.S0G2;
          break;

        case 3: //Cambiamos la entrada 3
          lutmod = S0G3.S0G3;
          break;

        case 4: //Cambiamos la entrada 4
          lutmod = S0G4.S0G4;
          break;
      }
      break;

    case 4: // Cambiamos la LUT F de la slice 0
      switch (entrada)
      {
        case 1:  //Cambiamos la entrada 1
          lutmod= S0F1.S0F1;
          break;

        case 2: //Cambiamos la entrada 2
          lutmod= S0F2.S0F2;
          break;

        case 3: //Cambiamos la entrada 3
          lutmod = S0F3.S0F3;
          break;

        case 4: //Cambiamos la entrada 4
          lutmod = S0F4.S0F4;
          break;
      }
      break;
  }

  //Seleccionamos el nuevo valor de la entrada
  aux = S0F1.Name[nvalor][1].toCharArray();

  if (aux != null)
  {
    nv = new int[aux.length];

    // Convertirlo a array de enteros
    for(i = 0; i < aux.length; i++)
      nv[i] = (aux[i] == '1')?1:0;

    // Modificamos la entrada de la LUT
    int[] lut=null;

    try
    {
      lut = miJbit.get(clbx,clby,lutmod);
    }
```

```
      catch(Exception e)
      {
        System.out.println ("No podemos leer el valor de la entrada leido por Jbits");
      }

      System.out.println("Valor antiguo: " + mostrarVector(lut,8));

      try
      {
        miJbit.set( clby, clbx, lutmod,nv);
      }
      catch (ConfigurationException e)
      {
        System.out.println("jbits no puede modificar el bitstream");
        System.exit(-1);
      }

      try
      {
        lut= miJbit.get(clby,clbx,lutmod);
      }
      catch(Exception e)
      {
        System.out.println ("No podemos leer el valor de la entrada modificado por Jbits");
      }

      System.out.println("Valor nuevo: " + mostrarVector(lut,8));

      this.escribirContenido();
    }
    else
    {
      System.out.println("No se ha podido obtener la lista de constantes");
    }
  }

  public void modificarModoBiestable(int clbx, int clby, int bloque, int nvalor){
    this.leerContenido();

    // MODIFICAR EL MODO DEL BIESTABLE
    int[][] bmod = null;
    int[] nv = null;
    char[] aux = null;
    int i;

    System.out.println("Modificando biestable a modo " + nvalor + " del bloque " + bloque);

    switch (bloque)
    {
      case 1: // Slice 1
      case 2: // Slice 1
        bmod= S1Control.LatchMode;
        break;

      case 3: // Slice 0
      case 4: // Slice 0
        bmod= S0Control.LatchMode;
        break;
    }

    //Seleccionamos el nuevo valor de la entrada
    if(nvalor == 0)
      nv = S0Control.OFF;
    else
      nv = S0Control.ON;

    // Modificamos el modo del biestable
    int[] bm=null;

    try
    {
      bm = miJbit.get(clby,clbx,bmod);
    }
    catch(Exception e)
    {
```

```
      System.out.println ("No podemos leer el valor de la entrada modificado por Jbits");
    }

    System.out.println("Valor antiguo: " + mostrarVector(bm, bm.length));

    try
    {
      miJbit.set( clby, clbx, bmod, nv);
    }
    catch (ConfigurationException e)
    {
      System.out.println("jbits no puede modificar el bitstream");
      System.exit(-1);
    }

    try
    {
      bm = miJbit.get(clby,clbx,bmod);
    }
    catch(Exception e)
    {
      System.out.println ("No podemos leer el valor de la entrada modificado por Jbits");
    }

    System.out.println("Valor nuevo: " + mostrarVector(bm, bm.length));

    this.escribirContenido();
  }

  public void modificarResetBiestable(int clbx, int clby, int bloque, int nvalor){
    this.leerContenido();

    // MODIFICAR EL MODO DEL BIESTABLE
    int[][] bmod = null;
    int[] nv = null;
    char[] aux = null;
    int i;

    System.out.println("Modificando reset biestable a modo " + nvalor + " del bloque " +
bloque);

    switch (bloque)
    {
      case 1: // Slice 1
      case 2: // Slice 1
        bmod= S1Control.Sync;
        break;

      case 3: // Slice 0
      case 4: // Slice 0
        bmod= S0Control.Sync;
        break;
    }

    //Seleccionamos el nuevo valor de la entrada
    if(nvalor == 0)
      nv = S0Control.OFF;
    else
      nv = S0Control.ON;

    // Modificamos el modo del biestable
    int[] bm=null;

    try
    {
      bm = miJbit.get(clby,clbx,bmod);
    }
    catch(Exception e)
    {
      System.out.println ("No podemos leer el valor de la entrada modificado por Jbits");
    }

    System.out.println("Valor antiguo: " + mostrarVector(bm, bm.length));

    try
```

```
    {
      miJbit.set( clby, clbx, bmod, nv);
    }
    catch (ConfigurationException e)
    {
      System.out.println("jbits no puede modificar el bitstream");
      System.exit(-1);
    }

    try
    {
      bm = miJbit.get(clby,clbx,bmod);
    }
    catch(Exception e)
    {
      System.out.println ("No podemos leer el valor de la entrada modificado por Jbits");
    }

    System.out.println("Valor nuevo: " + mostrarVector(bm, bm.length));

    this.escribirContenido();
  }

  public void cargarBitStreamSimulador(String nombreBitStream) {
    try {
      miJbit = new JBits(Devices.XCV1000);
      //leemos el bitstream y lo cargamos en un objeto JBits
      miJbit.readPartial(nombreBitStream);

      //Creamos la conexion con el simulador, obteniendo la Virtex virtual
      con = new XHWIFConnection();
      BoardScope bs = new BoardScope(con);
      bs.show();
      con.connectToBoard("VirtexDS:XCV1000");
      con.resetBoard();
      System.out.println("Simulador inicializado");
      // Obtenemos la interfaz Jbits<--->Simulador
      fpgaExt = con.getXHWIFWithEvents();

      // Cargamos en el simulador el contenido del Jbits
      fpgaExt.setConfiguration(0,miJbit.getPartial());
      System.out.println("Bitstream cargado correctamente en el simulador");
    }
    catch (FileNotFoundException fe) {
      System.out.println("File " + nombreBitStream + " not found");
      System.out.println(fe);
    }
    catch (IOException io) {
      System.out.println("Error reading the bitstream file");
      System.out.println(io);
    }
    catch (ConfigurationException ce) {
      System.out.println("Internal error detected in the Bitstream");
      System.out.println(ce);
    }
    catch (XHWIFException xe) {
      System.out.println("Error en el módulo XHWIF al cargar el bitstream");
    }

  }  //cargarBitStreamSimulador

  public void desconectarSimulador(){
    try{
      con.disconnectFromBoard();
      System.out.println("nos hemos desconectado de la fpga simulada");
    }
    catch (XHWIFException xe){
      System.out.println("Error al desconectar del simulador");
    }
  }

  public void desconectarFPGA(){
  try {
    fpga.reset();
    fpga.disconnect();
```

```
    System.out.println("nos hemos desconectado de la fpga física");
  }
  catch (Exception xe){
    System.out.println("Error al desconectar del simulador");
  }
}



  public String[] getRecursosEntradasLut(int entrada, int bloque)
  {
    // Enumera los recursos que pueden asignarse a una determinada entrada de una lut
    String[][] recursos=null;
    String[] salida = null;
    int i;

    // Determinar que slut y que bloque esta siendo pedido
    switch (bloque)
    {
      case 1: // LUT G de la slice 1
        switch (entrada)
        {
          case 1:  //entrada 1
            recursos = S1G1.Name;
            break;

          case 2: //entrada 2
            recursos= S1G2.Name;
            break;

          case 3: //entrada 3
            recursos = S1G3.Name;
            break;

          case 4: //entrada 4
            recursos = S1G4.Name;
            break;
        }
        break;

      case 2: // LUT F de la slice 1
        switch (entrada)
        {
          case 1:  //entrada 1
            recursos = S1F1.Name;
            break;

          case 2: //entrada 2
            recursos = S1F2.Name;
            break;

          case 3: //entrada 3
            recursos = S1F3.Name;
            break;

          case 4: //entrada 4
            recursos = S1F4.Name;
            break;
        }
        break;

      case 3: // LUT G de la slice 0
        switch (entrada)
        {
          case 1:  //entrada 1
            recursos = S0G1.Name;
            break;

          case 2: //entrada 2
            recursos = S0G2.Name;
            break;

          case 3: //entrada 3
            recursos = S0G3.Name;
            break;
```

```
        case 4: //entrada 4
          recursos = S0G4.Name;
          break;
      }
      break;

    case 4: //LUT F de la slice 0
      switch (entrada)
      {
        case 1:  //entrada 1
          recursos = S0F1.Name;
          break;

        case 2: //entrada 2
          recursos = S0F2.Name;
          break;

        case 3: //entrada 3
          recursos = S0F3.Name;
          break;

        case 4: //entrada 4
          recursos = S0F4.Name;
          break;
      }
      break;
  }

  salida = new String[recursos.length];

  // Crear la nueva lista de objetos
  for(i = 0; i < recursos.length; i++)
    salida[i] = recursos[i][0];

  return salida;
}

public String[] getRecursosEntradaBiestable(int bloque)
{
  // Enumera los recursos que pueden asignarse a una determinada entrada de una lut
  String[][] recursos=null;
  String[] salida = null;
  int i;

  // Determinar que slice y biestable se ha pedido
  switch (bloque)
  {
    case 1: // Slice 1 biestable Y
      recursos = S1Control.Y.Name;
      break;

    case 2: // Slice 1 biestable X
      recursos = S1Control.X.Name;
      break;

    case 3: // Slice 0 biestable y
      recursos = S0Control.Y.Name;
      break;

    case 4: // Slice 0 biestable x
      recursos = S0Control.X.Name;
      break;
  }

  salida = new String[recursos.length];

  // Crear la nueva lista de objetos
  for(i = 0; i < recursos.length; i++)
    salida[i] = recursos[i][0];

  return salida;
}

public void copiarCLB(int clbx, int clby, int clbxdest, int clbydest)
```

```
  {
    int[][][] recs = {
      S0F1.S0F1, S0F2.S0F2, S0F3.S0F3, S0F4.S0F4, S0G1.S0G1, S0G2.S0G2, S0G3.S0G3,
S0G4.S0G4, // Entradas de las LUTS
      S1F1.S1F1, S1F2.S1F2, S1F3.S1F3, S1F4.S1F4, S1G1.S1G1, S1G2.S1G2, S1G3.S1G3,
S1G4.S1G4,
      LUT.SLICE0_F, LUT.SLICE0_G, LUT.SLICE1_F, LUT.SLICE1_G, // El contenido de las LUTS
      CLB.SLICE0_XQ, CLB.SLICE0_YQ, CLB.SLICE1_XQ, CLB.SLICE1_YQ,
      S0Control.BxInvert, S0Control.ByInvert, S0Control.CeInvert, S0Control.ClockInvert, //
Control del Slice 0
      S0Control.InvertedSetReset, S0Control.LatchMode, S0Control.SrWeNotInvert,
S0Control.Sync,
      S0Control.XffSetResetSelect, S0Control.YffSetResetSelect, S0Control.AndMux.AndMux,
S0Control.Cin.Cin,
      S0Control.X.X, S0Control.XCarrySelect.XCarrySelect, S0Control.XDin.XDin,
      S0Control.Y.Y, S0Control.YB.YB, S0Control.YCarrySelect.YCarrySelect,
S0Control.YDin.YDin,
      S1Control.BxInvert, S1Control.ByInvert, S1Control.CeInvert, S1Control.ClockInvert, //
Control del Slice 1
      S1Control.InvertedSetReset, S1Control.LatchMode, S1Control.SrWeNotInvert,
S1Control.Sync,
      S1Control.XffSetResetSelect, S1Control.YffSetResetSelect, S1Control.AndMux.AndMux,
S1Control.Cin.Cin,
      S1Control.X.X, S1Control.XCarrySelect.XCarrySelect, S1Control.XDin.XDin,
      S1Control.Y.Y, S1Control.YB.YB, S1Control.YCarrySelect.YCarrySelect,
S1Control.YDin.YDin,
      S0BX.S0BX, S0BY.S0BY, S0CE.S0CE, S0Clk.S0Clk, // Entradas para reloj, de los
biestables y del reset
      S1BX.S1BX, S1BY.S1BY, S1CE.S1CE, S1Clk.S1Clk,
      S0RAM.DUAL_MODE, S0RAM.F_LUT_RAM, S0RAM.F_LUT_SHIFTER, S0RAM.G_LUT_RAM, // Memoria
      S0RAM.G_LUT_SHIFTER, S0RAM.LUT_MODE, S0RAM.RAM_32_X_1, S0RAM.UNUSED,
      S1RAM.DUAL_MODE, S1RAM.F_LUT_RAM, S1RAM.F_LUT_SHIFTER, S1RAM.G_LUT_RAM,
      S1RAM.G_LUT_SHIFTER, S1RAM.LUT_MODE, S1RAM.RAM_32_X_1, S1RAM.UNUSED,
      OUT0.OUT0, OUT1.OUT1, OUT2.OUT2, OUT3.OUT3, OUT4.OUT4, OUT5.OUT5, OUT6.OUT6,
OUT7.OUT7, // Salida
      SingleToSingle.SINGLE_WEST0_TO_SINGLE_EAST0,
SingleToSingle.SINGLE_EAST0_TO_SINGLE_WEST0, // Switch 'Sencillo'
      SingleToSingle.SINGLE_SOUTH1_TO_SINGLE_NORTH1,
SingleToSingle.SINGLE_NORTH1_TO_SINGLE_SOUTH1,
      SingleToSingle.SINGLE_SOUTH2_TO_SINGLE_NORTH2,
SingleToSingle.SINGLE_NORTH2_TO_SINGLE_SOUTH2,
      SingleToSingle.SINGLE_SOUTH5_TO_SINGLE_NORTH5,
SingleToSingle.SINGLE_NORTH5_TO_SINGLE_SOUTH5,
      SingleToSingle.SINGLE_SOUTH0_TO_SINGLE_NORTH0,
SingleToSingle.SINGLE_NORTH0_TO_SINGLE_SOUTH0,
      SingleToSingle.SINGLE_SOUTH6_TO_SINGLE_NORTH6,
SingleToSingle.SINGLE_NORTH6_TO_SINGLE_SOUTH6,
      SingleToSingle.SINGLE_SOUTH18_TO_SINGLE_NORTH18,
SingleToSingle.SINGLE_NORTH18_TO_SINGLE_SOUTH18,
      SingleToSingle.SINGLE_SOUTH19_TO_SINGLE_NORTH19,
SingleToSingle.SINGLE_NORTH19_TO_SINGLE_SOUTH19,
      SingleToSingle.SINGLE_SOUTH20_TO_SINGLE_NORTH20,
SingleToSingle.SINGLE_NORTH20_TO_SINGLE_SOUTH20,
      SingleToSingle.SINGLE_SOUTH23_TO_SINGLE_NORTH23,
SingleToSingle.SINGLE_NORTH23_TO_SINGLE_SOUTH23,
      SingleToSingle.SINGLE_WEST16_TO_SINGLE_EAST16,
SingleToSingle.SINGLE_EAST16_TO_SINGLE_WEST16,
      SingleToSingle.SINGLE_WEST19_TO_SINGLE_EAST19,
SingleToSingle.SINGLE_EAST19_TO_SINGLE_WEST19,
      SingleToSingle.SINGLE_WEST22_TO_SINGLE_EAST22,
SingleToSingle.SINGLE_EAST22_TO_SINGLE_WEST22,
      SingleToSingle.SINGLE_WEST21_TO_SINGLE_EAST21,
SingleToSingle.SINGLE_EAST21_TO_SINGLE_WEST21,
      SingleToSingle.SINGLE_WEST8_TO_SINGLE_EAST8,
SingleToSingle.SINGLE_EAST8_TO_SINGLE_WEST8,
      SingleToSingle.SINGLE_WEST9_TO_SINGLE_EAST9,
SingleToSingle.SINGLE_EAST9_TO_SINGLE_WEST9,
      SingleToSingle.SINGLE_WEST11_TO_SINGLE_EAST11,
SingleToSingle.SINGLE_EAST11_TO_SINGLE_WEST11,
      SingleToSingle.SINGLE_WEST10_TO_SINGLE_EAST10,
SingleToSingle.SINGLE_EAST10_TO_SINGLE_WEST10,
      SingleToSingle.SINGLE_WEST14_TO_SINGLE_EAST14,
SingleToSingle.SINGLE_EAST14_TO_SINGLE_WEST14,
      SingleToSingle.SINGLE_WEST15_TO_SINGLE_EAST15,
SingleToSingle.SINGLE_EAST15_TO_SINGLE_WEST15,
```

```
      SingleToSingle.SINGLE_WEST13_TO_SINGLE_EAST13,
SingleToSingle.SINGLE_EAST13_TO_SINGLE_WEST13,
      SingleToSingle.SINGLE_WEST12_TO_SINGLE_EAST12,
SingleToSingle.SINGLE_EAST12_TO_SINGLE_WEST12,
      SingleToSingle.SINGLE_WEST17_TO_SINGLE_EAST17,
SingleToSingle.SINGLE_EAST17_TO_SINGLE_WEST17,
      SingleToSingle.SINGLE_WEST18_TO_SINGLE_EAST18,
SingleToSingle.SINGLE_EAST18_TO_SINGLE_WEST18,
      SingleToSingle.SINGLE_WEST23_TO_SINGLE_EAST23,
SingleToSingle.SINGLE_EAST23_TO_SINGLE_WEST23,
      SingleToSingle.SINGLE_WEST20_TO_SINGLE_EAST20,
SingleToSingle.SINGLE_EAST20_TO_SINGLE_WEST20,
      SingleToSingle.SINGLE_WEST6_TO_SINGLE_EAST6,
SingleToSingle.SINGLE_EAST6_TO_SINGLE_WEST6,
      SingleToSingle.SINGLE_WEST7_TO_SINGLE_EAST7,
SingleToSingle.SINGLE_EAST7_TO_SINGLE_WEST7,
      SingleToSingle.SINGLE_WEST5_TO_SINGLE_EAST5,
SingleToSingle.SINGLE_EAST5_TO_SINGLE_WEST5,
      SingleToSingle.SINGLE_WEST4_TO_SINGLE_EAST4,
SingleToSingle.SINGLE_EAST4_TO_SINGLE_WEST4,
      SingleToSingle.SINGLE_WEST2_TO_SINGLE_EAST2,
SingleToSingle.SINGLE_EAST2_TO_SINGLE_WEST2,
      SingleToSingle.SINGLE_WEST3_TO_SINGLE_EAST3,
SingleToSingle.SINGLE_EAST3_TO_SINGLE_WEST3,
      SingleToSingle.SINGLE_WEST1_TO_SINGLE_EAST1,
SingleToSingle.SINGLE_EAST1_TO_SINGLE_WEST1,
      SingleToSingle.SINGLE_SOUTH4_TO_SINGLE_NORTH4,
SingleToSingle.SINGLE_NORTH4_TO_SINGLE_SOUTH4,
      SingleToSingle.SINGLE_SOUTH3_TO_SINGLE_NORTH3,
SingleToSingle.SINGLE_NORTH3_TO_SINGLE_SOUTH3,
      SingleToSingle.SINGLE_SOUTH22_TO_SINGLE_NORTH22,
SingleToSingle.SINGLE_NORTH22_TO_SINGLE_SOUTH22,
      SingleToSingle.SINGLE_SOUTH21_TO_SINGLE_NORTH21,
SingleToSingle.SINGLE_NORTH21_TO_SINGLE_SOUTH21,
      SingleToSingle.SINGLE_SOUTH9_TO_SINGLE_NORTH9,
SingleToSingle.SINGLE_NORTH9_TO_SINGLE_SOUTH9,
      SingleToSingle.SINGLE_SOUTH10_TO_SINGLE_NORTH10,
SingleToSingle.SINGLE_NORTH10_TO_SINGLE_SOUTH10,
      SingleToSingle.SINGLE_SOUTH11_TO_SINGLE_NORTH11,
SingleToSingle.SINGLE_NORTH11_TO_SINGLE_SOUTH11,
      SingleToSingle.SINGLE_SOUTH15_TO_SINGLE_NORTH15,
SingleToSingle.SINGLE_NORTH15_TO_SINGLE_SOUTH15,
      SingleToSingle.SINGLE_SOUTH16_TO_SINGLE_NORTH16,
SingleToSingle.SINGLE_NORTH16_TO_SINGLE_SOUTH16,
      SingleToSingle.SINGLE_SOUTH17_TO_SINGLE_NORTH17,
SingleToSingle.SINGLE_NORTH17_TO_SINGLE_SOUTH17,
      SingleToSingle.SINGLE_SOUTH14_TO_SINGLE_NORTH14,
SingleToSingle.SINGLE_NORTH14_TO_SINGLE_SOUTH14,
      SingleToSingle.SINGLE_SOUTH13_TO_SINGLE_NORTH13,
SingleToSingle.SINGLE_NORTH13_TO_SINGLE_SOUTH13,
      SingleToSingle.SINGLE_SOUTH12_TO_SINGLE_NORTH12,
SingleToSingle.SINGLE_NORTH12_TO_SINGLE_SOUTH12,
      SingleToSingle.SINGLE_SOUTH8_TO_SINGLE_NORTH8,
SingleToSingle.SINGLE_NORTH8_TO_SINGLE_SOUTH8,
      SingleToSingle.SINGLE_SOUTH7_TO_SINGLE_NORTH7,
SingleToSingle.SINGLE_NORTH7_TO_SINGLE_SOUTH7,
      SingleToSingle.SINGLE_WEST0_TO_SINGLE_SOUTH2,
SingleToSingle.SINGLE_SOUTH2_TO_SINGLE_WEST0,
      SingleToSingle.SINGLE_EAST20_TO_SINGLE_SOUTH2,
SingleToSingle.SINGLE_SOUTH2_TO_SINGLE_EAST20,
      SingleToSingle.SINGLE_EAST20_TO_SINGLE_NORTH0,
SingleToSingle.SINGLE_NORTH0_TO_SINGLE_EAST20,
      SingleToSingle.SINGLE_WEST5_TO_SINGLE_NORTH0,
SingleToSingle.SINGLE_NORTH0_TO_SINGLE_WEST5,
      SingleToSingle.SINGLE_WEST5_TO_SINGLE_SOUTH3,
SingleToSingle.SINGLE_SOUTH3_TO_SINGLE_WEST5,
      SingleToSingle.SINGLE_EAST1_TO_SINGLE_SOUTH3,
SingleToSingle.SINGLE_SOUTH3_TO_SINGLE_EAST1,
      SingleToSingle.SINGLE_EAST1_TO_SINGLE_NORTH1,
SingleToSingle.SINGLE_NORTH1_TO_SINGLE_EAST1,
      SingleToSingle.SINGLE_WEST2_TO_SINGLE_NORTH1,
SingleToSingle.SINGLE_NORTH1_TO_SINGLE_WEST2,
      SingleToSingle.SINGLE_WEST2_TO_SINGLE_SOUTH0,
SingleToSingle.SINGLE_SOUTH0_TO_SINGLE_WEST2,
```

```
      SingleToSingle.SINGLE_EAST22_TO_SINGLE_SOUTH0,
SingleToSingle.SINGLE_SOUTH0_TO_SINGLE_EAST22,
      SingleToSingle.SINGLE_EAST22_TO_SINGLE_NORTH2,
SingleToSingle.SINGLE_NORTH2_TO_SINGLE_EAST22,
      SingleToSingle.SINGLE_WEST7_TO_SINGLE_NORTH2,
SingleToSingle.SINGLE_NORTH2_TO_SINGLE_WEST7,
      SingleToSingle.SINGLE_WEST7_TO_SINGLE_SOUTH1,
SingleToSingle.SINGLE_SOUTH1_TO_SINGLE_WEST7,
      SingleToSingle.SINGLE_EAST3_TO_SINGLE_SOUTH1,
SingleToSingle.SINGLE_SOUTH1_TO_SINGLE_EAST3,
      SingleToSingle.SINGLE_EAST3_TO_SINGLE_NORTH3,
SingleToSingle.SINGLE_NORTH3_TO_SINGLE_EAST3,
      SingleToSingle.SINGLE_WEST4_TO_SINGLE_NORTH3,
SingleToSingle.SINGLE_NORTH3_TO_SINGLE_WEST4,
      SingleToSingle.SINGLE_WEST4_TO_SINGLE_SOUTH6,
SingleToSingle.SINGLE_SOUTH6_TO_SINGLE_WEST4,
      SingleToSingle.SINGLE_EAST0_TO_SINGLE_SOUTH6,
SingleToSingle.SINGLE_SOUTH6_TO_SINGLE_EAST0,
      SingleToSingle.SINGLE_EAST0_TO_SINGLE_NORTH4,
SingleToSingle.SINGLE_NORTH4_TO_SINGLE_EAST0,
      SingleToSingle.SINGLE_EAST19_TO_SINGLE_SOUTH17,
SingleToSingle.SINGLE_SOUTH17_TO_SINGLE_EAST19,
      SingleToSingle.SINGLE_WEST18_TO_SINGLE_SOUTH16,
SingleToSingle.SINGLE_SOUTH16_TO_SINGLE_WEST18,
      SingleToSingle.SINGLE_WEST18_TO_SINGLE_NORTH17,
SingleToSingle.SINGLE_NORTH17_TO_SINGLE_WEST18,
      SingleToSingle.SINGLE_WEST23_TO_SINGLE_NORTH18,
SingleToSingle.SINGLE_NORTH18_TO_SINGLE_WEST23,
      SingleToSingle.SINGLE_EAST17_TO_SINGLE_SOUTH19,
SingleToSingle.SINGLE_SOUTH19_TO_SINGLE_EAST17,
      SingleToSingle.SINGLE_WEST23_TO_SINGLE_SOUTH17,
SingleToSingle.SINGLE_SOUTH17_TO_SINGLE_WEST23,
      SingleToSingle.SINGLE_EAST17_TO_SINGLE_NORTH17,
SingleToSingle.SINGLE_NORTH17_TO_SINGLE_EAST17,
      SingleToSingle.SINGLE_EAST14_TO_SINGLE_NORTH18,
SingleToSingle.SINGLE_NORTH18_TO_SINGLE_EAST14,
      SingleToSingle.SINGLE_EAST14_TO_SINGLE_SOUTH16,
SingleToSingle.SINGLE_SOUTH16_TO_SINGLE_EAST14,
      SingleToSingle.SINGLE_WEST0_TO_SINGLE_NORTH23,
SingleToSingle.SINGLE_NORTH23_TO_SINGLE_WEST0,
      SingleToSingle.SINGLE_WEST21_TO_SINGLE_NORTH16,
SingleToSingle.SINGLE_NORTH16_TO_SINGLE_WEST21,
      SingleToSingle.SINGLE_WEST21_TO_SINGLE_SOUTH19,
SingleToSingle.SINGLE_SOUTH19_TO_SINGLE_WEST21,
      SingleToSingle.SINGLE_EAST12_TO_SINGLE_NORTH16,
SingleToSingle.SINGLE_NORTH16_TO_SINGLE_EAST12,
      SingleToSingle.SINGLE_EAST12_TO_SINGLE_SOUTH18,
SingleToSingle.SINGLE_SOUTH18_TO_SINGLE_EAST12,
      SingleToSingle.SINGLE_WEST16_TO_SINGLE_SOUTH18,
SingleToSingle.SINGLE_SOUTH18_TO_SINGLE_WEST16,
      SingleToSingle.SINGLE_WEST16_TO_SINGLE_NORTH15,
SingleToSingle.SINGLE_NORTH15_TO_SINGLE_WEST16,
      SingleToSingle.SINGLE_WEST19_TO_SINGLE_NORTH14,
SingleToSingle.SINGLE_NORTH14_TO_SINGLE_WEST19,
      SingleToSingle.SINGLE_EAST10_TO_SINGLE_NORTH14,
SingleToSingle.SINGLE_NORTH14_TO_SINGLE_EAST10,
      SingleToSingle.SINGLE_WEST19_TO_SINGLE_SOUTH13,
SingleToSingle.SINGLE_SOUTH13_TO_SINGLE_WEST19,
      SingleToSingle.SINGLE_EAST15_TO_SINGLE_NORTH15,
SingleToSingle.SINGLE_NORTH15_TO_SINGLE_EAST15,
      SingleToSingle.SINGLE_EAST15_TO_SINGLE_SOUTH13,
SingleToSingle.SINGLE_SOUTH13_TO_SINGLE_EAST15,
      SingleToSingle.SINGLE_EAST10_TO_SINGLE_SOUTH12,
SingleToSingle.SINGLE_SOUTH12_TO_SINGLE_EAST10,
      SingleToSingle.SINGLE_WEST14_TO_SINGLE_SOUTH12,
SingleToSingle.SINGLE_SOUTH12_TO_SINGLE_WEST14,
      SingleToSingle.SINGLE_EAST13_TO_SINGLE_NORTH13,
SingleToSingle.SINGLE_NORTH13_TO_SINGLE_EAST13,
      SingleToSingle.SINGLE_WEST14_TO_SINGLE_NORTH13,
SingleToSingle.SINGLE_NORTH13_TO_SINGLE_WEST14,
      SingleToSingle.SINGLE_EAST13_TO_SINGLE_SOUTH15,
SingleToSingle.SINGLE_SOUTH15_TO_SINGLE_EAST13,
      SingleToSingle.SINGLE_EAST23_TO_SINGLE_NORTH23,
SingleToSingle.SINGLE_NORTH23_TO_SINGLE_EAST23,
```

```
      SingleToSingle.SINGLE_EAST23_TO_SINGLE_SOUTH21,
SingleToSingle.SINGLE_SOUTH21_TO_SINGLE_EAST23,
      SingleToSingle.SINGLE_WEST3_TO_SINGLE_SOUTH21,
SingleToSingle.SINGLE_SOUTH21_TO_SINGLE_WEST3,
      SingleToSingle.SINGLE_WEST3_TO_SINGLE_NORTH22,
SingleToSingle.SINGLE_NORTH22_TO_SINGLE_WEST3,
      SingleToSingle.SINGLE_EAST18_TO_SINGLE_NORTH22,
SingleToSingle.SINGLE_NORTH22_TO_SINGLE_EAST18,
      SingleToSingle.SINGLE_EAST18_TO_SINGLE_SOUTH20,
SingleToSingle.SINGLE_SOUTH20_TO_SINGLE_EAST18,
      SingleToSingle.SINGLE_WEST22_TO_SINGLE_SOUTH20,
SingleToSingle.SINGLE_SOUTH20_TO_SINGLE_WEST22,
      SingleToSingle.SINGLE_EAST21_TO_SINGLE_SOUTH23,
SingleToSingle.SINGLE_SOUTH23_TO_SINGLE_EAST21,
      SingleToSingle.SINGLE_WEST1_TO_SINGLE_SOUTH23,
SingleToSingle.SINGLE_SOUTH23_TO_SINGLE_WEST1,
      SingleToSingle.SINGLE_WEST1_TO_SINGLE_NORTH20,
SingleToSingle.SINGLE_NORTH20_TO_SINGLE_WEST1,
      SingleToSingle.SINGLE_EAST16_TO_SINGLE_NORTH20,
SingleToSingle.SINGLE_NORTH20_TO_SINGLE_EAST16,
      SingleToSingle.SINGLE_WEST22_TO_SINGLE_NORTH21,
SingleToSingle.SINGLE_NORTH21_TO_SINGLE_WEST22,
      SingleToSingle.SINGLE_WEST20_TO_SINGLE_SOUTH22,
SingleToSingle.SINGLE_SOUTH22_TO_SINGLE_WEST20,
      SingleToSingle.SINGLE_EAST21_TO_SINGLE_NORTH21,
SingleToSingle.SINGLE_NORTH21_TO_SINGLE_EAST21,
      SingleToSingle.SINGLE_EAST16_TO_SINGLE_SOUTH22,
SingleToSingle.SINGLE_SOUTH22_TO_SINGLE_EAST16,
      SingleToSingle.SINGLE_WEST20_TO_SINGLE_NORTH19,
SingleToSingle.SINGLE_NORTH19_TO_SINGLE_WEST20,
      SingleToSingle.SINGLE_EAST19_TO_SINGLE_NORTH19,
SingleToSingle.SINGLE_NORTH19_TO_SINGLE_EAST19,
      SingleToSingle.SINGLE_WEST17_TO_SINGLE_SOUTH15,
SingleToSingle.SINGLE_SOUTH15_TO_SINGLE_WEST17,
      SingleToSingle.SINGLE_WEST17_TO_SINGLE_NORTH12,
SingleToSingle.SINGLE_NORTH12_TO_SINGLE_WEST17,
      SingleToSingle.SINGLE_EAST9_TO_SINGLE_SOUTH11,
SingleToSingle.SINGLE_SOUTH11_TO_SINGLE_EAST9,
      SingleToSingle.SINGLE_EAST9_TO_SINGLE_NORTH9,
SingleToSingle.SINGLE_NORTH9_TO_SINGLE_EAST9,
      SingleToSingle.SINGLE_WEST10_TO_SINGLE_NORTH9,
SingleToSingle.SINGLE_NORTH9_TO_SINGLE_WEST10,
      SingleToSingle.SINGLE_WEST10_TO_SINGLE_SOUTH8,
SingleToSingle.SINGLE_SOUTH8_TO_SINGLE_WEST10,
      SingleToSingle.SINGLE_EAST6_TO_SINGLE_SOUTH8,
SingleToSingle.SINGLE_SOUTH8_TO_SINGLE_EAST6,
      SingleToSingle.SINGLE_EAST6_TO_SINGLE_NORTH10,
SingleToSingle.SINGLE_NORTH10_TO_SINGLE_EAST6,
      SingleToSingle.SINGLE_WEST15_TO_SINGLE_NORTH10,
SingleToSingle.SINGLE_NORTH10_TO_SINGLE_WEST15,
      SingleToSingle.SINGLE_WEST15_TO_SINGLE_SOUTH9,
SingleToSingle.SINGLE_SOUTH9_TO_SINGLE_WEST15,
      SingleToSingle.SINGLE_EAST11_TO_SINGLE_SOUTH9,
SingleToSingle.SINGLE_SOUTH9_TO_SINGLE_EAST11,
      SingleToSingle.SINGLE_EAST11_TO_SINGLE_NORTH11,
SingleToSingle.SINGLE_NORTH11_TO_SINGLE_EAST11,
      SingleToSingle.SINGLE_WEST12_TO_SINGLE_NORTH11,
SingleToSingle.SINGLE_NORTH11_TO_SINGLE_WEST12,
      SingleToSingle.SINGLE_WEST12_TO_SINGLE_SOUTH14,
SingleToSingle.SINGLE_SOUTH14_TO_SINGLE_WEST12,
      SingleToSingle.SINGLE_EAST8_TO_SINGLE_SOUTH14,
SingleToSingle.SINGLE_SOUTH14_TO_SINGLE_EAST8,
      SingleToSingle.SINGLE_EAST8_TO_SINGLE_NORTH12,
SingleToSingle.SINGLE_NORTH12_TO_SINGLE_EAST8,
      SingleToSingle.SINGLE_EAST7_TO_SINGLE_SOUTH5,
SingleToSingle.SINGLE_SOUTH5_TO_SINGLE_EAST7,
      SingleToSingle.SINGLE_EAST7_TO_SINGLE_NORTH7,
SingleToSingle.SINGLE_NORTH7_TO_SINGLE_EAST7,
      SingleToSingle.SINGLE_WEST8_TO_SINGLE_NORTH7,
SingleToSingle.SINGLE_NORTH7_TO_SINGLE_WEST8,
      SingleToSingle.SINGLE_WEST8_TO_SINGLE_SOUTH10,
SingleToSingle.SINGLE_SOUTH10_TO_SINGLE_WEST8,
      SingleToSingle.SINGLE_EAST4_TO_SINGLE_SOUTH10,
SingleToSingle.SINGLE_SOUTH10_TO_SINGLE_EAST4,
```

```
      SingleToSingle.SINGLE_EAST4_TO_SINGLE_NORTH8,
SingleToSingle.SINGLE_NORTH8_TO_SINGLE_EAST4,
      SingleToSingle.SINGLE_WEST13_TO_SINGLE_NORTH8,
SingleToSingle.SINGLE_NORTH8_TO_SINGLE_WEST13,
      SingleToSingle.SINGLE_WEST13_TO_SINGLE_SOUTH11,
SingleToSingle.SINGLE_SOUTH11_TO_SINGLE_WEST13,
      SingleToSingle.SINGLE_EAST2_TO_SINGLE_SOUTH4,
SingleToSingle.SINGLE_SOUTH4_TO_SINGLE_EAST2,
      SingleToSingle.SINGLE_EAST2_TO_SINGLE_NORTH6,
SingleToSingle.SINGLE_NORTH6_TO_SINGLE_EAST2,
      SingleToSingle.SINGLE_WEST11_TO_SINGLE_NORTH6,
SingleToSingle.SINGLE_NORTH6_TO_SINGLE_WEST11,
      SingleToSingle.SINGLE_WEST11_TO_SINGLE_SOUTH5,
SingleToSingle.SINGLE_SOUTH5_TO_SINGLE_WEST11,
      SingleToSingle.SINGLE_EAST5_TO_SINGLE_SOUTH7,
SingleToSingle.SINGLE_SOUTH7_TO_SINGLE_EAST5,
      SingleToSingle.SINGLE_EAST5_TO_SINGLE_NORTH5,
SingleToSingle.SINGLE_NORTH5_TO_SINGLE_EAST5,
      SingleToSingle.SINGLE_WEST6_TO_SINGLE_NORTH5,
SingleToSingle.SINGLE_NORTH5_TO_SINGLE_WEST6,
      SingleToSingle.SINGLE_WEST6_TO_SINGLE_SOUTH4,
SingleToSingle.SINGLE_SOUTH4_TO_SINGLE_WEST6,
      SingleToSingle.SINGLE_WEST9_TO_SINGLE_NORTH4,
SingleToSingle.SINGLE_NORTH4_TO_SINGLE_WEST9,
      SingleToSingle.SINGLE_WEST9_TO_SINGLE_SOUTH7,
SingleToSingle.SINGLE_SOUTH7_TO_SINGLE_WEST9,
      BiHexToSingle.SINGLE_EAST1_TO_HEX_VERT_M0, BiHexToSingle.SINGLE_WEST3_TO_HEX_VERT_M0,
// Conexiones simples entre switches
      BiHexToSingle.SINGLE_EAST0_TO_HEX_SOUTH0, BiHexToSingle.SINGLE_WEST4_TO_HEX_NORTH0,
      BiHexToSingle.SINGLE_WEST5_TO_HEX_EAST0, BiHexToSingle.SINGLE_WEST6_TO_HEX_EAST1,
      BiHexToSingle.SINGLE_WEST8_TO_HEX_VERT_M1, BiHexToSingle.SINGLE_EAST7_TO_HEX_VERT_M1,
      BiHexToSingle.SINGLE_WEST9_TO_HEX_SOUTH1, BiHexToSingle.SINGLE_EAST9_TO_HEX_NORTH1,
      BiHexToSingle.SINGLE_EAST11_TO_HEX_WEST1, BiHexToSingle.SINGLE_WEST17_TO_HEX_EAST2,
      BiHexToSingle.SINGLE_WEST18_TO_HEX_EAST3, BiHexToSingle.SINGLE_EAST23_TO_HEX_WEST3,
      BiHexToSingle.SINGLE_EAST10_TO_HEX_WEST2, BiHexToSingle.SINGLE_WEST16_TO_HEX_NORTH2,
      BiHexToSingle.SINGLE_EAST21_TO_HEX_NORTH3, BiHexToSingle.SINGLE_EAST12_TO_HEX_SOUTH2,
      BiHexToSingle.SINGLE_WEST21_TO_HEX_SOUTH3,
BiHexToSingle.HEX_SOUTH2_TO_SINGLE_NORTH13,
      BiHexToSingle.HEX_SOUTH3_TO_SINGLE_NORTH19,
BiHexToSingle.HEX_NORTH2_TO_SINGLE_SOUTH14,
      BiHexToSingle.HEX_NORTH3_TO_SINGLE_SOUTH22,
BiHexToSingle.HEX_WEST2_TO_SINGLE_NORTH12,
      BiHexToSingle.HEX_WEST3_TO_SINGLE_SOUTH20, BiHexToSingle.HEX_EAST3_TO_SINGLE_NORTH18,
      BiHexToSingle.HEX_EAST2_TO_SINGLE_SOUTH16,
BiHexToSingle.HEX_HORIZ_M3_TO_SINGLE_NORTH22,
      BiHexToSingle.HEX_HORIZ_M3_TO_SINGLE_SOUTH21,
BiHexToSingle.HEX_HORIZ_M2_TO_SINGLE_SOUTH18,
      BiHexToSingle.HEX_HORIZ_M2_TO_SINGLE_NORTH16,
BiHexToSingle.HEX_WEST0_TO_SINGLE_NORTH0,
      BiHexToSingle.SINGLE_WEST20_TO_HEX_VERT_M3,
BiHexToSingle.HEX_NORTH0_TO_SINGLE_SOUTH2,
      BiHexToSingle.SINGLE_WEST15_TO_HEX_VERT_M2,
BiHexToSingle.HEX_HORIZ_M1_TO_SINGLE_NORTH10,
      BiHexToSingle.HEX_HORIZ_M1_TO_SINGLE_SOUTH9,
BiHexToSingle.HEX_WEST1_TO_SINGLE_SOUTH8,
      BiHexToSingle.HEX_SOUTH0_TO_SINGLE_NORTH1, BiHexToSingle.HEX_EAST1_TO_SINGLE_NORTH6,
      BiHexToSingle.HEX_HORIZ_M0_TO_SINGLE_SOUTH6,
BiHexToSingle.SINGLE_EAST22_TO_HEX_WEST0,
      BiHexToSingle.SINGLE_EAST13_TO_HEX_VERT_M2,
BiHexToSingle.HEX_HORIZ_M0_TO_SINGLE_NORTH4,
      BiHexToSingle.HEX_SOUTH1_TO_SINGLE_NORTH7,
BiHexToSingle.HEX_NORTH1_TO_SINGLE_SOUTH10,
      BiHexToSingle.SINGLE_EAST19_TO_HEX_VERT_M3, BiHexToSingle.HEX_EAST0_TO_SINGLE_SOUTH4,
      OutMuxToSingle.OUT0_TO_SINGLE_WEST7, OutMuxToSingle.OUT0_TO_SINGLE_EAST2, // Switch
pequeño para las salidas
      OutMuxToSingle.OUT1_TO_SINGLE_WEST4, OutMuxToSingle.OUT1_TO_SINGLE_EAST3,
      OutMuxToSingle.OUT1_TO_SINGLE_EAST5, OutMuxToSingle.OUT1_TO_SINGLE_WEST5,
      OutMuxToSingle.OUT2_TO_SINGLE_WEST9, OutMuxToSingle.OUT2_TO_SINGLE_EAST6,
      OutMuxToSingle.OUT3_TO_SINGLE_EAST8, OutMuxToSingle.OUT3_TO_SINGLE_WEST11,
      OutMuxToSingle.OUT3_TO_SINGLE_WEST10, OutMuxToSingle.OUT3_TO_SINGLE_EAST11,
      OutMuxToSingle.OUT5_TO_SINGLE_WEST16, OutMuxToSingle.OUT5_TO_SINGLE_EAST15,
      OutMuxToSingle.OUT5_TO_SINGLE_EAST17, OutMuxToSingle.OUT5_TO_SINGLE_WEST17,
      OutMuxToSingle.OUT6_TO_SINGLE_WEST21, OutMuxToSingle.OUT6_TO_SINGLE_EAST18,
      OutMuxToSingle.OUT7_TO_SINGLE_EAST20, OutMuxToSingle.OUT7_TO_SINGLE_WEST23,
      OutMuxToSingle.OUT7_TO_SINGLE_WEST22, OutMuxToSingle.OUT7_TO_SINGLE_EAST23,
```

```
      OutMuxToSingle.OUT4_TO_SINGLE_WEST19, OutMuxToSingle.OUT4_TO_SINGLE_EAST14,
      OutMuxToSingle.OUT4_TO_SINGLE_NORTH12, OutMuxToSingle.OUT4_TO_SINGLE_SOUTH13,
      OutMuxToSingle.OUT6_TO_SINGLE_NORTH18, OutMuxToSingle.OUT6_TO_SINGLE_SOUTH19,
      OutMuxToSingle.OUT4_TO_SINGLE_SOUTH15, OutMuxToSingle.OUT4_TO_SINGLE_NORTH13,
      OutMuxToSingle.OUT5_TO_SINGLE_NORTH14, OutMuxToSingle.OUT6_TO_SINGLE_SOUTH17,
      OutMuxToSingle.OUT5_TO_SINGLE_SOUTH12, OutMuxToSingle.OUT6_TO_SINGLE_NORTH20,
      OutMuxToSingle.OUT7_TO_SINGLE_NORTH21, OutMuxToSingle.OUT7_TO_SINGLE_SOUTH22,
      OutMuxToSingle.OUT0_TO_SINGLE_NORTH0, OutMuxToSingle.OUT0_TO_SINGLE_SOUTH1,
      OutMuxToSingle.OUT2_TO_SINGLE_NORTH6, OutMuxToSingle.OUT2_TO_SINGLE_SOUTH7,
      OutMuxToSingle.OUT0_TO_SINGLE_SOUTH3, OutMuxToSingle.OUT0_TO_SINGLE_NORTH1,
      OutMuxToSingle.OUT1_TO_SINGLE_NORTH2, OutMuxToSingle.OUT2_TO_SINGLE_SOUTH5,
      OutMuxToSingle.OUT1_TO_SINGLE_SOUTH0, OutMuxToSingle.OUT2_TO_SINGLE_NORTH8,
      OutMuxToSingle.OUT3_TO_SINGLE_NORTH9, OutMuxToSingle.OUT3_TO_SINGLE_SOUTH10,
      UniHexToSingle.SINGLE_WEST3_TO_HEX_EAST4, UniHexToSingle.SINGLE_EAST0_TO_HEX_VERT_M4,
// Switch sencillo a 6 unidireccional
      UniHexToSingle.SINGLE_WEST2_TO_HEX_VERT_M4, UniHexToSingle.SINGLE_EAST3_TO_HEX_WEST5,
      UniHexToSingle.SINGLE_EAST5_TO_HEX_SOUTH5,
UniHexToSingle.SINGLE_EAST4_TO_HEX_VERT_M9,
      UniHexToSingle.SINGLE_WEST6_TO_HEX_VERT_M9,
UniHexToSingle.SINGLE_WEST8_TO_HEX_NORTH8,
      UniHexToSingle.SINGLE_EAST6_TO_HEX_WEST9, UniHexToSingle.SINGLE_WEST11_TO_HEX_EAST8,
      UniHexToSingle.SINGLE_EAST9_TO_HEX_VERT_M8,
UniHexToSingle.SINGLE_WEST13_TO_HEX_VERT_M8,
      UniHexToSingle.SINGLE_WEST10_TO_HEX_SOUTH9,
UniHexToSingle.SINGLE_EAST12_TO_HEX_VERT_M6,
      UniHexToSingle.SINGLE_WEST14_TO_HEX_VERT_M6,
UniHexToSingle.SINGLE_EAST21_TO_HEX_VERT_M10,
      UniHexToSingle.SINGLE_WEST1_TO_HEX_VERT_M10,
UniHexToSingle.SINGLE_EAST16_TO_HEX_VERT_M11,
      UniHexToSingle.SINGLE_WEST18_TO_HEX_VERT_M11,
UniHexToSingle.SINGLE_EAST10_TO_HEX_VERT_M7,
      UniHexToSingle.SINGLE_WEST12_TO_HEX_VERT_M7,
UniHexToSingle.SINGLE_WEST15_TO_HEX_EAST6,
      UniHexToSingle.SINGLE_WEST23_TO_HEX_EAST10,
UniHexToSingle.SINGLE_EAST15_TO_HEX_WEST7,
      UniHexToSingle.SINGLE_EAST18_TO_HEX_WEST11,
UniHexToSingle.SINGLE_WEST20_TO_HEX_NORTH10,
      UniHexToSingle.SINGLE_EAST13_TO_HEX_NORTH6,
UniHexToSingle.SINGLE_EAST17_TO_HEX_SOUTH7,
      UniHexToSingle.SINGLE_WEST22_TO_HEX_SOUTH11,
UniHexToSingle.HEX_SOUTH11_TO_SINGLE_NORTH22,
      UniHexToSingle.HEX_SOUTH7_TO_SINGLE_NORTH16,
UniHexToSingle.HEX_NORTH10_TO_SINGLE_SOUTH17,
      UniHexToSingle.HEX_NORTH6_TO_SINGLE_SOUTH13,
UniHexToSingle.HEX_WEST11_TO_SINGLE_NORTH20,
      UniHexToSingle.HEX_WEST7_TO_SINGLE_SOUTH18,
UniHexToSingle.HEX_EAST10_TO_SINGLE_NORTH21,
      UniHexToSingle.HEX_EAST6_TO_SINGLE_SOUTH15,
UniHexToSingle.HEX_HORIZ_M6_TO_SINGLE_SOUTH14,
      UniHexToSingle.HEX_HORIZ_M7_TO_SINGLE_NORTH17,
UniHexToSingle.HEX_HORIZ_M11_TO_SINGLE_NORTH23,
      UniHexToSingle.HEX_HORIZ_M10_TO_SINGLE_SOUTH23,
UniHexToSingle.HEX_HORIZ_M7_TO_SINGLE_SOUTH16,
      UniHexToSingle.HEX_HORIZ_M11_TO_SINGLE_SOUTH20,
UniHexToSingle.HEX_HORIZ_M10_TO_SINGLE_NORTH19,
      UniHexToSingle.HEX_HORIZ_M6_TO_SINGLE_NORTH15,
UniHexToSingle.HEX_WEST9_TO_SINGLE_NORTH8,
      UniHexToSingle.HEX_EAST8_TO_SINGLE_NORTH9, UniHexToSingle.HEX_WEST5_TO_SINGLE_SOUTH6,
      UniHexToSingle.HEX_SOUTH9_TO_SINGLE_NORTH10,
UniHexToSingle.HEX_NORTH8_TO_SINGLE_SOUTH5,
      UniHexToSingle.HEX_EAST4_TO_SINGLE_SOUTH3,
UniHexToSingle.HEX_HORIZ_M4_TO_SINGLE_SOUTH2,
      UniHexToSingle.HEX_SOUTH5_TO_SINGLE_NORTH4,
UniHexToSingle.HEX_HORIZ_M5_TO_SINGLE_NORTH5,
      UniHexToSingle.SINGLE_EAST22_TO_HEX_VERT_M5,
UniHexToSingle.HEX_HORIZ_M9_TO_SINGLE_NORTH11,
      UniHexToSingle.SINGLE_EAST1_TO_HEX_NORTH4,
UniHexToSingle.SINGLE_WEST0_TO_HEX_VERT_M5,
      UniHexToSingle.HEX_HORIZ_M8_TO_SINGLE_SOUTH11,
UniHexToSingle.HEX_HORIZ_M5_TO_SINGLE_SOUTH4,
      UniHexToSingle.HEX_HORIZ_M9_TO_SINGLE_SOUTH8,
UniHexToSingle.HEX_HORIZ_M8_TO_SINGLE_NORTH7,
      UniHexToSingle.HEX_HORIZ_M4_TO_SINGLE_NORTH3,
UniHexToSingle.HEX_NORTH4_TO_SINGLE_SOUTH1,
```

```
      HexEast0.HexEast0, HexEast1.HexEast1, HexEast2.HexEast2, HexEast3.HexEast3, // Switch
longitud 6
      HexEast5.HexEast5, HexEast7.HexEast7, HexEast9.HexEast9, HexEast11.HexEast11,
      HexNorth0.HexNorth0, HexNorth1.HexNorth1, HexNorth2.HexNorth2, HexNorth3.HexNorth3,
      HexNorth5.HexNorth5, HexNorth7.HexNorth7, HexNorth9.HexNorth9, HexNorth11.HexNorth11,
      HexWest0.HexWest0, HexWest1.HexWest1, HexWest2.HexWest2, HexWest3.HexWest3,
      HexWest4.HexWest4, HexWest6.HexWest6, HexWest8.HexWest8, HexWest10.HexWest10,
      HexSouth0.HexSouth0, HexSouth1.HexSouth1, HexSouth2.HexSouth2, HexSouth3.HexSouth3,
      HexSouth4.HexSouth4, HexSouth6.HexSouth6, HexSouth8.HexSouth8, HexSouth10.HexSouth10
    };

    int i;

    this.leerContenido();

    // Copiar todos los recursos de una clb a otra
    for(i = 0; i < recs.length; i++)
    {
      try
      {
        miJbit.set(clbydest, clbxdest, recs[i], miJbit.get(clby,clbx,recs[i]));
      }
      catch(Exception e)
      {
        System.out.println ("Imposible copiar recurso " + i);
      }
    }

    System.out.println("Copiada CLB (" + clbx + "," + clby + ") a (" + clbxdest + "," +
clbydest + ")");

    this.escribirContenido();
  }
}
```

Clase InterfazGrafica

```
package proyectofpga;

import javax.swing.UIManager;
import java.awt.*;

/**
 * <p>Título: </p>
 * <p>Descripción: </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Empresa: </p>
 * @author sin atribuir
 * @version 1.0
 */

public class InterfazGrafica {
  boolean packFrame = false;

  //Construir la aplicación
  public InterfazGrafica() {

    MarcoPrincipal frame = new MarcoPrincipal();
    frame.setSize(600,550);
    //Validar marcos que tienen tamaños preestablecidos
    //Empaquetar marcos que cuentan con información de tamaño preferente útil. Ej. de su
diseño.
    if (packFrame) {
      frame.pack();
    }
    else {
      frame.validate();
    }
    //Centrar la ventana
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
```

```
      frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height -
frameSize.height) / 2);
    frame.setVisible(true);
  }
  //Método Main
  public static void main(String[] args) {
    try {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    new InterfazGrafica();
  }
}
```

Apéndice B: Código del segundo prototipo

Libreria mtIOB

Clase Arbiter

```
package com.xilinx.JBits.Virtex.RTPCore.mtIOB;

import com.xilinx.JBits.Virtex.Expr;
import com.xilinx.JBits.Virtex.RTPCore.Tags;
import com.xilinx.JBits.Virtex.Bits.SliceControl;
import com.xilinx.JBits.Virtex.Bits.LUT;
import com.xilinx.JBits.Virtex.Bits.S0CE;
import com.xilinx.JBits.Virtex.ConfigurationException;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Bus;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;

public class Arbiter extends RTPCore
{
  // Puertos del core
  private Port requestPort;
  private Port clkPort;
  private Port responsePort;

  public Arbiter(String instanceName, Bus request, Net clk, Bus response) throws
CoreException
  {
    super(instanceName);

    // Comprobar los parametros del core
    try
    {
      checkParameters(request, response);
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear los puerto de entrada
    requestPort = newInputPort("REQUEST", request);
    clkPort = newInputPort("CLK", clk);

    // Crear el puerto de salida
    responsePort = newOutputPort("RESPONSE", response);
  }

  private void checkParameters(Bus request, Bus response) throws CoreParameterException
  {
    // Comprobar que el ancho de ambos buses es 4
    if(request.getWidth() != 4)
      throw new CoreParameterException(this, "El ancho del bus de entrada tiene que ser
exactamente 4");
```

```
    if(response.getWidth() != 4)
      throw new CoreParameterException(this, "El ancho del bus de salida tiene que ser
exactamente 4");
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.CLB;
  }

  // Dimensiones del core
  public static int calcHeight()
  {
    return 8;
  }

  public static int calcWidth()
  {
    return 2;
  }

  // Crea el contenido del core
  public final void implement() throws CoreException
  {
    CounterMod4 counter1;
    CoderLP4_4 cod[];
    Mux4_1 mux[];
    int row, col, i;
    Net clkNet;
    Bus countBus, requestBus, responseBus, prioBus[];

    // Calcular el offset
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
    col = offset.getHorOffset(Gran.CLB);

    // Crear las señales internas
    prioBus = new Bus[4];

    for(i = 0; i < 4; i++)
      prioBus[i] = newBus("prio" + i, 4);

    clkNet = newNet("clk");
    countBus = newBus("count", 2);
    requestBus = newBus("request", 4);
    responseBus = newBus("response", 4);

    // Conectar los puertos con las señales correspondientes
    clkPort.setIntSig(clkNet);
    requestPort.setIntSig(requestBus);
    responsePort.setIntSig(responseBus);

    // Crear el contador
    counter1 = new CounterMod4("counter1", clkNet, countBus);

    cod = new CoderLP4_4[4];
    mux = new Mux4_1[4];

    for(i = 0; i < 4; i++)
    {
      // Crear un codificador
      cod[i] = new CoderLP4_4("cod" + i, requestBus, prioBus[i], i);

      // Crear un multiplexor
      mux[i] = new Mux4_1("mux" + i, prioBus[0].getNet(i), prioBus[1].getNet(i),
prioBus[2].getNet(i), prioBus[3].getNet(i), countBus, responseBus.getNet(i));
    }

    // Añadir los cores
```

```
    addChild(counter1);

    for(i = 0; i < 4; i++)
    {
      addChild(cod[i]);
      addChild(mux[i]);
    }

    // Situar el contador
    Offset counter1offset = counter1.getRelativeOffset();
    counter1offset.setVerOffset(Gran.CLB, 3);
    counter1offset.setHorOffset(Gran.CLB, 1);
    counter1offset.setHorOffset(Gran.SLICE, 1);

    for(i = 0; i < 4; i++)
    {
      // Situar un codificador
      Offset codoffset = cod[i].getRelativeOffset();
      codoffset.setVerOffset(Gran.CLB, i);
      codoffset.setHorOffset(Gran.CLB, 0);

      // Situar un multiplexor
      Offset muxoffset = mux[i].getRelativeOffset();
      muxoffset.setVerOffset(Gran.CLB, i);
      muxoffset.setHorOffset(Gran.CLB, 1);
      muxoffset.setHorOffset(Gran.SLICE, 0);
    }

    // Implementar
    counter1.implement();

    for(i = 0; i < 4; i++)
    {
      cod[i].implement();
      mux[i].implement();
    }

    // Rutar las señales internas
    Bitstream.connect(countBus);

    for(i = 0; i < 4; i++)
      Bitstream.connect(prioBus[i]);
  }

  private void tagCLB(int row, int col, int tag) throws CoreException
  {
    try
    {
      Bitstream.getVirtex().setTag(row, col, tag);
    }
    catch (ConfigurationException ce)
    {
      throw new CoreException(ce);
    }
  }
}
```

Clase Buffer

```
package com.xilinx.JBits.Virtex.RTPCore.mtIOB;

import com.xilinx.JBits.Virtex.Bits.LUT;
import com.xilinx.JBits.Virtex.Bits.S0CE;
import com.xilinx.JBits.Virtex.Bits.S1CE;
import com.xilinx.JBits.Virtex.Expr;
import com.xilinx.JBits.Virtex.ConfigurationException;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Place;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
```

```
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;

public class Buffer extends RTPCore
{
  // Puertos del core
  private Port dinPort;
  private Port doutPort;

  public Buffer(String instanceName, Net din, Net dout) throws CoreException
  {
    super(instanceName);

    // Comprobar los parametros del core
    try
    {
      checkParameters();
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear el puerto de entrada
    dinPort = newInputPort("DIN", din);

    // Crear el puerto de salida
    doutPort = newOutputPort("DOUT", dout);
  }

  private void checkParameters() throws CoreParameterException
  {
    // En este caso no es necesario comprobar nada
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.CLB;
  }

  // Dimensiones del core
  public static int calcHeight()
  {
    return 1;
  }

  public static int calcWidth()
  {
    return 1;
  }

  // Crea el contenido del core
  public final void implement() throws CoreException
  {
    int row, col;
    Pin dinPin, doutPin;

    // Calcular el offset
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
    col = offset.getHorOffset(Gran.CLB);
```

```
    // La entrada es la salida
    Bitstream.set(row, col, LUT.F[0], Expr.F_LUT("~(F1)"));
    Bitstream.set(row,col, S0CE.S0CE, S0CE.OFF);

    // Crear el pin de la entrada
    dinPin = new Pin(Pin.CLB, row, col, CenterWires.SliceF1[0]);

    // Crear el pin de la salida
    doutPin = new Pin(Pin.CLB, row, col, CenterWires.Slice_X[0]);

    // Asignar los pines a los puertos
    dinPort.setPin(dinPin);
    doutPort.setPin(doutPin);

    // Marcar el buffer
    tagCLB(row,col, 0x30);
  }

  private void tagCLB(int row, int col, int tag) throws CoreException
  {
    try
    {
      Bitstream.getVirtex().setTag(row, col, tag);
    }
    catch (ConfigurationException ce)
    {
      throw new CoreException(ce);
    }
  }
}
```

Clase CoderLP4_4

```
package com.xilinx.JBits.Virtex.RTPCore.mtIOB;

import com.xilinx.JBits.Virtex.Expr;
import com.xilinx.JBits.Virtex.RTPCore.Tags;
import com.xilinx.JBits.Virtex.Bits.SliceControl;
import com.xilinx.JBits.Virtex.Bits.LUT;
import com.xilinx.JBits.Virtex.ConfigurationException;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Bus;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;

public class CoderLP4_4 extends RTPCore
{
  // Entrada con mayor prioridad
  private int prio;

  // Puertos del core
  private Port dinPort;
  private Port doutPort;

  public CoderLP4_4(String instanceName, Bus din, Bus dout, int prio) throws CoreException
  {
    super(instanceName);

    // Comprobar los parametros del core
    try
    {
      checkParameters(din, dout, prio);
    }
    catch (CoreParameterException cpe)
    {
```

```
      throw new CoreException(cpe);
    }

    // Establecer las propiedades
    this.prio = prio;

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear los puertos
    dinPort = newInputPort("DIN", din);
    doutPort = newOutputPort("DOUT", dout);
  }

  private void checkParameters(Bus din, Bus dout, int prio) throws CoreParameterException
  {
    // Comprobar que el ancho del bus de seleccion es exactamente 4
    if(din.getWidth() != 4)
      throw new CoreParameterException(this, "El ancho del bus de entrada tiene que ser
exactamente 4");

    // Comprobar que el ancho del bus de entrada es exactamente 4
    if(dout.getWidth() != 4)
      throw new CoreParameterException(this, "El ancho del bus de salida tiene que ser
exactamente 4");

    // Comprobar que la entrada prioritaria esta en [0, 3]
    if(prio < 0 || prio > 3)
      throw new CoreParameterException(this, "Entrada prioritaria fuera de rango 0 .. 3");
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.CLB;
  }

  // Dimensiones del core
  public static int calcHeight()
  {
    return 1;
  }

  public static int calcWidth()
  {
    return 1;
  }

  // Crea el contenido del core
  public final void implement() throws CoreException
  {
    int row, col, i, j;
    Pin dinPin[][], doutPin[];

    // Calcular el offset
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
    col = offset.getHorOffset(Gran.CLB);

    // Crear los pines
    dinPin = new Pin[4][4];
    doutPin = new Pin[4];

    for(i = 0; i < 2; i++)
    {
      dinPin[2 * i][0] = new Pin(Pin.CLB, row, col, CenterWires.SliceF1[i]);
      dinPin[2 * i][1] = new Pin(Pin.CLB, row, col, CenterWires.SliceF2[i]);
```

```
      dinPin[2 * i][2] = new Pin(Pin.CLB, row, col, CenterWires.SliceF3[i]);
      dinPin[2 * i][3] = new Pin(Pin.CLB, row, col, CenterWires.SliceF4[i]);

      dinPin[2 * i + 1][0] = new Pin(Pin.CLB, row, col, CenterWires.SliceG1[i]);
      dinPin[2 * i + 1][1] = new Pin(Pin.CLB, row, col, CenterWires.SliceG2[i]);
      dinPin[2 * i + 1][2] = new Pin(Pin.CLB, row, col, CenterWires.SliceG3[i]);
      dinPin[2 * i + 1][3] = new Pin(Pin.CLB, row, col, CenterWires.SliceG4[i]);

      doutPin[2 * i] = new Pin(Pin.CLB, row, col, CenterWires.Slice_X[i]);
      doutPin[2 * i + 1] = new Pin(Pin.CLB, row, col, CenterWires.Slice_Y[i]);
    }

    // Conectar los puertos con los pines correspondientes teniendo en cuenta la entrada
prioritaria
    for(i = 0; i < 4; i++)
    {
      j = (i + 3 - prio) % 4;

      dinPort.setPin(i, dinPin[j]);
      doutPort.setPin(i, doutPin[j]);
    }

    // Configurar la CLB para el codificador de prioridad izquierda 4 a 4

    // Configurar las luts
    Bitstream.set(row, col, LUT.F[0], Expr.F_LUT("~(F1 & ~F2 & ~F3 & ~F4)"));
    Bitstream.set(row, col, LUT.G[0], Expr.G_LUT("~(G2 & ~G3 & ~G4)"));
    Bitstream.set(row, col, LUT.F[1], Expr.F_LUT("~(F3 & ~F4)"));
    Bitstream.set(row, col, LUT.G[1], Expr.G_LUT("~(G4)"));

    // Las salidas X e Y viene de la LUT en ambos slices
    for(i = 0; i < 2; i++)
    {
      Bitstream.set(row, col, SliceControl.X.X[i], SliceControl.X.FOUT[i]);
      Bitstream.set(row, col, SliceControl.Y.Y[i], SliceControl.Y.GOUT[i]);
    }

    // Marcar el core como multiplexor
    tagCLB(row, col, Tags.BLACKBOX);
  }

  private void tagCLB(int row, int col, int tag) throws CoreException
  {
    try
    {
      Bitstream.getVirtex().setTag(row, col, tag);
    }
    catch (ConfigurationException ce)
    {
      throw new CoreException(ce);
    }
  }
}
```

Clase ConstantComparator

```
package com.xilinx.JBits.Virtex.RTPCore.mtIOB;

import com.xilinx.JBits.Virtex.Expr;
import com.xilinx.JBits.Virtex.RTPCore.Tags;
import com.xilinx.JBits.Virtex.Bits.SliceControl;
import com.xilinx.JBits.Virtex.Bits.LUT;
import com.xilinx.JBits.Virtex.Bits.S0CE;
import com.xilinx.JBits.Virtex.Bits.S0BX;
import com.xilinx.JBits.Virtex.Bits.S0BY;
import com.xilinx.JBits.Virtex.Bits.S1CE;
import com.xilinx.JBits.Virtex.Bits.S1BX;
import com.xilinx.JBits.Virtex.Bits.S1BY;
import com.xilinx.JBits.Virtex.ConfigurationException;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Bus;
import com.xilinx.JBits.CoreTemplate.Pin;
```

```
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Place;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;

public class ConstantComparator extends RTPCore
{
  // Propiedades del core
  private int cte;

  // Puertos del core
  private Port dinPort;
  private Port matchPort;

  public ConstantComparator(String instanceName, Bus din, Net match, int cte) throws
CoreException
  {
    super(instanceName);

    // Comprobar los parametros del core
    try
    {
      checkParameters(din, cte);
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Guardar la constante
    this.cte = cte;

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear los puertos
    dinPort = newInputPort("DIN", din);
    matchPort = newOutputPort("MATCH", match);
  }

  private void checkParameters(Bus din, int cte) throws CoreParameterException
  {
    // Comprobar que la longitud del bus es exactamente 4
    if(din.getWidth() != 4)
      throw new CoreParameterException(this, "La longitud del bus de entrada tiene que ser
exactamente 4");

    // Comprobar que la constante es un numero entre 0 y 15
    if(cte < 0 || cte > 15)
      throw new CoreParameterException(this, "Constante fuera del rango [0, 15]");
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.SLICE;
  }

  // Dimensiones del core
  public static int calcHeight()
  {
    return 1;
  }
```

```
  public static int calcWidth()
  {
    return 1;
  }

  // Crea el contenido del core
  public final void implement() throws CoreException
  {
    int row, col, slice, i, bitcte[];
    Pin dinPin[], matchPin;

    // Calcular el offset
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
    col = offset.getHorOffset(Gran.CLB);
    slice = offset.getHorOffset(Gran.SLICE);

    // Crear los pines
    dinPin = new Pin[4];
    dinPin[0] = new Pin(Pin.CLB, row, col, CenterWires.SliceF1[slice]);
    dinPin[1] = new Pin(Pin.CLB, row, col, CenterWires.SliceF2[slice]);
    dinPin[2] = new Pin(Pin.CLB, row, col, CenterWires.SliceF3[slice]);
    dinPin[3] = new Pin(Pin.CLB, row, col, CenterWires.SliceF4[slice]);
    matchPin = new Pin(Pin.CLB, row, col, CenterWires.Slice_X[slice]);

    // Conectar los puertos con los pines correspondientes
    matchPort.setPin(matchPin);

    for(i = 0; i < 4; i++)
      dinPort.setPin(i, dinPin[i]);

    // Configurar la CLB como comparador

    // Crear la expresion de la clb
    bitcte = new int[16];

    for(i = 0; i < 16; i++)
      bitcte[i] = (i == cte) ? 0 : 1;

    // Comparar con el numero indicado
    Bitstream.set(row, col, LUT.F[slice], bitcte);

    // Establecer que X sea la salida de la lut
    Bitstream.set(row, col, SliceControl.X.X[slice], SliceControl.X.FOUT[slice]);

    // Marcar el core como test (marca toda la clb, puede que lo pisotee otro core)
    tagCLB(row, col, Tags.TEST_INPUT);
  }

  private void tagCLB(int row, int col, int tag) throws CoreException
  {
    try
    {
      Bitstream.getVirtex().setTag(row, col, tag);
    }
    catch (ConfigurationException ce)
    {
      throw new CoreException(ce);
    }
  }
}
```

Clase Counter

```
package com.xilinx.JBits.Virtex.RTPCore.mtIOB;

import com.xilinx.JBits.Virtex.Expr;
import com.xilinx.JBits.Virtex.RTPCore.Tags;
import com.xilinx.JBits.Virtex.Bits.SliceControl;
import com.xilinx.JBits.Virtex.Bits.LUT;
import com.xilinx.JBits.Virtex.Bits.S0CE;
import com.xilinx.JBits.Virtex.Bits.S0BX;
import com.xilinx.JBits.Virtex.Bits.S0BY;
import com.xilinx.JBits.Virtex.Bits.S1CE;
```

```
import com.xilinx.JBits.Virtex.Bits.S1BX;
import com.xilinx.JBits.Virtex.Bits.S1BY;
import com.xilinx.JBits.Virtex.ConfigurationException;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Bus;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Place;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;

public class Counter extends RTPCore
{
  // Puertos del core
  private Port clkPort;
  private Port rstPort;
  private Port qPort;
  private Port qrPort; // Realimentacion del estado

  public Counter(String instanceName, Net clk, Net rst, Bus q) throws CoreException
  {
    super(instanceName);

    // Comprobar los parametros del core
    try
    {
      checkParameters(q);
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear los puertos de entrada
    clkPort = newInputPort("CLK", clk);
    rstPort = newInputPort("RST", rst);

    // Crear el puerto de salida
    qPort = newOutputPort("Q", q);
    qrPort = newInputPort("QR", q);
  }

  private void checkParameters(Bus q) throws CoreParameterException
  {
    // Comprobar que la longitud del bus es exactamente 4
    if(q.getWidth() != 4)
      throw new CoreParameterException(this, "La longitud del bus de salida tiene que ser
exactamente 4");
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.SLICE;
  }

  // Dimensiones del core
```

```
public static int calcHeight()
{
  return 2;
}

public static int calcWidth()
{
  return 1;
}

// Crea el contenido del core
public final void implement() throws CoreException
{
  int row, col, slice, i;
  Pin clkPin[], rstPin[], qPin[], qrPin[];

  // Calcular el offset
  Offset offset = calcAbsoluteOffset();
  row = offset.getVerOffset(Gran.CLB);
  col = offset.getHorOffset(Gran.CLB);
  slice = offset.getHorOffset(Gran.SLICE);

  // Crear los pines
  clkPin = new Pin[2];
  rstPin = new Pin[2];
  qPin = new Pin[4];
  qrPin = new Pin[4];

  for(i = 0; i < 2; i++)
  {
    // Reloj
    clkPin[i] = new Pin(Pin.CLB, row + i, col, CenterWires.SliceClk[slice]);

    // Reset
    rstPin[i] = new Pin(Pin.CLB, row + i, col, CenterWires.SliceSR[slice]);

    // Estado actual
    qPin[2 * i] = new Pin(Pin.CLB, row + i, col, CenterWires.Slice_XQ[slice]);
    qPin[2 * i + 1] = new Pin(Pin.CLB, row + i, col, CenterWires.Slice_YQ[slice]);

    // Realimentacion del estado actual
    qrPin[2 * i] = new Pin(Pin.CLB, row + i, col, CenterWires.SliceF1[slice]);
    qrPin[2 * i + 1] = new Pin(Pin.CLB, row + i, col, CenterWires.SliceG1[slice]);
  }

  // Conectar los puertos con los pines correspondientes
  clkPort.setPin(clkPin);
  rstPort.setPin(rstPin);

  for(i = 0; i < 4; i++)
  {
    qPort.setPin(i, qPin[i]);
    qrPort.setPin(i, qrPin[i]);
  }

  // Configurar las CLBs para el contador

  // Acarreo de la primera etapa siempre 1 por entrada BX
  Bitstream.set(row, col, SliceControl.Cin.Cin[slice], SliceControl.Cin.BX[slice]);
  Bitstream.set(row, col, SliceControl.BxInvert[slice], SliceControl.OFF[slice]);

  if(slice == 0)
    Bitstream.set(row, col, S0BX.S0BX, S0BX.OFF);
  else
    Bitstream.set(row, col, S1BX.S1BX, S1BX.OFF);


  // Acarreo de la segunda se obtiene desde la primera etapa
  Bitstream.set(row + 1, col, SliceControl.Cin.Cin[slice], SliceControl.Cin.CIN[slice]);

  for(i = 0; i < 2; i++)
  {
    // Poner el modo flip flop
    Bitstream.set(row + i, col, SliceControl.LatchMode[slice], SliceControl.OFF[slice]);
```

```
      // Reloj no invertido
      Bitstream.set(row + i, col, SliceControl.ClockInvert[slice],
SliceControl.OFF[slice]);

      // SR es la señal de reset para ambos flip flps
      Bitstream.set(row + i, col, SliceControl.XffSetResetSelect[slice] ,
SliceControl.OFF[slice]);
      Bitstream.set(row + i, col, SliceControl.YffSetResetSelect[slice] ,
SliceControl.OFF[slice]);

      // Reset no invertido en ambas etapas
      Bitstream.set(row + i, col, SliceControl.SrWeNotInvert[slice],
SliceControl.ON[slice]);

      // Set/Reset sincrono
      Bitstream.set(row + i, col, SliceControl.Sync[slice], SliceControl.ON[slice]);

      // BY es la señal de set en ambos flip flops. Desconectarla
      // CE siempre activo
      if(slice == 0)
      {
        Bitstream.set(row + i, col, S0BY.S0BY , S0BY.OFF);
        Bitstream.set(row + i, col, S0CE.S0CE, S0CE.OFF);
      }
      else
      {
        Bitstream.set(row + i, col, S1BY.S1BY , S1BY.OFF);
        Bitstream.set(row + i, col, S1CE.S1CE, S1CE.OFF);
      }

      // By no invertido
      Bitstream.set(row + i, col, SliceControl.ByInvert[slice], SliceControl.OFF[slice]);

      // CE no invertido
      Bitstream.set(row + i, col, SliceControl.CeInvert[slice] , SliceControl.OFF[slice]);

      // Configurar las luts para que sean buffers de la entrada F1 y G1
      Bitstream.set(row + i, col, LUT.F[slice], Expr.F_LUT("~F1"));
      Bitstream.set(row + i, col, LUT.G[slice], Expr.G_LUT("~G1"));

      // Hacer que la salida X sea la salida de la lut XOR el acarreo
      Bitstream.set(row + i, col, SliceControl.X.X[slice],
SliceControl.X.FOUT_XOR_CARRY[slice]);

      // Hacer que la salida Y sea la salida de la lut XOR el acarreo
      Bitstream.set(row + i, col, SliceControl.Y.Y[slice],
SliceControl.Y.GOUT_XOR_CARRY[slice]);

      // Pasar X e Y a la entrada del flip flop
      Bitstream.set(row + i, col, SliceControl.XDin.XDin[slice],
SliceControl.XDin.X[slice]);
      Bitstream.set(row + i, col, SliceControl.YDin.YDin[slice],
SliceControl.YDin.Y[slice]);

      // Hacer que el multiplexor And devuelva siempre 0
      Bitstream.set(row + i, col, SliceControl.AndMux.AndMux[slice],
SliceControl.AndMux.ZERO[slice]);

      // Seleccion del origen del acarreo controlado por la lut
      Bitstream.set(row + i, col, SliceControl.XCarrySelect.XCarrySelect[slice],
SliceControl.XCarrySelect.LUT_CONTROL[slice]);
      Bitstream.set(row + i, col, SliceControl.YCarrySelect.YCarrySelect[slice],
SliceControl.YCarrySelect.LUT_CONTROL[slice]);

      // Marcar el core como contador (marca toda la clb, puede que lo pisotee otro core)
      tagCLB(row + i, col, Tags.COUNTER);
    }
  }

  private void tagCLB(int row, int col, int tag) throws CoreException
  {
    try
    {
      Bitstream.getVirtex().setTag(row, col, tag);
    }
```

```
    catch (ConfigurationException ce)
    {
      throw new CoreException(ce);
    }
  }
}
```

Clase CounterMod4

```
package com.xilinx.JBits.Virtex.RTPCore.mtIOB;

import com.xilinx.JBits.Virtex.Expr;
import com.xilinx.JBits.Virtex.RTPCore.Tags;
import com.xilinx.JBits.Virtex.Bits.SliceControl;
import com.xilinx.JBits.Virtex.Bits.LUT;
import com.xilinx.JBits.Virtex.Bits.S0CE;
import com.xilinx.JBits.Virtex.Bits.S0SR;
import com.xilinx.JBits.Virtex.Bits.S0BX;
import com.xilinx.JBits.Virtex.Bits.S0BY;
import com.xilinx.JBits.Virtex.Bits.S1CE;
import com.xilinx.JBits.Virtex.Bits.S1SR;
import com.xilinx.JBits.Virtex.Bits.S1BX;
import com.xilinx.JBits.Virtex.Bits.S1BY;
import com.xilinx.JBits.Virtex.ConfigurationException;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Bus;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Place;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;

public class CounterMod4 extends RTPCore
{
  // Puertos del core
  private Port clkPort;
  private Port qPort;
  private Port qrPort; // Realimentacion del estado

  public CounterMod4(String instanceName, Net clk, Bus q) throws CoreException
  {
    super(instanceName);

    // Comprobar los parametros del core
    try
    {
      checkParameters(q);
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear el puerto de entrada
    clkPort = newInputPort("CLK", clk);

    // Crear el puerto de salida
    qPort = newOutputPort("Q", q);
    qrPort = newInputPort("QR", q);
  }

  private void checkParameters(Bus q) throws CoreParameterException
```

```
  {
    // Comprobar que la longitud del bus es exactamente 4
    if(q.getWidth() != 2)
      throw new CoreParameterException(this, "La longitud del bus de salida tiene que ser
exactamente 2");
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.SLICE;
  }

  // Dimensiones del core
  public static int calcHeight()
  {
    return 1;
  }

  public static int calcWidth()
  {
    return 1;
  }

  // Crea el contenido del core
  public final void implement() throws CoreException
  {
    int row, col, slice, i;
    Pin clkPin, qPin[], qrPin[];

    // Calcular el offset
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
    col = offset.getHorOffset(Gran.CLB);
    slice = offset.getHorOffset(Gran.SLICE);

    // Crear los pines
    qPin = new Pin[2];
    qrPin = new Pin[2];

    // Reloj
    clkPin = new Pin(Pin.CLB, row, col, CenterWires.SliceClk[slice]);

    // Estado actual
    qPin[0] = new Pin(Pin.CLB, row, col, CenterWires.Slice_XQ[slice]);
    qPin[1] = new Pin(Pin.CLB, row, col, CenterWires.Slice_YQ[slice]);

    // Realimentacion del estado actual
    qrPin[0] = new Pin(Pin.CLB, row, col, CenterWires.SliceF1[slice]);
    qrPin[1] = new Pin(Pin.CLB, row, col, CenterWires.SliceG1[slice]);

    // Conectar los puertos con los pines correspondientes
    clkPort.setPin(clkPin);

    for(i = 0; i < 2; i++)
    {
      qPort.setPin(i, qPin[i]);
      qrPort.setPin(i, qrPin[i]);
    }

    // Configurar las CLBs para el contador

    // Acarreo siempre 1 por entrada BX
    Bitstream.set(row, col, SliceControl.Cin.Cin[slice], SliceControl.Cin.BX[slice]);
    Bitstream.set(row, col, SliceControl.BxInvert[slice], SliceControl.OFF[slice]);

    if(slice == 0)
      Bitstream.set(row, col, S0BX.S0BX, S0BX.OFF);
    else
      Bitstream.set(row, col, S1BX.S1BX, S1BX.OFF);
```

```
    // Poner el modo flip flop
    Bitstream.set(row, col, SliceControl.LatchMode[slice], SliceControl.OFF[slice]);

    // Reloj no invertido
    Bitstream.set(row, col, SliceControl.ClockInvert[slice], SliceControl.OFF[slice]);

    // SR es la señal de reset para ambos flip flps
    Bitstream.set(row, col, SliceControl.XffSetResetSelect[slice] ,
SliceControl.OFF[slice]);
    Bitstream.set(row, col, SliceControl.YffSetResetSelect[slice] ,
SliceControl.OFF[slice]);

    // Reset invertido en ambas etapas
    Bitstream.set(row, col, SliceControl.SrWeNotInvert[slice], SliceControl.OFF[slice]);

    // Set/Reset asincrono
    Bitstream.set(row, col, SliceControl.Sync[slice], SliceControl.OFF[slice]);

    // BY es la señal de set en ambos flip flops. Desconectarla
    // CE siempre activo, SR siempre activa
    if(slice == 0)
    {
      Bitstream.set(row, col, S0BY.S0BY , S0BY.OFF);
      Bitstream.set(row, col, S0CE.S0CE, S0CE.OFF);
      Bitstream.set(row, col, S0CE.S0CE, S0CE.OFF);
    }
    else
    {
      Bitstream.set(row, col, S1BY.S1BY , S1BY.OFF);
      Bitstream.set(row, col, S1CE.S1CE, S1CE.OFF);
      Bitstream.set(row, col, S1CE.S1CE, S1CE.OFF);
    }

    // By no invertido
    Bitstream.set(row, col, SliceControl.ByInvert[slice], SliceControl.OFF[slice]);

    // CE no invertido
    Bitstream.set(row, col, SliceControl.CeInvert[slice] , SliceControl.OFF[slice]);

    // Configurar las luts para que sean buffers de la entrada F1 y G1
    Bitstream.set(row, col, LUT.F[slice], Expr.F_LUT("~F1"));
    Bitstream.set(row, col, LUT.G[slice], Expr.G_LUT("~G1"));

    // Hacer que la salida X sea la salida de la lut XOR el acarreo
    Bitstream.set(row, col, SliceControl.X.X[slice], SliceControl.X.FOUT_XOR_CARRY[slice]);

    // Hacer que la salida Y sea la salida de la lut XOR el acarreo
    Bitstream.set(row, col, SliceControl.Y.Y[slice], SliceControl.Y.GOUT_XOR_CARRY[slice]);

    // Pasar X e Y a la entrada del flip flop
    Bitstream.set(row, col, SliceControl.XDin.XDin[slice], SliceControl.XDin.X[slice]);
    Bitstream.set(row, col, SliceControl.YDin.YDin[slice], SliceControl.YDin.Y[slice]);

    // Hacer que el multiplexor And devuelva siempre 0
    Bitstream.set(row, col, SliceControl.AndMux.AndMux[slice],
SliceControl.AndMux.ZERO[slice]);

    // Seleccion del origen del acarreo controlado por la lut
    Bitstream.set(row, col, SliceControl.XCarrySelect.XCarrySelect[slice],
SliceControl.XCarrySelect.LUT_CONTROL[slice]);
    Bitstream.set(row, col, SliceControl.YCarrySelect.YCarrySelect[slice],
SliceControl.YCarrySelect.LUT_CONTROL[slice]);

    // Marcar el core como contador (marca toda la clb, puede que lo pisotee otro core)
    tagCLB(row, col, Tags.COUNTER);
  }

  private void tagCLB(int row, int col, int tag) throws CoreException
  {
    try
    {
      Bitstream.getVirtex().setTag(row, col, tag);
    }
    catch (ConfigurationException ce)
```

```
    {
      throw new CoreException(ce);
    }
  }
}
```

Clase mtBoard

```
package com.xilinx.JBits.Virtex.RTPCore.mtIOB;

import com.xilinx.JBits.Virtex.RTPCore.Iob.Board;
import com.xilinx.JBits.Virtex.RTPCore.Iob.XCVPackage;
import com.xilinx.JBits.Virtex.RTPCore.Iob.xcv1000_bg560;

import com.xilinx.JBits.CoreTemplate.CoreException;

public class mtBoard extends Board
{
  // Empaquetado. Uno por cada fpga. Solo tenemos 1 en la placa.
  private XCVPackage xcvPackage[] = { new xcv1000_bg560() };

  // El reloj global
  private int GCLK = 1;

  public mtBoard(String instanceName) throws CoreException
  {
    super(instanceName);

    // Establecer el empaquetado de la placa
    setXcvPackage(xcvPackage);

    // Definir la configuracion de la fpga
    setGCLK(GCLK);
  }
}
```

Clase mtIOB

```
package com.xilinx.JBits.Virtex.RTPCore.mtIOB;

import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Bus;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JBits.Virtex.RTPCore.Tags;
import com.xilinx.JBits.Virtex.ConfigurationException;
import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;

public class mtIOB extends RTPCore
{
  // Definicion de la altura del core
  private final static int rows = 1;

  // Puertos de entrada del core
  private Port dinAPort; // Entrada desde la tarea A
  private Port dinBPort; // Entrada desde la tarea B
  private Port dinCPort; // Entrada desde la tarea C
  private Port dinDPort; // Entrada desde la tarea D
  private Port clkPort; // Entrada del reloj

  // Puertos de salida del core
  private Port doutPort; // Salida a los leds
  private Port clkAPort; // Reloj para la tarea A
  private Port clkBPort; // Reloj para la tarea B
  private Port clkCPort; // Reloj para la tarea C
  private Port clkDPort; // Reloj para la tarea D

  public mtIOB(String instanceName,
```

```
              Net dinA,
              Net clkA,
              Net dinB,
              Net clkB,
              Net dinC,
              Net clkC,
              Net dinD,
              Net clkD,
              Net clk,
              Bus dout) throws CoreException
  {
    super(instanceName);

    // Comprobar los parametros del core
    try
    {
      checkParameters(dout);
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear los puertos de entrada
    dinAPort = newInputPort("DINA", dinA);
    dinBPort = newInputPort("DINB", dinB);
    dinCPort = newInputPort("DINC", dinC);
    dinDPort = newInputPort("DIND", dinD);
    clkPort = newInputPort("CLK", clk);

    // Crear los puertos de salida
    doutPort = newOutputPort("DOUT", dout);
    clkAPort = newOutputPort("CLKA", clkA);
    clkBPort = newOutputPort("CLKB", clkB);
    clkCPort = newOutputPort("CLKC", clkC);
    clkDPort = newOutputPort("CLKD", clkD);
  }

  private void checkParameters(Bus dout) throws CoreParameterException
  {
    // Comprobar que el ancho del bus de salida es exactamente 4
    // (uno por cada tarea).
    if(dout.getWidth() != 4)
      throw new CoreParameterException(this, "El ancho del bus de salida tiene que ser
4.");
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.CLB;
  }

  // Dimensiones del core
  public static int calcHeight()
  {
    // Este valor se define al principio de la clase
    return rows;
  }

  public static int calcWidth()
  {
    // Ocupa toda una fila de la FPGA
    return Bitstream.getVirtex().getClbColumns();
```

```
}

public final void implement() throws CoreException
{
  int row, col, i, j;
  Offset bufferOffset;
  Register registerDinA, registerDinB, registerDinC, registerDinD;
  Net dinANet, dinBNet, dinCNet, dinDNet;
  Net clkNet;
  Bus doutBus;

  // Calcular el offset
  Offset offset = calcAbsoluteOffset();
  row = offset.getVerOffset(Gran.CLB);
  col = offset.getHorOffset(Gran.CLB);

  // Crear las señales internas
  dinANet = newNet("dinA");
  dinBNet = newNet("dinB");
  dinCNet = newNet("dinC");
  dinDNet = newNet("dinD");
  doutBus = newBus("data", 4);
  clkNet = newNet("clk");

  // Conectar puertos a la net correspondiente
  dinAPort.setIntSig(dinANet);
  clkAPort.setIntSig(clkNet);

  dinBPort.setIntSig(dinBNet);
  clkBPort.setIntSig(clkNet);

  dinCPort.setIntSig(dinCNet);
  clkCPort.setIntSig(clkNet);

  dinDPort.setIntSig(dinDNet);
  clkDPort.setIntSig(clkNet);

  doutPort.setIntSig(doutBus);
  clkPort.setIntSig(clkNet);

  // Crear los buffers para las entradas y salidas con las tareas
  registerDinA = new Register("registerDinA", dinANet, clkNet, doutBus.getNet(0));
  registerDinB = new Register("registerDinB", dinBNet, clkNet, doutBus.getNet(1));
  registerDinC = new Register("registerDinC", dinCNet, clkNet, doutBus.getNet(2));
  registerDinD = new Register("registerDinD", dinDNet, clkNet, doutBus.getNet(3));

  // Situar los buffers a intervalos regulares
  bufferOffset = registerDinA.getRelativeOffset();
  bufferOffset.setVerOffset(Gran.CLB, 0);
  bufferOffset.setHorOffset(Gran.CLB, 0);

  bufferOffset = registerDinB.getRelativeOffset();
  bufferOffset.setVerOffset(Gran.CLB, 0);
  bufferOffset.setHorOffset(Gran.CLB, 24);

  bufferOffset = registerDinC.getRelativeOffset();
  bufferOffset.setVerOffset(Gran.CLB, 0);
  bufferOffset.setHorOffset(Gran.CLB, 48);

  bufferOffset = registerDinD.getRelativeOffset();
  bufferOffset.setVerOffset(Gran.CLB, 0);
  bufferOffset.setHorOffset(Gran.CLB, 72);

  // Añadir los buffers
  addChild(registerDinA);
  addChild(registerDinB);
  addChild(registerDinC);
  addChild(registerDinD);

  // Marcar todas las CLBs
  for (i = col; i < col + getWidth(); i++)
    for (j = row; j < row + getHeight(); j++)
      tagCLB(j, i, Tags.BLACKBOX);

  // Implementarlos
```

```
    registerDinA.implement();
    registerDinB.implement();
    registerDinC.implement();
    registerDinD.implement();
  }

  private void tagCLB(int row, int col, int tag) throws CoreException
  {
    try
    {
      Bitstream.getVirtex().setTag(row, col, tag);
    }
    catch (ConfigurationException ce)
    {
      throw new CoreException(ce);
    }
  }
}
```

Clase Mux4_1

```
package com.xilinx.JBits.Virtex.RTPCore.mtIOB;

import com.xilinx.JBits.Virtex.Expr;
import com.xilinx.JBits.Virtex.RTPCore.Tags;
import com.xilinx.JBits.Virtex.Bits.SliceControl;
import com.xilinx.JBits.Virtex.Bits.LUT;
import com.xilinx.JBits.Virtex.ConfigurationException;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Bus;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;

public class Mux4_1 extends RTPCore
{
  // Puertos del core
  private Port din0Port;
  private Port din1Port;
  private Port din2Port;
  private Port din3Port;
  private Port selPort;
  private Port doutPort;

  public Mux4_1(String instanceName, Net din0, Net din1, Net din2, Net din3, Bus sel, Net
dout) throws CoreException
  {
    super(instanceName);

    // Comprobar los parametros del core
    try
    {
      checkParameters(sel);
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear los puertos de entrada
    din0Port = newInputPort("DIN0", din0);
```

```
    din1Port = newInputPort("DIN1", din1);
    din2Port = newInputPort("DIN2", din2);
    din3Port = newInputPort("DIN3", din3);
    selPort = newInputPort("SEL", sel);

    // Crear el puerto de salida
    doutPort = newOutputPort("DOUT", dout);
  }

  private void checkParameters(Bus sel) throws CoreParameterException
  {
    // Comprobar que el ancho del bus de seleccion es exactamente 2
    if(sel.getWidth() != 2)
      throw new CoreParameterException(this, "El ancho del bus de seleccion tiene que ser
exactamente 2");
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.SLICE;
  }

  // Dimensiones del core
  public static int calcHeight()
  {
    return 2;
  }

  public static int calcWidth()
  {
    return 1;
  }

  // Crea el contenido del core
  public final void implement() throws CoreException
  {
    int row, col, slice, i;
    Pin din0Pin, din1Pin, din2Pin, din3Pin, sel0Pin[], sel1Pin, doutPin;

    // Calcular el offset
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
    col = offset.getHorOffset(Gran.CLB);
    slice = offset.getHorOffset(Gran.SLICE);

    // Crear los pines
    sel0Pin = new Pin[2];

    din0Pin = new Pin(Pin.CLB, row, col, CenterWires.SliceG3[slice]);
    din1Pin = new Pin(Pin.CLB, row, col, CenterWires.SliceG2[slice]);
    din2Pin = new Pin(Pin.CLB, row, col, CenterWires.SliceF3[slice]);
    din3Pin = new Pin(Pin.CLB, row, col, CenterWires.SliceF2[slice]);
    doutPin = new Pin(Pin.CLB, row,col, CenterWires.Slice_X[slice]);
    sel0Pin[0] = new Pin(Pin.CLB, row, col, CenterWires.SliceG1[slice]);
    sel0Pin[1] = new Pin(Pin.CLB, row, col, CenterWires.SliceF1[slice]);
    sel1Pin = new Pin(Pin.CLB, row, col, CenterWires.SliceBX[slice]);

    // Conectar los puertos con los pines correspondientes
    din0Port.setPin(din0Pin);
    din1Port.setPin(din1Pin);
    din2Port.setPin(din2Pin);
    din3Port.setPin(din3Pin);
    selPort.setPin(0, sel0Pin);
    selPort.setPin(1, sel1Pin);
    doutPort.setPin(doutPin);

    // Configurar las CLBs para el multiplexor 4 a 1

    // BX no invertido
```

```
    Bitstream.set(row, col, SliceControl.BxInvert[slice], SliceControl.OFF[slice]);

    // Configurar las luts para que sean multiplexores 2 a 1
    Bitstream.set(row, col, LUT.F[slice], Expr.F_LUT("~((F1&F2)|(~F1&F3))"));
    Bitstream.set(row, col, LUT.G[slice], Expr.G_LUT("~((G1&G2)|(~G1&G3))"));

    // La salida X viene de F5
    Bitstream.set(row, col, SliceControl.X.X[slice], SliceControl.X.F5[slice]);

    // Marcar el core como multiplexor
    tagCLB(row, col, Tags.BLACKBOX);
  }

  private void tagCLB(int row, int col, int tag) throws CoreException
  {
    try
    {
      Bitstream.getVirtex().setTag(row, col, tag);
    }
    catch (ConfigurationException ce)
    {
      throw new CoreException(ce);
    }
  }
}
```

Clase Register

```
package com.xilinx.JBits.Virtex.RTPCore.mtIOB;

import com.xilinx.JBits.Virtex.Expr;
import com.xilinx.JBits.Virtex.RTPCore.Tags;
import com.xilinx.JBits.Virtex.Bits.SliceControl;
import com.xilinx.JBits.Virtex.Bits.LUT;
import com.xilinx.JBits.Virtex.Bits.S0CE;
import com.xilinx.JBits.Virtex.ConfigurationException;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;

public class Register extends RTPCore
{
  // Puertos del core
  private Port dinPort;
  private Port clkPort;
  private Port doutPort;

  public Register(String instanceName, Net din, Net clk, Net dout) throws CoreException
  {
    super(instanceName);

    // Comprobar los parametros del core
    try
    {
      checkParameters();
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());
```

```
  // Crear los puerto de entrada
  dinPort = newInputPort("DIN", din);
  clkPort = newInputPort("CLK", clk);

  // Crear el puerto de salida
  doutPort = newOutputPort("DOUT", dout);
}

private void checkParameters() throws CoreParameterException
{
  // En este caso no es necesario comprobar nada
}

// Calculo de las dimensiones de la unidad
public static int calcHeightGran()
{
  return Gran.CLB;
}

public static int calcWidthGran()
{
  return Gran.CLB;
}

// Dimensiones del core
public static int calcHeight()
{
  return 1;
}

public static int calcWidth()
{
  return 1;
}

// Crea el contenido del core
public final void implement() throws CoreException
{
  int row, col;
  Pin dinPin, clkPin, doutPin;

  // Calcular el offset
  Offset offset = calcAbsoluteOffset();
  row = offset.getVerOffset(Gran.CLB);
  col = offset.getHorOffset(Gran.CLB);

  // Configurar la CLB
  Bitstream.set(row, col, LUT.SLICE0_F, Expr.F_LUT("~F1"));
  Bitstream.set(row,col, S0CE.S0CE, S0CE.OFF);
  Bitstream.set(row, col, SliceControl.CeInvert[0], SliceControl.OFF[0]);
  Bitstream.set(row,col, SliceControl.LatchMode[0], SliceControl.OFF[0]);
  Bitstream.set(row,col, SliceControl.ClockInvert[0], SliceControl.OFF[0]);
  Bitstream.set(row, col, SliceControl.X.X[0], SliceControl.X.FOUT[0]);
  Bitstream.set(row,col, SliceControl.XDin.XDin[0], SliceControl.XDin.X[0]);

  // Crear los pines de las entradas
  dinPin = new Pin(Pin.CLB, row, col, CenterWires.SliceF1[0]);
  clkPin = new Pin(Pin.CLB, row, col, CenterWires.SliceClk[0]);

  // Crear el pin de la salida
  doutPin = new Pin(Pin.CLB, row, col, CenterWires.Slice_XQ[0]);

  // Asignar los pines a los puertos
  dinPort.setPin(dinPin);
  clkPort.setPin(clkPin);
  doutPort.setPin(doutPin);

  // Marcar la clb como registro
  tagCLB(row, col, Tags.BUFFER);
}

private void tagCLB(int row, int col, int tag) throws CoreException
{
  try
```

```
    {
      Bitstream.getVirtex().setTag(row, col, tag);
    }
    catch (ConfigurationException ce)
    {
      throw new CoreException(ce);
    }
  }
}
```

Clase Task

```
package com.xilinx.JBits.Virtex.RTPCore.mtIOB;

import java.io.Serializable;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JBits.Virtex.ConfigurationException;

public abstract class Task extends RTPCore implements Serializable
{
  // Puertos del core
  protected Port doutPort;
  protected Port clkPort;

  public Task(String instanceName, Net clk, Net dout) throws CoreException
  {
    super(instanceName);

    // Comprobar los parametros
    try
    {
      checkParameters();
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear los puertos
    clkPort = newInputPort("CLK", clk);
    doutPort = newOutputPort("DOUT", dout);
  }

  private void checkParameters() throws CoreParameterException
  {
    // En este caso no hace falta comprobar nada
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.CLB;
  }

  // Dimensiones del core
  public static int calcHeight()
```

```
  {
    return 63;
  }

  public static int calcWidth()
  {
    return 24;
  }

  // Crea el contenido del core
  public abstract void implement() throws CoreException;

  protected void tagTask(int row, int col, int tag) throws CoreException
  {
    int i, rows, cols;

    rows = row + getHeight();
    cols = col + getWidth();

    for(i = col; i < cols; i++)
    {
      tagCLB(row, i, tag);
      tagCLB(rows - 1, i, tag);
    }

    for(i = row + 1; i < rows - 1; i++)
    {
      tagCLB(i, col, tag);
      tagCLB(i, cols - 1, tag);
    }

  }

  private void tagCLB(int row, int col, int tag) throws CoreException
  {
    try
    {
      Bitstream.getVirtex().setTag(row, col, tag);
    }
    catch (ConfigurationException ce)
    {
      throw new CoreException(ce);
    }
  }
}
```

Clase TaskCounter

```
package com.xilinx.JBits.Virtex.RTPCore.mtIOB;

import com.xilinx.JBits.Virtex.RTPCore.Tags;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Bus;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

public class TaskCounter extends Task
{
  // Parametros del core
  private int modulo;

  public TaskCounter(String instanceName, Net clk, Net dout, int modulo) throws
CoreException
  {
    super(instanceName, clk, dout);

    // Comprobar los parametros
    try
    {
```

```
      checkParameters(modulo);
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Recordar las propiedades del core
    this.modulo = modulo;
  }

  private void checkParameters(int modulo) throws CoreParameterException
  {
    // Comprobar que el modulo esta entre 0 y 15
    if(modulo < 0 || modulo > 15)
    {
      throw new CoreParameterException(this, "El modulo del contador tiene que estar entre
0 y 15");
    }
  }

  public void implement() throws com.xilinx.JBits.CoreTemplate.CoreException
  {
    ConstantComparator ctecmp1;
    Counter counter1;
    int row, col, i;
    Net rstNet, clkNet;
    Bus qBus;

    // Calcular el offset
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
    col = offset.getHorOffset(Gran.CLB);

    // Crear las señales internas
    rstNet = newNet("reset");
    clkNet = newNet("clk");
    qBus = newBus("q", 4);

    // Conectar los puertos con las nets correspondientes
    clkPort.setIntSig(clkNet);
    doutPort.setIntSig(rstNet);

    // Crear el contador
    counter1 = new Counter("counter1", clkNet, rstNet, qBus);

    // Crear el comparador
    ctecmp1 = new ConstantComparator("ctecmp1", qBus, rstNet, modulo);

    // Añadir ambos cores
    addChild(counter1);
    addChild(ctecmp1);

    // Situar el contador
    Offset counter1offset = counter1.getRelativeOffset();
    counter1offset.setVerOffset(Gran.CLB, 1);
    counter1offset.setHorOffset(Gran.CLB, 1);
    counter1offset.setHorOffset(Gran.SLICE, 0);

    // Situar el comparador
    Offset ctecmp1offset = ctecmp1.getRelativeOffset();
    ctecmp1offset.setVerOffset(Gran.CLB, 1);
    ctecmp1offset.setHorOffset(Gran.CLB, 1);
    ctecmp1offset.setHorOffset(Gran.SLICE, 1);

    // Marcar todas las CLBs
    tagTask(row, col, Tags.CONSTANT);

    // Implementar
    counter1.implement();
    ctecmp1.implement();

    // Rutar las señales internas
    Bitstream.connect(qBus);
  }
```

```
}
```

Clase TaskHigh

```
package com.xilinx.JBits.Virtex.RTPCore.mtIOB;

import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JBits.Virtex.RTPCore.Basic.Constant;

public class TaskHigh extends Task
{
  public TaskHigh(String instanceName, Net clk, Net dout) throws CoreException
  {
    super(instanceName, clk, dout);
  }

  public void implement() throws com.xilinx.JBits.CoreTemplate.CoreException
  {
    int row, col;
    Constant constant1;
    Net doutNet;

    // Calcular el offset
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
    col = offset.getHorOffset(Gran.CLB);

    // Crear la señal interna
    doutNet = newNet("dout");

    // Conectar el puerto a la net correspondiente
    super.doutPort.setIntSig(doutNet);

    // Crear el core para la salida a alta
    constant1 = new Constant("constant1", doutNet);

    // Situar el core
    Offset constant1Offset = constant1.getRelativeOffset();
    constant1Offset.setVerOffset(Gran.CLB, 0);
    constant1Offset.setHorOffset(Gran.CLB, 0);

    // Añadir el core constante
    addChild(constant1);

    // Marcar todas las CLBs
    tagTask(row, col, 0x40);

    // Implementarlo
    constant1.implement(1);
  }
}
```

Clase TaskLow

```
package com.xilinx.JBits.Virtex.RTPCore.mtIOB;

import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JBits.Virtex.RTPCore.Basic.Constant;
```

```
public class TaskLow extends Task
{
  public TaskLow(String instanceName, Net clk, Net dout) throws CoreException
  {
    super(instanceName, clk, dout);
  }

  public void implement() throws com.xilinx.JBits.CoreTemplate.CoreException
  {
    int row, col;
    Constant constant1;
    Net doutNet;

    // Calcular el offset
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
    col = offset.getHorOffset(Gran.CLB);

    // Crear la señal interna
    doutNet = newNet("dout");

    // Conectar el puerto a la net correspondiente
    super.doutPort.setIntSig(doutNet);

    // Crear el core para la salida a alta
    constant1 = new Constant("constant1", doutNet);

    // Situar el core
    Offset constant1Offset = constant1.getRelativeOffset();
    constant1Offset.setVerOffset(Gran.CLB, 0);
    constant1Offset.setHorOffset(Gran.CLB, 0);

    // Añadir el core constante
    addChild(constant1);

    // Marcar todas las CLBs
    tagTask(row, col, 0x25);

    // Implementarlo
    constant1.implement(0);
  }
}
```

Prueba TestmtIOBRT

```
package testmtiobrt;

// Swing
import javax.swing.Timer;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import javax.swing.JToggleButton;
import javax.swing.BorderFactory;
import javax.swing.border.LineBorder;

// Awt
import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.FlowLayout;

import com.borland.jbcl.layout.VerticalFlowLayout;

// Jbits
import com.xilinx.XHWIF.XHWIFConnection;
import com.xilinx.XHWIF.XHWIFWithEvents;
import com.xilinx.XHWIF.XHWIFException;

import com.xilinx.JBits.Virtex.Devices;
import com.xilinx.JBits.Virtex.JBits;
import com.xilinx.JBits.Virtex.Bits.CLB;
import com.xilinx.JBits.Virtex.ReadbackCommand;
```

```
import com.xilinx.JBits.Virtex.ConfigurationException;

import com.xilinx.JBits.CoreTemplate.NetPinsList;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Bus;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.CoreOutput;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JRoute2.Virtex.JRoute;
import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;
import com.xilinx.JBits.Virtex.RTPCore.Basic.Clock;
import com.xilinx.DeviceSimulator.Virtex.SimulatorClient;

// mtIOB
import com.xilinx.JBits.Virtex.RTPCore.mtIOB.mtBoard;
import com.xilinx.JBits.Virtex.RTPCore.mtIOB.mtIOB;
import com.xilinx.JBits.Virtex.RTPCore.mtIOB.TaskCounter;
import com.xilinx.JBits.Virtex.RTPCore.mtIOB.TaskBitstream;
import java.awt.event.*;

public class frmMain extends JFrame
{
  // Definicion de la placa
  private final String boardName = "VirtexDS:xcv1000";
  private final int deviceType = Devices.XCV1000;

  // Objetos para trabajar con jbits
  private XHWIFWithEvents board;
  private JBits jbits;
  private JRoute jroute;
  private SimulatorClient simulatorclient;

  // Puntos de prueba
  private Pin[] ledPin;

  // Señales del bus
  private Net taskDinANet;
  private Net taskClkANet;
  private Net taskDinBNet;
  private Net taskClkBNet;
  private Net taskDinCNet;
  private Net taskClkCNet;
  private Net taskDinDNet;
  private Net taskClkDNet;

  // Recursos usados por cada tarea
  private NetPinsList taskANetPinsList;
  private NetPinsList taskBNetPinsList;
  private NetPinsList taskCNetPinsList;
  private NetPinsList taskDNetPinsList;

  // Constante necesaria para cerrar la ventana a terminar
  private final int EXIT_ON_CLOSE = 3;

  // Componentes
  private Timer timer;

  private JPanel jContentPane = new JPanel();
  private JPanel jPTasks = new JPanel();
  private JScrollPane jSPLog = new JScrollPane();
  private JToggleButton jTBTaskA = new JToggleButton();
  private JToggleButton jTBTaskB = new JToggleButton();
  private JToggleButton jTBTaskC = new JToggleButton();
  private JToggleButton jTBTaskD = new JToggleButton();
  private GridLayout gridLayout1 = new GridLayout();
  private VerticalFlowLayout verticalFlowLayout1 = new VerticalFlowLayout();
  private JTextArea jTALog = new JTextArea();
  private JPanel jPEstado = new JPanel();
  private JLabel jLLed[] = new JLabel[4];
  private JPanel jPLeds = new JPanel();
  private FlowLayout flowLayout1 = new FlowLayout();
```

```
private JToggleButton jTBRun = new JToggleButton();
private GridLayout gridLayout2 = new GridLayout();

private final String titulo = "Prueba de mtIOB en tiempo real";

public frmMain()
{
  try
  {
    jbInit();
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
}

public static void main(String[] args)
{
  frmMain frmMain = new frmMain();

  // Deshabilitar los eventos
  frmMain.enableInputMethods(false);

  // Mostrar el dialogo
  frmMain.show();

  // Inicializar jBits
  try
  {
    frmMain.jbitsInit();
    frmMain.crearmtIOBus();
  }
  catch(Exception e)
  {
    frmMain.error(e);
  }

  // Habilitar los eventos
  frmMain.enableInputMethods(true);
}

private void jbInit() throws Exception
{
  int i;

  // Configurar la ventana
  this.setTitle(titulo);
  this.setDefaultCloseOperation(EXIT_ON_CLOSE);
  this.setSize(new Dimension(600, 500));
  this.setResizable(false);
  this.getContentPane().add(jContentPane, null);
  this.setContentPane(jContentPane);

  // Contenedor principal
  jContentPane.setLayout(verticalFlowLayout1);

  // Tareas
  jPTasks.setLayout(gridLayout1);

  jTBTaskA.setText("A");
  jTBTaskB.setText("B");
  jTBTaskC.setText("C");
  jTBTaskD.setText("D");

  jTBTaskA.setFont(new java.awt.Font("Dialog", 0, 24));
  jTBTaskB.setFont(new java.awt.Font("Dialog", 0, 24));
  jTBTaskC.setFont(new java.awt.Font("Dialog", 0, 24));
  jTBTaskD.setFont(new java.awt.Font("Dialog", 0, 24));

  // Leds
  jPLeds.setLayout(flowLayout1);
  jPLeds.setBorder(BorderFactory.createRaisedBevelBorder());

  for(i = 0; i < 4; i++)
```

```
{
  jLLed[i] = new JLabel();
  jLLed[i].setText("  ");
  jLLed[i].setOpaque(true);
  jLLed[i].setBorder(BorderFactory.createLineBorder(Color.black));
}

// Estado de la ejecucion
jTBRun.setText("Ejecutar");
jPEstado.setLayout(gridLayout2);

// Log
jSPLog.setAutoscrolls(true);
jTALog.setEditable(false);
jTALog.setText("");
jTALog.setRows(22);

// Añadir cada cosa a su sitio
jContentPane.add(jPTasks, null);
jContentPane.add(jPEstado, null);
jContentPane.add(jSPLog, null);
jPEstado.add(jTBRun, null);
jPEstado.add(jPLeds, null);
jPTasks.add(jTBTaskA, null);
jPTasks.add(jTBTaskB, null);
jPTasks.add(jTBTaskC, null);
jPTasks.add(jTBTaskD, null);
jSPLog.getViewport().add(jTALog, null);

for(i = 0; i < 4; i++)
  jPLeds.add(jLLed[i], null);


// Eventos
this.addWindowListener(new java.awt.event.WindowAdapter() {
  public void windowClosing(WindowEvent e) {
    this_windowClosing(e);
  }
});

jTBTaskA.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    jTBTaskA_actionPerformed(e);
  }
});

jTBTaskB.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    jTBTaskB_actionPerformed(e);
  }
});

jTBTaskC.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    jTBTaskC_actionPerformed(e);
  }
});

jTBTaskD.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    jTBTaskD_actionPerformed(e);
  }
});

jTBRun.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    jTBRun_actionPerformed(e);
```

```
    }
  });

  // Autoscroll
  jSPLog.getVerticalScrollBar().addAdjustmentListener(new AdjustmentListener()
  {
    public void adjustmentValueChanged(AdjustmentEvent e)
    {
      jTALog.select(jTALog.getDocument().getLength(),jTALog.getDocument().getLength());
    }
  });

  // Contador
  timer = new Timer(3000,new ActionListener(){
    public void actionPerformed(ActionEvent evt)
    {
      try
      {
        // Pasar un ciclo
        board.clockStep(1);
        leerLeds();
      }
      catch(Exception ex)
      {
        error(ex);
        timer.stop();
      }
    }
  });
}

private void jbitsInit() throws Exception
{
  int i;
  XHWIFConnection xc;

  accion("INICIANDO JBITS");

  // Conectar con la placa
  mensaje("Conectando con " + boardName + " ... ");
  xc = new XHWIFConnection();
  xc.connectToBoard(boardName);
  board = xc.getXHWIFWithEvents();
  hecho();

  mensaje("Reseteando la placa ... ");
  board.reset();
  hecho();

  mensaje("Cargando el bitstream nulo en la placa ...");
  jbits = new JBits(deviceType);
  jbits.readPartial("null1000.bit");
  board.setConfiguration(0, jbits.getPartial());
  hecho();

  // Inicializar el interfaz
  mensaje("Preparando el bitstream ...");
  jroute = new JRoute(jbits);
  Bitstream.setVirtex(jbits, jroute);
  CoreOutput.generateBitstream(true);
  CoreOutput.generateSYM(false);
  CoreOutput.generateXDL(false);

  // Crear el simulador
  simulatorclient = new SimulatorClient();

  // Establecer los puntos de prueba
  ledPin = new Pin[4];

  for(i = 0; i < 4; i++)
    ledPin[i] = new Pin(Pin.CLB, 0, 24 * i, CenterWires.S0_XQ);

  hecho();

  completado();
```

```
  }

  private void crearmtIOBus() throws Exception
  {
    int ledbusOutput;
    mtBoard mtboard;
    mtIOB mtiobus;
    Clock clock;
    Bus ledBus;
    Net clkNet;

    // Crear las conexiones internas
    taskDinANet = new Net("taskDinA", null);
    taskClkANet = new Net("taskClkA", null);

    taskDinBNet = new Net("taskDinB", null);
    taskClkBNet = new Net("taskClkB", null);

    taskDinCNet = new Net("taskDinC", null);
    taskClkCNet = new Net("taskClkC", null);

    taskDinDNet = new Net("taskDinD", null);
    taskClkDNet = new Net("taskClkD", null);

    clkNet = new Net("CLK", null);
    ledBus = new Bus("LED", null, 4);

    accion("CREANDO EL BUS");

    mensaje("Configurando los IOBs de la placa ... ");

    // Crear las conexiones con los leds
    mtboard = new mtBoard("XCV1000_BG560");

    // Añadir la salida de los leds a la placa
    ledbusOutput = mtboard.addOutput(ledBus.getName(), ledBus);

    // Configurar el IOB seleccionado para salida no invertida
    mtboard.setOutputClk(ledbusOutput, clkNet);
    mtboard.setOutputInvertT(ledbusOutput, true);
    mtboard.setOutputInvertO(ledbusOutput, false);
    hecho();

    // Implementar la placa con el ucf especificado
    mensaje("Cargando las posiciones de los leds ... ");
    mtboard.implement(0, "xcv1000_bg560.ucf");
    hecho();

    // Crear el reloj
    mensaje("Creando el reloj ... ");
    clock = new Clock("clock", clkNet);
    clock.implement(mtboard.getGCLK());
    hecho();

    // Crear el bus
    mensaje("Preparando el bus ... ");
    mtiobus = new mtIOB("mtiobus", taskDinANet, taskClkANet, taskDinBNet, taskClkBNet,
taskDinCNet, taskClkCNet, taskDinDNet, taskClkDNet, clkNet, ledBus);

    // Establecer la posicion del nuevo core
    Offset mtiobusOffset = mtiobus.getRelativeOffset();
    mtiobusOffset.setHorOffset(Gran.CLB, 0);
    mtiobusOffset.setVerOffset(Gran.CLB, 0);

    // Implementarlo
    mtiobus.implement();
    hecho();

    // Rutar las señales internas
    mensaje("Rutando la señal de reloj ... ");
    Bitstream.connect(clkNet);
    hecho();

    mensaje("Rutando las conexiones con los leds ... ");
    Bitstream.connect(ledBus);
```

```
    hecho();

    reconfigurar();

    completado();
  }

  private void leerLeds()
  {
    int i, valor;
    Color estado;

    mensaje("Leyendo el estado de los leds ... ");

    for(i = 0; i < 4; i++)
    {
      // Obtener el valor actual
      valor = simulatorclient.probePinValue(ledPin[i]);

      // Determinar el nuevo estado
      estado = (valor == 1) ? Color.red : Color.black;

      // Cambiar el color del led
      jLLed[i].setBackground(estado);
    }

    hecho();
  }

  private void reconfigurar() throws Exception
  {
    // Escribir la configuracion parcial en la placa
    mensaje("Cargando el bitstream parcial en la placa ... ");
    board.setConfiguration(0, jbits.getPartial());
    hecho();

    // Actualizar el estado de los leds
    leerLeds();
  }

  private void accion(String texto)
  {
    // Mostrar el estado actual de la aplicacion en la barra de titulo
    setTitle(texto);

    // Mostrar un mensaje en el cuadro de log
    jTALog.append("\n" + texto + "\n\n");
  }

  private void completado()
  {
    // Volver al titulo original
    setTitle(titulo);

    // Mostrar un mensaje en el cuadro de log
    jTALog.append("\nCOMPLETADO.\n\n");
  }

  private void mensaje(String texto)
  {
    // Mostrar un mensaje en el cuadro de log
    jTALog.append(texto);
  }

  private void hecho()
  {
    // Mostrar un mensaje en el cuadro de log
    jTALog.append("hecho\n");
  }

  private void error(Exception e)
  {
    // Volver al titulo original
    setTitle(titulo);
```

```
  // Mostrar un mensaje de error
  jTALog.append("error\n" + e.getClass() + "\n" + e.getMessage() + "\n\n");
}

void this_windowClosing(WindowEvent e)
{
  try
  {
    //Parar el timer si se esta ejecutando
    if(timer.isRunning())
      timer.stop();

    // Desconectar la placa
    mensaje("Desconectando de la placa " + boardName);
    simulatorclient.close();
    board.disconnect();
    hecho();
  }
  catch(Exception ex)
  {
    error(ex);
  }
}

void jTBTaskA_actionPerformed(ActionEvent e)
{
  try
  {
    // Comprobar que el boton ha sido pulsado
    if(jTBTaskA.isSelected())
    {
      // Crear la tarea A
      accion("CREANDO LA TAREA A");

      mensaje("Creando una instancia para la tarea A ... ");

      // Crear el core
      TaskCounter taskA = new TaskCounter("taskA", taskClkANet, taskDinANet, 4);

      // Establecer la posicion
      Offset taskAOffset = taskA.getRelativeOffset();
      taskAOffset.setHorOffset(Gran.CLB, 0);
      taskAOffset.setVerOffset(Gran.CLB, 1);

      hecho();

      // Iniciar la grabacion de los recursos usados
      taskANetPinsList = Bitstream.startRouteRecording();

      // Configurar la logica del core
      mensaje("Implementando la tarea ... ");
      taskA.implement();
      hecho();

      // Conectar las señales con el bus
      mensaje("Realizando las conexiones con el bus ... ");
      Bitstream.connect(taskDinANet);

      // Detener la grabacion
      Bitstream.endRouteRecording();

      // Conectar el reloj
      Bitstream.connect(taskClkANet);
      hecho();

      reconfigurar();

      completado();
    }
    else
    {
      accion("ELIMINANDO LA TAREA A");

      // Desconectar la tarea del bus
      mensaje("Desconectando las señales ... ");
```

```
      Bitstream.unroute(taskANetPinsList);
      hecho();

      // Limpiar la netlist
      Port p = taskDinANet.getFirstSink();
      taskDinANet = new Net("taskDinA", null);
      p.setExtSig(taskDinANet);
      reconfigurar();

      completado();
    }
  }
  catch(Exception ex)
  {
    // Se ha producido un error
    error(ex);
  }
}

void jTBTaskB_actionPerformed(ActionEvent e)
{
  // Comprobar que el boton ha sido pulsado
  if(jTBTaskB.isSelected())
  {
    try
    {
      // Crear la tarea B
      accion("CREANDO LA TAREA B");

      mensaje("Creando una instancia para la tarea B ... ");

      // Crear el core
      TaskCounter taskB = new TaskCounter("taskB", taskClkBNet, taskDinBNet, 2);

      // Establecer la posicion
      Offset taskBOffset = taskB.getRelativeOffset();
      taskBOffset.setHorOffset(Gran.CLB, 24);
      taskBOffset.setVerOffset(Gran.CLB, 1);

      hecho();

      mensaje("Implementando la tarea ... ");
      taskB.implement();
      hecho();

      // Conectar las señales con el bus
      mensaje("Realizando las conexiones con el bus ... ");
      Bitstream.connect(taskDinBNet);
      Bitstream.connect(taskClkBNet);
      hecho();

      reconfigurar();

      completado();
    }
    catch(Exception ex)
    {
      // Se ha producido un error
      error(ex);
    }
  }
}

void jTBTaskC_actionPerformed(ActionEvent e)
{
  // Comprobar que el boton ha sido pulsado
  if(jTBTaskC.isSelected())
  {
    try
    {
      // Crear la tarea C
      accion("CREANDO LA TAREA C");

      mensaje("Creando una instancia para la tarea C ... ");
```

```
      // Crear el core
      TaskCounter taskC = new TaskCounter("taskC", taskClkCNet, taskDinCNet, 7);

      // Establecer la posicion
      Offset taskCOffset = taskC.getRelativeOffset();
      taskCOffset.setHorOffset(Gran.CLB, 48);
      taskCOffset.setVerOffset(Gran.CLB, 1);

      hecho();

      mensaje("Implementando la tarea ... ");
      taskC.implement();
      hecho();

      // Conectar las señales con el bus
      mensaje("Realizando las conexiones con el bus ... ");
      Bitstream.connect(taskDinCNet);
      Bitstream.connect(taskClkCNet);
      hecho();

      reconfigurar();

      completado();
    }
    catch(Exception ex)
    {
      // Se ha producido un error
      error(ex);
    }
  }
}

void jTBTaskD_actionPerformed(ActionEvent e)
{
  // Comprobar que el boton ha sido pulsado
  if(jTBTaskD.isSelected())
  {
    try
    {
      // Crear la tarea D
      accion("CREANDO LA TAREA D");

      mensaje("Creando una instancia para la tarea D ... ");

      // Crear el core
      TaskCounter taskD = new TaskCounter("taskD", taskClkDNet, taskDinDNet, 13);

      // Establecer la posicion
      Offset taskDOffset = taskD.getRelativeOffset();
      taskDOffset.setHorOffset(Gran.CLB, 72);
      taskDOffset.setVerOffset(Gran.CLB, 1);

      hecho();

      mensaje("Implementando la tarea ... ");
      taskD.implement();
      hecho();

      // Conectar las señales con el bus
      mensaje("Realizando las conexiones con el bus ... ");
      Bitstream.connect(taskDinDNet);
      Bitstream.connect(taskClkDNet);
      hecho();

      reconfigurar();

      completado();
    }
    catch(Exception ex)
    {
      // Se ha producido un error
      error(ex);
    }
  }
}
```

```
  void jTBRun_actionPerformed(ActionEvent e)
  {
    // Comprobar si se ha marcado el boton
    if(jTBRun.isSelected())
      timer.start();
    else
      timer.stop();
  }
}
```

Prueba TestTracer

```
package testtracer;

import java.util.Vector;

import com.xilinx.JBits.Virtex.JBits;
import com.xilinx.JBits.Virtex.Devices;

import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Pin;

import com.xilinx.JRoute2.Virtex.JRoute;
import com.xilinx.JRoute2.Virtex.ResourceFactory;
import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;

import com.xilinx.DeviceSimulator.Virtex.RouteTracer;
import com.xilinx.DeviceSimulator.Virtex.RouteTree;

public class TestTracer
{
  // Poner a falso para quitar los mensajes de error
  private static final boolean debug = true;

  private static final int[] sourcePins = {
    CenterWires.S0_X,
    CenterWires.S0_Y,
    CenterWires.S0_XQ,
    CenterWires.S0_YQ,
    CenterWires.S0_XB,
    CenterWires.S0_YB,
    CenterWires.S1_X,
    CenterWires.S1_Y,
    CenterWires.S1_XQ,
    CenterWires.S1_YQ,
    CenterWires.S1_XB,
    CenterWires.S1_YB,
  };

  public static void routeCLB(RouteTracer tracer, Vector nets, int row, int col)
  {
    int i, j;
    Vector net;
    Pin p;

    for(i = 0; i < sourcePins.length; i++)
    {
      p = new Pin(Pin.CLB, row, col, sourcePins[i]);
      RouteTree t = new RouteTree(p);

      net = new Vector();

      net.addElement(p);

      try
      {
        tracer.trace(t);
      }
      catch(Exception e)
      {
        System.out.println("Error al trazar");
        System.out.println(e);
      }
```

```
      RouteTree[] ts = t.getBottom();

      for(j = 0; j < ts.length; j++)
      {
        if(ts[j].sink())
          net.addElement(ts[j].getPin());
      }

      if(net.size() > 1)
        nets.addElement(net);
    }
  }

  public static void main(String[] args)
  {
    int row, col;

    Vector nets = new Vector();

    // Crear el Bitstream
    JBits jbits = new JBits(Devices.XCV1000);
    JRoute jroute = new JRoute(jbits, System.out);

    // Leer el bitstream del archivo
    try
    {
      jbits.readPartial("cout.bit");
    }
    catch(Exception e)
    {
      System.out.println("No se puede abrir el archivo");
      System.out.println(e);
    }

    // Rutar
    RouteTracer tracer = new RouteTracer(jbits);

    for(row = 0; row < 10; row++)
      for(col = 0; col < 10; col++)
        routeCLB(tracer, nets, row, col);

    System.out.println(nets);
  }
}
```

Apéndice C: Código del Sistema Multitarea

Libreria mtBits para el simulador

Clase mtbException

```
package mtbits;

import java.lang.Throwable;

public class mtbException extends Exception
{
  private Throwable cause;

  public mtbException(Throwable e)
  {
    super();

    this.cause = e;
  }

  public mtbException(String text)
  {
    super(text);

    cause = null;
  }
```

```
  public Throwable getCause()
  {
    return cause;
  }

  public String toString()
  {
    if(cause != null)
      return cause.toString();
    else
      return this.toString();
  }

  public String getMessage()
  {
    if(cause != null)
      return cause.getMessage();
    else
      return super.getMessage();
  }
}
```

Clase mtBits

```
package mtbits;

// Java
import java.io.PrintStream;
import java.io.IOException;

// Jbits
import com.xilinx.XHWIF.XHWIFConnection;
import com.xilinx.XHWIF.XHWIFWithEvents;
import com.xilinx.XHWIF.XHWIFException;

import com.xilinx.JBits.Virtex.Devices;
import com.xilinx.JBits.Virtex.JBits;
import com.xilinx.JBits.Virtex.Bits.CLB;
import com.xilinx.JBits.Virtex.ReadbackCommand;
import com.xilinx.JBits.Virtex.ConfigurationException;

import com.xilinx.JBits.Virtex.RTPCore.Basic.Clock;

import com.xilinx.JBits.CoreTemplate.NetPinsList;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Bus;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.CoreOutput;
import com.xilinx.JBits.CoreTemplate.CoreException;

import com.xilinx.JRoute2.Virtex.JRoute;
import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;

import com.xilinx.DeviceSimulator.Virtex.SimulatorClient;

// Cores
import mtbits.RTPCore.mtBoard;
import mtbits.RTPCore.mtIOB;
import mtbits.RTPCore.Register;
import mtbits.RTPCore.TaskBitstream;

public class mtBits
{
  // Registro de errores
  private PrintStream debugOutput;

  // Definicion de la placa
  private final String boardName = "VirtexDS:xcv1000";
  private final int deviceType = Devices.XCV1000;
```

```
// Objetos para trabajar con jbits
private XHWIFWithEvents board;
private JBits jbits;
private JRoute jroute;
private SimulatorClient simulatorclient;

// Puntos de prueba
private Pin[] ledPin;

// Registro de las rutas usadas por cada tarea
private NetPinsList[] taskNetPinsList;

// Cores usados
private mtIOB mtiobus;

public mtBits() throws mtbException, IOException
{
  this(null);
}

public mtBits(PrintStream debugOutput) throws mtbException
{
  XHWIFConnection xc;
  int i, ledbusOutput;
  mtBoard mtboard;
  Clock clock;
  Bus ledBus;
  Net clkNet;

  // Inicializar las listas de pines
  taskNetPinsList = new NetPinsList[4];

  // Preparar la salida
  this.debugOutput = debugOutput;

  try
  {
    // Crear la conexion con la placa
    message("Conectar con " + boardName);
    xc = new XHWIFConnection();
    xc.connectToBoard(boardName);
    board = xc.getXHWIFWithEvents();
    done();

    message("Resetear la placa");
    board.reset();
    done();

    message("Cargar el bitstream nulo en la placa");
    jbits = new JBits(deviceType);
    jbits.readPartial("null1000.bit");
    board.setConfiguration(0, jbits.getPartial());
    done();

    // Inicializar el interfaz
    message("Preparar el bitstream");
    jroute = new JRoute(jbits);
    Bitstream.setVirtex(jbits, jroute);
    CoreOutput.generateBitstream(true);
    CoreOutput.generateSYM(false);
    CoreOutput.generateXDL(false);


    // Crear el simulador
    simulatorclient = new SimulatorClient();

    // Establecer los puntos de prueba
    ledPin = new Pin[4];

    for(i = 0; i < 4; i++)
      ledPin[i] = new Pin(Pin.CLB, 0, 24 * i, CenterWires.S0_XQ);

    done();

    // Crear las señales internas
```

```
      clkNet = new Net("CLK", null);
      ledBus = new Bus("LED", null, 4);

      message("Configurar los IOBs de la placa");

      // Crear las conexiones con los leds
      mtboard = new mtBoard("XCV1000_BG560");

      // Añadir la salida de los leds a la placa
      ledbusOutput = mtboard.addOutput(ledBus.getName(), ledBus);

      // Configurar el IOB seleccionado para salida no invertida
      mtboard.setOutputClk(ledbusOutput, clkNet);
      mtboard.setOutputInvertT(ledbusOutput, true);
      mtboard.setOutputInvertO(ledbusOutput, false);
      done();

      // Implementar la placa con el ucf especificado
      message("Cargar las posiciones de los leds");
      mtboard.implement(0, "xcv1000_bg560.ucf");
      done();

      // Crear el reloj
      message("Crear el reloj");
      clock = new Clock("clock", clkNet);
      clock.implement(mtboard.getGCLK());
      done();

      // Crear el bus
      message("Preparar el bus");
      mtiobus = new mtIOB("mtiobus", clkNet, ledBus);

      // Establecer la posicion del nuevo core
      Offset mtiobusOffset = mtiobus.getRelativeOffset();
      mtiobusOffset.setHorOffset(Gran.CLB, 0);
      mtiobusOffset.setVerOffset(Gran.CLB, 0);

      // Implementarlo
      mtiobus.implement();
      done();

      // Rutar las señales internas
      message("Rutar la señal de reloj");
      Bitstream.connect(clkNet);
      done();

      message("Rutar las conexiones con los leds");
      Bitstream.connect(ledBus);
      done();

      // Actualizar la placa
      update();
    }
    catch(Exception e)
    {
      fail();

      throw new mtbException(e);
    }
  }

  public void step() throws mtbException
  {
    try
    {
      message("Ciclo de reloj");
      board.clockStep(1);
      done();
    }
    catch(XHWIFException xhwife)
    {
      fail();

      throw new mtbException(xhwife);
    }
```

```
  }

  public void load(String bitstream, int taskIndex) throws mtbException
  {
    Net dinTaskNet, clkTaskNet;

    try
    {
      message("Crear una instancia para la tarea " + taskIndex);

      // Crear las conexiones internas
      dinTaskNet = new Net("Din" + taskIndex, null);
      clkTaskNet = new Net("Clk"+ taskIndex, null);

      // Crear el core
      TaskBitstream task = new TaskBitstream("task" + taskIndex, clkTaskNet, dinTaskNet,
bitstream);

      // Añadir el core al bus
      mtiobus.setTask(dinTaskNet, clkTaskNet, taskIndex);

      // Establecer la posicion
      Offset taskOffset = task.getRelativeOffset();
      taskOffset.setHorOffset(Gran.CLB, taskIndex * 24);
      taskOffset.setVerOffset(Gran.CLB, 1);
      done();

      // Iniciar la grabacion de los recursos usados
      taskNetPinsList[taskIndex] = Bitstream.startRouteRecording();

      // Configurar la logica del core
      message("Implementar");
      task.implement();
      done();

      // Conectar las señales con el bus
      message("Realizar las conexiones con el bus");
      Bitstream.connect(dinTaskNet);

      // Detener la grabacion
      Bitstream.endRouteRecording();

      // Conectar el reloj
      Bitstream.connect(clkTaskNet);
      done();

      // Reconfigurar la placa
      update();
    }
    catch(Exception e)
    {
      fail();

      throw new mtbException(e);
    }
  }

  public void clear(int taskIndex) throws mtbException
  {
    try
    {
      message("Limpiar la tarea " + taskIndex);
      Bitstream.unroute(taskNetPinsList[taskIndex]);
      done();

      // Borrar la lista de pines usados
      taskNetPinsList[taskIndex] = null;

      // Reconfigurar
      update();
    }
    catch(Exception e)
    {
      fail();
```

```
      throw new mtbException(e);
    }
  }

  public int[] getStatus()
  {
    int [] estado;
    int i;

    estado = new int[4];

    message("Leer el estado de los leds");

    // Leer el valor de cada led
    for(i = 0; i < 4; i++)
      estado[i] = simulatorclient.probePinValue(ledPin[i]);

    done();

    return estado;
  }

  public void close() throws mtbException
  {
    try
    {
      // Desconectar la placa
      message("Desconectar de la placa " + boardName);
      simulatorclient.close();
      board.disconnect();
      done();
    }
    catch(Exception e)
    {
      fail();

      throw new mtbException(e);
    }
  }

  private void update() throws XHWIFException, ConfigurationException
  {
    // Escribir la configuracion parcial en la placa
    message("Cargando el bitstream parcial en la placa");
    board.setConfiguration(0, jbits.getPartial());
    done();
  }

  private void message(String text)
  {
    if(debugOutput != null)
      debugOutput.print(text + " ... ");
  }

  private void done()
  {
    if(debugOutput != null)
      debugOutput.println("hecho");
  }

  private void fail()
  {
    if(debugOutput != null)
      debugOutput.println("error");
  }
}
```

Clase mtBoard

```
package mtbits.RTPCore;

import com.xilinx.JBits.Virtex.RTPCore.Iob.Board;
import com.xilinx.JBits.Virtex.RTPCore.Iob.XCVPackage;
import com.xilinx.JBits.Virtex.RTPCore.Iob.xcv1000_bg560;
```

```
import com.xilinx.JBits.CoreTemplate.CoreException;

public class mtBoard extends Board
{
  // Empaquetado. Uno por cada fpga. Solo tenemos 1 en la placa.
  private XCVPackage xcvPackage[] = { new xcv1000_bg560() };

  // El reloj global
  private int GCLK = 1;

  public mtBoard(String instanceName) throws CoreException
  {
    super(instanceName);

    // Establecer el empaquetado de la placa
    setXcvPackage(xcvPackage);

    // Definir la configuracion de la fpga
    setGCLK(GCLK);
  }
}
```

Clase mtIOB

```
package mtbits.RTPCore;

import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Bus;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JBits.Virtex.ConfigurationException;
import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;

public class mtIOB extends RTPCore
{
  // Definicion de la altura del core
  private final static int rows = 1;

  // Puertos de entrada del core
  private Port[] dinTaskPort; // Entrada desde una tarea
  private Port clkPort; // Entrada del reloj

  // Puertos de salida del core
  private Port doutPort; // Salida a los leds
  private Port[] clkTaskPort; // Reloj para una tarea A

  // Registros de entrada/salida
  private Register[] registerDinTask;

  public mtIOB(String instanceName,
               Net clk,
               Bus dout) throws CoreException
  {
    super(instanceName);

    int i;

    // Comprobar los parametros del core
    try
    {
      checkParameters(dout);
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Calcular las dimensiones del core
```

```
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear los puertos de entrada
    dinTaskPort = new Port[4];

    for(i = 0; i < 4; i++)
      dinTaskPort[i] = newInputPort("DIN" + i, Net.NoConnect);

    clkPort = newInputPort("CLK", clk);

    // Crear los puertos de salida
    doutPort = newOutputPort("DOUT", dout);

    clkTaskPort = new Port[4];

    for(i = 0; i < 4; i++)
      clkTaskPort[i] = newOutputPort("CLK" + i, Net.NoConnect);
  }

  private void checkParameters(Bus dout) throws CoreParameterException
  {
    // Comprobar que el ancho del bus de salida es exactamente 4
    // (uno por cada tarea).
    if(dout.getWidth() != 4)
      throw new CoreParameterException(this, "El ancho del bus de salida tiene que ser
4.");
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.CLB;
  }

  // Dimensiones del core
  public static int calcHeight()
  {
    // Este valor se define al principio de la clase
    return rows;
  }

  public static int calcWidth()
  {
    // Ocupa toda una fila de la FPGA
    return Bitstream.getVirtex().getClbColumns();
  }

  public final void implement() throws CoreException
  {
    int row, col, i, j;
    Offset bufferOffset;
    Net[] dinTaskNet;
    Net clkNet;
    Bus doutBus;

    // Calcular el offset
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
    col = offset.getHorOffset(Gran.CLB);

    // Crear las señales internas
    dinTaskNet = new Net[4];

    for(i = 0; i < 4; i++)
      dinTaskNet[i] = newNet("din" + i);

    doutBus = newBus("data", 4);
```

```
    clkNet = newNet("clk");

    // Conectar puertos a la net correspondiente
    doutPort.setIntSig(doutBus);
    clkPort.setIntSig(clkNet);

    for(i = 0; i < 4; i++)
    {
      dinTaskPort[i].setIntSig(dinTaskNet[i]);
      clkTaskPort[i].setIntSig(clkNet);
    }

    // Crear el vector de registros
    registerDinTask = new Register[4];

    // Crear los buffers para las entradas y salidas con las tareas
    for(i = 0; i < 4; i++)
    {
      registerDinTask[i] = new Register("registerDin" + i, dinTaskNet[i], clkNet,
doutBus.getNet(i));

      // Situar los buffers a intervalos regulares
      bufferOffset = registerDinTask[i].getRelativeOffset();
      bufferOffset.setVerOffset(Gran.CLB, 0);
      bufferOffset.setHorOffset(Gran.CLB, i * 24);

      // Añadir el buffer
      addChild(registerDinTask[i]);

      // Implementar
      registerDinTask[i].implement();
    }
  }

  public void setTask(Net din, Net clk, int taskIndex) throws CoreException
  {
    Net dinNet, clkNet;

    // Comprobar que el indice de tarea no excede 4
    if(taskIndex > 4 || taskIndex < 0)
      throw new CoreException("Indice de la tarea fuera del rango [0, 4]");

    // Actualizar los puertos
    dinTaskPort[taskIndex] = newInputPort("DIN" + taskIndex, din);
    clkTaskPort[taskIndex] = newOutputPort("CLK" + taskIndex, clk);

    // Establecer las nuevas señales internas
    dinNet = newNet("din" + taskIndex);
    clkNet = (Net)clkPort.getIntSig();

    // Asignar las señales a los puertos
    dinTaskPort[taskIndex].setIntSig(dinNet);
    clkTaskPort[taskIndex].setIntSig(clkNet);

    // Actualizar el registro
    registerDinTask[taskIndex].setDin(dinNet);
  }
}
```

Clase Register

```
package mtbits.RTPCore;

import com.xilinx.JBits.Virtex.Expr;
import com.xilinx.JBits.Virtex.Bits.SliceControl;
import com.xilinx.JBits.Virtex.Bits.LUT;
import com.xilinx.JBits.Virtex.Bits.S0CE;
import com.xilinx.JBits.Virtex.ConfigurationException;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Port;
```

```
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;

public class Register extends RTPCore
{
  // Puertos del core
  private Port dinPort;
  private Port clkPort;
  private Port doutPort;

  public Register(String instanceName, Net din, Net clk, Net dout) throws CoreException
  {
    super(instanceName);

    // Comprobar los parametros del core
    try
    {
      checkParameters();
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear los puerto de entrada
    dinPort = newInputPort("DIN", din);
    clkPort = newInputPort("CLK", clk);

    // Crear el puerto de salida
    doutPort = newOutputPort("DOUT", dout);
  }

  private void checkParameters() throws CoreParameterException
  {
    // En este caso no es necesario comprobar nada
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.CLB;
  }

  // Dimensiones del core
  public static int calcHeight()
  {
    return 1;
  }

  public static int calcWidth()
  {
    return 1;
  }

  // Crea el contenido del core
  public final void implement() throws CoreException
  {
    int row, col;
    Pin dinPin, clkPin, doutPin;

    // Calcular el offset
```

```
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
    col = offset.getHorOffset(Gran.CLB);

    // Configurar la CLB
    Bitstream.set(row, col, LUT.SLICE0_F, Expr.F_LUT("~F1"));
    Bitstream.set(row,col, S0CE.S0CE, S0CE.OFF);
    Bitstream.set(row, col, SliceControl.CeInvert[0], SliceControl.OFF[0]);
    Bitstream.set(row,col, SliceControl.LatchMode[0], SliceControl.OFF[0]);
    Bitstream.set(row,col, SliceControl.ClockInvert[0], SliceControl.OFF[0]);
    Bitstream.set(row, col, SliceControl.X.X[0], SliceControl.X.FOUT[0]);
    Bitstream.set(row,col, SliceControl.XDin.XDin[0], SliceControl.XDin.X[0]);

    // Crear los pines de las entradas
    dinPin = new Pin(Pin.CLB, row, col, CenterWires.SliceF1[0]);
    clkPin = new Pin(Pin.CLB, row, col, CenterWires.SliceClk[0]);

    // Crear el pin de la salida
    doutPin = new Pin(Pin.CLB, row, col, CenterWires.Slice_XQ[0]);

    // Asignar los pines a los puertos
    dinPort.setPin(dinPin);
    clkPort.setPin(clkPin);
    doutPort.setPin(doutPin);
  }

  public void setDin(Net din) throws CoreException
  {
    Pin dinPin;

    if(dinPort.hasPins() == true)
    {
      // Recuperar el pin de entrada
      dinPin = dinPort.getPins(0)[0];
    }
    else
    {
      throw new CoreException("No se ha implementado el registro");
    }

    // Crear el nuevo puerto
    dinPort = newInputPort("DIN", din);

    // Asignar el pin al puerto
    dinPort.setPin(dinPin);
  }

  public void setDout(Net dout) throws CoreException
  {
    Pin doutPin;

    if(doutPort.hasPins() == true)
    {
      // Recuperar el pin de entrada
      doutPin = doutPort.getPins(0)[0];
    }
    else
    {
      throw new CoreException("No se ha implementado el registro");
    }

    // Crear el nuevo puerto
    doutPort = newOutputPort("DOUT", dout);

    // Asignar el pin al puerto
    doutPort.setPin(doutPin);
  }
}
```

Clase Task

```
package mtbits.RTPCore;

import java.io.Serializable;
```

```
import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JBits.Virtex.ConfigurationException;

public abstract class Task extends RTPCore implements Serializable
{
  // Puertos del core
  protected Port doutPort;
  protected Port clkPort;

  public Task(String instanceName, Net clk, Net dout) throws CoreException
  {
    super(instanceName);

    // Comprobar los parametros
    try
    {
      checkParameters();
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear los puertos
    clkPort = newInputPort("CLK", clk);
    doutPort = newOutputPort("DOUT", dout);
  }

  private void checkParameters() throws CoreParameterException
  {
    // En este caso no hace falta comprobar nada
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.CLB;
  }

  // Dimensiones del core
  public static int calcHeight()
  {
    return 63;
  }

  public static int calcWidth()
  {
    return 24;
  }

  // Crea el contenido del core
  public abstract void implement() throws CoreException;
}
```

Clase TaskBitstream

```
package mtbits.RTPCore;
```

```
import java.util.Vector;

import com.xilinx.JBits.Virtex.JBits;
import com.xilinx.JBits.Virtex.Devices;

import com.xilinx.JBits.Virtex.Bits.S0Control;
import com.xilinx.JBits.Virtex.Bits.S1Control;
import com.xilinx.JBits.Virtex.Bits.IOB;
import com.xilinx.JBits.Virtex.Bits.LUT;
import com.xilinx.JBits.Virtex.Bits.S0RAM;
import com.xilinx.JBits.Virtex.Bits.S1RAM;
import com.xilinx.JBits.Virtex.Bits.S0Clk;
import com.xilinx.JBits.Virtex.Bits.S1Clk;

import com.xilinx.DeviceSimulator.Virtex.RouteTracer;
import com.xilinx.DeviceSimulator.Virtex.RouteTree;

import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;
import com.xilinx.JRoute2.Virtex.ResourceDB.IobWiresBottom;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Bus;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

public class TaskBitstream extends Task
{
  // Lista de parametros de la configuracion de una clb
  private static final int [][][] clbParams = {
    S0Control.BxInvert, S0Control.ByInvert, S0Control.CeInvert, S0Control.ClockInvert,
    S0Control.InvertedSetReset, S0Control.LatchMode, S0Control.SrWeNotInvert,
    S0Control.Sync, S0Control.XffSetResetSelect, S0Control.YffSetResetSelect,
    S0Control.AndMux.AndMux, S0Control.Cin.Cin, S0Control.X.X,
    S0Control.XCarrySelect.XCarrySelect, S0Control.XDin.XDin, S0Control.Y.Y,
    S0Control.YB.YB, S0Control.YCarrySelect.YCarrySelect, S0Control.YDin.YDin,
    S1Control.BxInvert, S1Control.ByInvert, S1Control.CeInvert, S1Control.ClockInvert,
    S1Control.InvertedSetReset, S1Control.LatchMode, S1Control.SrWeNotInvert,
    S1Control.Sync, S1Control.XffSetResetSelect, S1Control.YffSetResetSelect,
    S1Control.AndMux.AndMux, S1Control.Cin.Cin, S1Control.X.X,
    S1Control.XCarrySelect.XCarrySelect, S1Control.XDin.XDin, S1Control.Y.Y,
    S1Control.YB.YB, S1Control.YCarrySelect.YCarrySelect, S1Control.YDin.YDin,
    LUT.SLICE0_F, LUT.SLICE0_G, LUT.SLICE1_F, LUT.SLICE1_G,
    S0RAM.DUAL_MODE, S0RAM.F_LUT_RAM, S0RAM.F_LUT_SHIFTER, S0RAM.G_LUT_RAM,
    S0RAM.G_LUT_SHIFTER, S0RAM.LUT_MODE, S0RAM.RAM_32_X_1, S0RAM.UNUSED,
    S1RAM.DUAL_MODE, S1RAM.F_LUT_RAM, S1RAM.F_LUT_SHIFTER, S1RAM.G_LUT_RAM,
    S1RAM.G_LUT_SHIFTER, S1RAM.LUT_MODE, S1RAM.RAM_32_X_1, S1RAM.UNUSED
  };

  // Lista de fines fuente en una CLB
  private static final int[] sourcePins = {
    CenterWires.S0_X,
    CenterWires.S0_Y,
    CenterWires.S0_XQ,
    CenterWires.S0_YQ,
    CenterWires.S0_XB,
    CenterWires.S0_YB,
    CenterWires.S1_X,
    CenterWires.S1_Y,
    CenterWires.S1_XQ,
    CenterWires.S1_YQ,
    CenterWires.S1_XB,
    CenterWires.S1_YB,
  };

  // El pin que se usa como referencia para el puerto DOUT
  private static final Pin doutIobPin = new Pin(Pin.IOB, IOB.BOTTOM, 14,
IobWiresBottom.O[1]);
```

```
  // Parametros del core
  private JBits jbits;
  private RouteTracer tracer;

  public TaskBitstream(String instanceName, Net clk, Net dout, String nombreArchivo) throws
CoreException
  {
    super(instanceName, clk, dout);

    // Cargar el bitstream con el core
    this.jbits = new JBits(Devices.XCV1000);

    try
    {
      jbits.read(nombreArchivo);
    }
    catch(Exception e)
    {
      throw new CoreException(e);
    }

    // Establecer las propiedades del core
    this.tracer = new RouteTracer(jbits);
  }

  public void implement() throws CoreException
  {
    int [] valor;
    int row, col;
    int i, j, k;
    Pin doutPin;
    Net clkNet, doutNet;
    wireNet wireClk, wireDout;
    Offset wireOffset;

    // Calcular el offset
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
    col = offset.getHorOffset(Gran.CLB);

    // Crear las señales internas
    clkNet = newNet("clk");
    doutNet = newNet("dout");

    // Conectar las señales con el puerto correspondiente
    clkPort.setIntSig(clkNet);
    doutPort.setIntSig(doutNet);

    // Copiar la configuracion de las CLBs en la posicion indicada
    try
    {
      for(i = 0; i < super.calcHeight(); i++)
        for(j = 0; j < super.calcWidth(); j++)
          for(k = 0; k < clbParams.length; k++)
          {
            valor = jbits.get(i, j, clbParams[k]);
            Bitstream.set(row + i, col + j, clbParams[k], valor);
          }
    }
    catch(Exception e)
    {
      throw new CoreException(e);
    }

    // Rutar la salida
    doutPin = routeDout();

    // Crear las conexiones con el bus
    wireClk = new wireNet("wireClk", clkNet, null, routeClk());
    wireDout = new wireNet("wireDout", doutNet, doutPin, routeDout(doutPin));

    // Añadir los cores
    addChild(wireClk);
    addChild(wireDout);
```

```
    // Situar
    wireOffset = wireClk.getRelativeOffset();
    wireOffset.setVerOffset(Gran.CLB, 0);
    wireOffset.setHorOffset(Gran.CLB, 0);

    wireOffset = wireDout.getRelativeOffset();
    wireOffset.setVerOffset(Gran.CLB, 0);
    wireOffset.setHorOffset(Gran.CLB, 0);

    // Implementar
    wireClk.implement();
    wireDout.implement();

    // Crear las rutas internas
    for(i = 0; i < super.calcHeight(); i++)
      for(j = 0; j < super.calcWidth(); j++)
        routeCLB(i, j, routeDout());
  }

  private void routeCLB(int row, int col, Pin doutPin) throws CoreException
  {
    int i, j;
    Pin source;
    Pin[] sinks;
    wireNet wire;
    Net net;

    for(i = 0; i < sourcePins.length; i++)
    {
      // Determinar el pin fuente
      source = new Pin(Pin.CLB, row, col, sourcePins[i]);

      // Comprobar que no se trata de la salida
      if(!equal(source, doutPin))
      {
        // Obtener el conjunto de pines que estan conectados a él
        sinks = routePin(source);

        // Comprobar si hay algun pin conectado
        if(sinks.length > 0)
        {
          // Crear la nueva net
          net = new Net("net" + i, this);

          // Crear el core para unir la net
          wire = new wireNet("wire" + i, net, source, sinks);

          // Añadirlo
          addChild(wire);

          // Situar el cable
          Offset wireOffset = wire.getRelativeOffset();
          wireOffset.setVerOffset(Gran.CLB, 0);
          wireOffset.setHorOffset(Gran.CLB, 0);

          // Implementar el core
          wire.implement();

          // Realizar las conexiones
          Bitstream.connect(net);
        }
      }
    }
  }

  private Pin[] routePin(Pin source) throws CoreException
  {
    Pin[] resultado;
    RouteTree t;
    RouteTree[] ts;
    Vector sinks;
    int i;

    // Crear el vector de pines destino
    sinks = new Vector();
```

```
  // Crear el arbol de ruta para el pin
  t = new RouteTree(source);

  try
  {
    // Obtener la traza
    tracer.trace(t);
  }
  catch(Exception e)
  {
    throw new CoreException(e);
  }

  // Hacer un recorrido en profundidad
  ts = t.getBottom();

  // Obtener los pines que son destino
  for(i = 0; i < ts.length; i++)
  {
    if(ts[i].sink())
      sinks.addElement(ts[i].getPin());
  }

  // Devolver la lista de pines
  resultado = new Pin[sinks.size()];

  for(i = 0; i < sinks.size(); i++)
    resultado[i] = (Pin)sinks.get(i);

  return resultado;
}

private Pin[] routeClk() throws CoreException
{
  int i, j;
  int[] valor;
  Vector sinks;
  Pin[] resultado;

  // Crear el vector de pines de reloj
  sinks = new Vector();

  try
  {
    for(i = 0; i < super.calcHeight(); i++)
      for(j = 0; j < super.calcWidth(); j++)
      {
        // Comprobar si el reloj es usado en el slice 0
        valor = jbits.get(i, j, S0Clk.S0Clk);

        if(valor != S0Clk.OFF)
          sinks.add(new Pin(Pin.CLB, i, j, CenterWires.S0_CLK));

        // Comprobar si el reloj es usado en el slice 1
        valor = jbits.get(i, j, S1Clk.S1Clk);

        if(valor != S1Clk.OFF)
          sinks.add(new Pin(Pin.CLB, i, j, CenterWires.S1_CLK));
      }
  }
  catch(Exception e)
  {
    throw new CoreException(e);
  }

  // Devolver la lista de pines
  resultado = new Pin[sinks.size()];

  for(i = 0; i < sinks.size(); i++)
    resultado[i] = (Pin)sinks.get(i);

  return resultado;
}
```

```
  private Pin routeDout()
  {
    RouteTree t;

    // Localizar el pin fuente que conecta con Dout
    t = new RouteTree(doutIobPin);
    tracer.reverseTrace(t);

    return t.getTop().getPin();
  }

  private Pin[] routeDout(Pin source) throws CoreException
  {
    Pin pin;
    Pin[] resultado;
    RouteTree t;
    RouteTree[] ts;
    Vector sinks;
    int i;

    // Crear el vector de pines destino
    sinks = new Vector();

    // Crear el arbol de ruta para el pin
    t = new RouteTree(source);

    try
    {
      // Obtener la traza
      tracer.trace(t);
    }
    catch(Exception e)
    {
      throw new CoreException(e);
    }

    // Hacer un recorrido en profundidad
    ts = t.getBottom();

    // Obtener los pines que son destino y no son la salida
    for(i = 0; i < ts.length; i++)
    {
      if(ts[i].sink())
      {
        pin = ts[i].getPin();

        if(!equal(pin, doutIobPin))
          sinks.add(pin);
      }
    }

    // Devolver la lista de pines
    resultado = new Pin[sinks.size()];

    for(i = 0; i < sinks.size(); i++)
      resultado[i] = (Pin)sinks.get(i);

    return resultado;
  }

  private boolean equal(Pin a, Pin b)
  {
    return a.equals(b);
  }
}
```

Clase wireNet

```
package mtbits.RTPCore;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Offset;
```

```
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

public class wireNet extends RTPCore
{
  // Puertos del core
  private Port dinPort;
  private Port doutPort;

  // Parametros del core
  private Pin sourcePin;
  private Pin[] sinksPin;

  // Configura una net a partir de los pines especificados de forma que pueda ser
  // utilizada por otro core de nivel superior
  public wireNet(String instanceName, Net net, Pin sourcePin, Pin[] sinksPin) throws
CoreException
  {
    super(instanceName);

    // Comprobar los parametros del core
    try
    {
      checkParameters(sourcePin, sinksPin);
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Asignar los parametros del core
    this.sourcePin = sourcePin;
    this.sinksPin = sinksPin;

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear los puertos
    if(sourcePin != null)
      doutPort = newOutputPort("DOUT", net);

    if(sinksPin != null)
      if(sinksPin.length > 0)
        dinPort = newInputPort("DIN", net);
  }

  private void checkParameters(Pin sourcePin, Pin[] sinksPin) throws CoreParameterException
  {
    // Comprobar que el numero de pines destino no es cero
    if(sinksPin == null)
    {
      if(sourcePin == null)
        throw new CoreParameterException(this, "No hay pines en la net.");
    }
    else
    {
      if(sinksPin.length == 0 && sourcePin == null)
        throw new CoreParameterException(this, "No hay pines en la net.");
    }
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.CLB;
```

```
  }

  // Dimensiones del core
  public static int calcHeight()
  {
    return 0;
  }

  public static int calcWidth()
  {
    return 0;
  }

  // Crea el contenido del core
  public final void implement() throws CoreException
  {
    int i, row, col;
    Pin nSourcePin;
    Pin[] nSinksPin;

    // Calcular el offset
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
    col = offset.getHorOffset(Gran.CLB);

    if(sourcePin != null)
    {
      // Calcular la nueva posicion del pin fuente
      nSourcePin = sourcePin;
      nSourcePin.setCol(sourcePin.getCol() + col);
      nSourcePin.setRow(sourcePin.getRow() + row);

      // Asignar el pin al puerto correspondiente
      doutPort.setPin(nSourcePin);
    }

    if(sinksPin.length > 0)
    {
      // Calcular las nuevas posiciones de los pines destino
      nSinksPin = new Pin[sinksPin.length];

      for(i = 0; i < sinksPin.length; i++)
      {
        nSinksPin[i] = sinksPin[i];
        nSinksPin[i].setCol(sinksPin[i].getCol() + col);
        nSinksPin[i].setRow(sinksPin[i].getRow() + row);
      }

      // Asignar los pines a los puertos
      dinPort.setPin(nSinksPin);
    }
  }
}
```

Programa TestmtBits para el simulador

```
package testmtbits;

// Swing
import javax.swing.Timer;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import javax.swing.JToggleButton;
import javax.swing.BorderFactory;
import javax.swing.border.LineBorder;

// Awt
import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.FlowLayout;
import java.awt.event.WindowEvent;
```

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.AdjustmentEvent;
import java.awt.event.AdjustmentListener;

import com.borland.jbcl.layout.VerticalFlowLayout;

// mtbits
import mtbits.mtBits;
import mtbits.mtbException;

public class frmMain extends JFrame
{
  // Interfaz con la fpga
  private mtBits mtb;

  // Constante necesaria para cerrar la ventana a terminar
  private final int EXIT_ON_CLOSE = 3;

  // Componentes
  private Timer timer;

  private JPanel jContentPane = new JPanel();
  private JPanel jPTasks = new JPanel();
  private JScrollPane jSPLog = new JScrollPane();
  private JToggleButton jTBTaskA = new JToggleButton();
  private JToggleButton jTBTaskB = new JToggleButton();
  private JToggleButton jTBTaskC = new JToggleButton();
  private JToggleButton jTBTaskD = new JToggleButton();
  private GridLayout gridLayout1 = new GridLayout();
  private VerticalFlowLayout verticalFlowLayout1 = new VerticalFlowLayout();
  private JTextArea jTALog = new JTextArea();
  private JPanel jPEstado = new JPanel();
  private JLabel jLLed[] = new JLabel[4];
  private JPanel jPLeds = new JPanel();
  private FlowLayout flowLayout1 = new FlowLayout();
  private JToggleButton jTBRun = new JToggleButton();
  private GridLayout gridLayout2 = new GridLayout();

  private final String titulo = "Prueba de mtIOB en tiempo real";

  public frmMain()
  {
    try
    {
      jbInit();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
  }

  public static void main(String[] args)
  {
    frmMain frmMain = new frmMain();

    // Deshabilitar los eventos
    frmMain.enableInputMethods(false);

    // Mostrar el dialogo
    frmMain.show();

    frmMain.mtBitsInit();

    // Habilitar los eventos
    frmMain.enableInputMethods(true);
  }

  private void mtBitsInit()
  {
    try
    {
      mtb = new mtBits(System.out);
    }
```

```
    catch(Exception e)
    {
      error(e);
    }
  }

  private void jbInit() throws Exception
  {
    int i;

    // Configurar la ventana
    this.setTitle(titulo);
    this.setDefaultCloseOperation(EXIT_ON_CLOSE);
    this.setSize(new Dimension(600, 500));
    this.setResizable(false);
    this.getContentPane().add(jContentPane, null);
    this.setContentPane(jContentPane);

    // Contenedor principal
    jContentPane.setLayout(verticalFlowLayout1);

    // Tareas
    jPTasks.setLayout(gridLayout1);

    jTBTaskA.setText("A");
    jTBTaskB.setText("B");
    jTBTaskC.setText("C");
    jTBTaskD.setText("D");

    jTBTaskA.setFont(new java.awt.Font("Dialog", 0, 24));
    jTBTaskB.setFont(new java.awt.Font("Dialog", 0, 24));
    jTBTaskC.setFont(new java.awt.Font("Dialog", 0, 24));
    jTBTaskD.setFont(new java.awt.Font("Dialog", 0, 24));

    // Leds
    jPLeds.setLayout(flowLayout1);
    jPLeds.setBorder(BorderFactory.createRaisedBevelBorder());

    for(i = 0; i < 4; i++)
    {
      jLLed[i] = new JLabel();
      jLLed[i].setText("  ");
      jLLed[i].setOpaque(true);
      jLLed[i].setBorder(BorderFactory.createLineBorder(Color.black));
    }

    // Estado de la ejecucion
    jTBRun.setText("Ejecutar");
    jPEstado.setLayout(gridLayout2);

    // Log
    jSPLog.setAutoscrolls(true);
    jTALog.setEditable(false);
    jTALog.setText("");
    jTALog.setRows(22);

    // Añadir cada cosa a su sitio
    jContentPane.add(jPTasks, null);
    jContentPane.add(jPEstado, null);
    jContentPane.add(jSPLog, null);
    jPEstado.add(jTBRun, null);
    jPEstado.add(jPLeds, null);
    jPTasks.add(jTBTaskA, null);
    jPTasks.add(jTBTaskB, null);
    jPTasks.add(jTBTaskC, null);
    jPTasks.add(jTBTaskD, null);
    jSPLog.getViewport().add(jTALog, null);

    for(i = 0; i < 4; i++)
      jPLeds.add(jLLed[i], null);

    // Eventos
    this.addWindowListener(new java.awt.event.WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        this_windowClosing(e);
```

```
    }
  });

  jTBTaskA.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jTBTaskA_actionPerformed(e);
    }
  });

  jTBTaskB.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      jTBTaskB_actionPerformed(e);
    }
  });

  jTBTaskC.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      jTBTaskC_actionPerformed(e);
    }
  });

  jTBTaskD.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      jTBTaskD_actionPerformed(e);
    }
  });

  jTBRun.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      jTBRun_actionPerformed(e);
    }
  });

  // Autoscroll
  jSPLog.getVerticalScrollBar().addAdjustmentListener(new AdjustmentListener()
  {
    public void adjustmentValueChanged(AdjustmentEvent e)
    {
      jTALog.select(jTALog.getDocument().getLength(),jTALog.getDocument().getLength());
    }
  });

  // Contador
  timer = new Timer(3000,new ActionListener(){
    public void actionPerformed(ActionEvent evt)
    {
      try
      {
        // Pasar un ciclo
        mtb.step();
        leerLeds();
      }
      catch(Exception ex)
      {
        error(ex);
        timer.stop();
      }
    }
  });
}

private void leerLeds()
{
  int[] valor;
  int i;
  Color estado;
```

```
    valor = mtb.getStatus();

    for(i = 0; i < 4; i++)
    {
      // Determinar el nuevo estado
      estado = (valor[i] == 1) ? Color.red : Color.black;

      // Cambiar el color del led
      jLLed[i].setBackground(estado);
    }
  }

  private void error(Exception e)
  {
    // Volver al titulo original
    setTitle(titulo);

    // Mostrar un mensaje de error
    jTALog.append("error\n" + e.getClass() + "\n" + e.toString() + "\n\n");
  }

  void this_windowClosing(WindowEvent e)
  {
    try
    {
      //Parar el timer si se esta ejecutando
      if(timer.isRunning())
        timer.stop();

      // Desconectar la placa
      mtb.close();
    }
    catch(Exception ex)
    {
      error(ex);
    }
  }

  void jTBTaskA_actionPerformed(ActionEvent e)
  {
    try
    {
      // Comprobar que el boton ha sido pulsado
      if(jTBTaskA.isSelected())
      {
        mtb.load("counter.bit", 0);
      }
      else
      {
        mtb.clear(0);
      }
    }
    catch(Exception ex)
    {
      // Se ha producido un error
      error(ex);
    }
  }

  void jTBTaskB_actionPerformed(ActionEvent e)
  {
    try
    {
      // Comprobar que el boton ha sido pulsado
      if(jTBTaskB.isSelected())
      {
        mtb.load("counter.bit", 1);
      }
      else
      {
        mtb.clear(1);
      }
    }
    catch(Exception ex)
    {
```

```
      // Se ha producido un error
      error(ex);
    }
  }

  void jTBTaskC_actionPerformed(ActionEvent e)
  {
    try
    {
      // Comprobar que el boton ha sido pulsado
      if(jTBTaskC.isSelected())
      {
        mtb.load("counter.bit", 2);
      }
      else
      {
        mtb.clear(2);
      }
    }
    catch(Exception ex)
    {
      // Se ha producido un error
      error(ex);
    }
  }

  void jTBTaskD_actionPerformed(ActionEvent e)
  {
    try
    {
      // Comprobar que el boton ha sido pulsado
      if(jTBTaskD.isSelected())
      {
        mtb.load("counter.bit", 3);
      }
      else
      {
        mtb.clear(3);
      }
    }
    catch(Exception ex)
    {
      // Se ha producido un error
      error(ex);
    }
  }

  void jTBRun_actionPerformed(ActionEvent e)
  {
    // Comprobar si se ha marcado el boton
    if(jTBRun.isSelected())
      timer.start();
    else
      timer.stop();
  }
}
```

Libreria mtBits para la Virtex 1000

Clase mtbException

```
package mtbits;

import java.lang.Throwable;

public class mtbException extends Exception
{
  private Throwable cause;

  public mtbException(Throwable e)
  {
    super();

    this.cause = e;
```

```
  }

  public mtbException(String text)
  {
    super(text);

    cause = null;
  }

  public Throwable getCause()
  {
    return cause;
  }

  public String toString()
  {
    if(cause != null)
      return cause.toString();
    else
      return this.toString();
  }

  public String getMessage()
  {
    if(cause != null)
      return cause.getMessage();
    else
      return super.getMessage();
  }
}
```

Clase mtBits

```
package mtbits;

// Java
import java.io.PrintStream;
import java.io.IOException;

// Jbits
import com.xilinx.XHWIF.XHWIF;
import com.xilinx.XHWIF.XHWIFException;
import com.xilinx.XHWIF.Boards.xcv1000;

import com.xilinx.JBits.Virtex.Devices;
import com.xilinx.JBits.Virtex.JBits;
import com.xilinx.JBits.Virtex.Bits.CLB;
import com.xilinx.JBits.Virtex.ReadbackCommand;
import com.xilinx.JBits.Virtex.ConfigurationException;

import com.xilinx.JBits.Virtex.RTPCore.Basic.Clock;

import com.xilinx.JBits.CoreTemplate.NetPinsList;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Bus;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.CoreOutput;
import com.xilinx.JBits.CoreTemplate.CoreException;

import com.xilinx.JRoute2.Virtex.JRoute;
import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;

import com.xilinx.DeviceSimulator.Virtex.SimulatorClient;

// Cores
import mtbits.RTPCore.mtBoard;
import mtbits.RTPCore.mtIOB;
import mtbits.RTPCore.Register;
import mtbits.RTPCore.TaskBitstream;

public class mtBits
```

```
{
  // Registro de errores
  private PrintStream debugOutput;

  // Definicion de la placa
  private final String boardName = "rc1000pp:xcv1000";
  private final int deviceType = Devices.XCV1000;

  // Objetos para trabajar con jbits
  private XHWIF board;
  private JBits jbits;
  private JRoute jroute;
  private int boardId;

  // Puntos de prueba
  private Pin[] ledPin;

  // Registro de las rutas usadas por cada tarea
  private NetPinsList[] taskNetPinsList;

  // Cores usados
  private mtIOB mtiobus;

  public mtBits() throws mtbException, IOException
  {
    this(null);
  }

  public mtBits(PrintStream debugOutput) throws mtbException
  {
    int i, ledbusOutput;
    mtBoard mtboard;
    Clock clock;
    Bus ledBus;
    Net clkNet;

    // Inicializar las listas de pines
    taskNetPinsList = new NetPinsList[4];

    // Preparar la salida
    this.debugOutput = debugOutput;

    try
    {
      // Crear la conexion con la placa
      message("Conectar con " + boardName);
      board = XHWIF.Get(boardName);
      boardId = board.connect();

      // Comprobar que se ha encontrado la placa
      if(boardId < 0)
        throw new mtbException("No se ha encontrado la placa");
      done();

      message("Resetear la placa");
      board.reset();
      done();

      message("Cargar el bitstream nulo en la placa " + boardId);
      jbits = new JBits(deviceType);
      jbits.readPartial("null1000.bit");
      board.setConfiguration(boardId, jbits.getPartial());
      done();

      // Inicializar el interfaz
      message("Preparar el bitstream");
      jroute = new JRoute(jbits, System.out);
      Bitstream.setVirtex(jbits, jroute);
      CoreOutput.generateBitstream(true);
      CoreOutput.generateSYM(false);
      CoreOutput.generateXDL(false);

      // Establecer los puntos de prueba
      ledPin = new Pin[4];
```

```
    for(i = 0; i < 4; i++)
      ledPin[i] = new Pin(Pin.CLB, 0, 24 * i, CenterWires.S0_XQ);

    done();

    // Crear las señales internas
    clkNet = new Net("CLK", null);
    ledBus = new Bus("LED", null, 4);

    message("Configurar los IOBs de la placa");

    // Crear las conexiones con los leds
    mtboard = new mtBoard("XCV1000_BG560");

    // Añadir la salida de los leds a la placa
    ledbusOutput = mtboard.addOutput(ledBus.getName(), ledBus);

    // Configurar el IOB seleccionado para salida no invertida
    mtboard.setOutputClk(ledbusOutput, clkNet);
    mtboard.setOutputInvertT(ledbusOutput, true);
    mtboard.setOutputInvertO(ledbusOutput, false);
    done();

    // Implementar la placa con el ucf especificado
    message("Cargar las posiciones de los leds");
    mtboard.implement(boardId, "xcv1000_bg560.ucf");
    done();

    // Crear el reloj
    message("Crear el reloj");
    clock = new Clock("clock", clkNet);
    clock.implement(mtboard.getGCLK());
    done();

    // Crear el bus
    message("Preparar el bus");
    mtiobus = new mtIOB("mtiobus", clkNet, ledBus);

    // Establecer la posicion del nuevo core
    Offset mtiobusOffset = mtiobus.getRelativeOffset();
    mtiobusOffset.setHorOffset(Gran.CLB, 0);
    mtiobusOffset.setVerOffset(Gran.CLB, 0);

    // Implementarlo
    mtiobus.implement();
    done();

    // Rutar las señales internas
    message("Rutar la señal de reloj");
    Bitstream.connect(clkNet);
    done();

    message("Rutar las conexiones con los leds");
    Bitstream.connect(ledBus);
    done();

    // Actualizar la placa
    update();
  }
  catch(Exception e)
  {
    fail();

    throw new mtbException(e);
  }
}

public void step() throws mtbException
{
  message("Ciclo de reloj");
  board.clockOn();
  board.clockStep(1);
  done();
}
```

```
  public void load(String bitstream, int taskIndex) throws mtbException
  {
    Net dinTaskNet, clkTaskNet;

    try
    {
      message("Crear una instancia para la tarea " + taskIndex);

      // Crear las conexiones internas
      dinTaskNet = new Net("Din" + taskIndex, null);
      clkTaskNet = new Net("Clk"+ taskIndex, null);

      // Crear el core
      TaskBitstream task = new TaskBitstream("task" + taskIndex, clkTaskNet, dinTaskNet,
bitstream);

      // Añadir el core al bus
      mtiobus.setTask(dinTaskNet, clkTaskNet, taskIndex);

      // Establecer la posicion
      Offset taskOffset = task.getRelativeOffset();
      taskOffset.setHorOffset(Gran.CLB, taskIndex * 24);
      taskOffset.setVerOffset(Gran.CLB, 1);
      done();

      // Iniciar la grabacion de los recursos usados
      taskNetPinsList[taskIndex] = Bitstream.startRouteRecording();

      // Configurar la logica del core
      message("Implementar");
      task.implement();
      done();

      // Conectar las señales con el bus
      message("Realizar las conexiones con el bus");
      Bitstream.connect(dinTaskNet);

      // Detener la grabacion
      Bitstream.endRouteRecording();

      // Conectar el reloj
      Bitstream.connect(clkTaskNet);
      done();

      // Reconfigurar la placa
      update();
    }
    catch(Exception e)
    {
      fail();

      throw new mtbException(e);
    }
  }

  public void clear(int taskIndex) throws mtbException
  {
    try
    {
      message("Limpiar la tarea " + taskIndex);
      Bitstream.unroute(taskNetPinsList[taskIndex]);
      done();

      // Borrar la lista de pines usados
      taskNetPinsList[taskIndex] = null;

      // Reconfigurar
      update();
    }
    catch(Exception e)
    {
      fail();

      throw new mtbException(e);
    }
```

```
  }

  public int[] getStatus()
  {
    int [] estado;
    int i;

    estado = new int[4];

    message("Leer el estado de los leds");

    // Leer el valor de cada led
//    for(i = 0; i < 4; i++)
//      estado[i] = simulatorclient.probePinValue(ledPin[i]);

    done();

    return estado;
  }

  public void close() throws mtbException
  {
    try
    {
      // Desconectar la placa
      message("Desconectar de la placa " + boardName);
      board.reset();
      board.disconnect();
      done();
    }
    catch(Exception e)
    {
      fail();

      throw new mtbException(e);
    }
  }

  private void update() throws XHWIFException, ConfigurationException
  {
    // Escribir la configuracion parcial en la placa
    message("Cargando el bitstream parcial en la placa");
    board.setConfiguration(boardId, jbits.getPartial());
    done();
  }

  private void message(String text)
  {
    if(debugOutput != null)
      debugOutput.print(text + " ... ");
  }

  private void done()
  {
    if(debugOutput != null)
      debugOutput.println("hecho");
  }

  private void fail()
  {
    if(debugOutput != null)
      debugOutput.println("error");
  }
}
```

Clase mtBoard

```
package mtbits.RTPCore;

import com.xilinx.JBits.Virtex.RTPCore.Iob.Board;
import com.xilinx.JBits.Virtex.RTPCore.Iob.XCVPackage;
import com.xilinx.JBits.Virtex.RTPCore.Iob.xcv1000_bg560;

import com.xilinx.JBits.CoreTemplate.CoreException;
```

```
public class mtBoard extends Board
{
  // Empaquetado. Uno por cada fpga. Solo tenemos 1 en la placa.
  private XCVPackage xcvPackage[] = { new xcv1000_bg560() };

  // El reloj global
  private int GCLK = 3;

  public mtBoard(String instanceName) throws CoreException
  {
    super(instanceName);

    // Establecer el empaquetado de la placa
    setXcvPackage(xcvPackage);

    // Definir la configuracion de la fpga
    setGCLK(GCLK);
  }
}
```

Clase mtIOB

```
package mtbits.RTPCore;

import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Bus;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JBits.Virtex.ConfigurationException;
import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;

public class mtIOB extends RTPCore
{
  // Definicion de la altura del core
  private final static int rows = 1;

  // Puertos de entrada del core
  private Port[] dinTaskPort; // Entrada desde una tarea
  private Port clkPort; // Entrada del reloj

  // Puertos de salida del core
  private Port doutPort; // Salida a los leds
  private Port[] clkTaskPort; // Reloj para una tarea A

  // Registros de entrada/salida
  private Register[] registerDinTask;

  public mtIOB(String instanceName,
               Net clk,
               Bus dout) throws CoreException
  {
    super(instanceName);

    int i;

    // Comprobar los parametros del core
    try
    {
      checkParameters(dout);
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
```

```
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear los puertos de entrada
    dinTaskPort = new Port[4];

    for(i = 0; i < 4; i++)
      dinTaskPort[i] = newInputPort("DIN" + i, Net.NoConnect);

    clkPort = newInputPort("CLK", clk);

    // Crear los puertos de salida
    doutPort = newOutputPort("DOUT", dout);

    clkTaskPort = new Port[4];

    for(i = 0; i < 4; i++)
      clkTaskPort[i] = newOutputPort("CLK" + i, Net.NoConnect);
  }

  private void checkParameters(Bus dout) throws CoreParameterException
  {
    // Comprobar que el ancho del bus de salida es exactamente 4
    // (uno por cada tarea).
    if(dout.getWidth() != 4)
      throw new CoreParameterException(this, "El ancho del bus de salida tiene que ser
4.");
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.CLB;
  }

  // Dimensiones del core
  public static int calcHeight()
  {
    // Este valor se define al principio de la clase
    return rows;
  }

  public static int calcWidth()
  {
    // Ocupa toda una fila de la FPGA
    return Bitstream.getVirtex().getClbColumns();
  }

  public final void implement() throws CoreException
  {
    int row, col, i, j;
    Offset bufferOffset;
    Net[] dinTaskNet;
    Net clkNet;
    Bus doutBus;

    // Calcular el offset
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
    col = offset.getHorOffset(Gran.CLB);

    // Crear las señales internas
    dinTaskNet = new Net[4];

    for(i = 0; i < 4; i++)
      dinTaskNet[i] = newNet("din" + i);

    doutBus = newBus("data", 4);
    clkNet = newNet("clk");
```

```
    // Conectar puertos a la net correspondiente
    doutPort.setIntSig(doutBus);
    clkPort.setIntSig(clkNet);

    for(i = 0; i < 4; i++)
    {
      dinTaskPort[i].setIntSig(dinTaskNet[i]);
      clkTaskPort[i].setIntSig(clkNet);
    }

    // Crear el vector de registros
    registerDinTask = new Register[4];

    // Crear los buffers para las entradas y salidas con las tareas
    for(i = 0; i < 4; i++)
    {
      registerDinTask[i] = new Register("registerDin" + i, dinTaskNet[i], clkNet,
doutBus.getNet(i));

      // Situar los buffers a intervalos regulares
      bufferOffset = registerDinTask[i].getRelativeOffset();
      bufferOffset.setVerOffset(Gran.CLB, 0);
      bufferOffset.setHorOffset(Gran.CLB, i * 24);

      // Añadir el buffer
      addChild(registerDinTask[i]);

      // Implementar
      registerDinTask[i].implement();
    }
  }

  public void setTask(Net din, Net clk, int taskIndex) throws CoreException
  {
    Net dinNet, clkNet;

    // Comprobar que el indice de tarea no excede 4
    if(taskIndex > 4 || taskIndex < 0)
      throw new CoreException("Indice de la tarea fuera del rango [0, 4]");

    // Actualizar los puertos
    dinTaskPort[taskIndex] = newInputPort("DIN" + taskIndex, din);
    clkTaskPort[taskIndex] = newOutputPort("CLK" + taskIndex, clk);

    // Establecer las nuevas señales internas
    dinNet = newNet("din" + taskIndex);
    clkNet = (Net)clkPort.getIntSig();

    // Asignar las señales a los puertos
    dinTaskPort[taskIndex].setIntSig(dinNet);
    clkTaskPort[taskIndex].setIntSig(clkNet);

    // Actualizar el registro
    registerDinTask[taskIndex].setDin(dinNet);
  }
}
```

Clase Register

```
package mtbits.RTPCore;

import com.xilinx.JBits.Virtex.Expr;
import com.xilinx.JBits.Virtex.Bits.SliceControl;
import com.xilinx.JBits.Virtex.Bits.LUT;
import com.xilinx.JBits.Virtex.Bits.S0CE;
import com.xilinx.JBits.Virtex.ConfigurationException;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
```

```
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;

public class Register extends RTPCore
{
  // Puertos del core
  private Port dinPort;
  private Port clkPort;
  private Port doutPort;

  public Register(String instanceName, Net din, Net clk, Net dout) throws CoreException
  {
    super(instanceName);

    // Comprobar los parametros del core
    try
    {
      checkParameters();
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear los puerto de entrada
    dinPort = newInputPort("DIN", din);
    clkPort = newInputPort("CLK", clk);

    // Crear el puerto de salida
    doutPort = newOutputPort("DOUT", dout);
  }

  private void checkParameters() throws CoreParameterException
  {
    // En este caso no es necesario comprobar nada
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.CLB;
  }

  // Dimensiones del core
  public static int calcHeight()
  {
    return 1;
  }

  public static int calcWidth()
  {
    return 1;
  }

  // Crea el contenido del core
  public final void implement() throws CoreException
  {
    int row, col;
    Pin dinPin, clkPin, doutPin;

    // Calcular el offset
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
```

```
    col = offset.getHorOffset(Gran.CLB);

    // Configurar la CLB
    Bitstream.set(row, col, LUT.SLICE0_F, Expr.F_LUT("~F1"));
    Bitstream.set(row,col, S0CE.S0CE, S0CE.OFF);
    Bitstream.set(row, col, SliceControl.CeInvert[0], SliceControl.OFF[0]);
    Bitstream.set(row,col, SliceControl.LatchMode[0], SliceControl.OFF[0]);
    Bitstream.set(row,col, SliceControl.ClockInvert[0], SliceControl.OFF[0]);
    Bitstream.set(row, col, SliceControl.X.X[0], SliceControl.X.FOUT[0]);
    Bitstream.set(row,col, SliceControl.XDin.XDin[0], SliceControl.XDin.X[0]);

    // Crear los pines de las entradas
    dinPin = new Pin(Pin.CLB, row, col, CenterWires.SliceF1[0]);
    clkPin = new Pin(Pin.CLB, row, col, CenterWires.SliceClk[0]);

    // Crear el pin de la salida
    doutPin = new Pin(Pin.CLB, row, col, CenterWires.Slice_XQ[0]);

    // Asignar los pines a los puertos
    dinPort.setPin(dinPin);
    clkPort.setPin(clkPin);
    doutPort.setPin(doutPin);
  }

  public void setDin(Net din) throws CoreException
  {
    Pin dinPin;

    if(dinPort.hasPins() == true)
    {
      // Recuperar el pin de entrada
      dinPin = dinPort.getPins(0)[0];
    }
    else
    {
      throw new CoreException("No se ha implementado el registro");
    }

    // Crear el nuevo puerto
    dinPort = newInputPort("DIN", din);

    // Asignar el pin al puerto
    dinPort.setPin(dinPin);
  }

  public void setDout(Net dout) throws CoreException
  {
    Pin doutPin;

    if(doutPort.hasPins() == true)
    {
      // Recuperar el pin de entrada
      doutPin = doutPort.getPins(0)[0];
    }
    else
    {
      throw new CoreException("No se ha implementado el registro");
    }

    // Crear el nuevo puerto
    doutPort = newOutputPort("DOUT", dout);

    // Asignar el pin al puerto
    doutPort.setPin(doutPin);
  }
}
```

Clase Task

```
package mtbits.RTPCore;

import java.io.Serializable;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
```

```
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

import com.xilinx.JBits.Virtex.ConfigurationException;

public abstract class Task extends RTPCore implements Serializable
{
  // Puertos del core
  protected Port doutPort;
  protected Port clkPort;

  public Task(String instanceName, Net clk, Net dout) throws CoreException
  {
    super(instanceName);

    // Comprobar los parametros
    try
    {
      checkParameters();
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear los puertos
    clkPort = newInputPort("CLK", clk);
    doutPort = newOutputPort("DOUT", dout);
  }

  private void checkParameters() throws CoreParameterException
  {
    // En este caso no hace falta comprobar nada
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.CLB;
  }

  // Dimensiones del core
  public static int calcHeight()
  {
    return 63;
  }

  public static int calcWidth()
  {
    return 24;
  }

  // Crea el contenido del core
  public abstract void implement() throws CoreException;
}
```

Clase TaskBitstream

```
package mtbits.RTPCore;

import java.util.Vector;
```

```
import com.xilinx.JBits.Virtex.JBits;
import com.xilinx.JBits.Virtex.Devices;

import com.xilinx.JBits.Virtex.Bits.S0Control;
import com.xilinx.JBits.Virtex.Bits.S1Control;
import com.xilinx.JBits.Virtex.Bits.IOB;
import com.xilinx.JBits.Virtex.Bits.LUT;
import com.xilinx.JBits.Virtex.Bits.S0RAM;
import com.xilinx.JBits.Virtex.Bits.S1RAM;
import com.xilinx.JBits.Virtex.Bits.S0Clk;
import com.xilinx.JBits.Virtex.Bits.S1Clk;

import com.xilinx.DeviceSimulator.Virtex.RouteTracer;
import com.xilinx.DeviceSimulator.Virtex.RouteTree;

import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;
import com.xilinx.JRoute2.Virtex.ResourceDB.IobWiresBottom;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Bus;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

public class TaskBitstream extends Task
{
  // Lista de parametros de la configuracion de una clb
  private static final int [][][] clbParams = {
    S0Control.BxInvert, S0Control.ByInvert, S0Control.CeInvert, S0Control.ClockInvert,
    S0Control.InvertedSetReset, S0Control.LatchMode, S0Control.SrWeNotInvert,
    S0Control.Sync, S0Control.XffSetResetSelect, S0Control.YffSetResetSelect,
    S0Control.AndMux.AndMux, S0Control.Cin.Cin, S0Control.X.X,
    S0Control.XCarrySelect.XCarrySelect, S0Control.XDin.XDin, S0Control.Y.Y,
    S0Control.YB.YB, S0Control.YCarrySelect.YCarrySelect, S0Control.YDin.YDin,
    S1Control.BxInvert, S1Control.ByInvert, S1Control.CeInvert, S1Control.ClockInvert,
    S1Control.InvertedSetReset, S1Control.LatchMode, S1Control.SrWeNotInvert,
    S1Control.Sync, S1Control.XffSetResetSelect, S1Control.YffSetResetSelect,
    S1Control.AndMux.AndMux, S1Control.Cin.Cin, S1Control.X.X,
    S1Control.XCarrySelect.XCarrySelect, S1Control.XDin.XDin, S1Control.Y.Y,
    S1Control.YB.YB, S1Control.YCarrySelect.YCarrySelect, S1Control.YDin.YDin,
    LUT.SLICE0_F, LUT.SLICE0_G, LUT.SLICE1_F, LUT.SLICE1_G,
    S0RAM.DUAL_MODE, S0RAM.F_LUT_RAM, S0RAM.F_LUT_SHIFTER, S0RAM.G_LUT_RAM,
    S0RAM.G_LUT_SHIFTER, S0RAM.LUT_MODE, S0RAM.RAM_32_X_1, S0RAM.UNUSED,
    S1RAM.DUAL_MODE, S1RAM.F_LUT_RAM, S1RAM.F_LUT_SHIFTER, S1RAM.G_LUT_RAM,
    S1RAM.G_LUT_SHIFTER, S1RAM.LUT_MODE, S1RAM.RAM_32_X_1, S1RAM.UNUSED
  };

  // Lista de fines fuente en una CLB
  private static final int[] sourcePins = {
    CenterWires.S0_X,
    CenterWires.S0_Y,
    CenterWires.S0_XQ,
    CenterWires.S0_YQ,
    CenterWires.S0_XB,
    CenterWires.S0_YB,
    CenterWires.S1_X,
    CenterWires.S1_Y,
    CenterWires.S1_XQ,
    CenterWires.S1_YQ,
    CenterWires.S1_XB,
    CenterWires.S1_YB,
  };

  // El pin que se usa como referencia para el puerto DOUT
  private static final Pin doutIobPin = new Pin(Pin.IOB, IOB.BOTTOM, 14,
IobWiresBottom.O[1]);

  // Parametros del core
  private JBits jbits;
```

```
  private RouteTracer tracer;

  public TaskBitstream(String instanceName, Net clk, Net dout, String nombreArchivo) throws
CoreException
  {
    super(instanceName, clk, dout);

    // Cargar el bitstream con el core
    this.jbits = new JBits(Devices.XCV1000);

    try
    {
      jbits.read(nombreArchivo);
    }
    catch(Exception e)
    {
      throw new CoreException(e);
    }

    // Establecer las propiedades del core
    this.tracer = new RouteTracer(jbits);
  }

  public void implement() throws CoreException
  {
    int [] valor;
    int row, col;
    int i, j, k;
    Pin doutPin;
    Net clkNet, doutNet;
    wireNet wireClk, wireDout;
    Offset wireOffset;

    // Calcular el offset
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
    col = offset.getHorOffset(Gran.CLB);

    // Crear las señales internas
    clkNet = newNet("clk");
    doutNet = newNet("dout");

    // Conectar las señales con el puerto correspondiente
    clkPort.setIntSig(clkNet);
    doutPort.setIntSig(doutNet);

    // Copiar la configuracion de las CLBs en la posicion indicada
    try
    {
      for(i = 0; i < super.calcHeight(); i++)
        for(j = 0; j < super.calcWidth(); j++)
          for(k = 0; k < clbParams.length; k++)
          {
            valor = jbits.get(i, j, clbParams[k]);
            Bitstream.set(row + i, col + j, clbParams[k], valor);
          }
    }
    catch(Exception e)
    {
      throw new CoreException(e);
    }

    // Rutar la salida
    doutPin = routeDout();

    // Crear las conexiones con el bus
    wireClk = new wireNet("wireClk", clkNet, null, routeClk());
    wireDout = new wireNet("wireDout", doutNet, doutPin, routeDout(doutPin));

    // Añadir los cores
    addChild(wireClk);
    addChild(wireDout);

    // Situar
    wireOffset = wireClk.getRelativeOffset();
```

```
    wireOffset.setVerOffset(Gran.CLB, 0);
    wireOffset.setHorOffset(Gran.CLB, 0);

    wireOffset = wireDout.getRelativeOffset();
    wireOffset.setVerOffset(Gran.CLB, 0);
    wireOffset.setHorOffset(Gran.CLB, 0);

    // Implementar
    wireClk.implement();
    wireDout.implement();

    // Crear las rutas internas
    for(i = 0; i < super.calcHeight(); i++)
      for(j = 0; j < super.calcWidth(); j++)
        routeCLB(i, j, routeDout());
  }

  private void routeCLB(int row, int col, Pin doutPin) throws CoreException
  {
    int i, j;
    Pin source;
    Pin[] sinks;
    wireNet wire;
    Net net;

    for(i = 0; i < sourcePins.length; i++)
    {
      // Determinar el pin fuente
      source = new Pin(Pin.CLB, row, col, sourcePins[i]);

      // Comprobar que no se trata de la salida
      if(!equal(source, doutPin))
      {
        // Obtener el conjunto de pines que estan conectados a él
        sinks = routePin(source);

        // Comprobar si hay algun pin conectado
        if(sinks.length > 0)
        {
          // Crear la nueva net
          net = new Net("net" + i, this);

          // Crear el core para unir la net
          wire = new wireNet("wire" + i, net, source, sinks);

          // Añadirlo
          addChild(wire);

          // Situar el cable
          Offset wireOffset = wire.getRelativeOffset();
          wireOffset.setVerOffset(Gran.CLB, 0);
          wireOffset.setHorOffset(Gran.CLB, 0);

          // Implementar el core
          wire.implement();

          // Realizar las conexiones
          Bitstream.connect(net);
        }
      }
    }
  }

  private Pin[] routePin(Pin source) throws CoreException
  {
    Pin[] resultado;
    RouteTree t;
    RouteTree[] ts;
    Vector sinks;
    int i;

    // Crear el vector de pines destino
    sinks = new Vector();

    // Crear el arbol de ruta para el pin
```

```
    t = new RouteTree(source);

    try
    {
      // Obtener la traza
      tracer.trace(t);
    }
    catch(Exception e)
    {
      throw new CoreException(e);
    }

    // Hacer un recorrido en profundidad
    ts = t.getBottom();

    // Obtener los pines que son destino
    for(i = 0; i < ts.length; i++)
    {
      if(ts[i].sink())
        sinks.addElement(ts[i].getPin());
    }

    // Devolver la lista de pines
    resultado = new Pin[sinks.size()];

    for(i = 0; i < sinks.size(); i++)
      resultado[i] = (Pin)sinks.get(i);

    return resultado;
  }

  private Pin[] routeClk() throws CoreException
  {
    int i, j;
    int[] valor;
    Vector sinks;
    Pin[] resultado;

    // Crear el vector de pines de reloj
    sinks = new Vector();

    try
    {
      for(i = 0; i < super.calcHeight(); i++)
        for(j = 0; j < super.calcWidth(); j++)
        {
          // Comprobar si el reloj es usado en el slice 0
          valor = jbits.get(i, j, S0Clk.S0Clk);

          if(valor != S0Clk.OFF)
            sinks.add(new Pin(Pin.CLB, i, j, CenterWires.S0_CLK));

          // Comprobar si el reloj es usado en el slice 1
          valor = jbits.get(i, j, S1Clk.S1Clk);

          if(valor != S1Clk.OFF)
            sinks.add(new Pin(Pin.CLB, i, j, CenterWires.S1_CLK));
        }
    }
    catch(Exception e)
    {
      throw new CoreException(e);
    }

    // Devolver la lista de pines
    resultado = new Pin[sinks.size()];

    for(i = 0; i < sinks.size(); i++)
      resultado[i] = (Pin)sinks.get(i);

    return resultado;
  }

  private Pin routeDout()
  {
```

```
    RouteTree t;

    // Localizar el pin fuente que conecta con Dout
    t = new RouteTree(doutIobPin);
    tracer.reverseTrace(t);

    return t.getTop().getPin();
  }

  private Pin[] routeDout(Pin source) throws CoreException
  {
    Pin pin;
    Pin[] resultado;
    RouteTree t;
    RouteTree[] ts;
    Vector sinks;
    int i;

    // Crear el vector de pines destino
    sinks = new Vector();

    // Crear el arbol de ruta para el pin
    t = new RouteTree(source);

    try
    {
      // Obtener la traza
      tracer.trace(t);
    }
    catch(Exception e)
    {
      throw new CoreException(e);
    }

    // Hacer un recorrido en profundidad
    ts = t.getBottom();

    // Obtener los pines que son destino y no son la salida
    for(i = 0; i < ts.length; i++)
    {
      if(ts[i].sink())
      {
        pin = ts[i].getPin();

        if(!equal(pin, doutIobPin))
          sinks.add(pin);
      }
    }

    // Devolver la lista de pines
    resultado = new Pin[sinks.size()];

    for(i = 0; i < sinks.size(); i++)
      resultado[i] = (Pin)sinks.get(i);

    return resultado;
  }

  private boolean equal(Pin a, Pin b)
  {
    return a.equals(b);
  }
}
```

Clase wireNet

```
package mtbits.RTPCore;

import com.xilinx.JBits.CoreTemplate.Bitstream;
import com.xilinx.JBits.CoreTemplate.Gran;
import com.xilinx.JBits.CoreTemplate.Net;
import com.xilinx.JBits.CoreTemplate.Pin;
import com.xilinx.JBits.CoreTemplate.Offset;
import com.xilinx.JBits.CoreTemplate.Port;
import com.xilinx.JBits.CoreTemplate.RTPCore;
```

```
import com.xilinx.JBits.CoreTemplate.CoreException;
import com.xilinx.JBits.CoreTemplate.CoreParameterException;

public class wireNet extends RTPCore
{
  // Puertos del core
  private Port dinPort;
  private Port doutPort;

  // Parametros del core
  private Pin sourcePin;
  private Pin[] sinksPin;

  // Configura una net a partir de los pines especificados de forma que pueda ser
  // utilizada por otro core de nivel superior
  public wireNet(String instanceName, Net net, Pin sourcePin, Pin[] sinksPin) throws
CoreException
  {
    super(instanceName);

    // Comprobar los parametros del core
    try
    {
      checkParameters(sourcePin, sinksPin);
    }
    catch (CoreParameterException cpe)
    {
      throw new CoreException(cpe);
    }

    // Asignar los parametros del core
    this.sourcePin = sourcePin;
    this.sinksPin = sinksPin;

    // Calcular las dimensiones del core
    setHeightGran(calcHeightGran());
    setWidthGran(calcWidthGran());
    setHeight(calcHeight());
    setWidth(calcWidth());

    // Crear los puertos
    if(sourcePin != null)
      doutPort = newOutputPort("DOUT", net);

    if(sinksPin != null)
      if(sinksPin.length > 0)
        dinPort = newInputPort("DIN", net);
  }

  private void checkParameters(Pin sourcePin, Pin[] sinksPin) throws CoreParameterException
  {
    // Comprobar que el numero de pines destino no es cero
    if(sinksPin == null)
    {
      if(sourcePin == null)
        throw new CoreParameterException(this, "No hay pines en la net.");
    }
    else
    {
      if(sinksPin.length == 0 && sourcePin == null)
        throw new CoreParameterException(this, "No hay pines en la net.");
    }
  }

  // Calculo de las dimensiones de la unidad
  public static int calcHeightGran()
  {
    return Gran.CLB;
  }

  public static int calcWidthGran()
  {
    return Gran.CLB;
  }
```

```
  // Dimensiones del core
  public static int calcHeight()
  {
    return 0;
  }

  public static int calcWidth()
  {
    return 0;
  }

  // Crea el contenido del core
  public final void implement() throws CoreException
  {
    int i, row, col;
    Pin nSourcePin;
    Pin[] nSinksPin;

    // Calcular el offset
    Offset offset = calcAbsoluteOffset();
    row = offset.getVerOffset(Gran.CLB);
    col = offset.getHorOffset(Gran.CLB);

    if(sourcePin != null)
    {
      // Calcular la nueva posicion del pin fuente
      nSourcePin = sourcePin;
      nSourcePin.setCol(sourcePin.getCol() + col);
      nSourcePin.setRow(sourcePin.getRow() + row);

      // Asignar el pin al puerto correspondiente
      doutPort.setPin(nSourcePin);
    }

    if(sinksPin.length > 0)
    {
      // Calcular las nuevas posiciones de los pines destino
      nSinksPin = new Pin[sinksPin.length];

      for(i = 0; i < sinksPin.length; i++)
      {
        nSinksPin[i] = sinksPin[i];
        nSinksPin[i].setCol(sinksPin[i].getCol() + col);
        nSinksPin[i].setRow(sinksPin[i].getRow() + row);
      }

      // Asignar los pines a los puertos
      dinPort.setPin(nSinksPin);
    }
  }
}
```

Programa TestmtBits para la Virtex 1000

```
package testmtbits;

// Swing
import javax.swing.Timer;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import javax.swing.JToggleButton;
import javax.swing.BorderFactory;
import javax.swing.border.LineBorder;

// Awt
import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.FlowLayout;
import java.awt.event.WindowEvent;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
import java.awt.event.AdjustmentEvent;
import java.awt.event.AdjustmentListener;

import com.borland.jbcl.layout.VerticalFlowLayout;

// mtbits
import mtbits.mtBits;
import mtbits.mtbException;

public class frmMain extends JFrame
{
  // Interfaz con la fpga
  private mtBits mtb;

  // Constante necesaria para cerrar la ventana a terminar
  private final int EXIT_ON_CLOSE = 3;

  // Componentes
  private Timer timer;
  private int ciclo;

  private JPanel jContentPane = new JPanel();
  private JPanel jPTasks = new JPanel();
  private JScrollPane jSPLog = new JScrollPane();
  private JToggleButton jTBTaskA = new JToggleButton();
  private JToggleButton jTBTaskB = new JToggleButton();
  private JToggleButton jTBTaskC = new JToggleButton();
  private JToggleButton jTBTaskD = new JToggleButton();
  private GridLayout gridLayout1 = new GridLayout();
  private VerticalFlowLayout verticalFlowLayout1 = new VerticalFlowLayout();
  private JTextArea jTALog = new JTextArea();
  private JPanel jPEstado = new JPanel();
  private JLabel jLLed[] = new JLabel[4];
  private JPanel jPLeds = new JPanel();
  private FlowLayout flowLayout1 = new FlowLayout();
  private JToggleButton jTBRun = new JToggleButton();
  private GridLayout gridLayout2 = new GridLayout();

  private final String titulo = "Prueba de mtIOB en tiempo real";

  public frmMain()
  {
    try
    {
      jbInit();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
  }

  public static void main(String[] args)
  {
    frmMain frmMain = new frmMain();

    // Deshabilitar los eventos
    frmMain.enableInputMethods(false);

    // Mostrar el dialogo
    frmMain.show();

    frmMain.mtBitsInit();

    // Habilitar los eventos
    frmMain.enableInputMethods(true);
  }

  private void mtBitsInit()
  {
    try
    {
      mtb = new mtBits(System.out);
    }
    catch(Exception e)
```

```
    {
      error(e);
    }
  }

  private void jbInit() throws Exception
  {
    int i;

    // Configurar la ventana
    this.setTitle(titulo);
    this.setDefaultCloseOperation(EXIT_ON_CLOSE);
    this.setSize(new Dimension(600, 500));
    this.setResizable(false);
    this.getContentPane().add(jContentPane, null);
    this.setContentPane(jContentPane);

    // Contenedor principal
    jContentPane.setLayout(verticalFlowLayout1);

    // Tareas
    jPTasks.setLayout(gridLayout1);

    jTBTaskA.setText("A");
    jTBTaskB.setText("B");
    jTBTaskC.setText("C");
    jTBTaskD.setText("D");

    jTBTaskA.setFont(new java.awt.Font("Dialog", 0, 24));
    jTBTaskB.setFont(new java.awt.Font("Dialog", 0, 24));
    jTBTaskC.setFont(new java.awt.Font("Dialog", 0, 24));
    jTBTaskD.setFont(new java.awt.Font("Dialog", 0, 24));

    // Leds
    jPLeds.setLayout(flowLayout1);
    jPLeds.setBorder(BorderFactory.createRaisedBevelBorder());

    for(i = 0; i < 4; i++)
    {
      jLLed[i] = new JLabel();
      jLLed[i].setText("  ");
      jLLed[i].setOpaque(true);
      jLLed[i].setBorder(BorderFactory.createLineBorder(Color.black));
    }

    // Estado de la ejecucion
    jTBRun.setText("Ejecutar");
    jPEstado.setLayout(gridLayout2);

    // Log
    jSPLog.setAutoscrolls(true);
    jTALog.setEditable(false);
    jTALog.setText("");
    jTALog.setRows(22);

    // Añadir cada cosa a su sitio
    jContentPane.add(jPTasks, null);
    jContentPane.add(jPEstado, null);
    jContentPane.add(jSPLog, null);
    jPEstado.add(jTBRun, null);
    jPEstado.add(jPLeds, null);
    jPTasks.add(jTBTaskA, null);
    jPTasks.add(jTBTaskB, null);
    jPTasks.add(jTBTaskC, null);
    jPTasks.add(jTBTaskD, null);
    jSPLog.getViewport().add(jTALog, null);

    for(i = 0; i < 4; i++)
      jPLeds.add(jLLed[i], null);

    // Eventos
    this.addWindowListener(new java.awt.event.WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        this_windowClosing(e);
      }
```

```
    });

    jTBTaskA.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jTBTaskA_actionPerformed(e);
      }
    });

    jTBTaskB.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        jTBTaskB_actionPerformed(e);
      }
    });

    jTBTaskC.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        jTBTaskC_actionPerformed(e);
      }
    });

    jTBTaskD.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        jTBTaskD_actionPerformed(e);
      }
    });

    jTBRun.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        jTBRun_actionPerformed(e);
      }
    });

    // Autoscroll
    jSPLog.getVerticalScrollBar().addAdjustmentListener(new AdjustmentListener()
    {
      public void adjustmentValueChanged(AdjustmentEvent e)
      {
        jTALog.select(jTALog.getDocument().getLength(),jTALog.getDocument().getLength());
      }
    });

    // Contador
    timer = new Timer(3000,new ActionListener(){
      public void actionPerformed(ActionEvent evt)
      {
        try
        {
          // Pasar un ciclo
          ciclo++;
          System.out.print(ciclo + " ");
          mtb.step();
          leerLeds();
        }
        catch(Exception ex)
        {
          error(ex);
          timer.stop();
        }
      }
    });
  }

  private void leerLeds()
  {
    int[] valor;
    int i;
    Color estado;
```

```
}

private void error(Exception e)
{
  // Volver al titulo original
  setTitle(titulo);

  // Mostrar un mensaje de error
  jTALog.append("error\n" + e.getClass() + "\n" + e.toString() + "\n\n");
}

void this_windowClosing(WindowEvent e)
{
  try
  {
    //Parar el timer si se esta ejecutando
    if(timer.isRunning())
      timer.stop();

    // Desconectar la placa
    mtb.close();
  }
  catch(Exception ex)
  {
    error(ex);
  }
}

void jTBTaskA_actionPerformed(ActionEvent e)
{
  try
  {
    // Comprobar que el boton ha sido pulsado
    if(jTBTaskA.isSelected())
    {
      mtb.load("counter.bit", 0);
    }
    else
    {
      mtb.clear(0);
    }
  }
  catch(Exception ex)
  {
    // Se ha producido un error
    error(ex);
  }
}

void jTBTaskB_actionPerformed(ActionEvent e)
{
  try
  {
    // Comprobar que el boton ha sido pulsado
    if(jTBTaskB.isSelected())
    {
      mtb.load("counter.bit", 1);
    }
    else
    {
      mtb.clear(1);
    }
  }
  catch(Exception ex)
  {
    // Se ha producido un error
    error(ex);
  }
}

void jTBTaskC_actionPerformed(ActionEvent e)
{
  try
  {
    // Comprobar que el boton ha sido pulsado
```

```
      if(jTBTaskC.isSelected())
      {
        mtb.load("counter.bit", 2);
      }
      else
      {
        mtb.clear(2);
      }
    }
    catch(Exception ex)
    {
      // Se ha producido un error
      error(ex);
    }
  }

  void jTBTaskD_actionPerformed(ActionEvent e)
  {
    try
    {
      // Comprobar que el boton ha sido pulsado
      if(jTBTaskD.isSelected())
      {
        mtb.load("counter.bit", 3);
      }
      else
      {
        mtb.clear(3);
      }
    }
    catch(Exception ex)
    {
      // Se ha producido un error
      error(ex);
    }
  }

  void jTBRun_actionPerformed(ActionEvent e)
  {
    // Comprobar si se ha marcado el boton
    if(jTBRun.isSelected())
      timer.start();
    else
      timer.stop();
  }
}
```

Apéndice D: Sintaxis JBits

public class **JBits** extends **Virtex**

La primera clase que comentaremos es la propia JBits. Hay que reseñar que algunos de los métodos que explicamos aquí no son propios de JBits, sino heredados de clases padre. A continuación incluimos un listado en orden alfabético de los métodos que hemos usado de esta clase para la consecución de nuestro proyecto.

`JBits(int _deviceType)`

El constructor inicializa el objeto JBits pasándole como parámetro el nombre del dispositivo; en nuestro caso concreto: Devices.XCV1000

`Int[] get(int clbRow, int clbColumn, int[][] bits)`

Esta función devuelve la configuración asociada a un recurso concreto de una clb determinada por su posición en la FPGA. Dicha configuración se representa por medio de un array de enteros.

`Byte[] getPartial()`

Esta función devuelve una secuencia de paquetes que pueden ser usados para configurar un dispositivo, devolviendo los paquetes marcados como "sucios" si se trata de una reconfiguración parcial, o todos en caso contrario. Después de completarse la llamada a este método, la configuración de la memoria estará sincronizada con el dispositivo.

`void parsePartial(byte[] command, byte[] data)`

Este método es usado para parsear un bitstream, proporcionando el comando usado para obtener el bitstream desde el dispositivo, así como los datos devueltos como parámetros. Es necesario usar la utilidad ReadbackCommand que comentaremos más adelante para generar el comando requerido.

`void read(java.lang.String bitFileName)`

Este método lee un bitstream de una Virtex y lo carga en el objeto JBits

`void readPartial(java.lang.String bitFileName)`

Este método lee y parsea un bitstream. Usado en reconfiguración parcial, hace uso de "bloques sucios" para comparar las diferencias entre el archivo leído y la configuración en memoria.

`void set(int clbRow, int clbColumn, int[][] bits, int val)`

Este método modifica un recurso de una clb concreta indicada por sus coordenadas y lo sustituye por el valor val. Cada recurso se representa mediante un array bidimensional de enteros, aunque puede ser sustituido por el nombre de una constante ya definida, como por ejemplo *S0F1.S0F1*.

`void write(java.lang.String bitFileName)`

Este método escribe el contenido del objeto JBits a un archivo bitstream .bit

`void writePartial(java.lang.String bitFileName)`

Este método escribe el contenido del objeto JBits a un archive bitstream. Para ello, y al igual que readPartial, compara la configuración en memoria con la del dispositivo, escribiendo al .bit sólo lo modificado.

public abstract class **XHWIF** extends java.lang.Object

Esta clase define un interfaz genérico al hardware de una Virtex reconfigurable. Esta pensada para facilitar el portar herramientas y aplicaciones de una plataforma hardware a otra. Nosotros la usaremos en nuestro proyecto para conectar con la FPGA Virtex 1000. Los métodos más comunes y usados en nuestro proyecto son los siguientes:

`abstract int clockStep(int count)`
Esta función ejecuta count ciclos de reloj en la FPGA física.

`abstract int connect(java.lang.String serverName, int port)`
Esta función establece la comunicación con el hardware. Se pueden omitir los parámetros del nombre del servidor y el puerto en caso de que estemos accediendo a un dispositivo local.

`abstract byte[] getConfiguration(int device, int byteCount)`
Esta función devuelve los datos de configuración leídos de la Virtex. La función normalmente devuelve datos desde el hardware que previamente han sido mandados a través de una llamada a *setConfiguration()*.

`abstract int reset()`
Esta función resetea la FPGA.

`abastract int setConfiguration(int device, byte[] data)`
Esta función escribe datos de configuración indicados por *data* en la Virtex. Es sobre todo útil para la reconfiguración parcial, obteniendo el array de bytes mediante la llamada a *JBit.getPartial()*.

public class **XHWIFConnection** extends java.lang.Object implements java.io.Serializable, ConnectionListener

Esta clase representa una conexión a una placa compatible o al simulador VirtexDS (es para esto último para lo que la usaremos en el proyecto). Asimismo, la usaremos como parámetro pasado al crear una instancia de la aplicación BoardScope.

`void connectToBoard(java.lang.String boardString)`
Este método trata de crear una placa XHWIF con el nombre pasado como parámetro, conecta a ella y crea los correspondientes objetos JBits.

`void resetBoard()`
Este método resetea los dispositivos de la placa (la FPGA).

`XHWIFWithEvents getXHWIFWithEvents()`
Esta función proporciona acceso al objeto *XHWIFWithEvents* accesible desde la clase *XHWIFConnection*. Es importante reseñar que la clase *XHWIFWithEvents* proporciona los mismos métodos que la clase *XHWIF*, pero además permite usar la herramienta BoardScope a través de su clase "padre" *XHWIFConnection*.

`void disconnectFromBoard()`
Este método cierra la conexión con la placa (y por tanto se desconecta de la FPGA).

public class **XHWIFWithEvents** extends java.lang.Object

Los métodos y funciones usadas en el desarrollo del proyecto son los mismos que para la clase *XHWIF*, por lo que no los listamos de nuevo.

public class **ReadbackCommand** extends java.lang.Object

Esta clase creará bitstreams usados para recuperar datos desde la FPGA.

`static byte[] getClbConfig(int deviceType)`

Esta función genera un comando u orden que servirá luego para obtener la configuración completa de una CLB. En nuestro caso, el nombre del dispositivo será: *Devices.XCV1000*.

`static int getReadLength()`

Esta función devuelve la longitud expresada en palabras del último paquete generado. Combinado con la orden obtenida mediante *getClbConfig*, permite llamar con los parámetros adecuados a los métodos *setConfiguration(numDispositivo, orden)* y a *getConfiguration(numDispositivo, longitud)* de la clase *XHWIF*.

public class **JRoute** extends java.lang.Object

Esta clase proporciona métodos que además soportan la especificación de los RTP Cores. La usaremos para conectar varios pines de la FPGA entre sí, marcándolos como usados para posteriores reconfiguraciones.

`JRoute(JBits jbits)`

El constructor crea la utilidad de rutado a partir de un objeto JBits.

`void route(Pin src, Pin sink)`

Este método ruta un pin origen a un pin destino de la manera más directa posible.

`void route(Pin src, Pin[] sink)`

Este método ruta de un pin origen a una lista de pines destino, usando un algoritmo basado en A* que minimiza el rutado para el grupo entero de destinos.

`void unroute(Pin src)`

Este método elimina una ruta que comience en el pin origen src.

Apéndice E: Constantes asociadas a elementos funcionales de un CLB de una Virtex 1000

```
int[][][] recs = {
    S0F1.S0F1, S0F2.S0F2, S0F3.S0F3, S0F4.S0F4, S0G1.S0G1, S0G2.S0G2, S0G3.S0G3, S0G4.S0G4, // Entradas de las LUTS
    S1F1.S1F1, S1F2.S1F2, S1F3.S1F3, S1F4.S1F4, S1G1.S1G1, S1G2.S1G2, S1G3.S1G3, S1G4.S1G4,
    LUT.SLICE0_F, LUT.SLICE0_G, LUT.SLICE1_F, LUT.SLICE1_G, // El contenido de las LUTS
    CLB.SLICE0_XQ, CLB.SLICE0_YQ, CLB.SLICE1_XQ, CLB.SLICE1_YQ,
    S0Control.BxInvert, S0Control.ByInvert, S0Control.CeInvert, S0Control.ClockInvert, // Control del Slice 0
    S0Control.InvertedSetReset, S0Control.LatchMode, S0Control.SrWeNotInvert, S0Control.Sync,
    S0Control.XffSetResetSelect, S0Control.YffSetResetSelect, S0Control.AndMux.AndMux, S0Control.Cin.Cin,
    S0Control.X.X, S0Control.XCarrySelect.XCarrySelect, S0Control.XDin.XDin,
    S0Control.Y.Y, S0Control.YB.YB, S0Control.YCarrySelect.YCarrySelect, S0Control.YDin.YDin,
    S1Control.BxInvert, S1Control.ByInvert, S1Control.CeInvert, S1Control.ClockInvert, // Control del Slice 1
```

S1Control.InvertedSetReset, S1Control.LatchMode, S1Control.SrWeNotInvert, S1Control.Sync,
S1Control.XffSetResetSelect, S1Control.YffSetResetSelect, S1Control.AndMux.AndMux, S1Control.Cin.Cin,
S1Control.X.X, S1Control.XCarrySelect.XCarrySelect, S1Control.XDin.XDin,
S1Control.Y.Y, S1Control.YB.YB, S1Control.YCarrySelect.YCarrySelect, S1Control.YDin.YDin,
S0BX.S0BX, S0BY.S0BY, S0CE.S0CE, S0Clk.S0Clk, // Entradas para reloj, de los biestables y del reset
S1BX.S1BX, S1BY.S1BY, S1CE.S1CE, S1Clk.S1Clk,
S0RAM.DUAL_MODE, S0RAM.F_LUT_RAM, S0RAM.F_LUT_SHIFTER, S0RAM.G_LUT_RAM, // Memoria
S0RAM.G_LUT_SHIFTER, S0RAM.LUT_MODE, S0RAM.RAM_32_X_1, S0RAM.UNUSED,
S1RAM.DUAL_MODE, S1RAM.F_LUT_RAM, S1RAM.F_LUT_SHIFTER, S1RAM.G_LUT_RAM,
S1RAM.G_LUT_SHIFTER, S1RAM.LUT_MODE, S1RAM.RAM_32_X_1, S1RAM.UNUSED,
OUT0.OUT0, OUT1.OUT1, OUT2.OUT2, OUT3.OUT3, OUT4.OUT4, OUT5.OUT5, OUT6.OUT6, OUT7.OUT7, // Salida
SingleToSingle.SINGLE_WEST0_TO_SINGLE_EAST0, SingleToSingle.SINGLE_EAST0_TO_SINGLE_WEST0, // Switch
SingleToSingle.SINGLE_SOUTH1_TO_SINGLE_NORTH1, SingleToSingle.SINGLE_NORTH1_TO_SINGLE_SOUTH1,
SingleToSingle.SINGLE_SOUTH2_TO_SINGLE_NORTH2, SingleToSingle.SINGLE_NORTH2_TO_SINGLE_SOUTH2,
SingleToSingle.SINGLE_SOUTH5_TO_SINGLE_NORTH5, SingleToSingle.SINGLE_NORTH5_TO_SINGLE_SOUTH5,
SingleToSingle.SINGLE_SOUTH0_TO_SINGLE_NORTH0, SingleToSingle.SINGLE_NORTH0_TO_SINGLE_SOUTH0,
SingleToSingle.SINGLE_SOUTH6_TO_SINGLE_NORTH6, SingleToSingle.SINGLE_NORTH6_TO_SINGLE_SOUTH6,
SingleToSingle.SINGLE_SOUTH18_TO_SINGLE_NORTH18, SingleToSingle.SINGLE_NORTH18_TO_SINGLE_SOUTH18,
SingleToSingle.SINGLE_SOUTH19_TO_SINGLE_NORTH19, SingleToSingle.SINGLE_NORTH19_TO_SINGLE_SOUTH19,
SingleToSingle.SINGLE_SOUTH20_TO_SINGLE_NORTH20, SingleToSingle.SINGLE_NORTH20_TO_SINGLE_SOUTH20,
SingleToSingle.SINGLE_SOUTH23_TO_SINGLE_NORTH23, SingleToSingle.SINGLE_NORTH23_TO_SINGLE_SOUTH23,
SingleToSingle.SINGLE_WEST16_TO_SINGLE_EAST16, SingleToSingle.SINGLE_EAST16_TO_SINGLE_WEST16,
SingleToSingle.SINGLE_WEST19_TO_SINGLE_EAST19, SingleToSingle.SINGLE_EAST19_TO_SINGLE_WEST19,
SingleToSingle.SINGLE_WEST22_TO_SINGLE_EAST22, SingleToSingle.SINGLE_EAST22_TO_SINGLE_WEST22,
SingleToSingle.SINGLE_WEST21_TO_SINGLE_EAST21, SingleToSingle.SINGLE_EAST21_TO_SINGLE_WEST21,
SingleToSingle.SINGLE_WEST8_TO_SINGLE_EAST8, SingleToSingle.SINGLE_EAST8_TO_SINGLE_WEST8,
SingleToSingle.SINGLE_WEST9_TO_SINGLE_EAST9, SingleToSingle.SINGLE_EAST9_TO_SINGLE_WEST9,
SingleToSingle.SINGLE_WEST11_TO_SINGLE_EAST11, SingleToSingle.SINGLE_EAST11_TO_SINGLE_WEST11,
SingleToSingle.SINGLE_WEST10_TO_SINGLE_EAST10, SingleToSingle.SINGLE_EAST10_TO_SINGLE_WEST10,
SingleToSingle.SINGLE_WEST14_TO_SINGLE_EAST14, SingleToSingle.SINGLE_EAST14_TO_SINGLE_WEST14,
SingleToSingle.SINGLE_WEST15_TO_SINGLE_EAST15, SingleToSingle.SINGLE_EAST15_TO_SINGLE_WEST15,
SingleToSingle.SINGLE_WEST13_TO_SINGLE_EAST13, SingleToSingle.SINGLE_EAST13_TO_SINGLE_WEST13,
SingleToSingle.SINGLE_WEST12_TO_SINGLE_EAST12, SingleToSingle.SINGLE_EAST12_TO_SINGLE_WEST12,
SingleToSingle.SINGLE_WEST17_TO_SINGLE_EAST17, SingleToSingle.SINGLE_EAST17_TO_SINGLE_WEST17,
SingleToSingle.SINGLE_WEST18_TO_SINGLE_EAST18, SingleToSingle.SINGLE_EAST18_TO_SINGLE_WEST18,
SingleToSingle.SINGLE_WEST23_TO_SINGLE_EAST23, SingleToSingle.SINGLE_EAST23_TO_SINGLE_WEST23,
SingleToSingle.SINGLE_WEST20_TO_SINGLE_EAST20, SingleToSingle.SINGLE_EAST20_TO_SINGLE_WEST20,
SingleToSingle.SINGLE_WEST6_TO_SINGLE_EAST6, SingleToSingle.SINGLE_EAST6_TO_SINGLE_WEST6,
SingleToSingle.SINGLE_WEST7_TO_SINGLE_EAST7, SingleToSingle.SINGLE_EAST7_TO_SINGLE_WEST7,
SingleToSingle.SINGLE_WEST5_TO_SINGLE_EAST5, SingleToSingle.SINGLE_EAST5_TO_SINGLE_WEST5,
SingleToSingle.SINGLE_WEST4_TO_SINGLE_EAST4, SingleToSingle.SINGLE_EAST4_TO_SINGLE_WEST4,
SingleToSingle.SINGLE_WEST2_TO_SINGLE_EAST2, SingleToSingle.SINGLE_EAST2_TO_SINGLE_WEST2,
SingleToSingle.SINGLE_WEST3_TO_SINGLE_EAST3, SingleToSingle.SINGLE_EAST3_TO_SINGLE_WEST3,
SingleToSingle.SINGLE_WEST1_TO_SINGLE_EAST1, SingleToSingle.SINGLE_EAST1_TO_SINGLE_WEST1,
SingleToSingle.SINGLE_SOUTH4_TO_SINGLE_NORTH4, SingleToSingle.SINGLE_NORTH4_TO_SINGLE_SOUTH4,
SingleToSingle.SINGLE_SOUTH3_TO_SINGLE_NORTH3, SingleToSingle.SINGLE_NORTH3_TO_SINGLE_SOUTH3,
SingleToSingle.SINGLE_SOUTH22_TO_SINGLE_NORTH22, SingleToSingle.SINGLE_NORTH22_TO_SINGLE_SOUTH22,
SingleToSingle.SINGLE_SOUTH21_TO_SINGLE_NORTH21, SingleToSingle.SINGLE_NORTH21_TO_SINGLE_SOUTH21,
SingleToSingle.SINGLE_SOUTH9_TO_SINGLE_NORTH9, SingleToSingle.SINGLE_NORTH9_TO_SINGLE_SOUTH9,
SingleToSingle.SINGLE_SOUTH10_TO_SINGLE_NORTH10, SingleToSingle.SINGLE_NORTH10_TO_SINGLE_SOUTH10,
SingleToSingle.SINGLE_SOUTH11_TO_SINGLE_NORTH11, SingleToSingle.SINGLE_NORTH11_TO_SINGLE_SOUTH11,
SingleToSingle.SINGLE_SOUTH15_TO_SINGLE_NORTH15, SingleToSingle.SINGLE_NORTH15_TO_SINGLE_SOUTH15,
SingleToSingle.SINGLE_SOUTH16_TO_SINGLE_NORTH16, SingleToSingle.SINGLE_NORTH16_TO_SINGLE_SOUTH16,
SingleToSingle.SINGLE_SOUTH17_TO_SINGLE_NORTH17, SingleToSingle.SINGLE_NORTH17_TO_SINGLE_SOUTH17,
SingleToSingle.SINGLE_SOUTH14_TO_SINGLE_NORTH14, SingleToSingle.SINGLE_NORTH14_TO_SINGLE_SOUTH14,
SingleToSingle.SINGLE_SOUTH13_TO_SINGLE_NORTH13, SingleToSingle.SINGLE_NORTH13_TO_SINGLE_SOUTH13,
SingleToSingle.SINGLE_SOUTH12_TO_SINGLE_NORTH12, SingleToSingle.SINGLE_NORTH12_TO_SINGLE_SOUTH12,
SingleToSingle.SINGLE_SOUTH8_TO_SINGLE_NORTH8, SingleToSingle.SINGLE_NORTH8_TO_SINGLE_SOUTH8,
SingleToSingle.SINGLE_SOUTH7_TO_SINGLE_NORTH7, SingleToSingle.SINGLE_NORTH7_TO_SINGLE_SOUTH7,
SingleToSingle.SINGLE_WEST0_TO_SINGLE_SOUTH2, SingleToSingle.SINGLE_SOUTH2_TO_SINGLE_WEST0,
SingleToSingle.SINGLE_EAST20_TO_SINGLE_SOUTH2, SingleToSingle.SINGLE_SOUTH2_TO_SINGLE_EAST20,
SingleToSingle.SINGLE_EAST20_TO_SINGLE_NORTH0, SingleToSingle.SINGLE_NORTH0_TO_SINGLE_EAST20,
SingleToSingle.SINGLE_WEST5_TO_SINGLE_NORTH0, SingleToSingle.SINGLE_NORTH0_TO_SINGLE_WEST5,
SingleToSingle.SINGLE_WEST5_TO_SINGLE_SOUTH3, SingleToSingle.SINGLE_SOUTH3_TO_SINGLE_WEST5,
SingleToSingle.SINGLE_EAST1_TO_SINGLE_SOUTH3, SingleToSingle.SINGLE_SOUTH3_TO_SINGLE_EAST1,
SingleToSingle.SINGLE_EAST1_TO_SINGLE_NORTH1, SingleToSingle.SINGLE_NORTH1_TO_SINGLE_EAST1,
SingleToSingle.SINGLE_WEST2_TO_SINGLE_NORTH1, SingleToSingle.SINGLE_NORTH1_TO_SINGLE_WEST2,
SingleToSingle.SINGLE_WEST2_TO_SINGLE_SOUTH0, SingleToSingle.SINGLE_SOUTH0_TO_SINGLE_WEST2,
SingleToSingle.SINGLE_EAST22_TO_SINGLE_SOUTH0, SingleToSingle.SINGLE_SOUTH0_TO_SINGLE_EAST22,
SingleToSingle.SINGLE_EAST22_TO_SINGLE_NORTH2, SingleToSingle.SINGLE_NORTH2_TO_SINGLE_EAST22,
SingleToSingle.SINGLE_WEST7_TO_SINGLE_NORTH2, SingleToSingle.SINGLE_NORTH2_TO_SINGLE_WEST7,
SingleToSingle.SINGLE_WEST7_TO_SINGLE_SOUTH1, SingleToSingle.SINGLE_SOUTH1_TO_SINGLE_WEST7,
SingleToSingle.SINGLE_EAST3_TO_SINGLE_SOUTH1, SingleToSingle.SINGLE_SOUTH1_TO_SINGLE_EAST3,
SingleToSingle.SINGLE_EAST3_TO_SINGLE_NORTH3, SingleToSingle.SINGLE_NORTH3_TO_SINGLE_EAST3,
SingleToSingle.SINGLE_WEST4_TO_SINGLE_NORTH3, SingleToSingle.SINGLE_NORTH3_TO_SINGLE_WEST4,
SingleToSingle.SINGLE_WEST4_TO_SINGLE_SOUTH6, SingleToSingle.SINGLE_SOUTH6_TO_SINGLE_WEST4,
SingleToSingle.SINGLE_EAST0_TO_SINGLE_SOUTH6, SingleToSingle.SINGLE_SOUTH6_TO_SINGLE_EAST0,

SingleToSingle.SINGLE_EAST0_TO_SINGLE_NORTH4, SingleToSingle.SINGLE_NORTH4_TO_SINGLE_EAST0,
SingleToSingle.SINGLE_EAST19_TO_SINGLE_SOUTH17, SingleToSingle.SINGLE_SOUTH17_TO_SINGLE_EAST19,
SingleToSingle.SINGLE_WEST18_TO_SINGLE_SOUTH16, SingleToSingle.SINGLE_SOUTH16_TO_SINGLE_WEST18,
SingleToSingle.SINGLE_WEST18_TO_SINGLE_NORTH17, SingleToSingle.SINGLE_NORTH17_TO_SINGLE_WEST18,
SingleToSingle.SINGLE_WEST23_TO_SINGLE_NORTH18, SingleToSingle.SINGLE_NORTH18_TO_SINGLE_WEST23,
SingleToSingle.SINGLE_EAST17_TO_SINGLE_SOUTH19, SingleToSingle.SINGLE_SOUTH19_TO_SINGLE_EAST17,
SingleToSingle.SINGLE_WEST23_TO_SINGLE_SOUTH17, SingleToSingle.SINGLE_SOUTH17_TO_SINGLE_WEST23,
SingleToSingle.SINGLE_EAST17_TO_SINGLE_NORTH17, SingleToSingle.SINGLE_NORTH17_TO_SINGLE_EAST17,
SingleToSingle.SINGLE_EAST14_TO_SINGLE_NORTH18, SingleToSingle.SINGLE_NORTH18_TO_SINGLE_EAST14,
SingleToSingle.SINGLE_EAST14_TO_SINGLE_SOUTH16, SingleToSingle.SINGLE_SOUTH16_TO_SINGLE_EAST14,
SingleToSingle.SINGLE_WEST0_TO_SINGLE_NORTH23, SingleToSingle.SINGLE_NORTH23_TO_SINGLE_WEST0,
SingleToSingle.SINGLE_WEST21_TO_SINGLE_NORTH16, SingleToSingle.SINGLE_NORTH16_TO_SINGLE_WEST21,
SingleToSingle.SINGLE_WEST21_TO_SINGLE_SOUTH19, SingleToSingle.SINGLE_SOUTH19_TO_SINGLE_WEST21,
SingleToSingle.SINGLE_EAST12_TO_SINGLE_NORTH16, SingleToSingle.SINGLE_NORTH16_TO_SINGLE_EAST12,
SingleToSingle.SINGLE_EAST12_TO_SINGLE_SOUTH18, SingleToSingle.SINGLE_SOUTH18_TO_SINGLE_EAST12,
SingleToSingle.SINGLE_WEST16_TO_SINGLE_SOUTH18, SingleToSingle.SINGLE_SOUTH18_TO_SINGLE_WEST16,
SingleToSingle.SINGLE_WEST16_TO_SINGLE_NORTH15, SingleToSingle.SINGLE_NORTH15_TO_SINGLE_WEST16,
SingleToSingle.SINGLE_WEST19_TO_SINGLE_NORTH14, SingleToSingle.SINGLE_NORTH14_TO_SINGLE_WEST19,
SingleToSingle.SINGLE_EAST10_TO_SINGLE_NORTH14, SingleToSingle.SINGLE_NORTH14_TO_SINGLE_EAST10,
SingleToSingle.SINGLE_WEST19_TO_SINGLE_SOUTH13, SingleToSingle.SINGLE_SOUTH13_TO_SINGLE_WEST19,
SingleToSingle.SINGLE_EAST15_TO_SINGLE_NORTH15, SingleToSingle.SINGLE_NORTH15_TO_SINGLE_EAST15,
SingleToSingle.SINGLE_EAST15_TO_SINGLE_SOUTH13, SingleToSingle.SINGLE_SOUTH13_TO_SINGLE_EAST15,
SingleToSingle.SINGLE_EAST10_TO_SINGLE_SOUTH12, SingleToSingle.SINGLE_SOUTH12_TO_SINGLE_EAST10,
SingleToSingle.SINGLE_WEST14_TO_SINGLE_SOUTH12, SingleToSingle.SINGLE_SOUTH12_TO_SINGLE_WEST14,
SingleToSingle.SINGLE_EAST13_TO_SINGLE_NORTH13, SingleToSingle.SINGLE_NORTH13_TO_SINGLE_EAST13,
SingleToSingle.SINGLE_WEST14_TO_SINGLE_NORTH13, SingleToSingle.SINGLE_NORTH13_TO_SINGLE_WEST14,
SingleToSingle.SINGLE_EAST13_TO_SINGLE_SOUTH15, SingleToSingle.SINGLE_SOUTH15_TO_SINGLE_EAST13,
SingleToSingle.SINGLE_EAST23_TO_SINGLE_NORTH23, SingleToSingle.SINGLE_NORTH23_TO_SINGLE_EAST23,
SingleToSingle.SINGLE_EAST23_TO_SINGLE_SOUTH21, SingleToSingle.SINGLE_SOUTH21_TO_SINGLE_EAST23,
SingleToSingle.SINGLE_WEST3_TO_SINGLE_SOUTH21, SingleToSingle.SINGLE_SOUTH21_TO_SINGLE_WEST3,
SingleToSingle.SINGLE_WEST3_TO_SINGLE_NORTH22, SingleToSingle.SINGLE_NORTH22_TO_SINGLE_WEST3,
SingleToSingle.SINGLE_EAST18_TO_SINGLE_NORTH22, SingleToSingle.SINGLE_NORTH22_TO_SINGLE_EAST18,
SingleToSingle.SINGLE_EAST18_TO_SINGLE_SOUTH20, SingleToSingle.SINGLE_SOUTH20_TO_SINGLE_EAST18,
SingleToSingle.SINGLE_WEST22_TO_SINGLE_SOUTH20, SingleToSingle.SINGLE_SOUTH20_TO_SINGLE_WEST22,
SingleToSingle.SINGLE_EAST21_TO_SINGLE_SOUTH23, SingleToSingle.SINGLE_SOUTH23_TO_SINGLE_EAST21,
SingleToSingle.SINGLE_WEST1_TO_SINGLE_SOUTH23, SingleToSingle.SINGLE_SOUTH23_TO_SINGLE_WEST1,
SingleToSingle.SINGLE_WEST1_TO_SINGLE_NORTH20, SingleToSingle.SINGLE_NORTH20_TO_SINGLE_WEST1,
SingleToSingle.SINGLE_EAST16_TO_SINGLE_NORTH20, SingleToSingle.SINGLE_NORTH20_TO_SINGLE_EAST16,
SingleToSingle.SINGLE_WEST22_TO_SINGLE_NORTH21, SingleToSingle.SINGLE_NORTH21_TO_SINGLE_WEST22,
SingleToSingle.SINGLE_WEST20_TO_SINGLE_SOUTH22, SingleToSingle.SINGLE_SOUTH22_TO_SINGLE_WEST20,
SingleToSingle.SINGLE_EAST21_TO_SINGLE_NORTH21, SingleToSingle.SINGLE_NORTH21_TO_SINGLE_EAST21,
SingleToSingle.SINGLE_EAST16_TO_SINGLE_SOUTH22, SingleToSingle.SINGLE_SOUTH22_TO_SINGLE_EAST16,
SingleToSingle.SINGLE_WEST20_TO_SINGLE_NORTH19, SingleToSingle.SINGLE_NORTH19_TO_SINGLE_WEST20,
SingleToSingle.SINGLE_EAST19_TO_SINGLE_NORTH19, SingleToSingle.SINGLE_NORTH19_TO_SINGLE_EAST19,
SingleToSingle.SINGLE_WEST17_TO_SINGLE_SOUTH15, SingleToSingle.SINGLE_SOUTH15_TO_SINGLE_WEST17,
SingleToSingle.SINGLE_WEST17_TO_SINGLE_NORTH12, SingleToSingle.SINGLE_NORTH12_TO_SINGLE_WEST17,
SingleToSingle.SINGLE_EAST9_TO_SINGLE_SOUTH11, SingleToSingle.SINGLE_SOUTH11_TO_SINGLE_EAST9,
SingleToSingle.SINGLE_EAST9_TO_SINGLE_NORTH9, SingleToSingle.SINGLE_NORTH9_TO_SINGLE_EAST9,
SingleToSingle.SINGLE_WEST10_TO_SINGLE_NORTH9, SingleToSingle.SINGLE_NORTH9_TO_SINGLE_WEST10,
SingleToSingle.SINGLE_WEST10_TO_SINGLE_SOUTH8, SingleToSingle.SINGLE_SOUTH8_TO_SINGLE_WEST10,
SingleToSingle.SINGLE_EAST6_TO_SINGLE_SOUTH8, SingleToSingle.SINGLE_SOUTH8_TO_SINGLE_EAST6,
SingleToSingle.SINGLE_EAST6_TO_SINGLE_NORTH10, SingleToSingle.SINGLE_NORTH10_TO_SINGLE_EAST6,
SingleToSingle.SINGLE_WEST15_TO_SINGLE_NORTH10, SingleToSingle.SINGLE_NORTH10_TO_SINGLE_WEST15,
SingleToSingle.SINGLE_WEST15_TO_SINGLE_SOUTH9, SingleToSingle.SINGLE_SOUTH9_TO_SINGLE_WEST15,
SingleToSingle.SINGLE_EAST11_TO_SINGLE_SOUTH9, SingleToSingle.SINGLE_SOUTH9_TO_SINGLE_EAST11,
SingleToSingle.SINGLE_EAST11_TO_SINGLE_NORTH11, SingleToSingle.SINGLE_NORTH11_TO_SINGLE_EAST11,
SingleToSingle.SINGLE_WEST12_TO_SINGLE_NORTH11, SingleToSingle.SINGLE_NORTH11_TO_SINGLE_WEST12,
SingleToSingle.SINGLE_WEST12_TO_SINGLE_SOUTH14, SingleToSingle.SINGLE_SOUTH14_TO_SINGLE_WEST12,
SingleToSingle.SINGLE_EAST8_TO_SINGLE_SOUTH14, SingleToSingle.SINGLE_SOUTH14_TO_SINGLE_EAST8,
SingleToSingle.SINGLE_EAST8_TO_SINGLE_NORTH12, SingleToSingle.SINGLE_NORTH12_TO_SINGLE_EAST8,
SingleToSingle.SINGLE_EAST7_TO_SINGLE_SOUTH5, SingleToSingle.SINGLE_SOUTH5_TO_SINGLE_EAST7,
SingleToSingle.SINGLE_EAST7_TO_SINGLE_NORTH7, SingleToSingle.SINGLE_NORTH7_TO_SINGLE_EAST7,
SingleToSingle.SINGLE_WEST8_TO_SINGLE_NORTH7, SingleToSingle.SINGLE_NORTH7_TO_SINGLE_WEST8,
SingleToSingle.SINGLE_WEST8_TO_SINGLE_SOUTH10, SingleToSingle.SINGLE_SOUTH10_TO_SINGLE_WEST8,
SingleToSingle.SINGLE_EAST4_TO_SINGLE_SOUTH10, SingleToSingle.SINGLE_SOUTH10_TO_SINGLE_EAST4,
SingleToSingle.SINGLE_EAST4_TO_SINGLE_NORTH8, SingleToSingle.SINGLE_NORTH8_TO_SINGLE_EAST4,
SingleToSingle.SINGLE_WEST13_TO_SINGLE_NORTH8, SingleToSingle.SINGLE_NORTH8_TO_SINGLE_WEST13,
SingleToSingle.SINGLE_WEST13_TO_SINGLE_SOUTH11, SingleToSingle.SINGLE_SOUTH11_TO_SINGLE_WEST13,
SingleToSingle.SINGLE_EAST2_TO_SINGLE_SOUTH4, SingleToSingle.SINGLE_SOUTH4_TO_SINGLE_EAST2,
SingleToSingle.SINGLE_EAST2_TO_SINGLE_NORTH6, SingleToSingle.SINGLE_NORTH6_TO_SINGLE_EAST2,
SingleToSingle.SINGLE_WEST11_TO_SINGLE_NORTH6, SingleToSingle.SINGLE_NORTH6_TO_SINGLE_WEST11,
SingleToSingle.SINGLE_WEST11_TO_SINGLE_SOUTH5, SingleToSingle.SINGLE_SOUTH5_TO_SINGLE_WEST11,
SingleToSingle.SINGLE_EAST5_TO_SINGLE_SOUTH7, SingleToSingle.SINGLE_SOUTH7_TO_SINGLE_EAST5,
SingleToSingle.SINGLE_EAST5_TO_SINGLE_NORTH5, SingleToSingle.SINGLE_NORTH5_TO_SINGLE_EAST5,
SingleToSingle.SINGLE_WEST6_TO_SINGLE_NORTH5, SingleToSingle.SINGLE_NORTH5_TO_SINGLE_WEST6,
SingleToSingle.SINGLE_WEST6_TO_SINGLE_SOUTH4, SingleToSingle.SINGLE_SOUTH4_TO_SINGLE_WEST6,
SingleToSingle.SINGLE_WEST9_TO_SINGLE_NORTH4, SingleToSingle.SINGLE_NORTH4_TO_SINGLE_WEST9,

SingleToSingle.SINGLE_WEST9_TO_SINGLE_SOUTH7, SingleToSingle.SINGLE_SOUTH7_TO_SINGLE_WEST9,
BiHexToSingle.SINGLE_EAST1_TO_HEX_VERT_M0, BiHexToSingle.SINGLE_WEST3_TO_HEX_VERT_M0,
BiHexToSingle.SINGLE_EAST0_TO_HEX_SOUTH0, BiHexToSingle.SINGLE_WEST4_TO_HEX_NORTH0,
BiHexToSingle.SINGLE_WEST5_TO_HEX_EAST0, BiHexToSingle.SINGLE_WEST6_TO_HEX_EAST1,
BiHexToSingle.SINGLE_WEST8_TO_HEX_VERT_M1, BiHexToSingle.SINGLE_EAST7_TO_HEX_VERT_M1,
BiHexToSingle.SINGLE_WEST9_TO_HEX_SOUTH1, BiHexToSingle.SINGLE_EAST9_TO_HEX_NORTH1,
BiHexToSingle.SINGLE_EAST11_TO_HEX_WEST1, BiHexToSingle.SINGLE_WEST17_TO_HEX_EAST2,
BiHexToSingle.SINGLE_WEST18_TO_HEX_EAST3, BiHexToSingle.SINGLE_EAST23_TO_HEX_WEST3,
BiHexToSingle.SINGLE_EAST10_TO_HEX_WEST2, BiHexToSingle.SINGLE_WEST16_TO_HEX_NORTH2,
BiHexToSingle.SINGLE_EAST21_TO_HEX_NORTH3, BiHexToSingle.SINGLE_EAST12_TO_HEX_SOUTH2,
BiHexToSingle.SINGLE_WEST21_TO_HEX_SOUTH3, BiHexToSingle.HEX_SOUTH2_TO_SINGLE_NORTH13,
BiHexToSingle.HEX_SOUTH3_TO_SINGLE_NORTH19, BiHexToSingle.HEX_NORTH2_TO_SINGLE_SOUTH14,
BiHexToSingle.HEX_NORTH3_TO_SINGLE_SOUTH22, BiHexToSingle.HEX_WEST2_TO_SINGLE_NORTH12,
BiHexToSingle.HEX_WEST3_TO_SINGLE_SOUTH20, BiHexToSingle.HEX_EAST3_TO_SINGLE_NORTH18,
BiHexToSingle.HEX_EAST2_TO_SINGLE_SOUTH16, BiHexToSingle.HEX_HORIZ_M3_TO_SINGLE_NORTH22,
BiHexToSingle.HEX_HORIZ_M3_TO_SINGLE_SOUTH21, BiHexToSingle.HEX_HORIZ_M2_TO_SINGLE_SOUTH18,
BiHexToSingle.HEX_HORIZ_M2_TO_SINGLE_NORTH16, BiHexToSingle.HEX_WEST0_TO_SINGLE_NORTH0,
BiHexToSingle.SINGLE_WEST20_TO_HEX_VERT_M3, BiHexToSingle.HEX_NORTH0_TO_SINGLE_SOUTH2,
BiHexToSingle.SINGLE_WEST15_TO_HEX_VERT_M2, BiHexToSingle.HEX_HORIZ_M1_TO_SINGLE_NORTH10,
BiHexToSingle.HEX_HORIZ_M1_TO_SINGLE_SOUTH9, BiHexToSingle.HEX_WEST1_TO_SINGLE_SOUTH8,
BiHexToSingle.HEX_SOUTH0_TO_SINGLE_NORTH1, BiHexToSingle.HEX_EAST1_TO_SINGLE_NORTH6,
BiHexToSingle.HEX_HORIZ_M0_TO_SINGLE_SOUTH6, BiHexToSingle.SINGLE_EAST22_TO_HEX_WEST0,
BiHexToSingle.SINGLE_EAST13_TO_HEX_VERT_M2, BiHexToSingle.HEX_HORIZ_M0_TO_SINGLE_NORTH4,
BiHexToSingle.HEX_SOUTH1_TO_SINGLE_NORTH7, BiHexToSingle.HEX_NORTH1_TO_SINGLE_SOUTH10,
BiHexToSingle.SINGLE_EAST19_TO_HEX_VERT_M3, BiHexToSingle.HEX_EAST0_TO_SINGLE_SOUTH4,
OutMuxToSingle.OUT0_TO_SINGLE_WEST7, OutMuxToSingle.OUT0_TO_SINGLE_EAST2, // Switch pequeño para las salidas
OutMuxToSingle.OUT1_TO_SINGLE_WEST4, OutMuxToSingle.OUT1_TO_SINGLE_EAST3,
OutMuxToSingle.OUT1_TO_SINGLE_EAST5, OutMuxToSingle.OUT1_TO_SINGLE_WEST5,
OutMuxToSingle.OUT2_TO_SINGLE_WEST9, OutMuxToSingle.OUT2_TO_SINGLE_EAST6,
OutMuxToSingle.OUT3_TO_SINGLE_EAST8, OutMuxToSingle.OUT3_TO_SINGLE_WEST11,
OutMuxToSingle.OUT3_TO_SINGLE_WEST10, OutMuxToSingle.OUT3_TO_SINGLE_EAST11,
OutMuxToSingle.OUT5_TO_SINGLE_WEST16, OutMuxToSingle.OUT5_TO_SINGLE_EAST15,
OutMuxToSingle.OUT5_TO_SINGLE_EAST17, OutMuxToSingle.OUT5_TO_SINGLE_WEST17,
OutMuxToSingle.OUT6_TO_SINGLE_WEST21, OutMuxToSingle.OUT6_TO_SINGLE_EAST18,
OutMuxToSingle.OUT7_TO_SINGLE_EAST20, OutMuxToSingle.OUT7_TO_SINGLE_WEST23,
OutMuxToSingle.OUT7_TO_SINGLE_WEST22, OutMuxToSingle.OUT7_TO_SINGLE_EAST23,
OutMuxToSingle.OUT4_TO_SINGLE_WEST19, OutMuxToSingle.OUT4_TO_SINGLE_EAST14,
OutMuxToSingle.OUT4_TO_SINGLE_NORTH12, OutMuxToSingle.OUT4_TO_SINGLE_SOUTH13,
OutMuxToSingle.OUT6_TO_SINGLE_NORTH18, OutMuxToSingle.OUT6_TO_SINGLE_SOUTH19,
OutMuxToSingle.OUT4_TO_SINGLE_SOUTH15, OutMuxToSingle.OUT4_TO_SINGLE_NORTH13,
OutMuxToSingle.OUT5_TO_SINGLE_NORTH14, OutMuxToSingle.OUT6_TO_SINGLE_SOUTH17,
OutMuxToSingle.OUT5_TO_SINGLE_SOUTH12, OutMuxToSingle.OUT6_TO_SINGLE_NORTH20,
OutMuxToSingle.OUT7_TO_SINGLE_NORTH21, OutMuxToSingle.OUT7_TO_SINGLE_SOUTH22,
OutMuxToSingle.OUT0_TO_SINGLE_NORTH0, OutMuxToSingle.OUT0_TO_SINGLE_SOUTH1,
OutMuxToSingle.OUT2_TO_SINGLE_NORTH6, OutMuxToSingle.OUT2_TO_SINGLE_SOUTH7,
OutMuxToSingle.OUT0_TO_SINGLE_SOUTH3, OutMuxToSingle.OUT0_TO_SINGLE_NORTH1,
OutMuxToSingle.OUT1_TO_SINGLE_NORTH2, OutMuxToSingle.OUT2_TO_SINGLE_SOUTH5,
OutMuxToSingle.OUT1_TO_SINGLE_SOUTH0, OutMuxToSingle.OUT2_TO_SINGLE_NORTH8,
OutMuxToSingle.OUT3_TO_SINGLE_NORTH9, OutMuxToSingle.OUT3_TO_SINGLE_SOUTH10,
UniHexToSingle.SINGLE_WEST3_TO_HEX_EAST4, UniHexToSingle.SINGLE_EAST0_TO_HEX_VERT_M4,
UniHexToSingle.SINGLE_WEST2_TO_HEX_VERT_M4, UniHexToSingle.SINGLE_EAST3_TO_HEX_WEST5,
UniHexToSingle.SINGLE_EAST5_TO_HEX_SOUTH5, UniHexToSingle.SINGLE_EAST4_TO_HEX_VERT_M9,
UniHexToSingle.SINGLE_WEST6_TO_HEX_VERT_M9, UniHexToSingle.SINGLE_WEST8_TO_HEX_NORTH8,
UniHexToSingle.SINGLE_EAST6_TO_HEX_WEST9, UniHexToSingle.SINGLE_WEST11_TO_HEX_EAST8,
UniHexToSingle.SINGLE_EAST9_TO_HEX_VERT_M8, UniHexToSingle.SINGLE_WEST13_TO_HEX_VERT_M8,
UniHexToSingle.SINGLE_WEST10_TO_HEX_SOUTH9, UniHexToSingle.SINGLE_EAST12_TO_HEX_VERT_M6,
UniHexToSingle.SINGLE_WEST14_TO_HEX_VERT_M6, UniHexToSingle.SINGLE_EAST21_TO_HEX_VERT_M10,
UniHexToSingle.SINGLE_WEST1_TO_HEX_VERT_M10, UniHexToSingle.SINGLE_EAST16_TO_HEX_VERT_M11,
UniHexToSingle.SINGLE_WEST18_TO_HEX_VERT_M11, UniHexToSingle.SINGLE_EAST10_TO_HEX_VERT_M7,
UniHexToSingle.SINGLE_WEST12_TO_HEX_VERT_M7, UniHexToSingle.SINGLE_WEST15_TO_HEX_EAST6,
UniHexToSingle.SINGLE_WEST23_TO_HEX_EAST10, UniHexToSingle.SINGLE_EAST15_TO_HEX_WEST7,
UniHexToSingle.SINGLE_EAST18_TO_HEX_WEST11, UniHexToSingle.SINGLE_WEST20_TO_HEX_NORTH10,
UniHexToSingle.SINGLE_EAST13_TO_HEX_NORTH6, UniHexToSingle.SINGLE_EAST17_TO_HEX_SOUTH7,
UniHexToSingle.SINGLE_WEST22_TO_HEX_SOUTH11, UniHexToSingle.HEX_SOUTH11_TO_SINGLE_NORTH22,
UniHexToSingle.HEX_SOUTH7_TO_SINGLE_NORTH16, UniHexToSingle.HEX_NORTH10_TO_SINGLE_SOUTH17,
UniHexToSingle.HEX_NORTH6_TO_SINGLE_SOUTH13, UniHexToSingle.HEX_WEST11_TO_SINGLE_NORTH20,
UniHexToSingle.HEX_WEST7_TO_SINGLE_SOUTH18, UniHexToSingle.HEX_EAST10_TO_SINGLE_NORTH21,
UniHexToSingle.HEX_EAST6_TO_SINGLE_SOUTH15, UniHexToSingle.HEX_HORIZ_M6_TO_SINGLE_SOUTH14,
UniHexToSingle.HEX_HORIZ_M7_TO_SINGLE_NORTH17, UniHexToSingle.HEX_HORIZ_M11_TO_SINGLE_NORTH23,
UniHexToSingle.HEX_HORIZ_M10_TO_SINGLE_SOUTH23, UniHexToSingle.HEX_HORIZ_M7_TO_SINGLE_SOUTH16,
UniHexToSingle.HEX_HORIZ_M11_TO_SINGLE_SOUTH20, UniHexToSingle.HEX_HORIZ_M10_TO_SINGLE_NORTH19,
UniHexToSingle.HEX_HORIZ_M6_TO_SINGLE_NORTH15, UniHexToSingle.HEX_WEST9_TO_SINGLE_NORTH8,
UniHexToSingle.HEX_EAST8_TO_SINGLE_NORTH9, UniHexToSingle.HEX_WEST5_TO_SINGLE_SOUTH6,
UniHexToSingle.HEX_SOUTH9_TO_SINGLE_NORTH10, UniHexToSingle.HEX_NORTH8_TO_SINGLE_SOUTH5,
UniHexToSingle.HEX_EAST4_TO_SINGLE_SOUTH3, UniHexToSingle.HEX_HORIZ_M4_TO_SINGLE_SOUTH2,
UniHexToSingle.HEX_SOUTH5_TO_SINGLE_NORTH4, UniHexToSingle.HEX_HORIZ_M5_TO_SINGLE_NORTH5,
UniHexToSingle.SINGLE_EAST22_TO_HEX_VERT_M5, UniHexToSingle.HEX_HORIZ_M9_TO_SINGLE_NORTH11,

```
    UniHexToSingle.SINGLE_EAST1_TO_HEX_NORTH4, UniHexToSingle.SINGLE_WEST0_TO_HEX_VERT_M5,
    UniHexToSingle.HEX_HORIZ_M8_TO_SINGLE_SOUTH11, UniHexToSingle.HEX_HORIZ_M5_TO_SINGLE_SOUTH4,
    UniHexToSingle.HEX_HORIZ_M9_TO_SINGLE_SOUTH8, UniHexToSingle.HEX_HORIZ_M8_TO_SINGLE_NORTH7,
    UniHexToSingle.HEX_HORIZ_M4_TO_SINGLE_NORTH3, UniHexToSingle.HEX_NORTH4_TO_SINGLE_SOUTH1,
    HexEast0.HexEast0, HexEast1.HexEast1, HexEast2.HexEast2, HexEast3.HexEast3, // Switch longitud 6
    HexEast5.HexEast5, HexEast7.HexEast7, HexEast9.HexEast9, HexEast11.HexEast11,
    HexNorth0.HexNorth0, HexNorth1.HexNorth1, HexNorth2.HexNorth2, HexNorth3.HexNorth3,
    HexNorth5.HexNorth5, HexNorth7.HexNorth7, HexNorth9.HexNorth9, HexNorth11.HexNorth11,
    HexWest0.HexWest0, HexWest1.HexWest1, HexWest2.HexWest2, HexWest3.HexWest3,
    HexWest4.HexWest4, HexWest6.HexWest6, HexWest8.HexWest8, HexWest10.HexWest10,
    HexSouth0.HexSouth0, HexSouth1.HexSouth1, HexSouth2.HexSouth2, HexSouth3.HexSouth3,
    HexSouth4.HexSouth4, HexSouth6.HexSouth6, HexSouth8.HexSouth8, HexSouth10.HexSouth10
  };
```

Bibliografía

JBits Tutorial, incluido en JBits SDK 2.8

JBits API, incluido en JBits SDK 2.8

Celoxica. RC1000 Hardware Reference Manual 2.3

Celoxica. RC1000 Software Reference Manual 1.3

Celoxica. RC1000 Functional Reference Manual 1.3

Xilinx. Libraries Guide

Xilinx. Constraints Guide

Prasanna Sundararajan, Scott McMillan, Steven A. Guccione. *Testing FPGA Devices Using JBits*. San Jose. California

Steven A. Guccione, Delon Levi. JBits: *A Java-Based Interface to FPGA Hardware*

José Carlos Palma, Aline Vieira de Mello, Leandro Möller, Fernando Moraes, Ney Calazans. *Core Communication Interface for FPGAs*. Pontifícia Universidade Católica do Rio Grande do Sul. Porto Alegre. Brasil

José Luis Camps, Vicente Herrero, Rafael Gadea, Joaquín Cerdà, Marcos Martínez, Ricardo Colom. *Aplicación de la reconfigurabilidad dinámica de la FPGA Virtex de Xilinx*. Universidad Politécnica de Valencia.

Juan Gonzalez. *Convirtiendo el hardware en software: FPGA's*

Matthias Dyer, Christian Plessl, Marco Platzner. *Partially Reconfigurable Cores for Xilinx Virtex***.** Computer Engineering and Networks Lab, Swiss Federal Institute of Technology, ETH Zurich, Switzerland