UNIVERSIDAD COMPLUTENSE DE MADRID
Facultad de Informática

Departamento de Arquitectura de Computadores y Automática

Protección de circuitos para aplicaciones espaciales contra los efectos de rayos cósmicos

Proyecto Fin de Grado en Ingeniería de Computadores

Carlos Cabañas García
Jaime Rodríguez Carmona

Profesora Directora: Hortensia Mecha López
Profesor Director: Juan Antonio Clemente Barreira

Madrid, 2014

Autorización

Los autores de este proyecto autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos no comerciales tanto la memoria, como el código, la documentación y el prototipo de la plataforma desarrollada.

Carlos Cabañas García

Jaime Rodríguez Carmona

Madrid, 2014

Agradecimientos

Queremos agradecer a nuestros directores de proyecto Hortensia Mecha y Juan Antonio Clemente por la ayuda y dedicación ofrecida en la realización del mismo, puesto que siempre hemos podido contar con ellos a cualquier hora y día cuando lo hemos necesitado. Agradecer también el habernos facilitado un despacho para poder llevarlo a cabo. Queremos agradecer especialmente a Felipe Serrano por la gran ayuda ofrecida en los comienzos, que fue un gran apoyo para nosotros.

Además, queremos hacer un agradecimiento especial a nuestras familias, amigos y pareja por estar a siempre a nuestro lado, apoyarnos y motivarnos en todo momento.

Índice de Figuras

Resumen

Los dispositivos implementados con tecnologías de última generación son cada vez más vulnerables a alteraciones en su comportamiento debido a la radiación cósmica. El objetivo de este trabajo es controlar esa posible vulnerabilidad. Esto sucede fundamentalmente en circuitos embarcados en aviones, satélites o cohetes.

Nuestro trabajo consiste en adaptar una herramienta de inyección de errores que sólo funcionaba para tecnologías basadas en FPGA para que pueda ser usada sobre ASIC (*Application-Specific Integrated Circuit*). Esta herramienta llamada Nessy fue desarrollada por otros compañeros en esta facultad en proyectos de SSII durante los cursos académicos 2009/2010 y 2010/2011. Para ello se ha necesitado modificar la herramienta y los ficheros de tal forma que permita inyectar errores sobre FlipFlops y de esta forma lograr emular errores en ASIC. Una de las principales diferencias entre una FPGA y un ASIC es que la primera se puede reconfigurar e implementar nuevas funcionalidades, mientras que los ASIC, al no tener una memoria de configuración no permiten, una vez implementado, tener un uso distinto, es decir no permiten ser reconfigurado. Sólo permite resetear su circuito, con lo que los fallos no lo desconfiguran, sólo alteran su funcionamiento.

Por otro lado, se ha querido ampliar la herramienta para que también permita proteger automáticamente los circuitos. El método utilizado es la triplicación del circuito (situando cada uno de los tres elementos en regiones distintas para que en el caso de que una partícula colisione, no dañe dos elementos a la vez), por medio del cual obtenemos tres señales iguales. En el momento en el que un circuito es dañado y crea una salida errónea, el sistema la detecta y saca una de las salidas correctas para que el resto de componentes no sufran ese fallo.

Palabras clave:

- Bitflip
- FlipFlop
- Inyección de errores
- Instrumentación
- TMR (Triple Modular Redundancy)
- Nessy
- FPGA
- Reconfiguración parcial

Abstract

The devices implemented with last-generation technologies are becoming more vulnerable to changes in their behavior induced by cosmic radiation. The objective of this project is to control their potential vulnerability. This happens basically in circuits on board planes, satellites or rockets.

Our work is to adapt a fault injection tool that only can work in FPGA based technologies so that they can be used on ASIC. This tool called Nessy, has been developed by others students at this faculty in several Ms theirs projects during the academic years 2009/2010 and 2010/2011. For that purpose it, was needed to modify the tool and the files so as to allow injecting errors on FlipFlops and thus can emulate ASIC. One of the main differences between a FPGA and an ASIC is that de first ones can be reconfigured and implement news functionalities, do not have any configuration memory. Here changing their functionality is not possible. They only support reset operations, so that failures do not alter their structure, but they only modify their behavior.

Moreover, we have decided to extend the tool to allow protecting the circuits automatically. The method used is based in triple redundancy the circuit (placing each of the three elements in different areas so that two of them are not affected by a collision particle simultaneously). When the circuit is damaged, the system detects it and it shows one of the correct outputs generated by one of the remaining components that do not suffer that failure.

Keywords:

- Bitflip
- FlipFlop
- Fault injection
- Instrumentation
- TMR (Triple Modular Redundancy)
- Nessy
- FPGA
- Partial reconfiguration

Índice

Capítulo 1: Introducción

1.1- Motivación

La caída de una partícula solar (iones, neutrones, protones…) puede alterar el comportamiento de un circuito de forma no transitoria si afecta a elementos de memoria. Esto se ve agravado en sistemas embarcados a grandes alturas o en entornos de alta radiación, puesto que se ven expuestos a problemas como son las colisiones con partículas cósmicas o las alteraciones debidas a las interferencias con elementos radiactivos. Esto puede provocar la alteración transitoria o definitiva del circuito ocasionando un fallo en su correcto funcionamiento y, como consecuencia, puede implicar un error total en el sistema, como por ejemplo, dejar de controlar la aleta de dirección de un avión.

Nuestra atmósfera evita que la mayoría de las radiaciones cósmicas lleguen a la superficie terrestre [ASCM12], actuando de este modo como un filtro. A pesar de esto, la atmosfera no logra filtrar el total de las partículas y por ello, los dispositivos electrónicos se ven expuestos a una posible colisión con alguna de estas partículas. Como consecuencia de esto, se vería afectado el funcionamiento de los mismos. Las partículas antes mencionadas son: protones, partículas alfa, iones pesados y neutrones. Estos últimos, a pesar de no estar apenas presentes en las radiaciones (debido a su corto tiempo de vida), son los responsables de la mayoría de los SEUs (*Single Event Upset*) apreciables y durante los últimos quince años son considerados los principales responsables de incidentes en los dispositivos electrónicos instalados tanto en aviones como en trenes.

La frecuencia de colisiones de partículas varía en función de la altura y la localización geográfica en la que se encuentre el dispositivo. En [Stest] se puede obtener la frecuencia de neutrones en la tierra a partir de unos simples datos facilitados, como pueden ser: la altitud, la latitud y la longitud. La frecuencia que nos da este sitio web corresponde con la frecuencia de colisiones de neutrones por centímetro cuadrado a nivel del mar en la ciudad de Nueva York. Para obtener el valor de esta frecuencia en otra localización geográfica es necesario multiplicar dicho valor por el número que corresponda a esa zona, el cual viene dado en dicha web. El máximo de intensidad en la radiación se alcanza aproximadamente a unos 20.000 metros de altura, y va disminuyendo a medida que nos acercamos a la superficie, hasta que a nivel del mar el flujo de neutrones es el único significativo. Se puede observar que el flujo de neutrones a 10.000 metros de altitud es unas 250 veces superior que al nivel del mar.

En el caso de sistemas embarcados es importante usar un método que permita detectar la vulnerabilidad de un circuito, es decir, que si se produce un fallo no suponga un riesgo para el control del dispositivo en el que se encuentra.

Los métodos más utilizados para medir la vulnerabilidad de un circuito son los basados en simulación, basados en emulación y los basados en radiación. Estos últimos resultan muy costosos puesto que supone colocar el circuito en un entorno radiactivo (acelerador de partículas). Los primeros son excesivamente lentos y son los basados en emulación los que permiten una relación coste/eficiencia mejor.

Durante los cursos académicos 2009/2010 y 2010/2011 en la Facultad de Informática se ha desarrollado Nessy. Esta herramienta es una plataforma de inyección de errores basada en FPGA que permite emular un fallo en todos los bits de la memoria de configuración utilizando el método de reconfiguración parcial, pero no inyecta en FlipFlops. En dicha plataforma se ejecuta el circuito al mismo tiempo que se realiza una inyección de errores. Nuestro objetivo es modificar esta herramienta para permitir que haga inyecciones en FlipFlops de forma directa e individual mediante modificaciones sobre el código original del circuito. Para poder llegar a este punto hay que modificar los ficheros VHDL originales incluyendo las señales necesarias para la inyección de errores en el circuito. Con esto Nessy será una plataforma de inyecciones tanto para tecnología basada en FPGA como en ASIC.

También hemos tratado de proteger los circuitos mediante el método de la triplicación, por medio del cual generamos tres circuitos iguales, lo cual produce tres conjuntos de señales de salida. Estas señales obtenidas son evaluadas por un comparador, el cual detectará si alguno de los circuitos está obteniendo un resultado erróneo, identificando al mismo y seleccionando el conjunto de señales procedentes de uno de los circuitos cuyo funcionamiento es el correcto. Es decir, se selecciona aquella señal o conjunto de señales que sean iguales a las procedentes de otro de los circuitos triplicados, ya que está comprobado que solo podrá haber una señal errónea al encontrarse estos circuitos situados en distintas regiones del componente y es estadísticamente muy poco probable que dos SEUs se produzcan al mismo tiempo y produzcan la misma salida errónea. Además se generan archivos de forma automática, en los cuales se aplicará el método de la triplicación con la respectiva localización de cada circuito. Esto sería una condición en el diseño del circuito global. En Nessy se emula usando regiones diferentes de la FPGA para cada circuito.

1.2- Entorno Software

En el desarrollo de este trabajo nos hemos ayudado de una serie de herramientas explicadas a continuación:

- Eclipse:

  Es un editor de código fuente que por debajo invoca a las herramientas JDK (*Java Development Kit*) y JRE (*Java Runtime Environment*), el cual nos permite crear aplicaciones y entornos de desarrollo integrados. Esta herramienta facilita la construcción de la interfaz de usuario y una amplia visión del proyecto. Debido a esto, nos hemos ayudado de esta herramienta para crear una serie de aplicaciones (que usaremos en Nessy) las cuales trabajan de forma automática con los ficheros VHDL originales para poder lograr nuestros objetivos.

- Xilinx ISE (*Integrated Software Environment*):

  Es una herramienta de diseño de sistemas digitales, la cual permite crear, verificar, simular y sintetizar diseños basados en una FPGA. Ha sido utilizada para crear los códigos en lenguaje VHDL de los diseños o circuitos que se pretenden testear,

verificarlos, sintetizarlos y simularlos, para comprobar su correctitud. Para ello, esta herramienta (además de llevar a cabo una serie de adaptaciones y transformaciones) genera un bitstream (fichero .bit) el cual contiene toda la información para que la FPGA sea configurada por completo, desde la localización de los dispositivos, hasta los elementos de rutado.

- Xilinx EDK (*Embedded Development Kit*):

  Es una herramienta que permite diseñar sistemas empotrados complejos para posteriormente implementarlos sobre la FPGA. Este software se integra con ISE y permite programar aplicaciones las cuales irán mapeadas en memoria y serán ejecutadas por los distintos procesadores que implementemos. Gracias a esta herramienta se ha generado el sistema empotrado que se implementará en la FPGA. A diferencia del ISE, provee de una forma de desarrollar hardware a más alto nivel, encapsulando cada módulo que desarrollemos en "IPcores". Además, nos permite ver cómo están relacionados los cores entre sí sin tener que entrar en detalle y nos da la opción de poder configurar los buses y establecer cómo estos se conectan con los diferentes módulos del sistema.

- Xilinx SDK (*Software Development Kit*):

  Es una versión modificada de eclipse la cual está integrada con Xilinx EDK. De este modo, la herramienta debe recibir un IDE (*Integrated Delopment Environment*) donde poder desarrollar el código en lenguaje C de los procesos software que el procesador del sistema empotrado hardware va a ejecutar. SDK mantiene toda la información importante relacionada con el hardware en un proyecto separado y además, proporciona una serie de funciones de manejo básico de los cores añadidos al sistema, para de este modo poder acceder de manera simple a ellos. Esta herramienta no puede ser considerada un sistema operativo (SO) puesto que sólo nos provee con funciones de alto nivel adaptadas a nuestro hardware. Para poder utilizar los dispositivos y en la memoria asociada a los procesadores, no hay nada más que nuestras aplicaciones.

- Xilinx PlanAhead:

  Herramienta utilizada para la generación de un bitstream parcial para un componente de nuestro sistema, es decir, permite decidir dónde ubicar cada uno de los módulos del sistema (definidos por las netlist correspondientes ".ngc") , así como decidir cuáles serán reconfigurables y cuales estáticos. Además, permite llevar a cabo el proceso de implementación y generar los bitstreams parciales y globales en caso de que estos existan.

1.3- Placa Virtex 5

Nuestro sistema de emulación se va diseñar sobre una FPGA de la familia Virtex 5, que se encuentra empotrada en la placa de desarrollo XUPV5-LX110T (ver Figura 1 obtenida en [Xil1]).

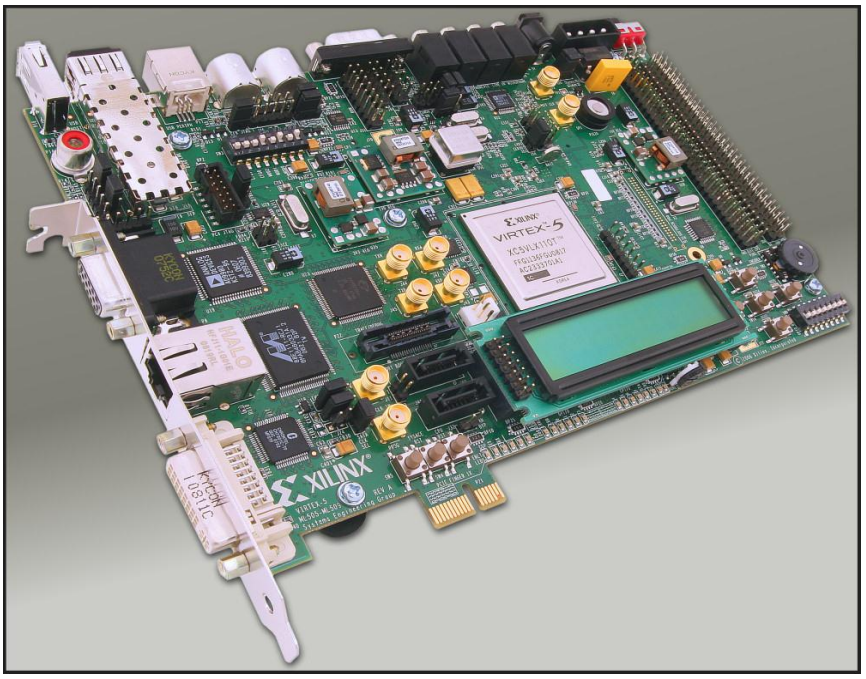


Figura 1: Placa Virtex 5

Las FPGA son circuitos electrónicos prefabricados reconfigurables que contienen bloques de lógica cuya funcionalidad e interconexión se puede determinar mediante una memoria de configuración. Esta característica nos da la posibilidad de generar diferentes circuitos cuyo comportamiento será definido de forma previa y personalizada. A esta característica se la denomina reconfigurabilidad.

Las FPGA aparecieron como una evolución de los circuitos CPLD (*Complex Progammable Logic Device*). Las mejoras que encontramos respecto a estos son su mayor densidad de puertas, la flexibilidad de su arquitectura y que habitualmente contienen módulos empotrados que implementan funciones más complejas, como pueden ser DSP (*Digital Signal Processing*) o multiplicadores. También contienen bloques de memoria empotrados (BRAM).

La principal ventaja de las FPGA es que al diseñar un sistema podemos reducir los costes puesto que nos permite implementar un circuito de forma directa sobre la placa, con lo que no es necesaria su fabricación durante la fase de desarrollo.

Debido a la flexibilidad, capacidad de proceso y a la rápida implementación de sistemas sobre FPGA, estos dispositivos se emplean habitualmente en:

- Procesamiento de señales digitales.

- Sistemas de visión artificial.

- Sistemas aeronáuticos y de defensa.

- Formación en el diseño de sistemas hardware.

- Simulación y depuración en el diseño de microprocesadores y microcontroladores.

- Sistemas de imágenes médicas.

- Codificación y encriptación.

Una FPGA es programable a nivel hardware, aunque actualmente existen dispositivos programables que llevan además procesadores empotrados capaces de ejecutar programas escritos en lenguajes de alto nivel. Debido a esto, proporciona las ventajas de un procesador de propósito general y de un circuito especializado (sistema empotrado).

Además, mediante la utilización de IPcores hardware, es posible emular el comportamiento de un procesador.

1.3.1- Arquitectura Virtex 5

La arquitectura de este tipo de FPGA consiste en una matriz con los siguientes elementos:

- IOBs (*Input Output Buffers*):

  Comunican el dispositivo con el exterior y están situados rodeando la matriz de bloques lógicos, próximos a las patillas del chip. Su funcionamiento se puede configurar para que se comporten como puertos de entrada, salida o entrada-salida. El esquema de un IOB se muestra en la Figura 2.

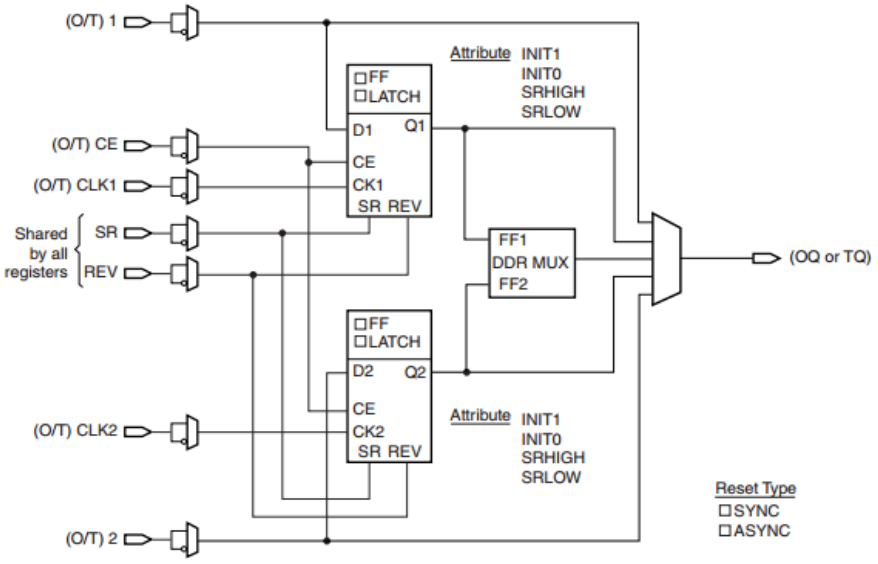


Figura 2: Esquema de un IOB

- CLB (*Configurable Logic Block*):

  En Virtex 5 existen dos tipos de slice: slicel y slicem. Los primeros se usan cuando lo que se va a implementar en ellos es lógica combinacional y los segundos, cuando se necesita almacenar en memoria. Un ejemplo de slicem es el que se puede ver en la Figura 3 (obtenido en [Xil2]).

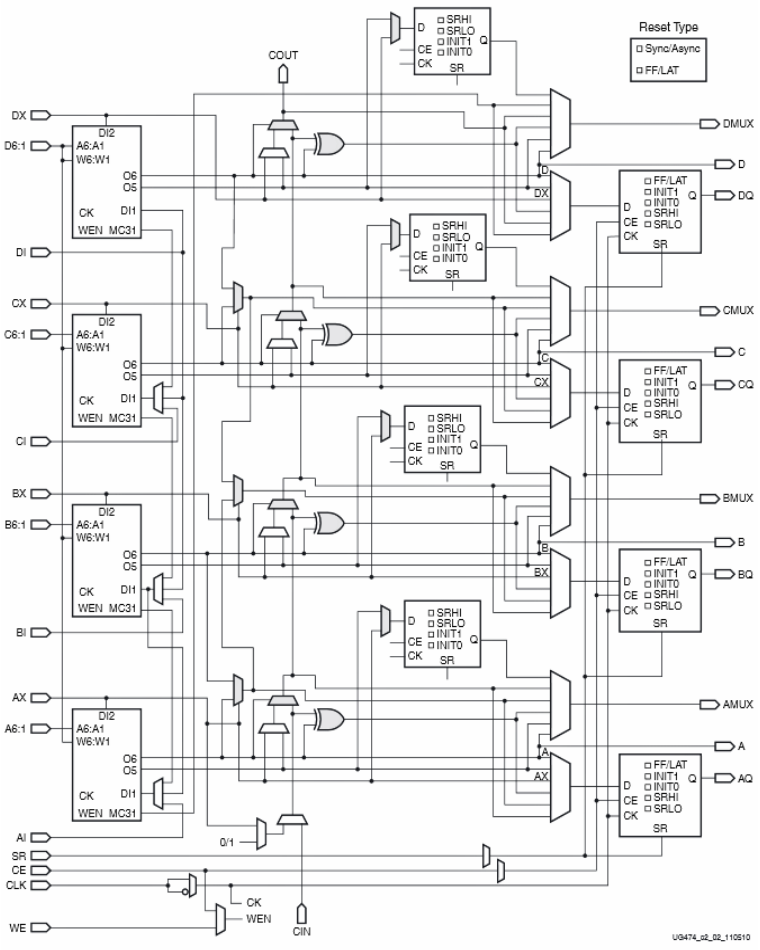


Figura 3: Esquema de un CLB

- BRAM (*Block RAM*):

  Estos elementos se encuentran integrados en la FPGA y son los encargados de evitar usar los CLB con la función de sintetizar los módulos de memoria menos eficientes. Son dispositivos de almacenamiento muy rápidos y en el caso de las FPGA Virtex 5, cada uno de ellos tiene una capacidad de 2KB.

- DSP (*Digital Signal Processing*):

  Se utilizan para operaciones aritmético-lógicas. Está compuesto por un multiplicador de 25 bits * 15 bits y un sumador de 48 bits.

- CMT (*Clock Manager Tiles*):

  Es un componente que agrupa los DCM (*Digital Clock Manager*), que se encargan de la gestión del reloj y realizan tareas de multiplicar o dividir frecuencias, aseguran una señal de reloj limpia y desfasan el reloj una cantidad fija.

1.4- Trabajo realizado

Utilizando Nessy, una plataforma de inyección de errores (que explicamos en el Capítulo 2) y gracias a las herramientas descritas en este capítulo, hemos creado una serie de aplicaciones capaces de adaptar y triplicar el código de un circuito cuya vulnerabilidad quiere probarse, de forma que se pueda ajustar a las nuevas necesidades para la inyección sobre FlipFlops y prueba de errores. Estas aplicaciones son:

- Adaptador VHDL:

  Esta aplicación es la encargada de adaptar el código original para poder realizar las inyecciones de forma directa en FlipFlops, introduciendo las señales necesarias y adaptando sintácticamente el fichero de forma que no suponga ningún problema para su posterior introducción en Nessy. A su vez, genera un fichero que contiene la localización espacial donde se colocarán todos sus elementos sobre la placa.

- Triplicador VHDL:

  Esta aplicación se encarga de generar un nuevo fichero a partir del generado por la aplicación "Adaptador VHDL", triplicando sus componentes de manera transparente al usuario. A su vez, genera un nuevo fichero que también contendrá la localización espacial donde se colocarán todos sus elementos.

- Modificar TB:

  Esta sencilla aplicación genera un nuevo Testbench (codificado en un fichero con extensión .txt) a partir de otro original, el cual contiene una lista de estímulos a las señales de entrada del circuito. Este nuevo fichero es igual que el original pero añadiéndole el valor de una nueva señal llamada Set_Nessy en la segunda columna. Cuando esta señal toma valor "1", indica en qué ciclo de reloj exacto se inyecta el bitflip.

Capítulo 2: Nessy

2.1 –Definición de la herramienta

Nessy es una plataforma de inyección de errores que cuenta con un alto rendimiento y un nivel de intrusión nulo y permite emular la caída de una partícula cósmica sobre la memoria de configuración de la FPGA.

La no intrusividad se garantiza considerando que la aplicación se ejecuta en un sistema hardware multitarea siguiendo el modelo arquitectónico representado en la Figura 4.

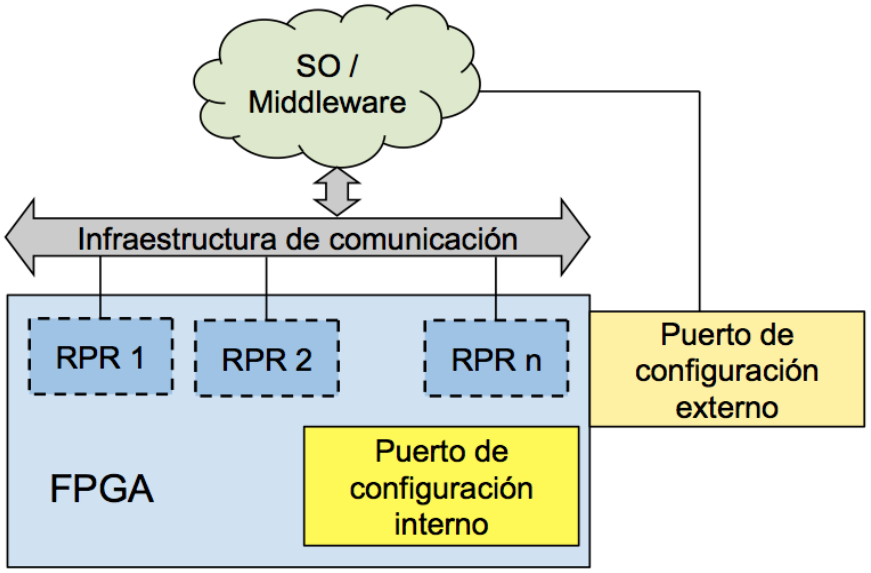


Figura 4: Sistema hardware multitarea sobre el cual los circuitos digitales testeados con Nessy se implementarán

En este sistema, la FPGA se divide en un conjunto de Regiones Parcialmente Reconfigurables (RPRs). Estas RPRs se conectan a una infraestructura de comunicación que puede ser implementada por medio de un bus o un *Network-on-Chip* (NoC) [BdM02]. Cada tarea a ejecutar en este sistema se coloca y enruta en una de estas RPRs y ésta se comunicará con el resto del sistema utilizando la infraestructura de comunicación mencionada. El control de la ejecución de las tareas (configuración, I/O, etc) se realiza a través de un sistema operativo o *midleware* y puede ser ejecutado en un procesador que se puede colocar tanto dentro como fuera de la FPGA. En cualquier caso, el procesador también está conectado a la infraestructura de comunicación. La I/O de las tareas se implementa junto con el hardware de la tarea asignada en la RPR correspondiente. Cada RPR se configura mediante reconfiguración parcial, ya sea mediante un puerto de configuración externo (JTAG, BPI, etc) o uno interno (ICAP).

Nessy dispone de una interfaz de usuario que permite introducir los ficheros necesarios, como son el VHDL del circuito a testear, el fichero de Testbench, el Golden, etc. A partir de ellos, genera de forma automática los bitstreams parciales y globales del sistema invocando

las herramientas de desarrollo de Xilinx PlanAhead y EDK 12.1. El bitstream parcial es el que corresponde únicamente al circuito a testear (circuito bajo prueba en Figura 5).

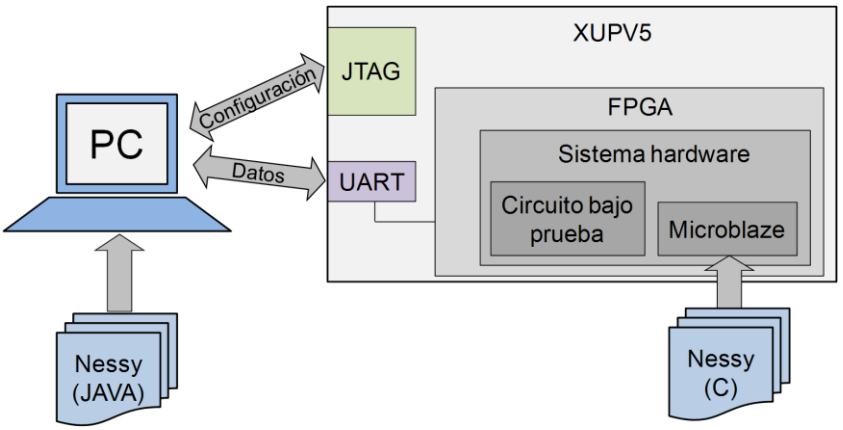


Figura 5: Arquitectura de Nessy

La Figura 5 presenta una visión de la arquitectura de Nessy. Este sistema se compone de un PC y una placa de desarrollo XUPV5LX110T, que se conectan mediante dos canales de comunicación. El canal de configuración gestiona toda la configuración inicial de la FPGA usando la interfaz de JTAG, mientras que el de datos se utiliza para el intercambio de ficheros bitstreams y Testbench entre el software del PC y la FPGA, y para tareas de sincronización. En este último caso el protocolo de comunicación utilizando es el RS232, por lo tanto las comunicaciones a través de este canal están restringidas en la medida de lo posible, con el fin de no degradar el rendimiento de la inyección de errores.

La implementación del sistema hardware en la FPGA Virtex 5 (Figura 5) es consistente con el modelo de arquitectura mostrado en la Figura 4. En este caso la inyección de errores se lleva a cabo mediante el microprocesador MicroBlaze [Xil5], el cual se implementa utilizando los recursos disponibles en la FPGA. Nessy tiene también un puerto de configuración interno (ICAP) y una interfaz que conecta el circuito testeado con la infraestructura de comunicación, que en este caso ha sido implementado usando un bus PLB (Processor Local Bus). En este sistema, los circuitos testeados se colocan en una RPR, exactamente con las mismas características (localización, anchura y altura) que la que se utilizará como parte del circuito en una misión real. Además el software que se ejecuta en la MicroBlaze se sincroniza con el resto de elementos de Nessy del mismo modo que en un sistema real.

La Figura 6 muestra el esquema general del sistema hardware de Nessy. Dicho sistema contiene un procesador MicroBlaze conectado a la memoria de instrucciones y datos (BRAM) mediante un bus LMB (*Local Memory Bus*). Esto se conecta al resto del sistema mediante un bus de tipo PLB. Además, a este bus se conectan otros elementos como la memoria DDR (*Double Data Rate*), el adaptador UART (*Universal Asynchronous Receiver-Transmitter*) y el puerto de configuración ICAP (*Internet Content Adaptation Protocol*). Tambien, se incluye el circuito que se quiere testear (circuito en Figura 6), el cual necesita de una interfaz para poder conectarse a dicho bus. Eso nos permite diseñar y emular el circuito sin necesidad de fabricarlo.

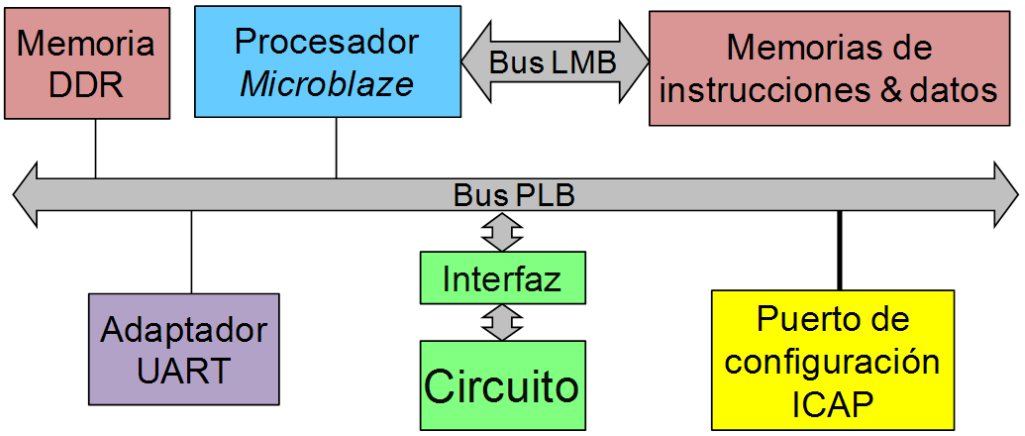


Figura 6: Estructura del sistema empotrado

Precisamente ese módulo que llamamos "circuito" será el módulo cuya vulnerabilidad se quiere evaluar mediante una inyección de errores. La utilización de FPGA se está extendiendo en todos los ámbitos del diseño de sistemas digitales, debido a que los costes y el tiempo de desarrollo son mucho menores. En la mayoría de casos, no se procede a la fabricación del circuito gracias a la posibilidad de poder generarlo en la propia FPGA. Finalmente, el sistema es conectado con la UART de la placa XUP mediante el módulo adaptador de la UART.

Este esquema permite al puerto ICAP realizar las emulaciones de SEUs mediante reconfiguración parcial. Con el fin de emular un SEU, el software que se ejecuta en la MicroBlaze inyecta un bitflip en uno de los bits de configuración que configura el circuito testeado. Para ello, Nessy realiza reconfiguración parcial de la *frame* (que es la mínima unidad de reconfiguración) correspondiente al bit que se quiere modificar. Como Nessy no realiza la reconfiguración total del circuito con el fin de emular el SEU, se logra un rendimiento muy alto.

La metodología descrita permite inyectar SEUs sin introducir ninguna modificación ni en la colocación, ni en el enrutado del circuito testeado excepto el bitflip inyectado. Por esta razón, Nessy es una plataforma de emulación de SEU no intrusiva. Nessy también cuenta con dos ventajas adicionales. Por un lado, un coste de implementación asequible. En efecto, Nessy no requiere una plataforma ad-hoc, sólo se utilizan unas FPGA y un PC para la comunicación con la MicroBlaze. Por otro lado, es fácil de portar a otras FPGA de Xilinx Virtex.

Finalmente, el PC (Figura 5) controla el funcionamiento global de los experimentos de inyección de errores a través de una GUI (*Graphical User Interface*), programada en Java. Dicha GUI se puede observar en la Figura 7.

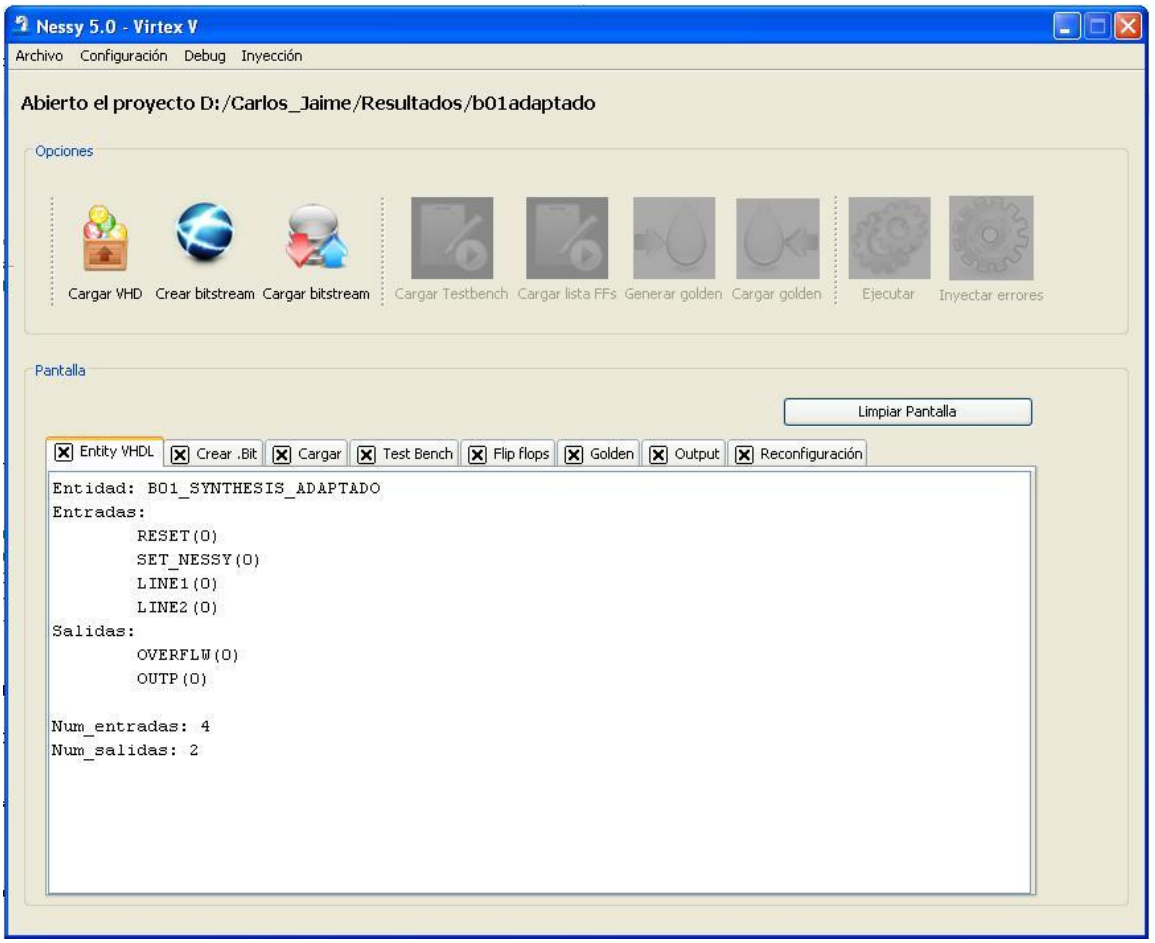


Figura 7: GUI de Nessy

Así, trabajar con Nessy es muy fácil e intuitivo. Pueden encontrarse más detalles de esta GUI en [ASCM12].

La *frame* es la mínima unidad de configuración. Esto significa que, para inyectar un bitflip en un circuito mediante reconfiguración parcial, se debe cargar en el puerto de configuración, al menos, la información de configuración de la *frame* que contiene el bit de configuración involucrado. Cada *frame* contiene 40 palabras de 32 bits (por lo tanto hay un total de 40 * 32=1280 bits de configuración).

Varios *frames consecutivos* contienen la información de configuración asociada a un conjunto de bloques de recursos adyacentes situados en la misma columna, de tal manera que una única *frame* contiene información de configuración de 20 CLB o de 4 DSP.

A través de un análisis más profundo, se puede observar que la matriz de bloques de recursos es irregular. En otras palabras, cada bloque de recursos no se configura a través del mismo número de *frames*. Así, se necesitan 36 *frames* para configurar un conjunto de 20 CLB, y 28 *frames* para un conjunto de 4 DSP. Por lo tanto, cada conjunto de 20 CLB se reconfiguran con 36 * 1280 = 46080 bits de configuración, mientras que para configurar un conjunto de 4 bloques DSP se necesitan 28 * 1280 = 35840 bits de configuración.

Con esta información, Nessy es capaz de iterar en todo el bitstream del circuito para inyectar SEU en los recursos de la FPGA de manera controlada.

En nuestro proyecto, utilizamos esta herramienta para realizar inyecciones sobre circuitos que supuestamente van a utilizar otras tecnologías, no necesariamente FPGA, en su implementación final y para poder comparar los resultados de las inyecciones antes y después de aplicar el método de la triplicación. Para poder llevar a cabo esto, tuvimos que efectuar una serie de modificaciones sobre Nessy. En primer lugar, añadimos una nueva opción a su interfaz gráfica, en la cual se permite seleccionar si la inyección va a ser sobre FlipFlops (como se puede observar en Figura 8).

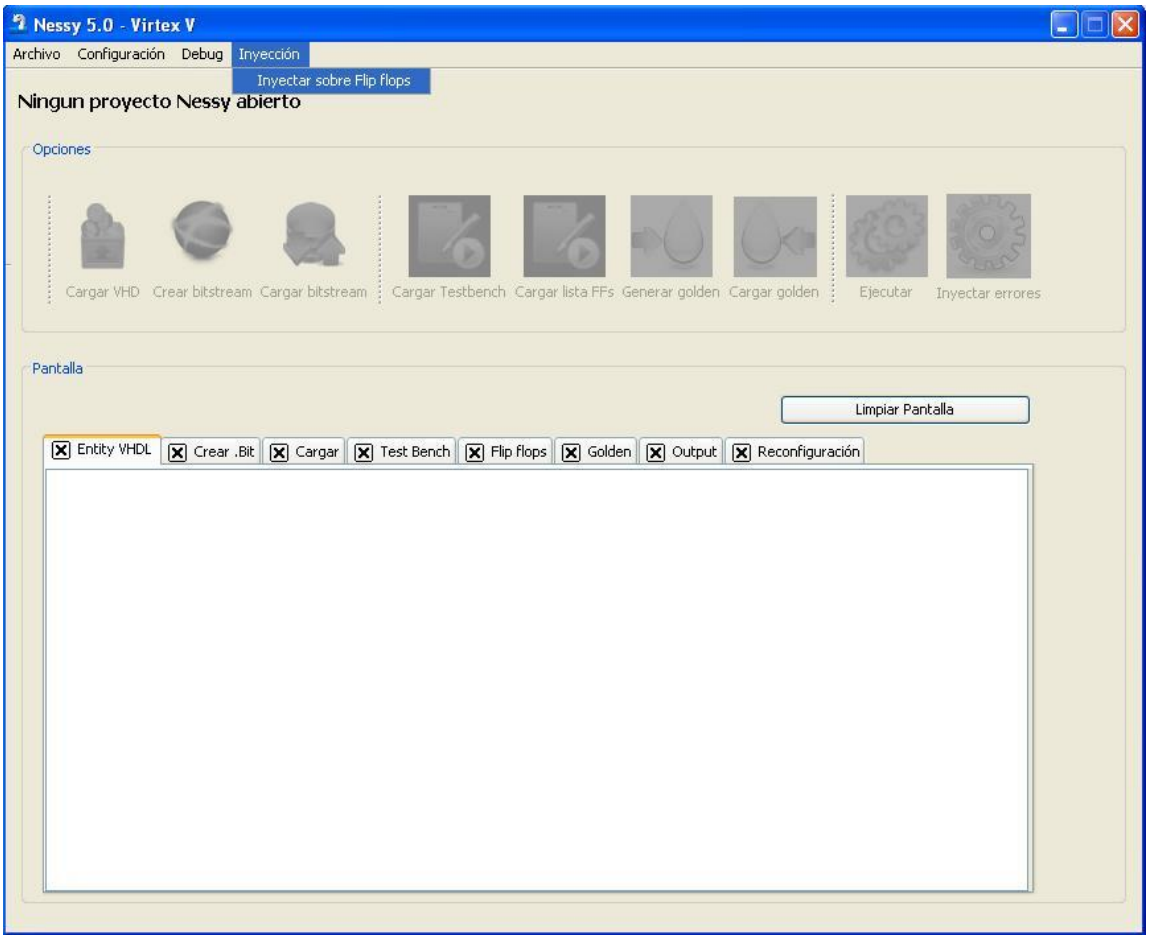


Figura 8: Botón Inyectar Errores en la GUI de Nessy

Tras esto, modificamos Nessy de tal manera que, al activar esta opción, la herramienta modifica su configuración para pasar a efectuar inyecciones sobre FlipFlops de forma directa e individual (en lugar de en la memoria de configuración) y por lo tanto una vez introducidos los ficheros VHDL, solicitará su fichero de restricciones UCF.

2.2 – Trabajar con la herramienta para inyectar en FlipFlops

En primer lugar, una vez abierta la aplicación, hay que seleccionar la opción "Inyección ->Inyectar sobre FlipFlops" para que la aplicación se configure de tal manera que solicite el fichero UCF de localización de estos. Esto es necesario porque la información acerca de la localización física de los FlipFlop utilizados por el circuito bajo test no se encuentra directamente en el bitstream del circuito. Por tanto, es necesario pasarle esta información a Nessy a través de un fichero de restricciones UCF. Tras esto, creamos un nuevo proyecto. Una vez creado, el programa nos da la opción "Cargar VHD" o cargar una serie de ficheros y seleccionar cual es el TOP (este caso se da cuando hacemos inyecciones en circuitos triplicados, puesto que tenemos que añadir el fichero original adaptado, el triplicado y el comparador). Una vez cargado el fichero VHDL, Nessy reconoce automáticamente las entradas y salidas del circuito y habilita la opción "Crear bitstream", la cual, tras ser seleccionada nos solicita el fichero UCF y las coordenadas "X" e "Y" de la FPGA donde se desea colocar el circuito bajo test. A continuación procede a la generación del bitstream. Cuando esta generación ha terminado, se habilita el botón "Cargar bitstream". Tras esto, procedemos a cargar el Testbench pulsando la opción "Cargar Testbench". Cuando se ha cargado correctamente este fichero, si en un principio hemos seleccionado la opción "Inyectar sobre FlipFlops", se verá habilitado el botón "Cargar lista FFs" el cual busca en el fichero UCF los FlipFlops declarados y los añade a una lista para posteriormente poder inyectar sobre ellos. El siguiente paso es seleccionar "Generar golden", el cual genera un fichero con las salidas del circuito en una ejecución normal sin errores. A continuación, se puede pulsar la opción "Ejecutar" para comprobar que el Golden ha sido generado de forma correcta. Por último, pulsamos el botón "Inyectar errores" el cual procede a inyectar un error en cada FlipFlop un determinado número de veces (en nuestro caso se hacen 100 inyecciones por FlipFlop). Cada uno de estos errores se inyecta de manera aleatoria en un ciclo de reloj del Testbench cargado anteriormente. Durante la ejecución de esta función, el programa nos muestra por pantalla los resultados obtenidos de la comparación entre el resultado del Golden y el resultado de la ejecución con errores. Un ejemplo de esto se muestra en la Figura 9.

```
FF(51,143,x,"RLAST_7"): bitflip en ciclo 501.
      Sin error
FF(51,143,x,"RLAST_7"): bitflip en ciclo 965.
      Sin error
FF(51,143,x,"RLAST_7"): bitflip en ciclo 980, error en ciclo 980.
      00010001 != 10010001 (10000000)
FF(51,143,x,"RLAST_7"): bitflip en ciclo 429, error en ciclo 430.
      00000101 != 10000101 (10000000)
```

Figura 9: Ejemplo de error en el proceso de inyección

Una vez finalizado este último proceso, se nos muestran los resultados de la ejecución, como son el tiempo tardado, y el porcentaje de resultados que han dado distinto a lo obtenido por el Golden. Un ejemplo de resultado de una inyección de errores es el mostrado en la Figura 10.

```
Tiempo tardado:                 0d:0h:10m:33s
NÃºmero de flip flops:                 67
Inyecciones por flip flop:            100
NÃºmero de reconfiguraciones:                6700
NÃºmero de reconfiguraciones errÃ³neas:    1026
Porcentaje de errores:                15.3134328358208948%
```

Figura 10: Resultado final de una inyección de errores

Capítulo 3: El método de inyección de errores sobre FlipFlops en FPGA

El método de inyección de errores en la memoria de configuración para las FPGA Xilinx Virtex 5 mediante reconfiguración parcial se aleja en gran medida de lo necesario para lograr modificar el contenido de un FlipFlop. En el primer caso, se puede acceder a la memoria de configuración y modificar directamente su contenido [ASCM12]. Sin embargo, desde la memoria de configuración del dispositivo no se puede modificar la información almacenada en un FlipFlop, a menos que se lleve a cabo un reseteo de la lógica secuencial de la FPGA [Xil4].

Existen para cada FlipFlop dos bits complementarios, llamados SRHIGH (*Set/Reset to High*) y SRLOW (*Set/Reset to Low*) los cuales son accesibles desde la memoria de configuración. Realmente estos bits son el mismo en la memoria de configuración. De este modo, sólo pueden tomar los siguientes valores: SRHIGH = 1, SRLOW = 0 y SRHIGH = 0, SRLOW = 1. La función de estos bits es establecer la funcionalidad de las señales de entrada SR (Set/Reset) y REV (REVerse) para cada FlipFlop de la FPGA, como se indica en la Figura 11. Así, si SRHIGH = 0 (SRLOW = 1) entonces SR puede ser utilizado para inicializar el FlipFlop a "0" (actuando así como una señal de Reset) y REV, para inicializarlo a "1" (como una señal de Set). Mientras que si SRHIGH = 1 (SRLOW = 0) el FlipFlop se comportará de forma opuesta (SR será como un Set y REV como un Reset). A menos que se indique lo contrario, las herramientas de síntesis de Xilinx establecen los valores de SRHIGH = 0 y SRLOW = 1 en los FlipFlop de Virtex 5.

SRHIGH=0; SRLOW=1			*SRHIGH=1; SRLOW=0*		
SR	*REV*	*Q (t+1)*	*SR*	*REV*	*Q (t+1)*
0	0	*Q (t)* [a]	0	0	*Q (t)* [a]
0	1	1 [b]	0	1	0 [b]
1	0	0 [b]	1	0	1 [b]
1	1	0 [b]	1	1	0 [b]

a - Actualiza solo si CE = "1"
b - Actualiza tanto si CE = "1" como CE = "0"

Figura 11: Tabla de verdad de un FF de Virtex 5

Las herramientas de síntesis de Xilinx permiten instanciar los FlipFlop existentes en la FPGA a través de primitivas del código fuente del circuito. La lista completa de las

primitivas disponibles para instanciar los FlipFlops de la FPGA Virtex 5 se pueden encontrar en [Xil4]. Se diferencian básicamente en la cantidad de entradas que son accesibles desde el código fuente (principalmente, la señal de entrada Reset y/o la señal de Set). Entre todas las formas de instanciación disponibles, FDRSE (un D-FlipFlop con Reset y Set síncronos) es una de las formas más directas de implementar un FlipFlop en una FPGA. La razón es que cuando SRHIGH = 0 y SRLOW = 1, el Reset de entrada está conectado a la señal SR y el Set, a la señal REV. Existen otras primitivas para instanciar FlipFlops, las cuales se implementan de manera similar, pero estableciendo las señales SR y/o REV a "0" a fin de cumplir con las especificaciones de las mismas. Sin embargo, todas ellas tienen en común que instancian al mismo (y único) tipo de FlipFlop de la FPGA.

La metodología implementada en Nessy para inyección de errores sobre FlipFlops utiliza la información de la Figura 11 y añade un hardware de instrumentación alrededor de todas las instancias de las primitivas de los FlipFlops en el circuito bajo test con el fin de emular los SEU simplemente modificando el bit SRHIGH/SRLOW en la memoria de configuración. Este nuevo hardware incluye una nueva señal de entrada Set_Nessy, cuyo funcionamiento se introdujo al final de la sección anterior. Así, si esta señal toma el valor "1", esto indica que se debe emular un SEU en cualquiera de los FlipFlops del circuito testeado. Esta señal es parte del Testbench del circuito, lo que permite controlar el ciclo de reloj en el que el SEU se emula.

En nuestro proyecto, hemos modificado Nessy para que utilice esta información de la siguiente manera: SRHIGH = 1, SRLOW = 0 del FlipFlop en el cual se inyectará el error, dejando a ese SRHIGH = 0 y SRLOW = 1 los restantes. Por tanto, para realizar esto no es necesario utilizar la captura y readback/modificación de todos los FlipFlop en la FPGA. En vez de esto, sólo se modifica un bit por error inyectado en el circuito bitstream. Nessy identifica la *frame* que contiene el bit que tiene que ser modificado y escribe la versión modificada de esta *frame* sobre la memoria de configuración de la FPGA. Como ya se comentó en el Capítulo 2, una *frame* es el segmento direccionable más pequeño en el espacio de direccionamiento de la memoria de configuración de Virtex 5, la cual contiene 1280 bits de configuración. Por lo tanto, esta estrategia garantiza que la penalización temporal adicional introducida debido a la reconfiguración parcial es muy baja.

Ha sido necesaria una nueva metodología puesto que cierta información no se podía obtener de la documentación de Xilinx [Xil3] y [Xil4].

La lógica extra añadida para cada primitiva FlipFlop, se implementa en tres LUT (*Look-Up Table*), las cuales se llaman FF_SR, FF_CE y FF_REV respectivamente. Cada una de ellas implementa las siguientes ecuaciones:

(1) $FF_{SR} = Set\_Nessy * \overline{Q} + \overline{Set\_Nessy} * original\_R$

(2) $FF_{CE} = \overline{Set\_Nessy} * original\_CE$

(3) $FF_{REV} = Set\_Nessy * Q + \overline{Set\_Nessy} * original\_S$

Donde original_CE, original_R y original_S son las señales de los FlipFlop originalmente conectadas al CE (*Clock Enable*), Reset y Set de las entradas primitivas de los FlipFlop, respectivamente. Esto es así para un FDRSE, donde estas tres entradas son accesibles desde el código fuente. En el caso de que una primitiva fuese instanciada en el código sin Reset y/o Set, ésta será remplazada por FDRSE, mediante el establecimiento de la señal inexistente original_reset y/o original_set a "0" buscando no alterar su comportamiento original.

Acorde con (1), (2) y (3) si no se inyecta ningún error (Set_Nessy = 0), el FlipFlop funciona normalmente (FF_{CE} = original_CE). De lo contrario, todos los FlipFlops del circuito testeado se actualizan de acuerdo con sus valores especificados de los bits de configuración SRHIGH y SRLOW y las entradas FF_{REV} y FF_{SR} en función de la Figura 11. Así, la metodología presentada hace que el FlipFlop seleccionado trabaje con la configuración SRHIGH=1 SRLOW=0, acorde con las ecuaciones (1) y (3) si Set_Nessy = 1, $FF_{SR}=\overline{Q}$ y FF_{REV}=Q. Por lo tanto, si Q = 0, FF_{SR} = 1 y FF_{REV} = 0, Q (t+1) toma el valor 1 (bitflip), en el FlipFlop deseado, y 0 (no cambia) en los restantes. Siguiendo el mismo razonamiento si Q = 1, FF_{SR} = 0 y FF_{REV} = 1, Q (t+1) toma el valor 0 (bitflip) en el FlipFlop deseado y 1 (no cambia) en los restantes. Dicho esto, si varios SEU se inyectan en varios FlipFlops, este proceso se realizará de forma paralela con todos los FlipFlops involucrados.

Como consecuencia de esta instrumentación hardware, el área requerida para implementar el diseño adaptado para la inyección de errores en FlipFlops es mayor que en la versión original. El factor de incremento del área frente al del circuito original depende del número de FlipFlops utilizados para la implementación de cada circuito. Esto se mostrará en detalle en el Capítulo 5.

Capítulo 4: Aplicaciones desarrolladas

4.1 - Adaptador VHDL

En este capítulo se describen las tres aplicaciones que se han desarrollado e integrado en Nessy para la inyección de errores en FlipFlops. Esta aplicación tiene como objetivo adaptar el código VHDL estructural de un circuito para poder hacer inyecciones de forma directa sobre los FlipFlops según se ha explicado en el Capítulo 3. Para ello es necesaria la creación de una nueva señal llamada Set_Nessy y el redireccionamiento de las señales antiguas para que la nueva señal pueda actuar sobre ellas. Añadimos cierta cantidad de hardware de instrumentación para cada FlipFlop, en el cual la nueva señal Set_Nessy, junto con el bit SRHIGH/SRLOW, permita modificar el valor de las señales de cada FlipFlop de forma independiente.

En primer lugar, el archivo recibido debe ser la versión sintetizada de un VHDL original. Para ello es necesario crear un nuevo proyecto ISE y seleccionar la opción "Generate Post-Synthesis Simulation Model" habiendo desactivado previamente la opción de incluir los I/O Buffers.

Una vez tenemos el fichero preparado, pasamos a introducirlo en el Adaptador. Para ello, nuestro programa nos pide una serie de datos que son, el nombre del fichero y la posición X e Y iniciales de las que se partirá para la posterior colocación de los elementos sobre la placa. Estos X e Y deben ser consistentes con la localización final que se seleccionará en Nessy para colocar finalmente el circuito a testear.

Una vez introducidos los datos, comienza la ejecución de nuestro programa. En primer lugar, se invoca a una función llamada "exploraArchivo" en la cual se lee el archivo origen buscando los las primitivas de FlipFlops existentes en este código e identifica sus señales guardándolas en una lista donde se refleja su nombre y a qué FlipFlop pertenecen. Las directivas tomadas para la identificación de los componentes y sus respectivas señales son:

** D(data input), CE(clock enable), C(clock), CLR(asynchronous clear), Q(data output), PRE(asynchronous preset), S(set), R(synchronous reset)*

- FDCE: D, CE, C, CLR, Q
- FDE: PRE, D, CE, C, CLR, Q
- FDCPE: PRE, D, CE, C, CLR, Q
- FDRSE: S, D, CE, C, R, Q
- FDCE_1: D, CE, C, CLR, Q
- FDCPE_1: PRE, D CE, C, CLR, Q
- FDRSE_1: S, D, CE,C, R, Q
- FDP: PRE, DC, Q

- FDR: S, D, CE, C, R, Q

- FDS: Q, D, C, S

En el caso de que el FlipFlop no disponga de nombre en el código VHDL, se le asigna uno siguiendo el patrón "FF_x" para que posteriormente pueda ser renombrado de forma correcta. Debido a que hay distintos tipos de FlipFlops, existen señales nombradas de distinto modo pero con una misma funcionalidad, por ello las señales "R o CLR" se nombrarán como "R" y las señales "S o PRE" se nombrarán como "S". Una vez terminado este paso, ya disponemos de una lista de señales para poder trabajar con ellas y procedemos a llamar a una segunda función llamada "generaArchivo". Esta función en primer lugar genera un fichero VHDL nuevo y tras esto, comienza a leer línea por línea el fichero origen. Primero elimina la línea "use UNISIM.VPKG" y las asignaciones de las variables iniciales contenidas en el "port" de la entidad como puede ser "*X: out stdlogic := 'X';*" transformándolas a " *X: out stdlogic;*" incluyendo la nueva señal Set_Nessy siempre en la segunda posición. Además de esto, renombra la entidad llamándola con su mismo nombre más "_synthesis_Adaptado". Tras esto, incluye la declaración de las nuevas señales necesarias para la adaptación del circuito.

Una vez finalizados estos pasos, continúa con la lectura del fichero origen. Si no se detecta la estructura de un FlipFlop, se copia de forma directa línea por línea, en caso contrario, procederá a su adaptación. En esta modificación, lo primero que hace es escribir el nombre del FlipFlop definiendo la nueva estructura, que en caso de ser síncrono será FDRSE y en caso contrario FDCPE, procediendo a redireccionar las señales antiguas y escribiendo las nuevas señales. De este modo, la estructura resultante sería la ilustrada en la Figura 12.

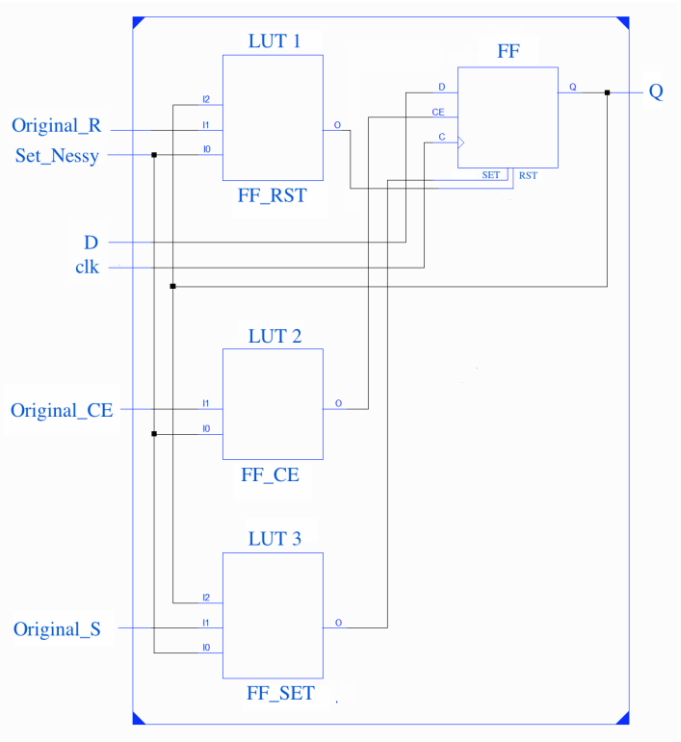


Figura 12: Esquema RTL de las modificaciones realizadas en un FF para poder inyectar errores en él

Los elementos señalados como LUT implementan las siguientes funciones que se corresponden con las ecuaciones (1), (2) y (3) del Capítulo 3:

- LUT1 (FF_RST): ((not Q) and Set_Nessy) or ((not Set_Nessy) and Original_R)

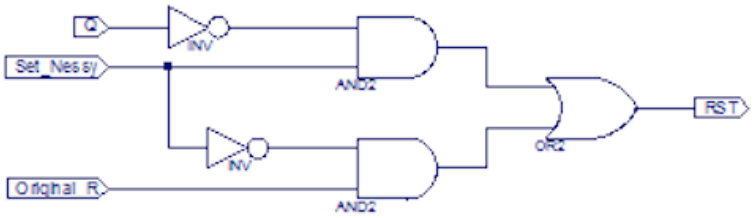


Figura 13: Implementación de la señal RST modificada

- LUT2 (FF_CE): (not Set_Nessy) and Original_CE

Figura 14: Implementación de la señal CE modificada

- LUT3 (FF_SET): (Q and Set_Nessy) or ((not Set_Nessy) and Original_S)

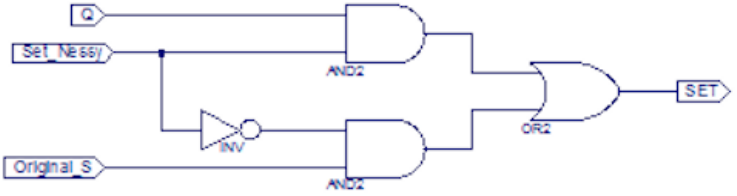


Figura 15: Implementación de la señal SET modificada

Existe la posibilidad de que alguno de los FlipFlops no contenga alguna de estas señales necesarias (RST, CE y/o SET), sin embargo en este caso, estas señales serían incluidas de forma automática en la información de las primitivas, aunque no aparecerían reflejadas en las funciones por simplicidad. Las señales son incluidas con la " , " delante para identificar que son señales puestas a posteriori y además, se muestra por consola un mensaje de error a modo informativo en el cual se refleja el nombre del FlipFlop y las señales que le faltan. De este modo, la estructura resultante sería la que se puede ver en la Figura 16.

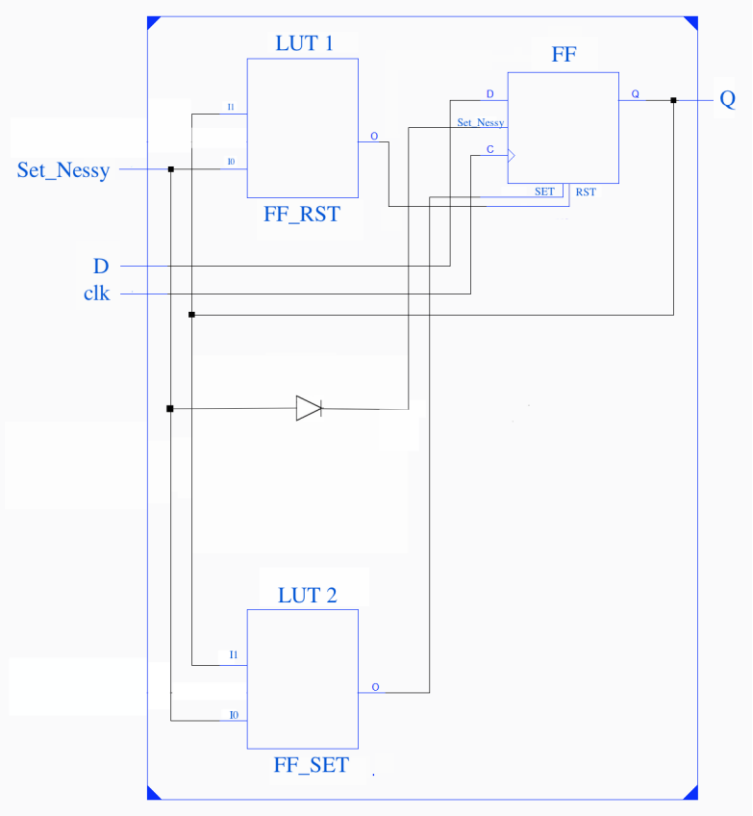


Figura 16: Esquema RTL de las modificaciones realizadas en un FF (cuado faltan las señales R, S y CE)

En esta nueva estructura, los elementos señalados como LUT implementan las siguientes funciones:

- LUT1 (FF_RST): (not Q) and Set_Nessy

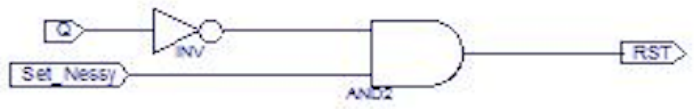


Figura 17: Implementación de la señal Reset (cuando faltan las señales R, S, CE)

- FF_CE: (not Set_Nessy)

Figura 18: Implementación de la señal CE (cuando faltan las señales R, S, CE)

- LUT2 (FF_SET): Q and Set_Nessy

Figura 19: Implementación de la señal Set (cuando faltan las señales R, S, CE)

Una vez terminada la generación del nuevo fichero VHDL, se procede a la generación de un fichero UCF en el cual se define la ubicación de los FlipFlops en función de los valores iniciales dados por el usuario previamente (observable en la Figura 20).

```
INST "circuito_0/circuito_0/instancia/Q_0" LOC=SLICE_X48Y140;
INST "circuito_0/circuito_0/instancia/Q_1" LOC=SLICE_X49Y140;
INST "circuito_0/circuito_0/instancia/Q_2" LOC=SLICE_X48Y141;
INST "circuito_0/circuito_0/instancia/Q_3" LOC=SLICE_X49Y141;
```

Figura 20: Ejemplo de fichero UCF

Para ubicar los elementos, cada línea de este fichero debe contener el nombre que tiene asignado el FlipFlop y su localización. El valor de la altura está establecido de forma que nunca supere el valor de Y = 159, por lo que si se alcanza esta altura, se pasará de forma automática a la siguiente posición de X y el valor de Y volverá a su origen.

Cuando finaliza el programa, nos muestra dos mensajes por consola, uno diciendo "Fin de Programa" y otro mostrando si ha terminado de forma correcta o con algún error, identificando el mismo. Un ejemplo de lo que se muestra durante la ejecución del programa es el ilustrado en la Figura 21.

```
Sennales que faltan en el flip flop Q_0: S CE
Sennales que faltan en el flip flop Q_1: S CE
Sennales que faltan en el flip flop Q_2: S CE
Sennales que faltan en el flip flop Q_3: S CE
Fin de programa
0 :Programa finalizado con exito
```

Figura 21: Ejemplo de mensaje por pantalla tras la ejecución del "AdaptadorVHDL"

Si hay error, el programa lo identifica y lo muestra por pantalla en función del valor devuelto al finalizar la ejecución, basándose en la clasificación que se muestra en la Figura 22.

```
switch (resul){
    case 0: System.out.println(resul+" :Programa finalizado con exito"); break;
    case -1: System.out.println(resul+" :Error al abrir fichero origen"); break;
    case -2: System.out.println(resul+" :Error al abrir fichero destino"); break;
    case -3: System.out.println(resul+" :Error al cerrar fichero destino"); break;
    case -4: System.out.println(resul+" :Error en el fichero destino de direcciones"); break;
}
```

Figura 22: Lista de posibles valores devueltos por la aplicación

4.2 - Triplicador VHDL

Esta aplicación tiene como objetivo crear un nuevo fichero VHDL en el cual se triplica un componente del fichero origen y se conectan sus salidas a un comparador que nos otorga la salida o conjunto de salidas correctas y una señal que indica si las tres señales procedentes de los distintos circuitos triplicados son iguales, si son distintas o identifica cuál es la señal o conjunto de señales distintas. Además, genera un nuevo fichero UCF con la correspondiente localización de todos los elementos generados sobre la placa.

En primer lugar, la aplicación recibe el nombre del fichero obtenido en la ejecución de la aplicación "Adaptador VHDL". Este debe estar en la carpeta de ejecución de la aplicación. Asimismo, en ese mismo directorio debe existir su correspondiente fichero UCF con la información de la localización de los FlipFlops en el dispositivo.

El fichero "comparador.vhd" mencionado anteriormente, contiene la descripción del comparador, es decir, la configuración del circuito que usaremos para comparar las señales recibidas por los elementos triplicados. Para ello, recibe tres vectores conteniendo las señales de salida de cada componente, las compara y genera dos salidas llamadas "Error" y "Salida". La señal "Salida" es un vector que contiene la salida seleccionada (Salida1, Salida2 o Salida3) como resultado de la comparación. La señal "Error" es un vector de tres elementos el cual nos indica el resultado de la comparación, como se muestra en la Figura 23.

```
000 - tres distintas
001 - X distinto
010 - Y distinto
011 - Z distinto
111 - tres iguales
```

Figura 23: Significado de la señal de salida "Error" del comparador en función del valor devuelto

Una vez obtenidos y situados estos documentos, se procede a la ejecución de la aplicación. En primer lugar, se invoca a la función "exploraArchivo" la cual recorre el fichero origen en busca del componente a triplicar. Una vez localizado este componente, identifica sus señales y las guarda en una lista, obteniendo sus características (como son el tipo, el tamaño del vector o si es de entrada o salida). Esta función está desarrollada de tal manera que es capaz de reconocer los datos a pesar de estar escritos de diversas maneras, como puede ser contener ";" al final de la línea, encontrarse la información de una misma señal en distintas líneas contiguas, e identificando el final de la definición ");" a pesar de estar en la misma línea que una señal normal o que una señal vectorial, aunque el paréntesis final de esta se encuentre junto a dicha expresión.

Después se llama a la función "creaArchivo" que lo primero que hace es incorporar la librería "IEEE" y "IEEE.STD_LOGIC_1164.ALL". Tras esto, genera la entidad añadiendo "_TMRtop" al nombre que tenía en la entidad origen y a continuación se define el "port" con las mismas señales que se han identificado anteriormente en la función "exploraArchivo". Esto se hace para que nuestro circuito de forma externa siga funcionando de forma individual, pero internamente lo haga de forma triplicada. A continuación se

procede a la definición de la arquitectura del circuito triplicado. Para ello se comienza definiendo las señales que el circuito va a necesitar. Estas señales son, una salida por cada rama del circuito triplicado (las cuales servirán de entrada al comparador), una salida de error (producida por el comparador) y la salida seleccionada por el comparador. Tras esto declara los componentes que usará el circuito, puesto que éstos están en ficheros externos. En este caso se declara un componente igual que el declarado en la entidad y otro del tipo comparador. Una vez hecho esto, se procede a la triplicación del componente. Para ello, se crean tres instancias del elemento origen, agrupando las respectivas salidas en vectores nombrados como "Salida1, Salida2 y Salida3", los cuales servirán de entradas al comparador. Tras crear estos elementos, se crea el último de los componentes, el comparador y se conectan sus entradas con las salidas anteriormente nombradas. Se puede ver un ejemplo de la arquitectura general resultante en la Figura 24.

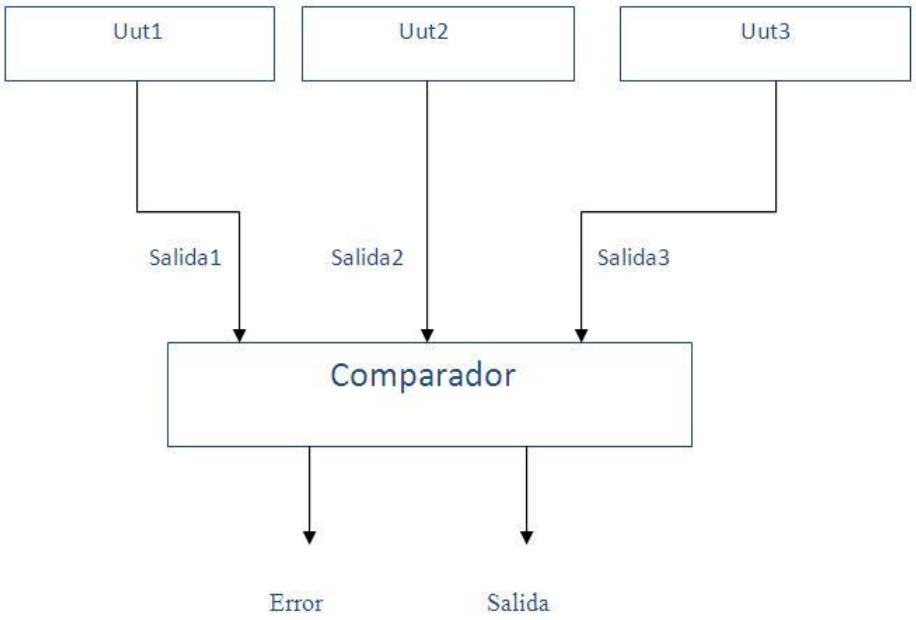


Figura 24: Esquema general de un circuito triplicado

Una vez finalizada la creación del fichero VHDL triplicado se comienza la generación del nuevo fichero UCF. Para ello, en primer lugar se invoca a la función "leerUcf", que se encarga de leer el documento de configuración del elemento adaptado original identificando el nombre del FlipFlop y sus coordenadas. Tras esto, se invoca a la función llamada "crearUCF". Dicha función genera un nuevo documento usando el nombre del fichero origen más "_Restricciones_Top.ucf". En este documento es donde se escribirá la localización de los elementos del circuito, pero en este caso de forma triplicada, es decir, una localización específica para cada uno de los elementos generados. En este caso se sigue manteniendo la restricción por la cual la coordenada Y no debe superar el valor de 159 y se tiene en cuenta una nueva, según la cual, si un componente termina en una coordenada X, la siguiente debe empezar en X+1, es decir, si un elemento termina en X = 61, el siguiente componente debe empezar en X = 62. Esto se muestra en la Figura 25.

```
INST "circuito_0/circuito_0/instancia/uut1/u_1" LOC=SLICE_X48Y140;
INST "circuito_0/circuito_0/instancia/uut1/stato_FSM_FFd3" LOC=SLICE_X49Y140;
INST "circuito_0/circuito_0/instancia/uut1/stato_FSM_FFd1" LOC=SLICE_X48Y141;
INST "circuito_0/circuito_0/instancia/uut1/stato_FSM_FFd2" LOC=SLICE_X49Y141;
INST "circuito_0/circuito_0/instancia/uut2/u_1" LOC=SLICE_X50Y140;
```

Figura 25: Ejemplo de UCF con cambio de variable "X" en el caso de un circuito triplicado

4.3 - Modifica TB

El fichero Testbench es el encargado de pasarle los valores a las señales de nuestro circuito durante le ejecución del programa Nessy.

Esta aplicación tiene como objetivo crear un nuevo fichero ".txt" partiendo de otro ya existente. El original contiene los valores que tomarán las señales de nuestro circuito. Será necesario añadirle un valor adicional, la señal Set_Nessy. Este valor será un cero y se coloca en la segunda posición empezando por la izquierda, puesto que es el propio programa Nessy quien tomará el valor de la segunda posición para dicha señal. Es importante destacar que, por defecto, este programa asigna un valor de "0" a la nueva señal Set_Nessy. Sin embargo, a la hora de realizar una inyección de errores, Nessy elige de manera aleatoria un ciclo de reloj donde se inyectará el error, y activa a "1" la entrada Set_Nessy de la línea del Testbench correspondiente. De este modo, el usuario no tiene que interactuar directamente con esta señal en el Testbench. Es Nessy quien se encarga de ello de una manera totalmente transparente.

En primer lugar la aplicación recibe el nombre del fichero Testbench.txt original y un argumento, el cual si es "derecha" significará que se debe introducir el nuevo valor por la derecha, en cualquier otro caso se añadirá por la izquierda.

El programa comienza creando un nuevo archivo con el mismo nombre que el original, añadiéndole "_nessy_izquierda.txt" o "_nessy_derecha.txt" dependiendo por dónde le añadamos el nuevo valor. A continuación el programa procederá a insertar un cero en todas las líneas del fichero en la posición indicada previamente.

Capítulo 5: Resultados obtenidos

Para poder comparar la diferencia en vulnerabilidad entre el circuito original y el circuito con nuestras modificaciones, se han utilizado doce circuitos (obtenidos del conjunto de ficheros de test contenidos en "ITC'99" [CMS00]). Para cada uno de ellos se ha creado un proyecto de ISE y una simulación del Testbench Nessy y de esta forma ir comparando que ambas salidas, antes de ser modificadas, obtenían los mismos resultados. Para ello, en primer lugar se elaboró un fichero Testbench con la configuración de los ficheros originales VHDL y se guardó una imagen obtenida de los resultados de la simulación. Una vez almacenadas estas imágenes, se procedió a generar los ficheros estructurales utilizando la opción "*Generate Post Synthesis Simulation*" de ISE y como consecuencia, su correspondiente fichero Testbench. De igual manera, se guardaron las imágenes de las simulaciones y se compararon con las de los ficheros originales, obteniendo algunas diferencias entre algunos de los circuitos. Un ejemplo de esto es el problema que se encontró en el fichero "b05.vhd", el cual contenía la instrucción "TM := -MAX mod 2**5;" a la cual se le aplicó la siguiente modificación "TM := (-MAX) mod 2**5;". Esto fue debido a que la instrucción original generaba un valor de TM erróneo, al devolver como negativo el resultado de la operación en vez de cambiarle el signo a MAX (cuando éste es negativo) antes de realizar la operación. Los resultados obtenidos antes de modificar esta instrucción se muestran en la Figura 26.

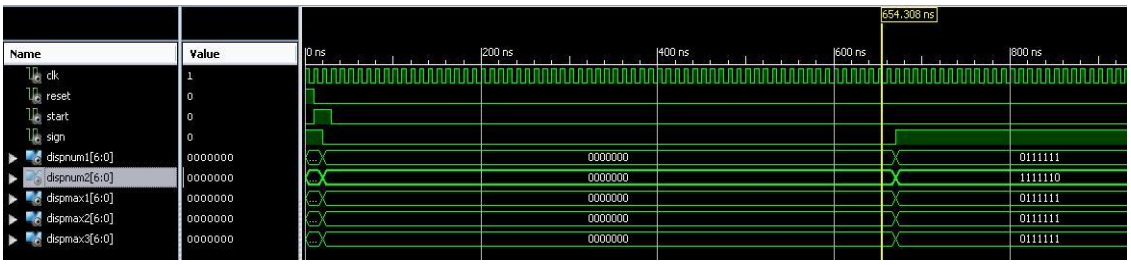


Figura 26: Testbench del circuito b05 cuando su salida era errónea

Se observa que la variable DISMAX3 tiene un cierto valor. A la hora de implementar el sintetizado y ver que el valor de las salidas no correspondía con el obtenido con anterioridad, se revisó el código para buscar dónde se producía ese valor erróneo. Tras la modificación de esa instrucción se obtuvo el resultado correcto de la variable, el cual se puede ver en la Figura 27.

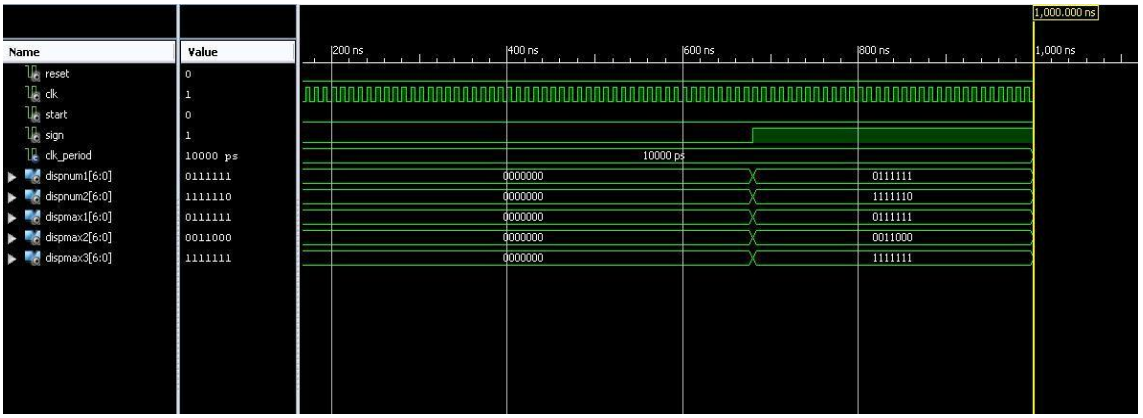


Figura 27: Salida del circuito b05 cuando su salida era correcta

Además de estos problemas se tuvieron diversas complicaciones con algunos de los ficheros (como puede ser el b07, b08, b12…) puesto que el VHDL no usaba el estándar establecido en el IEEE. Esto condujo a corregir primero estos errores antes de poder proceder con la ejecución de nuestras aplicaciones.

Una vez que todas las simulaciones fueron correctas, se introdujo cada fichero estructural en nuestro programa "Adaptador VHDL" para generar los nuevos ficheros adaptados con sus correspondientes ficheros de restricciones UCF.

Tras esto, se generaron unos nuevos Testbench que se adaptarían a los nuevos ficheros y de este modo poder obtener las nuevas simulaciones.

Una vez solucionados los problemas, se procedió a introducir el fichero VHDL y su correspondiente UCF por nuestro programa "Triplicador VHDL" y de este modo se generaron los ficheros triplicados con sus correspondientes ficheros UCF. Tras haber obtenido estos ficheros, se pasó a crear los proyectos en la aplicación Nessy (plataforma explicada en el Capítulo 2), primero los adaptados y después los triplicados. Una vez generados de forma correcta los proyectos, se procedió a la generación del Golden, el cual nos tenía que devolver los mismos resultados que las imágenes de las simulaciones obtenidas anteriormente, como se puede observar en las Figuras 28 y 29.

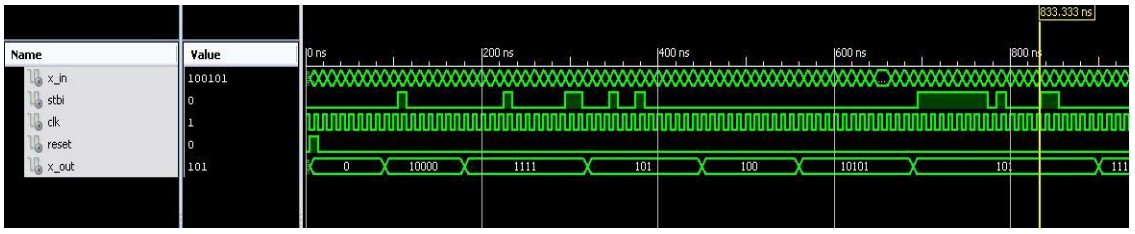


Figura 28: Salidas Testbench del circuito b11_Adaptado

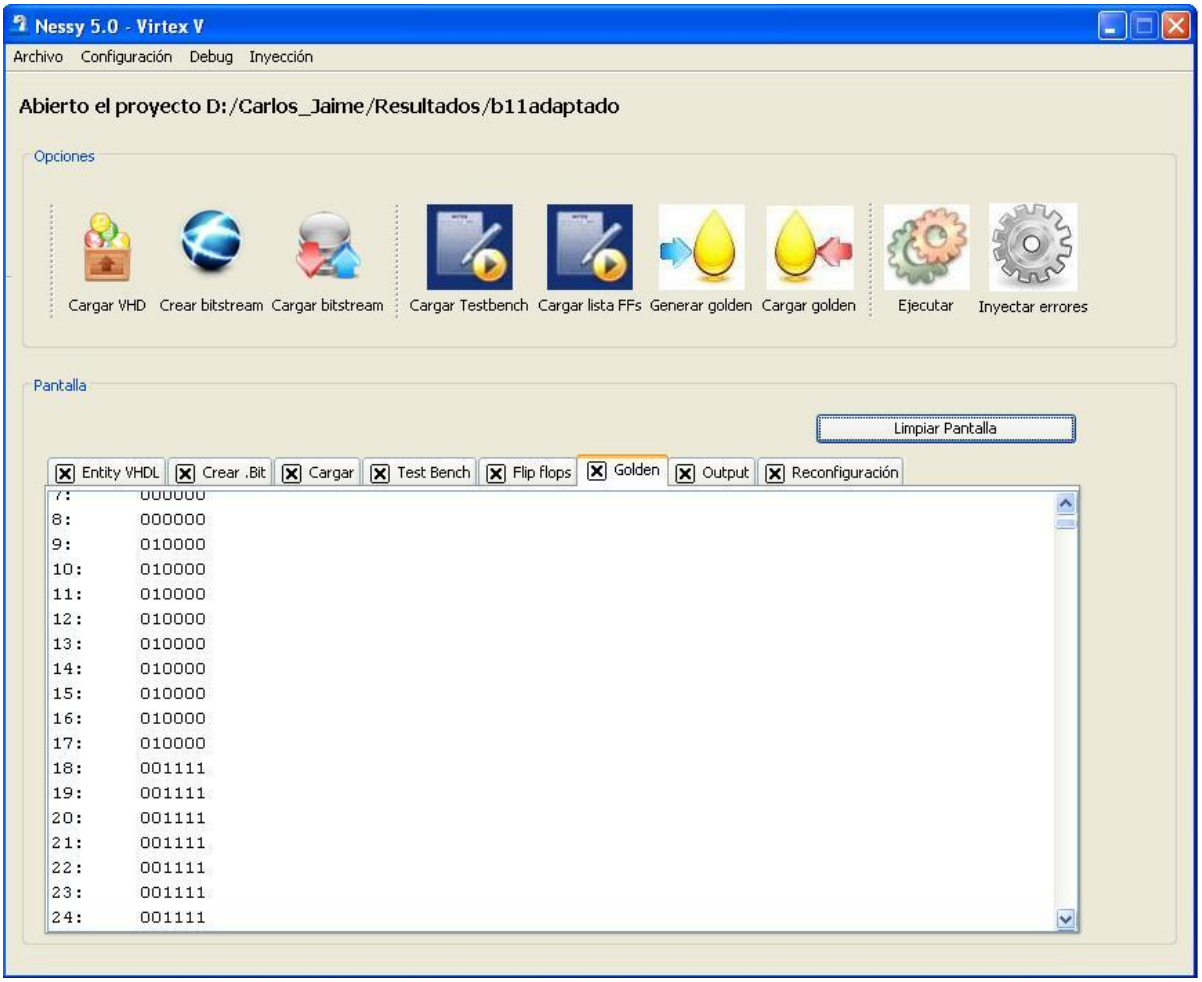


Figura 29: Golden del circuito b11_Adaptado

En este punto se encuentran una serie de problemas, puesto que o bien al generar el bitstream se producían errores, o bien la generación del Golden era errónea y no coincidía con el de las simulaciones anteriores. Esto era debido a una serie de problemas:

- Inexistencia del bit de configuración de la señal Set_Nessy en el fichero Testbench:

  Para solventar este problema, se creó una nueva aplicación llamada "ModificaTB", que insertaba los valores que tomará dicha señal durante la ejecución.

- El orden de las señales en la declaración de los componentes del fichero VHDL:

  En primer lugar se debe situar la señal Reset, puesto que la aplicación Nessy siempre la toma de esta manera sin identificarla previamente. En caso de no situarse la primera, la señal que se encuentre en ese lugar será utilizada como señal Reset, lo cual implicará un funcionamiento erróneo con los consecuentes resultados incorrectos. Una vez detectado este error, es necesario cambiar este orden en el fichero origen y volver a generar todos los documentos desde un principio, incluido el proyecto en Nessy.

- El orden de los bits de configuración en el fichero Testbench:

  Repetidas veces se encuentra en la situación en la que el orden de los valores de las señales establecidas en el fichero Testbench no coincidía con el orden de la declaración de las señales. Como consecuencia de esto, las señales tomaban valores incorrectos, lo que provocaba unas salidas erróneas.

A pesar de esto, el mayor de los problemas que se encontró fue el tiempo que tarda tanto en generarse el bitstream como la inyección de errores, puesto que los errores de cada "bXX" no se pudieron detectar hasta que estas acciones se completaban, lo cual suponía un gran período de tiempo debido a que algunas tardaban alrededor de una hora sólo en una de las acciones anteriormente nombradas, como se puede observar en la Figura 30.

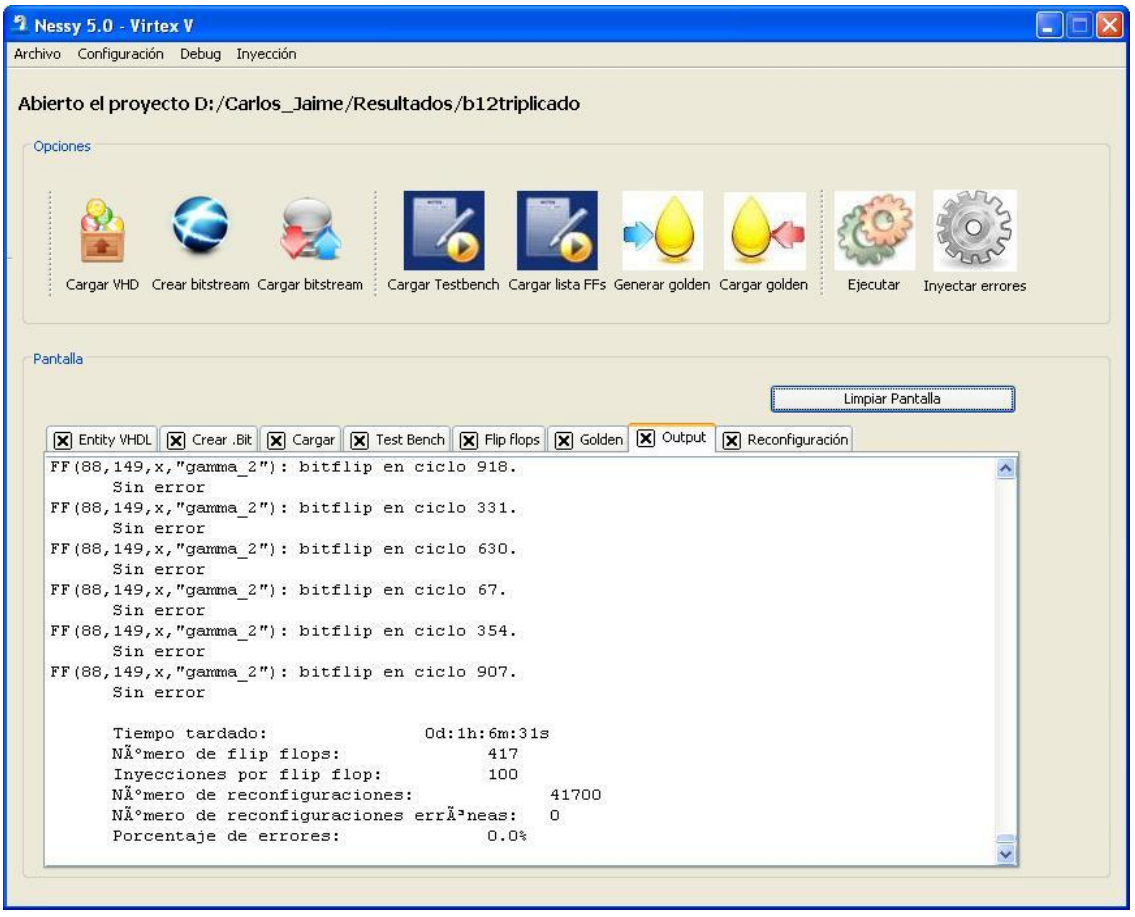


Figura 30: Resultado de una inyección de errores del circuito b12_Triplicado en Nessy

Una vez obtenido un Golden cuyo resultado era igual que las imágenes de las simulaciones anteriores, se procedió a la "Inyección de errores" para obtener qué porcentaje de errores hay en un circuito antes de nuestra protección y tras ella. Un ejemplo de esto es mostrado en la Figura 31.

```
Tiempo tardado:                 Od:Oh:Om:38s
NÃ°mero de flip flops:                4
Inyecciones por flip flop:           100
NÃ°mero de reconfiguraciones:                400
NÃ°mero de reconfiguraciones errÃ³neas:    180
Porcentaje de errores:               45.0%
```

Figura 31: Resultado de inyección de errores del circuito b02_Adaptado (sin triplicar)

Tras haber obtenido estos resultados, se procedió a la generación de los proyectos para los circuitos triplicados, para los cuales se encontraron los mismos problemas que los nombrados anteriormente. Y se obtuvo que los resultados de las inyecciones en los circuitos triplicados eran de un 0% de errores frente a los diversos errores anteriores, lo que significaba que el método de la triplicación funcionaba de forma correcta. Un ejemplo de esto se puede ver en la Figura 32.

```
Tiempo tardado:                 Od:Oh:1m:52s
NÃ°mero de flip flops:                12
Inyecciones por flip flop:           100
NÃ°mero de reconfiguraciones:                1200
NÃ°mero de reconfiguraciones errÃ³neas:    0
Porcentaje de errores:               0.0%
```

Figura 32: Resultado de inyección de errores del circuito b02_Triplicado

Se observa que el tiempo de ejecución del proceso de inyección de errores sobre el circuito sometido al método de la triplicación es prácticamente tres veces el tiempo de inyección del circuito adaptado, como se podía esperar.

Los porcentajes de errores obtenidos al efectuar las inyecciones sobre la placa Virtex 5 utilizando la aplicación Nessy fueron los ilustrados en la Figura 33.

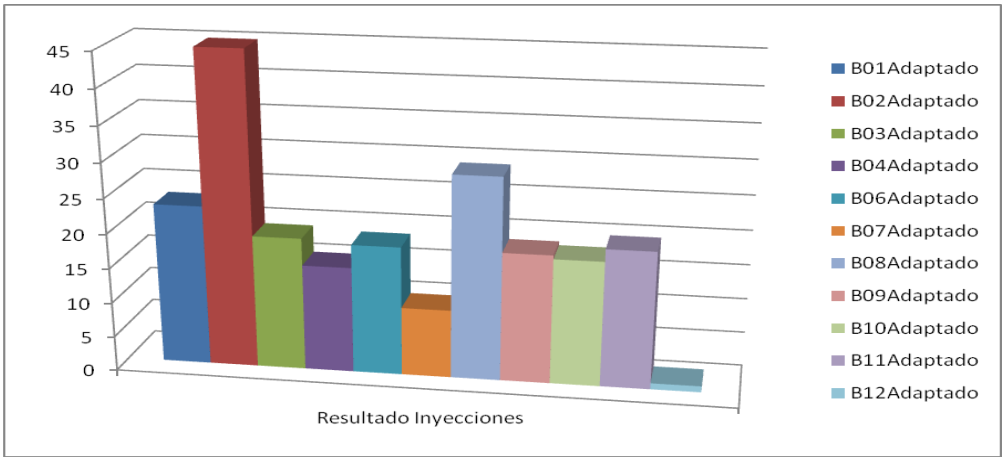


Figura 33: Gráfico resultado de inyecciones de todos los FlipFlops en los circuitos b01-b12

En todos los casos se pude observar que al aplicar el método de la triplicación, el nuevo resultado de las inyecciones es de un 0% de errores.

Capítulo 6: Conclusión

Podemos concluir que gracias al método de la triplicación obtenemos un gran sistema de seguridad puesto que vemos reducido el porcentaje de errores a un 0%, aunque debido a esto, también triplicamos el espacio físico necesario para la ubicación de los nuevos elementos, el consumo de los mismos y el tiempo necesario para realizar la inyección.

Por ello recomendamos utilizar este sistema de seguridad en todos aquellos dispositivos que vayan a ser trasladados a grandes alturas, lanzados al espacio o puedan llegar a tener colisiones o interferencias con partículas espaciales o rayos cósmicos, ya que una colisión de estos elementos en uno de los dispositivos podría modificar el comportamiento de sus circuitos y verse afectadas su configuración y sus salidas y no poder realizar las funciones para las que estaban diseñados.

En proyectos futuros, proponemos la búsqueda de métodos por los cuales no sea necesaria la triplicación de todos los elementos del circuito, sino solo aquéllos que supongan un grave riesgo para el correcto funcionamiento del dispositivo.

We can conclude that thanks to the triplication method we get a great security system since we have reduced the error rate to 0%, however because of this, we tripled the physical space required for the location of the new elements, their any energy consumption and the fault injection time.

Thus, we suggest using this security system in device that will be moved to great heights, launched to the space or when they may be affected by ac collision particle or cosmic rays. The reason is that a collision of the elements on the devices can modify their behaviour, as well as affect their settings and their outputs, there modifying their operation.

In the future, we propose the search for methods that do not necessarily involve the triplication of all the circuit elements, but only those that especially critical for the correct operation of the device.

Capítulo 7: Aportación de Carlos Cabañas García en el proyecto

En este proyecto he llevado a cabo una serie de tareas a la par con mi compañero. En primer lugar comenzamos creando la aplicación “Adaptador VHDL”, la cual obtiene un fichero VHDL sintetizado y genera dos ficheros nuevos. Uno de ellos es el VHDL con las modificaciones necesarias para inyectar errores en el circuito y otro con la localización espacial de los elementos en la placa. En este punto comencé yo creando el proyecto (debido a que nos dieron libertad sobre qué herramienta utilizar, seleccionamos Eclipse, ya que es una herramienta familiar y cómoda), comenzando a definir la estructura de la aplicación y creando las funciones de abrir, crear y cerrar los ficheros. Una vez hecho esto y junto a mi compañero, comenzamos a crear la función “exploraArchivo”, la cual forma una parte fundamental y muy importante de esta aplicación. Esta función requirió mucha dedicación puesto que es la encargada de leer el fichero origen, identificar las señales y guardar sus valores para un posterior uso. Una vez estaba prácticamente terminada esta función, mi trabajo consistió en terminar de matizarla en lo que mi compañero comenzaba la siguiente. Estos matices consistían en otorgar a esta función la posibilidad de reconocer las señales a pesar de estar escritas de diversas maneras, como puede ser en líneas contiguas, con la marca de fin de línea “;” junto al nombre (la cual había que eliminar para un posterior uso de ese nombre) o con el nombre de la señal junto con su valor, como por ejemplo en “clk:in STD_LOGIC := 'X';” donde el parseador reconocía la cadena “clk:in” como una sola, lo cual provocaba errores. Una vez finalizado esto, me incorporé a la función que estaba creando mi compañero, “generaArchivo”, que es la encargada de generar y escribir el nuevo fichero ya adaptado.

Una vez acabada esta función, y mientras mi compañero hacía la parte de generar el fichero UCF, comencé a llevar a cabo las modificaciones en Nessy para poder efectuar las inyecciones en FlipFlops. Para ello, en primer lugar hice las modificaciones necesarias en la interfaz añadiendo un nuevo botón y cuando mi compañero terminó, juntos efectuamos las modificaciones necesarias en el sistema para que se pudiese llevar a cabo esta inyección.

Cuando ya teníamos la herramienta preparada, el siguiente paso fue simular los circuitos adaptados y comprobar que su salida era correcta. Para ello, utilizando los ficheros que mi compañero ya había creado pasando los ficheros originales por nuestro programa, fuimos creando uno a uno los proyectos necesarios para poder simular los circuitos, adaptando cada uno de estos a la nueva configuración. Cuando obtuvimos los resultados de éstos, observamos que las salidas de algunos de los circuitos adaptados no coincidían con los originales y comenzamos a buscar los errores. Mi compañero buscó los errores de unos y yo de otros, aunque nos ayudamos mutuamente para solventar esto con mayor brevedad.

Por otro lado, mientras terminábamos de solucionar estos problemas, comencé a generar el proyecto para la nueva aplicación, “Triplicador VHDL”, que parte de los ficheros obtenidos del “Adaptador VHDL” para generar otros dos ficheros nuevos. Uno de ellos es el VHDL que utiliza la estructura del componente adaptado para crear tres instancias de él

y conectarlas a un comparador. De este modo buscamos aplicar un método de seguridad basada en triplicación. El otro fichero es el UCF con la nueva localización espacial de los nuevos elementos. Para esta aplicación reaprovechamos parte del código de la aplicación anterior.

Una vez logramos solventar todos los problemas que encontramos con los resultados de las simulaciones de los circuitos adaptados, mi compañero comenzó a definir la función de parseo de esta aplicación mientras yo creaba el archivo "comparador.vhd", que contiene la definición de un comparador que será utilizado en el sistema de seguridad a implementar. Cuando terminé esto, me incorporé al trabajo de mi compañero y continuamos haciendo la nueva función llamada "creaArchivo", la cual genera el nuevo fichero definiendo las estructuras necesarias para llevar a cabo el sistema de seguridad ya mencionado. Una vez finalizado esto, pasamos a modificar la función que generaba el fichero UCF para que generase las nuevas localizaciones en función de las que ya tenía el fichero original y basándose en una serie de requisitos especificados.

Cuando terminamos esta aplicación y comprobamos que generaba de forma correcta los nuevos archivos, comenzamos las pruebas en Nessy. Para ello generamos un proyecto para cada circuito "bxx_adaptado" y "bxx_triplicado". Esto nos llevó mucho tiempo, puesto que tanto la generación del bitstream como la inyección de errores (no permite realizar dos inyecciones a la vez puesto que utilizamos siempre una única placa) necesitaban bastante tiempo para llevarse a cabo. Debido a esto, fuimos pasando por el despacho de manera regular para dejar generando e inyectando nuevos proyectos y además, tuvimos que dejarlo varios fines de semana y conectarme yo desde mi ordenador (gracias a la aplicación TeamViewer) de forma regular para ir renovando las tareas (los que ya estaban generados a inyectar y los inyectados se cerraban para generar nuevos proyectos). Todo este proceso también nos supuso problemas puesto que no todos los ficheros cumplían el estándar establecido en el IEEE y nos daba problemas en la creación de los bitstream. Debido a ello, mi compañero se centró en solucionar los problemas existentes en unos ficheros mientras yo me encargaba de los otros. Una vez solucionado esto, detectamos nuevos problemas debido al orden de las señales en su definición y debido a la inexistencia del bit de configuración de la nueva señal Set_Nessy en fichero "TB.txt". Esto provocaba que las salidas de los Golden de la aplicación Nessy (que tenían que ser iguales que en la simulación de los circuitos originales, y la de los circuitos adaptados) fuesen erróneas, con lo que el resultado obtenido en la inyección de errores carecía de sentido.

Debido a este problema, y sabiendo que cada fichero contiene mil líneas, nos surgió la necesidad de crear una nueva aplicación que introdujera el bit inexistente en estos ficheros. Esta es la aplicación llamada "Modifica TB" y es una simple aplicación que nos hizo este trabajo de forma rápida. Además, sabiendo que esta señal puede ser situada en la segunda columna bien empezando por la izquierda o bien empezando por la derecha, mi compañero añadió un parámetro de entrada a la aplicación de forma que aparte de introducir el nombre del fichero original, se determina por qué lugar se insertara el mismo.

Debido a estos problemas llegamos a dos situaciones, si el problema estaba en el orden de las señales en el fichero VHDL, el proyecto debía ser generado de nuevo (con su respectivo coste de tiempo), si por el contrario el error estaba sólo en el fichero con los bits

de configuración, podíamos mantener el bitstream pero debíamos generar un nuevo Golden y una nueva inyección. Para estas tareas tuvimos que ir organizándonos (según la disponibilidad de cada uno) de modo que fuésemos poniendo a generar aquellos proyectos fallidos.

Cuando todos los proyectos fueron generados de forma correcta (todas las salidas eran correctas antes de las inyecciones, es decir, coincidían con las obtenidas anteriormente) pudimos comprobar que con el nuevo método obteníamos un muy buen resultado en cuanto al porcentaje de errores (del 0%) y comenzamos con la documentación del trabajo realizado, plasmándolo en la memoria.

Capítulo 8: Aportación de Jaime Rodríguez Carmona en el proyecto

El proyecto empezó con la tarea de hacer un código que adaptara los ficheros VHDL para que Nessy pudiera inyectar errores en ellos. Esta parte al ser la primera y única que teníamos por aquel entonces la hicimos los dos juntos, aunque fue mi compañero el que comenzó creando el proyecto. Como nos dieron libertad a la hora de elegir la plataforma de desarrollo de la aplicación, elegimos Eclipse que ya lo conocíamos y nos es fácil trabajar con ella. Una vez que teníamos casi terminada la función "exploraArchivo", función muy larga y compleja, mi compañero se encargó de pulir los detalles de ésta y yo empecé a desarrollar la función "generaArchivo". Por lo general, cuando uno terminaba se ponía con otra cosa para ir avanzando, aunque claro está, nos ayudábamos mutuamente cuando nos quedábamos bloqueados.

Una vez que terminé con esa función me puse con la función que se encarga de generar el fichero UCF, mientras mi compañero se encargaba de aplicar modificaciones sobre Nessy para poder incorporar nuestra aplicación, modificando la interfaz y demás. Una vez que terminé con la parte de generar los UCF ayudé a mi compañero a terminar de hacer las modificaciones necesarias en Nessy para que se pudiese llevar a cabo la inyección de errores. Los ficheros hay que sintetizarlos antes de hacer la inyección de errores.

Una vez terminada la aplicación y tener listo Nessy, nos pusimos a simular los circuitos y comprobar las salidas para comprobar si eran las correctas. Al principio dieron errores, en ese momento mi compañero y yo decidimos dividirnos estos errores para tratar de resolverlos con la mayor brevedad posible. A la hora de solucionar estos problemas tuvimos que consultarnos muchas cosas ya que implicaba modificar nuestra aplicación.

Mientras yo terminaba de solucionar los errores obtenidos tras las simulaciones, mi compañero empezó con el código de nuestra segunda aplicación, "Triplicador VHDL". Como siguieron dándonos libertad a la hora de elegir la plataforma, decidimos continuar con Eclipse. Cuando terminé con esos errores me puse a ayudarle con el código del Triplicador. Igual que en la aplicación anterior, en esta parte del desarrollo nos dividimos las tareas, mi compañero hacía unas funciones y yo iba haciendo otras. Algunas de estas funciones las obtuvimos modificando un poco las hechas en la aplicación anterior.

Una vez terminadas las dos aplicaciones comenzamos las pruebas en Nessy. Para ello generamos un proyecto para cada circuito "bxx_adaptado" y "bxx_triplicado". Las pruebas en Nessy nos llevaron mucho tiempo, ya que en generar el bitstream tardaba muchísimo, lo que nos hacía pasar por el despacho con mucha frecuencia para comprobar si habían terminado o no. En este punto mi compañero uso TeamWiever, para poder manejar el ordenador en el que se estaba ejecutando Nessy desde el suyo. Esto nos permitió ganar mucho tiempo ya que podíamos dejarlo en funcionamiento los fines de semana.

Como no todos los ficheros cumplían el estándar establecido en el IEEE, nos daban problemas a la hora de crear los bitstream. En este punto mi compañero se encargó de modificar unos ficheros mientras yo modificaba otros. Al terminar con estos errores nos

dimos cuenta que los Golden se generaban de forma incorrecta por que les faltaba un valor para la señal Set_Nessy. Para que funcionaran de forma correcta había que modificar (incorporando un valor más) todas las líneas de todos los ficheros "TB.txt". Como la modificación había que hacerla para todos los ficheros nos generamos una aplicación, también en Eclipse, la que recibe un fichero ".txt", devuelve otro prácticamente igual que el anterior pero con un valor más (el cual puede ser introducido por la derecha o por la izquierda, según se indique) que será para la señal Set_Nessy y siempre será "0". Otro de los errores que encontramos es que en algunos ficheros el orden de las entradas no era el correcto (la señal Reset tenía que ser la primera), por ello tuvimos que colocar las señales en el orden correcto y volver a ejecutar todos los pasos otra vez.

Cuando los proyectos fueron generados de forma correcta, la salida antes de la inyección coincidía con la generada en el Golden, pudimos comprobar que con el nuevo método se obtenía un buen resultado, un porcentaje de errores del 0%.

Al hacer la memoria, hemos trabajado prácticamente juntos. Casi siempre en un mismo ordenador, pensando los dos en alto y escribiendo lo que nos parecía más correcto. Aun así, también hemos hecho parte de la misma en casa, cada uno ponía lo que le parecía en el documento y luego se hablaba si al otro le parecía bien.

Apéndice 1

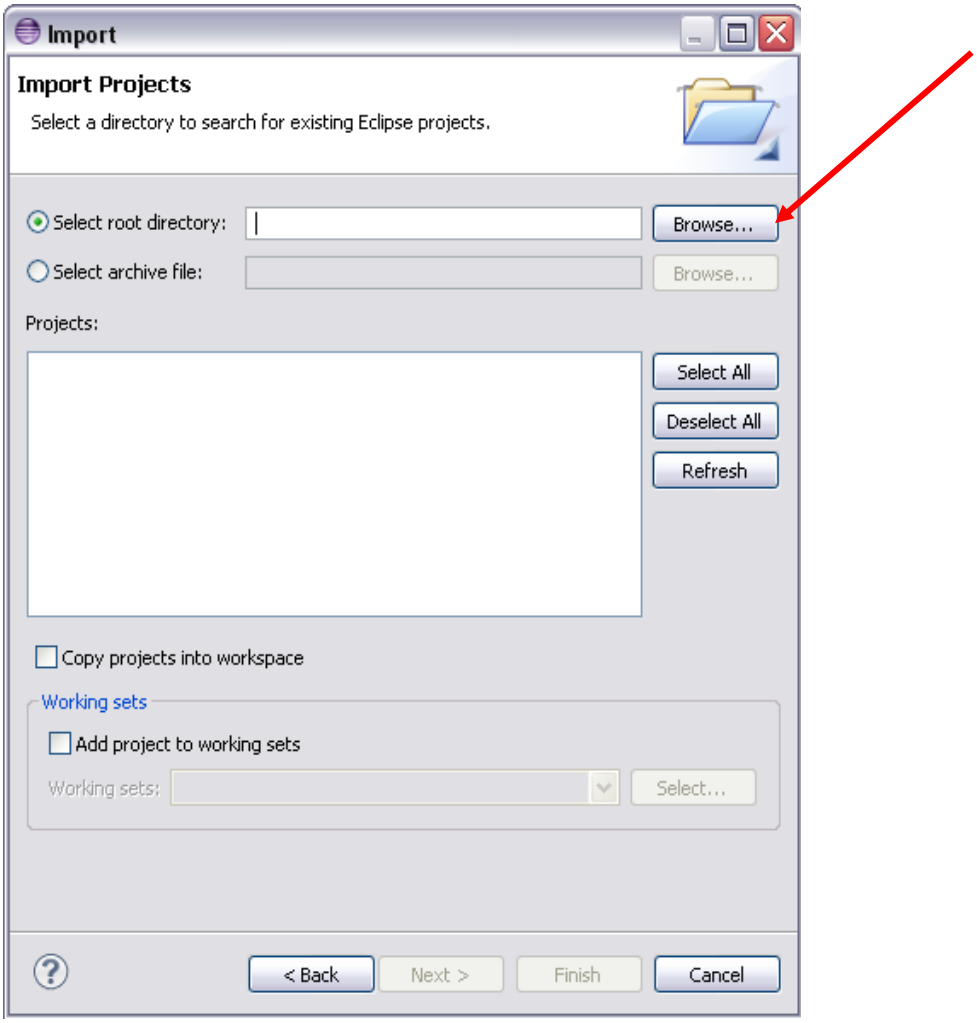
Manual Adaptador VHDL

1- Abrir Eclipse.

2- Hacer click en *File->Import…*

3- Desplegar la carpeta *general* y seleccionar *existing proyects into workspace*, pulsar *next*.

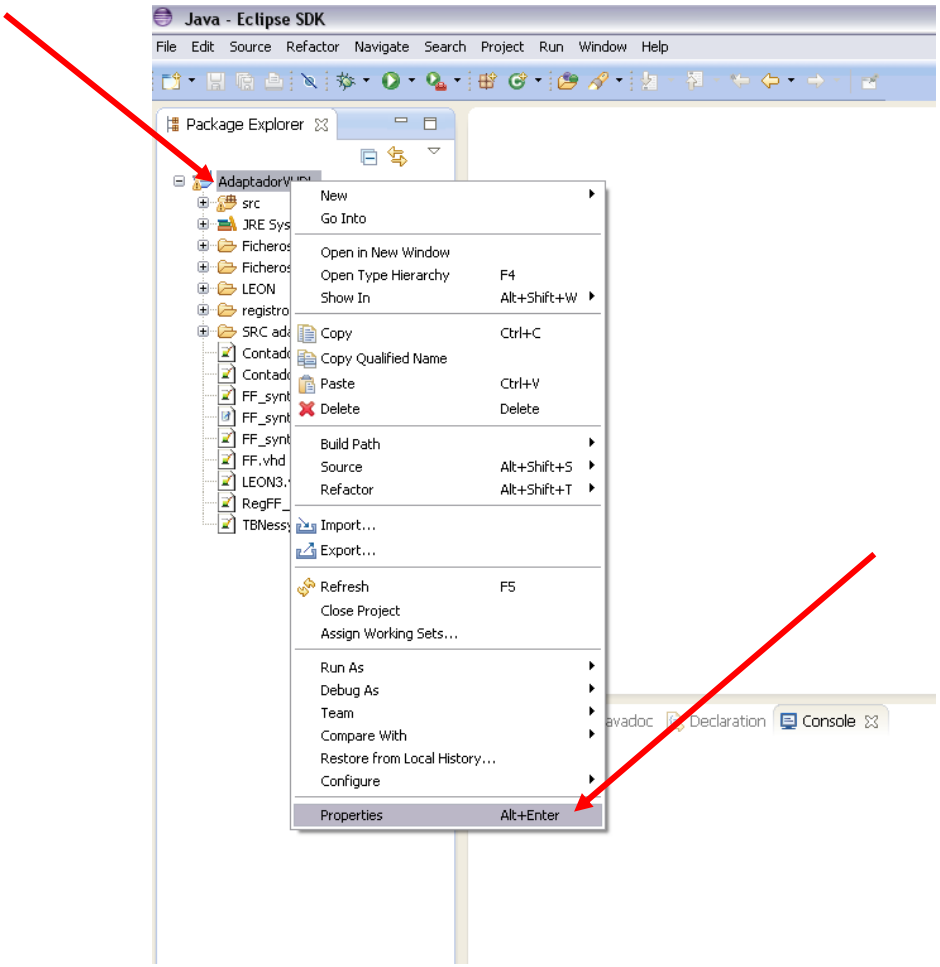
4- Aparecerá esta pantalla:



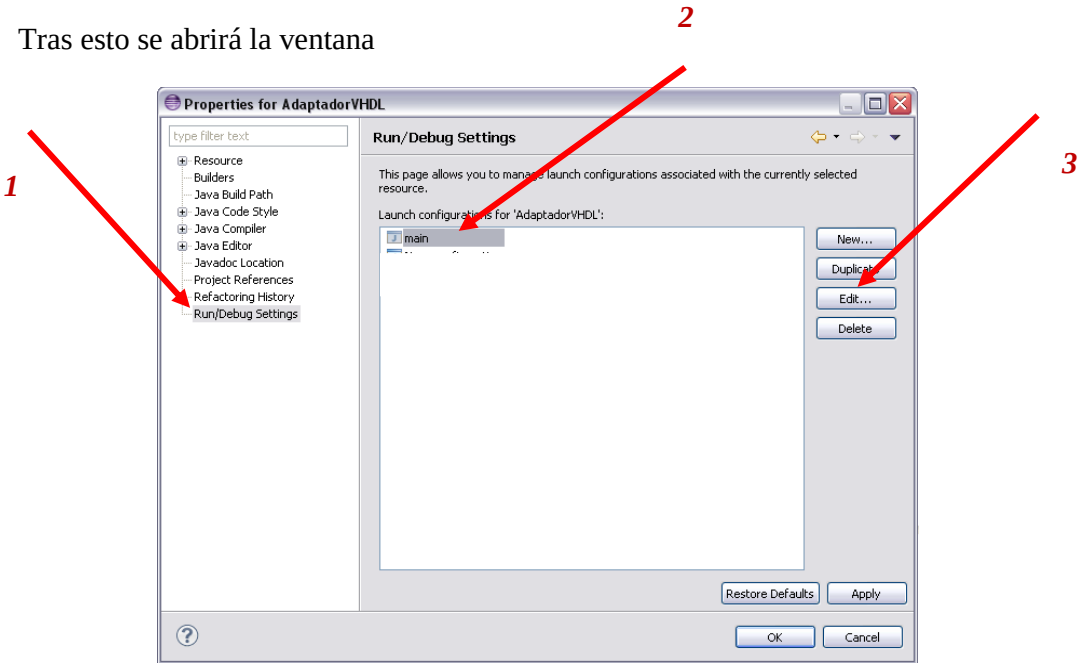
Pulsar *Browse* y buscar y seleccionar la carpeta contenedora de la aplicación y pulsar *finish*.

5- Introducir en la carpeta en la que se encuentra la aplicación el fichero VHDL sintetizado bxx_synthesis.vhd.

6- Hacer *click* derecho en la carpeta contenedora del proyecto en Eclipse, y seleccionar la opción *Properties*.

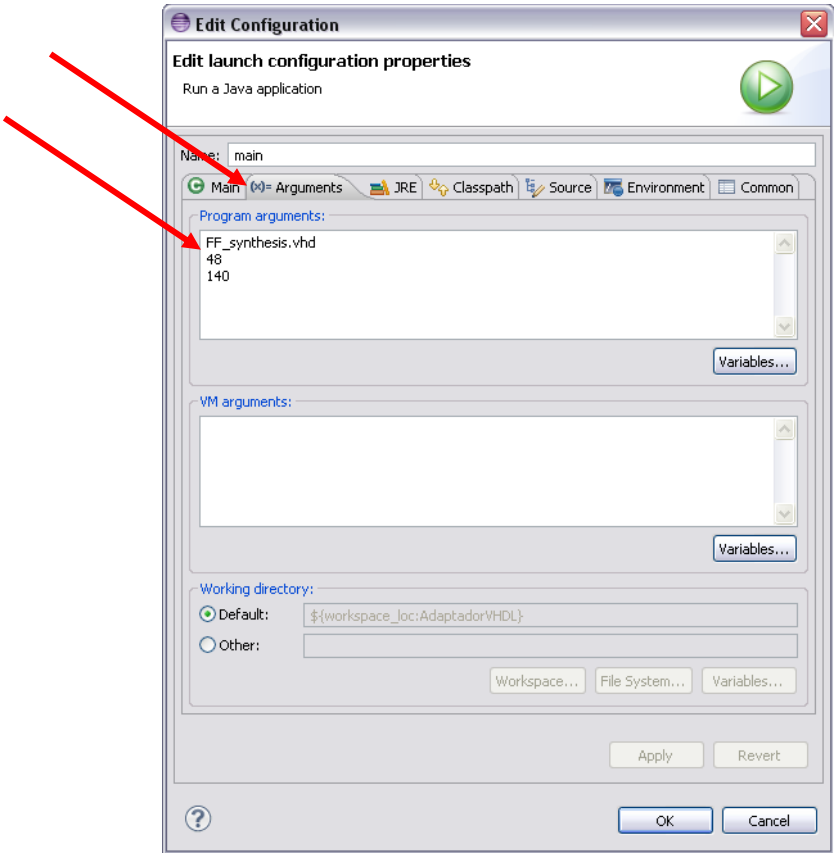


7- Tras esto se abrirá la ventana



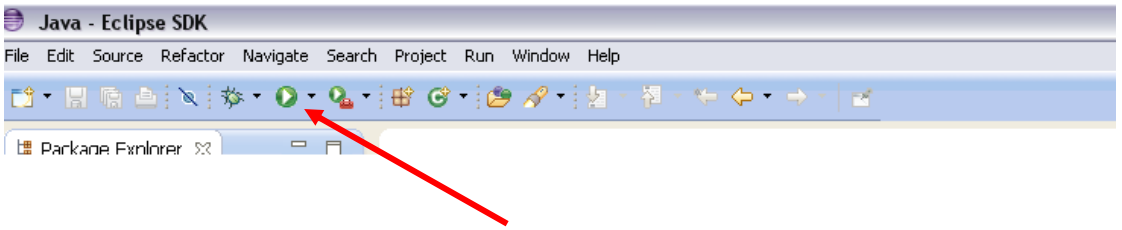
Seleccionamos *Run/Debug Settings*, después seleccionamos la configuración *Main*, y pinchamos en *Edit*…

8- Se abrirá la siguiente ventana:



Seleccionamos la pestaña *Arguments* e introducimos el nombre del fichero origen. Seleccionamos OK.

9- Por último iniciamos la ejecución del programa pulsando:



10- Al terminar la ejecución de la aplicación generara dos ficheros nuevos: bxx_synthesis_Adaptado.vhd y bxx_synthesis_Restricciones.ucf.

Apéndice 2

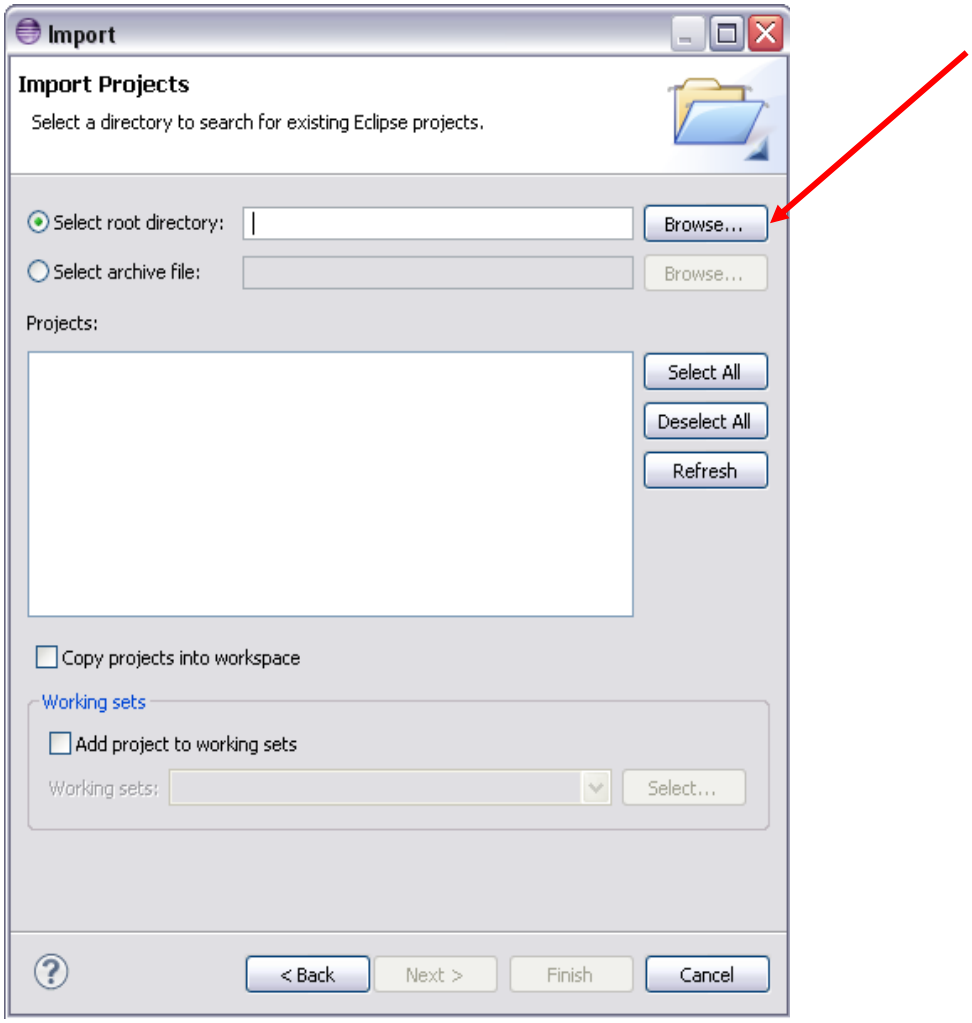
Manual Triplicador VHDL

1- Abrir Eclipse.

2- Hacer *click* en *File->Import…*

3- Desplegar la carpeta *general* y seleccionar *existing projects into workspace*, pulsar *next*.

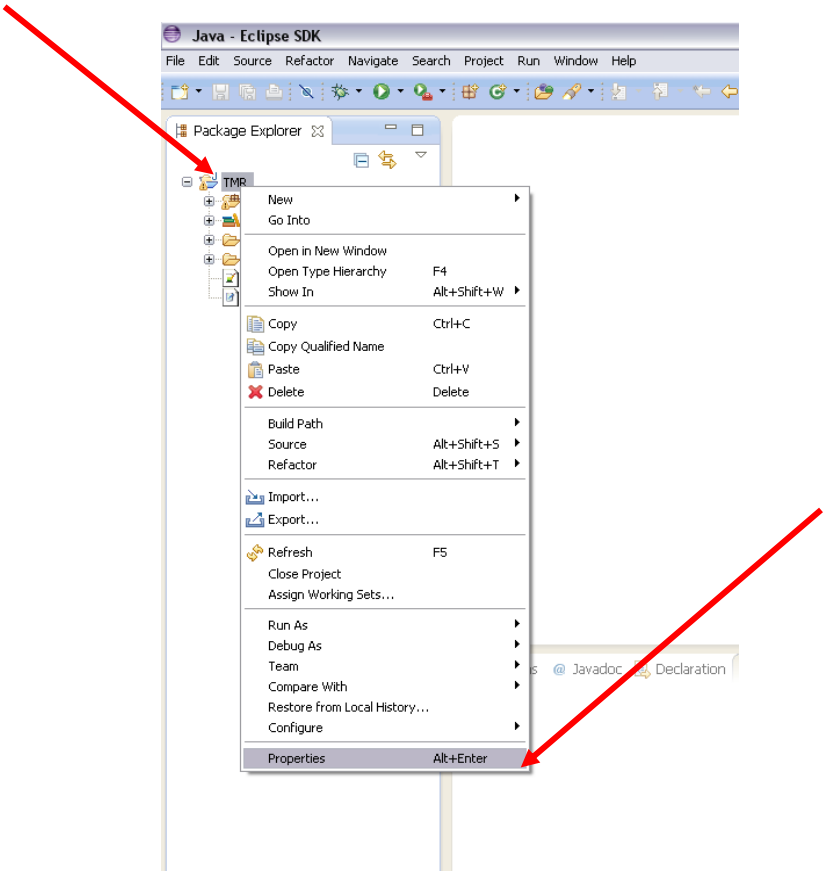
4- Aparecerá esta pantalla:



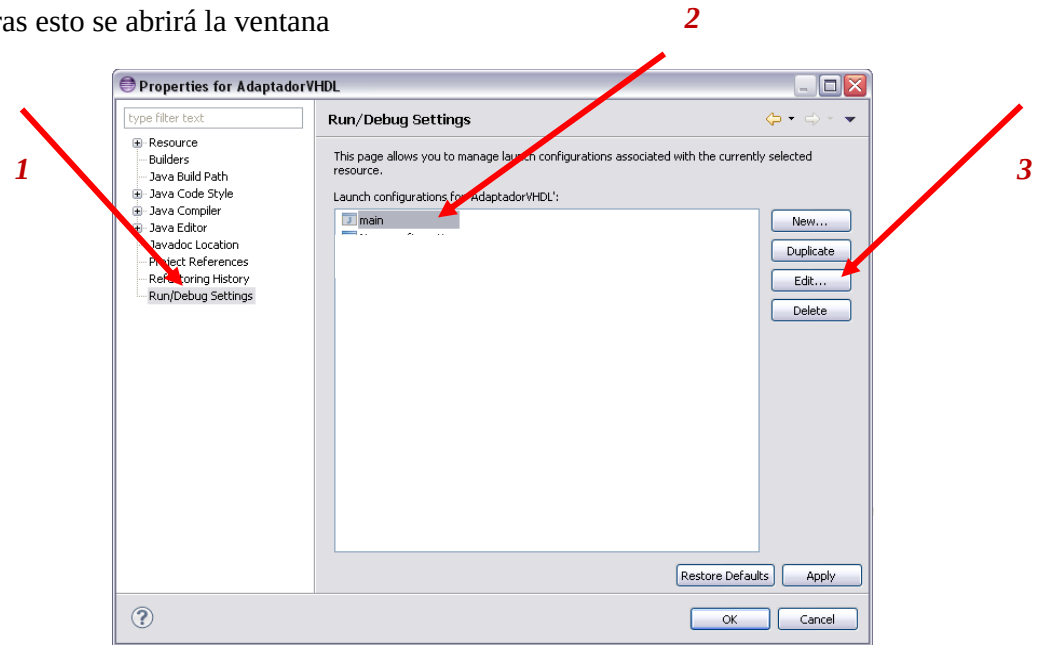
Pulsar *Browse* y buscar y seleccionar la carpeta contenedora de la aplicación y pulsar *finish*.

5- Introducir en la carpeta en la que se encuentra la aplicación los ficheros VHDL adaptado sintetizado bxx_synthesis_adaptado.vhd y el fichero UCF bxx_synthesis_Restricciones.ucf.

6- Hacer *click* derecho en la carpeta contenedora del proyecto en Eclipse, y seleccionar la opción *Properties*.

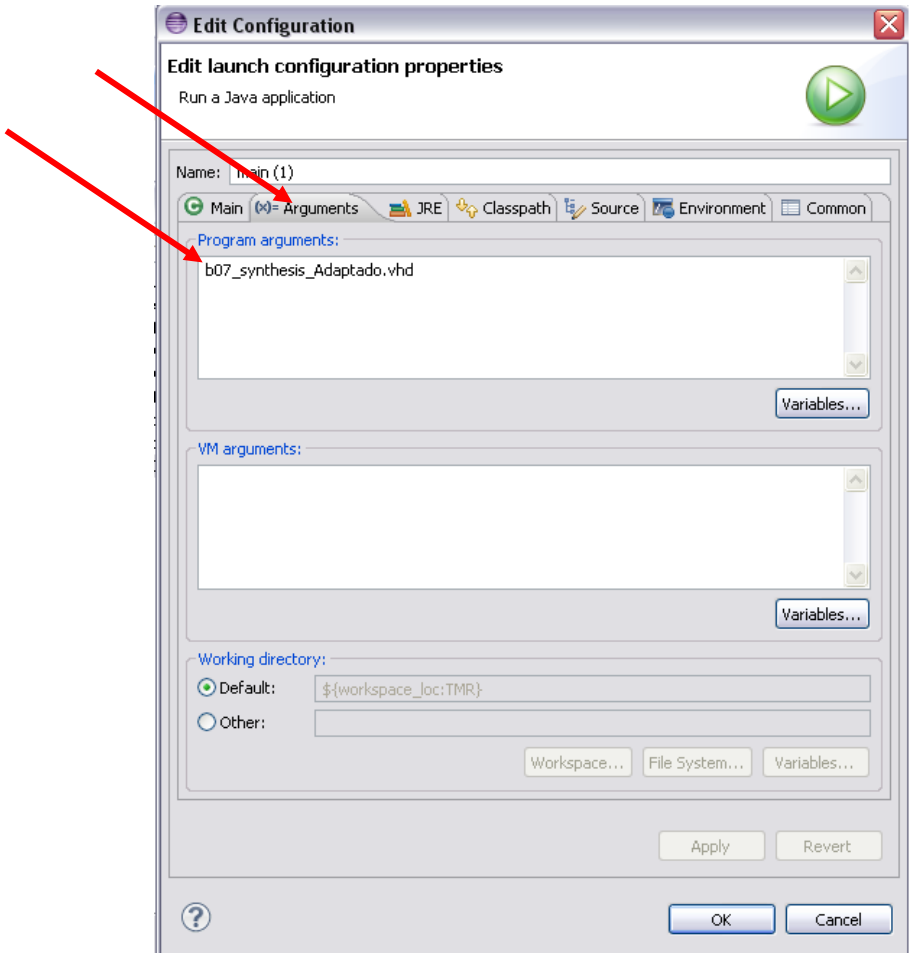


7- Tras esto se abrirá la ventana



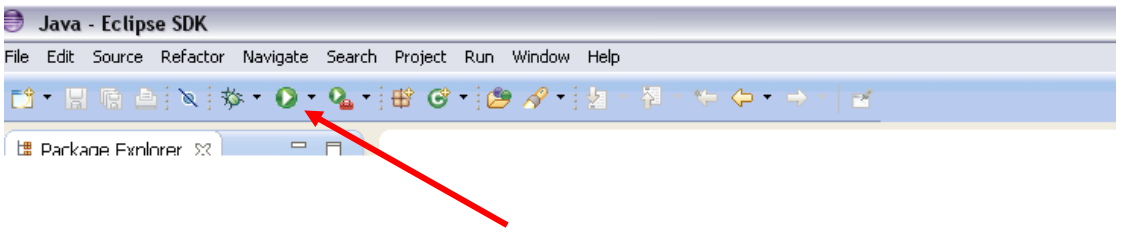
Seleccionamos *Run/Debug Settings*, después seleccionamos la configuración *Main*, y pinchamos en *Edit*…

8- Se abrirá la siguiente ventana:



Seleccionamos la pestaña *Arguments* e introducimos el nombre del fichero origen. Seleccionamos OK.

9- Por último iniciamos la ejecución del programa pulsando:



10- Al terminar la ejecución de la aplicación generara dos ficheros nuevos: bxx_synthesis_Adaptado_TMRtop.vhd y bxx_synthesis_Adaptado_Restricciones_Top.ucf.

Apéndice 3

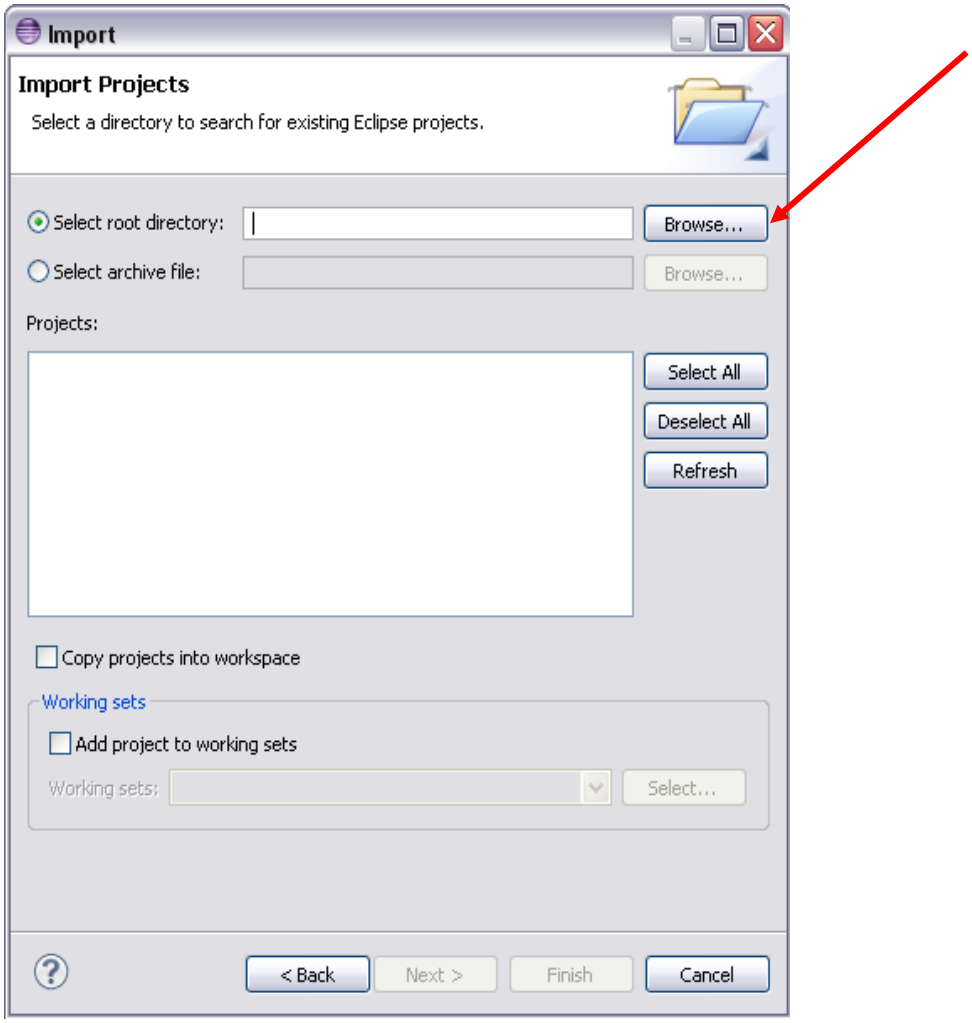
Manual Modifica TB

1- Abrir Eclipse.

2- Hacer *click* en *File->Import…*

3- Desplegar la carpeta *general* y seleccionar *existing proyects into workspace*, pulsar *next*.

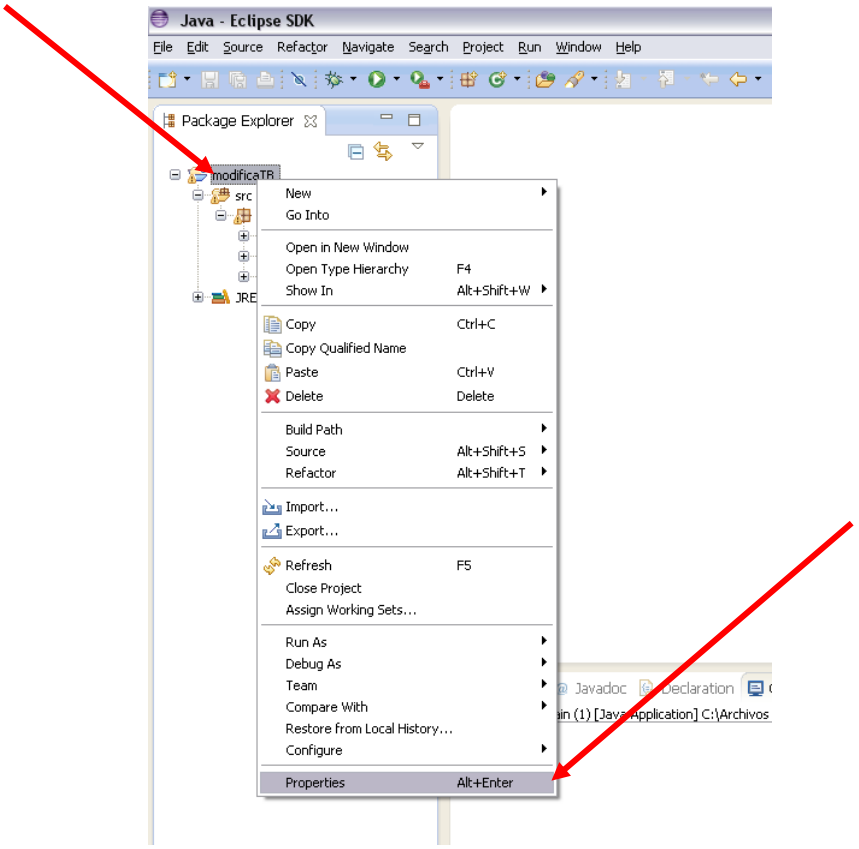
4- Aparecerá esta pantalla:



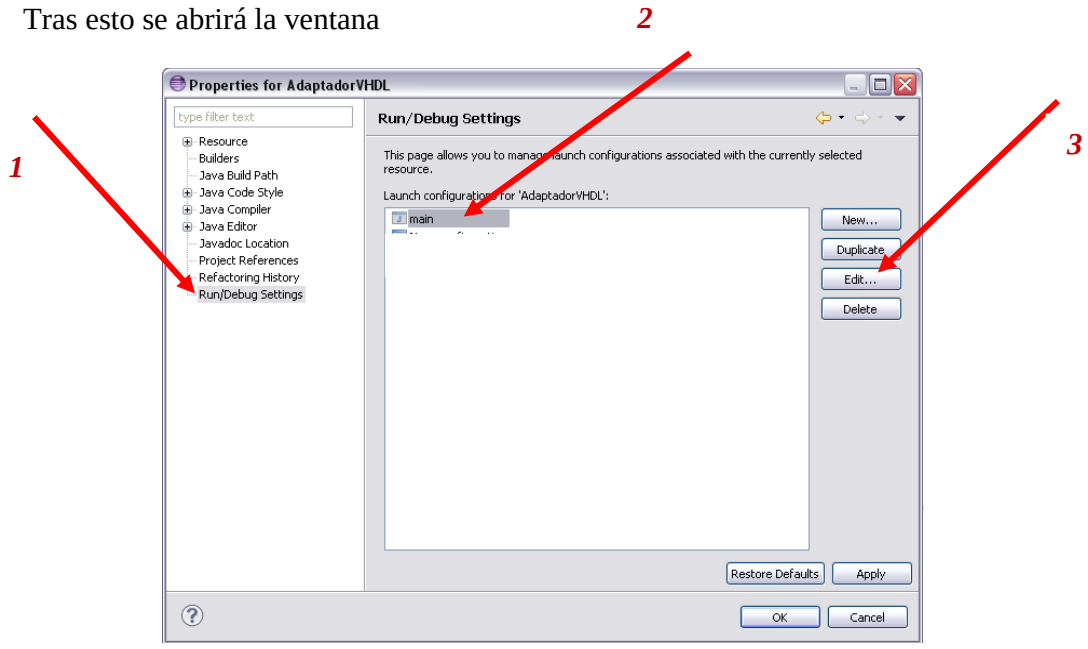
Pulsar *Browse* y buscar y seleccionar la carpeta contenedora de la aplicación y pulsar *finish*.

5- Introducir en la carpeta en la que se encuentra la aplicación Testbench NOMBRE.txt.

6- Hacer *click* derecho en la carpeta contenedora del proyecto en Eclipse, y seleccionar la opción *Properties*.

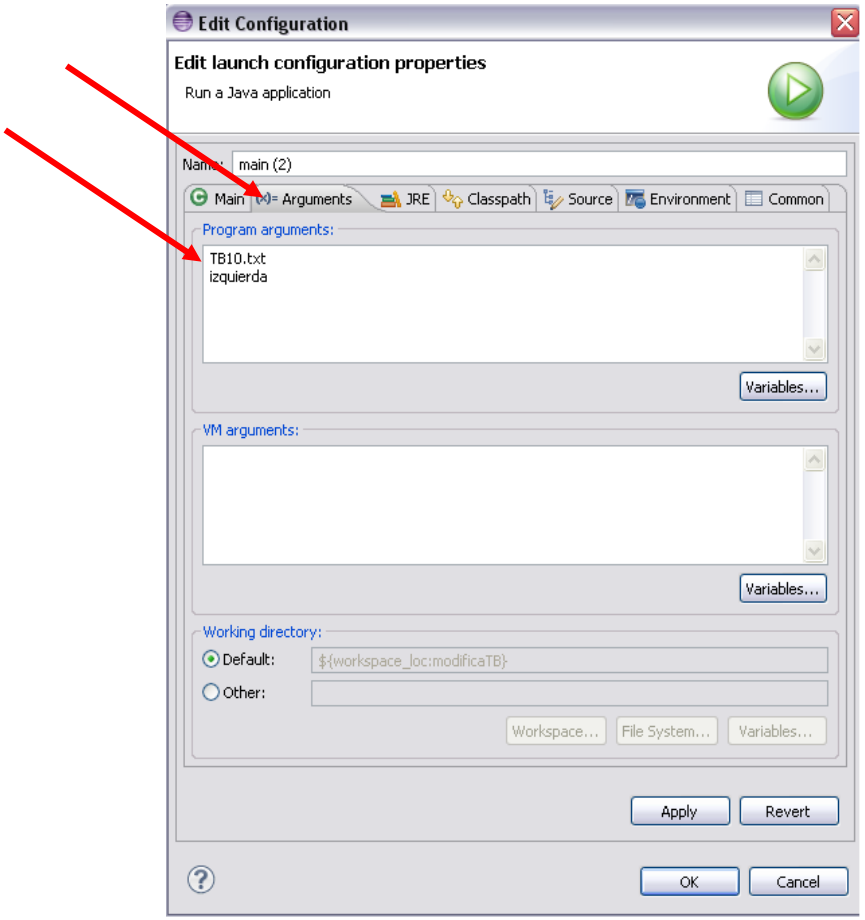


7- Tras esto se abrirá la ventana



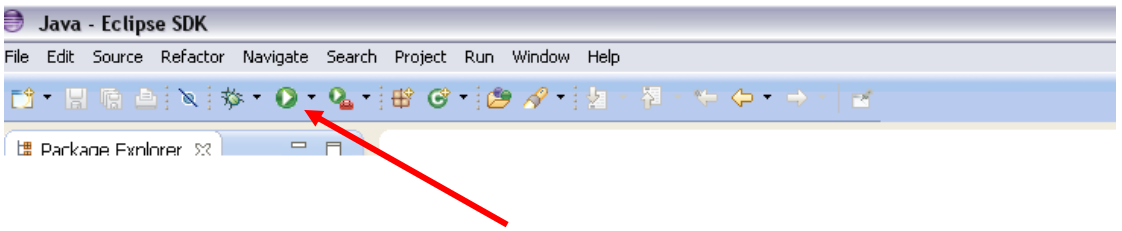
Seleccionamos *Run/Debug Settings*, después seleccionamos la configuración *Main*, y pinchamos en *Edit*…

8- Se abrirá la siguiente ventana:



Seleccionamos la pestaña *Arguments* e introducimos el nombre del fichero origen y el lugar por el cual insertar "0" correspondiente a la configuración de la señal Set_Nessy en la segunda columna. Ya sea por la derecha (escribiendo "derecha") o por la izquierda (escribiendo "izquierda"). Seleccionamos OK.

9- Por último iniciamos la ejecución del programa pulsando:



10- Al terminar la ejecución de la aplicación generara el fichero: NOMBRE_nessy_izquierda.txt o NOMBRE_nessy_derecha.txt dependiendo por donde se desee introducir la señal Set_Nessy.

Bibliografía

[ASCM12] V. Alaminos, F. Serrano, J. A. Clemente, H. Mecha, *"Nessy: An implementation of a low-cost fault-injection platform on a Virtex-5 FPGA"*, in The Conference on Radiation and its E_ects on Components and Systems (RADECS), 2012

[BdM02] L. Benini and G. De Micheli, *"Networks on chip: a new paradigm for systems on chip design"*, in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2002, pp. 418–419

[CMS00] F. Corno, M. Reorda, G. Squillero, *"Rt-level itc'99 benchmarks and _rst atpg results, Design Test of Computers"*, IEEE 17 (3) (2000) 44{53. doi: 10.1109/54.867894

[Stest] www.seutest.com

[Xil1] http://www.xilinx.com/univ/images/V5-Oblique-1000.jpg

[Xil2] http://forums.xilinx.com/t5/image/serverpage/image-id/12706i6F407BB5D988A43D/image-size/original?v=mpbl-1&px=-1

[Xil3] X. Corporation, *"Virtex-5 fpga configuration user guide, ug191 (v 3.11)"*, (2012)

[Xil4] X. Corporation, *"Virtex-5 libraries guide for schematic designs, ug622 (v13.1)",* (2011)

[Xil5] Xilinx, *"MicroBlaze processor reference guide, ug081 (v13.4)"*, 2012