



# Inserción y detección de errores en dispositivo FPGA

**Profesor director:** Hortensia Mecha

**Autores:** Carlos Sánchez-Vellisco Sánchez  
Antonio José García Martínez  
David Fernández Maiquez

**Curso académico:** 2009-2010

Resumen del proyecto .....	9
1. Marco de trabajo.....	10
1.1. FPGAs.....	10
1.2. Motivación del proyecto .....	13
2. Líneas de trabajo .....	13
1. Entrada/Salida.....	13
i. Diseño del transmisor Serie .....	16
ii. Diseño del receptor serie .....	20
2. Circuito inicial de prueba.....	22
2. Generación automática de fichero top vhdl .....	25
2.1. Lector de entidades de un fichero vhd.....	25
3. Ampliación a 32 bits de entrada/salida.....	28
3.1. Recepción de datos .....	28
3.2. Transmisión de datos .....	29
3.3. Cronograma de ejemplo.....	29
4. Fichero de restricciones de usuario (User Constraints File).....	30
5. Generación del fichero BIT.....	32
5.1. Pasos a seguir para la generación de un fichero .bit.....	33
6. Carga del fichero BIT en la FPGA.....	38
7. Comunicación de nuestra aplicación con el puerto RS232 .....	39
8. Ejecución de un circuito .....	40
9. Proceso de reconfiguración parcial.....	41
9.1. Reconfiguración en FPGAs de Xilinx.....	42
9.2. Tipos de configuración de una FPGA.....	42
9.3. Aplicación de reconfiguración.....	42
9.4. Descripción del proceso .....	42
3. Manual de Usuario .....	44
3.1. Requisitos del sistema.....	44
3.1.1. Requisitos software.....	44
3.1.2. Requisitos hardware.....	44
3.2. Requisitos de nuestra aplicación.....	44
3.2. Ventana Principal .....	45

3.2.1. Barra de menús .....	45
3.2.2. Barra de botones:.....	49
Más detalles a tener en cuenta:.....	63
4. Código fuente .....	65
4.1. Ficheros VHDL .....	65
RX_Serie .....	65
Tx_Serie .....	69
Contador inicial de prueba.....	72
Circuito_FPGA.vhd (generado automáticamente).....	75
5. Documentación del código.....	82
Package compiladorEntidad.....	82
Class Entidad .....	83
Entidad .....	85
getNombre .....	85
setNombre.....	85
getNumEntradas .....	85
getNumSalidas.....	85
getBitsEntrada.....	85
setBitsEntrada .....	86
getBitsSalida .....	86
setBitsSalida .....	86
getEntrada.....	86
getSalida .....	86
getPosicionReset .....	87
getPosicionClk .....	87
anadeEntrada .....	87
anadeSalida .....	87
muestra .....	88
getNombreReset .....	88
toString.....	88
Class Entrada .....	88
Entrada .....	89
getEsReloj.....	89

ponerComoReloj .....	90
quitarComoReloj .....	90
getEsReset .....	90
ponerComoReset .....	90
Class Errores .....	90
Errores .....	91
error .....	91
getErrores .....	92
Class EvaluadorExps .....	92
EvaluadorExps .....	93
esOperador .....	93
esOperando .....	93
esMenorIg .....	93
aplicar .....	94
Class LexicoEntidad .....	94
ENTITY .....	97
IS .....	97
PORT .....	98
STD_LOGIC .....	98
STD_LOGIC_VECTOR .....	98
IN .....	98
OUT .....	98
DOWNTO .....	98
END .....	99
PUNTO_Y_COMA .....	99
DOS_PUNTOS .....	99
ABRE_PARENTESIS .....	99
CIERRA_PARENTESIS .....	99
ENTERO .....	100
IDENTIFICADOR .....	100
EOF .....	100
OTRO .....	100
GENERIC .....	100

INTEGER.....	101
ASIG_GENERIC.....	101
SUMA.....	101
RESTA.....	101
MULT.....	101
DIV.....	101
LexicoEntidad .....	102
getNumLinea .....	102
iniciar .....	102
cerrar .....	103
sigToken .....	103
transita .....	103
getUltimoCharLeido .....	103
Class Pila<T> .....	104
Pila .....	105
esVacia.....	105
apilar.....	105
desapilar.....	105
getCima .....	105
numElems.....	106
Class Puerto.....	106
Puerto.....	107
getNombre .....	107
setNombre.....	107
getNumBits.....	107
setNumBits .....	108
Class Salida .....	108
Salida .....	109
Class SintacticoEntidad.....	109
MAX_ENTRADAS .....	111
MAX_SALIDAS.....	111
SintacticoEntidad.....	112
getEntidad .....	112

inicia .....	112
cerrar.....	112
Cabecera.....	113
Entidad .....	113
Generic.....	113
Variables.....	113
Variable .....	114
RVariables.....	114
Puertos .....	114
Senales .....	114
Senal .....	115
RSenales .....	115
Tipo.....	115
Exp .....	116
pasarAPostFija.....	116
evaluar.....	116
empareja .....	116
errorSint .....	117
Class Token.....	117
Token.....	118
Token.....	118
getCodigo .....	118
getLexema .....	119
getNumLinea .....	119
Package generadorVHDL.....	119
Class GeneraVhdl .....	119
GeneraVhdl.....	120
abrir .....	121
cerrar .....	121
crearFichero .....	121
Package IOFPGA .....	121
Class Ejecucion .....	122
Ejecucion .....	124

Ejecucion .....	125
setCadena.....	126
traduceString.....	126
setSetwait.....	127
setFileLogEjec.....	127
convierteCadenas.....	127
run .....	127
getejecutando .....	127
pararrecepcionfpga .....	128
ejecuta.....	128
CopiarSalida.....	128
formatoCorrectoFicheroTB .....	128
enviaReset.....	128
escribeEnLog .....	129
Class Reconfiguracion_Parcial.....	129
Reconfiguracion_Parcial.....	131
reconfiguracionParcial.....	131
run .....	132
pararreconfiguracionparcial.....	132
Package nessy20.....	132
Class CargaBit .....	133
existeFichero .....	133
cargar.....	134
Class Filtro .....	134
Filtro .....	135
Filtro .....	135
accept .....	135
getDescription .....	136
Class GUICargaTB .....	136
GUICargaTB .....	141
Class GUICargaVHDL.....	141
GUICargaVHDL.....	146
Class GUIConfig .....	146

GUIConfig .....	151
Class GUIPrincipal.....	151
GUIPrincipal.....	158
getFiles .....	158
setTop.....	158
getEntidad .....	158
setCerradoTop.....	159
generarGolden .....	159
cargarBitConChooser .....	159
getFichero_bit .....	159
compilarEntidad .....	159
inicializarPuertoSerie.....	160
ejec .....	160
setFwLog.....	160
escribirEnPantalla.....	161
cargarBit .....	161
setNumeroInst.....	161
seleccionaPanel.....	161
SeleccionTBModifFichero.....	162
getCom1 .....	162
Class GUISeleccionTop .....	162
GUISeleccionTop .....	167
Class Main .....	167
Main .....	168
main.....	168
Class Seleccion.....	168
seleccion.....	169
selTB .....	170
Seleccion.....	170
Enum SeleccionCargaVHD .....	170
NADA .....	171
SELECCION_VHDL_TOP .....	171
SELECCION_VARIOS_VHDL.....	171

values .....	171
valueOf .....	172
Enum SeleccionTB .....	172
NADA .....	173
CARGA_PANTALLA .....	173
CARGA_FICHERO .....	173
values .....	174
valueOf .....	174
6. Conclusiones.....	174
6.1. Sobre los conocimientos adquiridos y requeridos .....	174
6.2. Sobre la inserción de errores .....	175
7. Bibliografía .....	176

## Resumen del proyecto

El objeto de este proyecto es el de insertar errores en una FPGA -en este caso el modelo Virtex II Pro- para evaluar el posible impacto que pueden tener en esta. Para ello, cargaremos un circuito específico en la FPGA, conectándolo previamente a un módulo de entrada/salida que nos permita enviar y recibir datos de la placa. Una vez cargado, aplicaremos una serie de entradas al circuito y obtendremos sus salidas. Tras ello iremos insertando errores en la FPGA - mediante la modificación sucesiva de sus bits de configuración- y veremos si la salida es o no la esperada.

## 1. Marco de trabajo

### 1.1. FPGAs

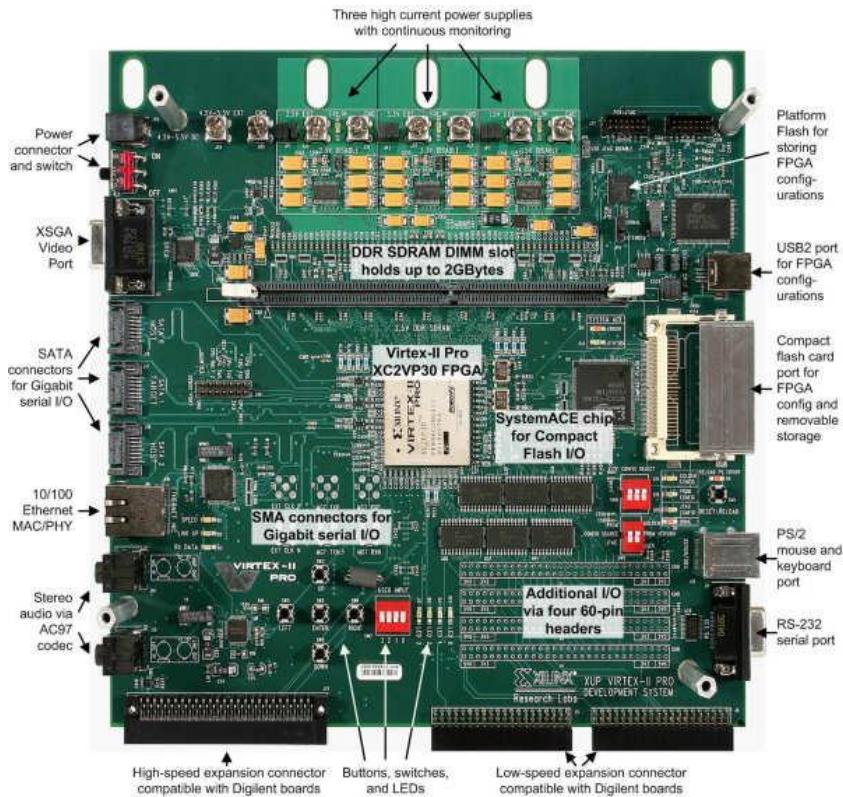
Las FPGA's (Field Programmable Gate Array) son dispositivos que contienen bloques de lógica cuya interconexión y funcionalidad es programable. En ellas se puede programar desde una simple puerta lógica hasta un complejo circuito secuencial.

Hay aplicaciones en las cuales usar un microcontrolador no es suficiente o usar una FPGA posee costos comparables (por ejemplo, codificar y decodificar un MPEG).

Existen otras formas de implementar circuitos digitales, tales como ASIC (Application Specific Integrated Circuit), microcontroladores (con un conjunto fijo de instrucciones), CPLDs (basado en memorias flash, pero más lentos y densos que una FPGA).

Las ventajas que tienen la FPGA es que son dispositivos reconfigurables, tienen un costo menor con respecto a los ASIC y sus circuitos se ejecutan más rápido que en otros dispositivos reprogramables. Además al tratarse de circuitos digitales los que se implementan en ella, su ejecución es “en paralelo”, cosa que no ocurre en un microcontrolador en el que las instrucciones se ejecutan de forma secuencial.

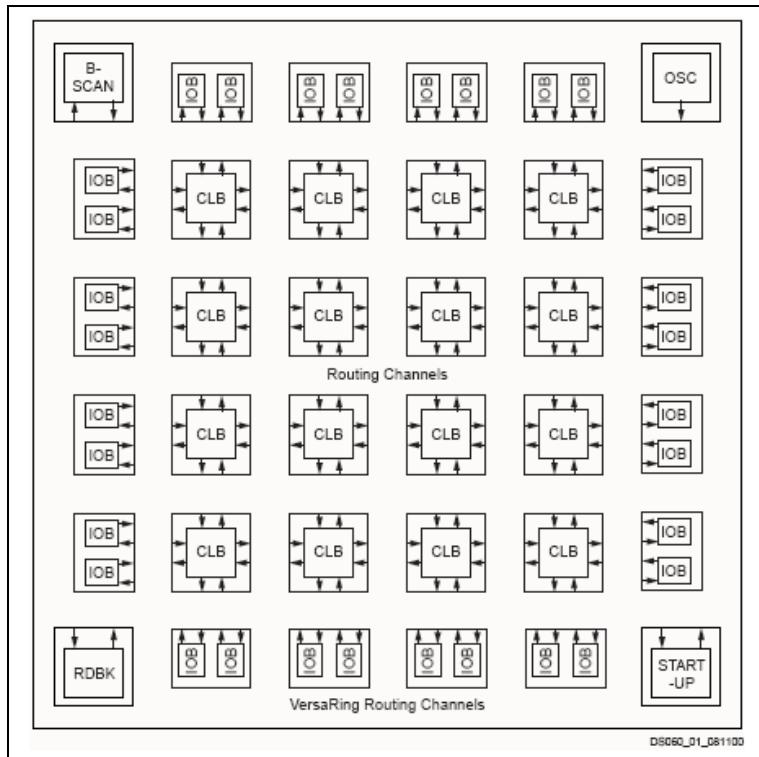
Nosotros trabajaremos con una FPGA del modelo Virtex-II Pro como la que se muestra a continuación:



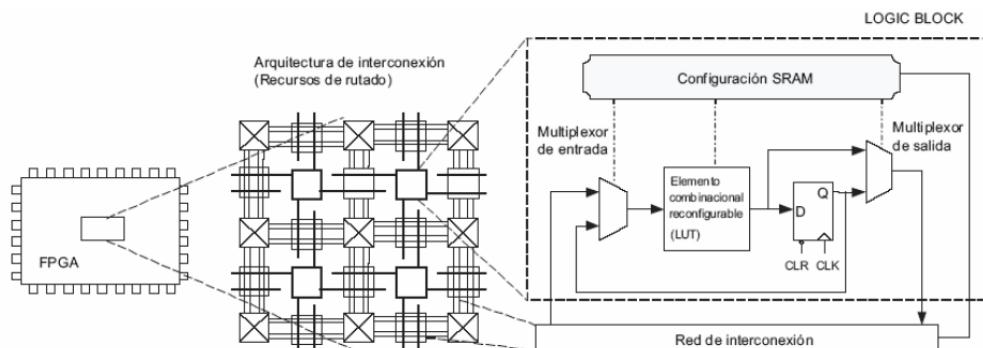
### **1.1.1. Arquitectura de una FPGA**

Una FPGA tiene al menos 3 bloques:

- CLB: Configurable-logic blocks, donde se implementan los circuitos lógicos.
- IOB: Input-output block, donde se conectan las configuraciones internas con pines de entrada/salida.
- DCM: Digital Clock Managers, que permiten entregar señales de reloj a toda la placa.



El elemento esencial de una FPGA es el CLB. En él se pueden implementar tanto circuitos combinacionales como secuenciales. Un CLB puede tener varias celdas lógicas (llamadas ALM, LE, Slice, etc). Los CLB de la Virtex tienen cuatro celdas lógicas distribuidas en dos slices tal y como se muestra a continuación.



Una LUT (Lookup Table) en esencia es una memoria RAM con valores predefinidos. Este es el elemento que nos debe preocupar.

### **1.1. Motivación del proyecto**

Debido a su bajo coste, a su capacidad de reprogramación y el poco espacio que ocupan sus circuitos, se está estudiando la posibilidad de utilizar las FPGAs en tecnología aeroespacial. Una de las aplicaciones implicaría el envío de estos dispositivos al espacio exterior. Aquí encontramos un posible problema. En el espacio exterior no hay protección atmosférica tal y como ocurre en la Tierra. Este hecho tiene como consecuencia que cualquier partícula solar, al chocar con la superficie de la FPGA podría modificar su contenido, alterando por tanto su comportamiento esperado. En concreto podría modificar un bit concreto de una LUT.

Nuestra tarea es, por consiguiente, emular el choque de una partícula, modificando para ello un bit concreto, y ver la cómo reacciona el circuito a dicha modificación. Nosotros conoceremos a priori cuál es la salida esperada de dicho circuito, por lo que si se produce cualquier comportamiento anómalo, sabremos detectarlo fácilmente.

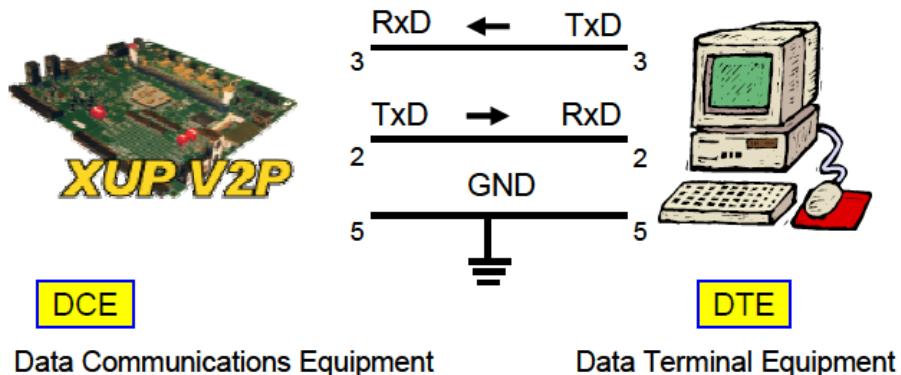
## **2. Líneas de trabajo**

### **1. Entrada/Salida**

Como en todo sistema, necesitábamos poder leer y escribir datos en nuestra FPGA. Teníamos que conseguir una forma de comunicarnos con la placa. En cursos anteriores, habíamos trabajado con FPGA's –en este caso del modelo Spartan III- para las cuales introducíamos las entradas utilizando los switches y los botones de los que viene provisto dicho modelo, y visualizábamos las salidas tanto a través de los leds como del display de 7 segmentos que tenía aquel modelo.

Esa metodología nos es insuficiente para nuestro propósito por varios motivos. En primer lugar, nos ofrece muy poca versatilidad y margen de maniobra para ejecutar y probar circuitos que tengan muchas entradas o salidas. En segundo lugar, es incómodo y poco *usable* estar utilizando en todo momento switches y botones si queremos, por ejemplo, introducir un gran número de instrucciones en la FPGA (simular muchos ciclos de reloj). Además no tendríamos forma de registrar las salidas obtenidas si no es *apuntándolas con papel y lápiz*. Por último, el modelo Virtex-II Pro ofrece únicamente 4 switches y 4 leds con lo que, definitivamente, se hace inviable el uso de la entrada/salida con este método.

Ha quedado claro que hay que comunicarse con la FPGA de otra forma. Una de las formas de comunicarse con la placa es mediante el puerto serie RS232. Por tanto tendremos que diseñar un módulo que nos permita comunicar nuestra placa con en puerto serie del PC.

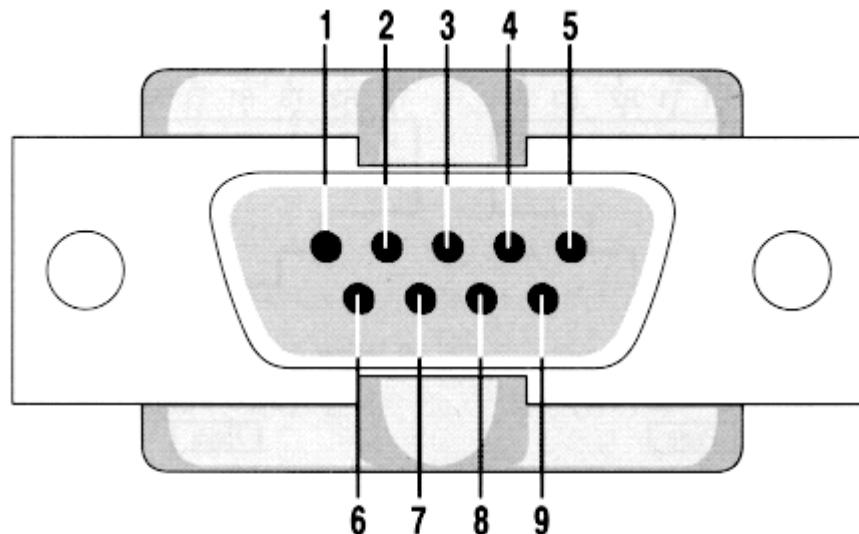


A continuación explicaremos el funcionamiento de este protocolo de comunicaciones.

En la norma se definen dos tipos de terminales: DTE y DCE, donde DTE es el equipo terminal de datos (en nuestro caso el PC) y DCE es el de comunicación de datos (en nuestro caso la placa).

El puerto serie se compone de 9 pines tal y como se muestra en el siguiente diagrama:

Para nosotros, sólo serán imprescindibles tres señales: la línea de transmisión de datos (Transmitted Data, TxD), la línea de recepción de datos (RxD) y la toma de tierra (Signal Ground, GND).



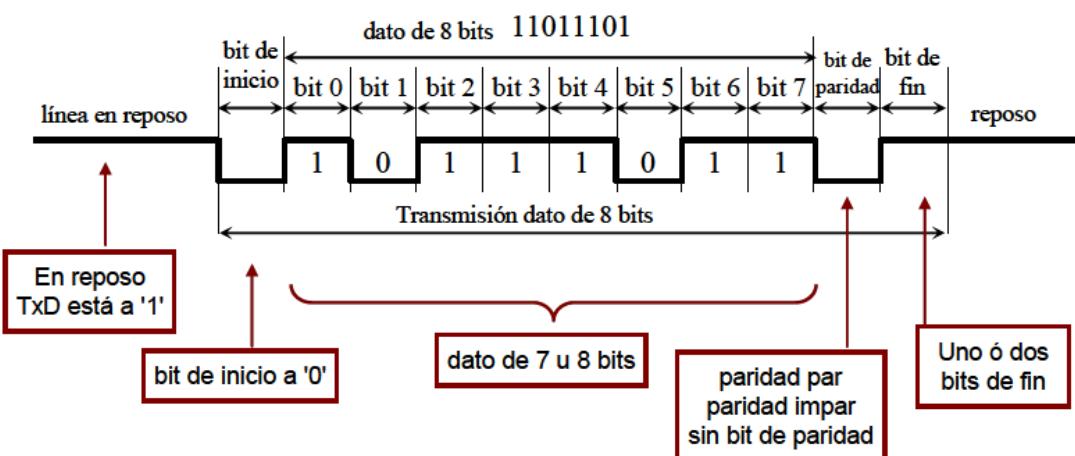
Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Para la conexión se utiliza un cable con conector db9 como el de la figura de arriba. Los pines de la FPGA que están conectados con el puerto RS232 se muestran en la siguiente tabla:

Nombre pin	Tipo	Nombre en FPGA (ucf)
RS232_TX_DATA	Salida	AE7
RS232_RX_DATA	Entrada	AJ8
RS232_DSR_OUT	Salida	AD10
RS232_CTS_OUT	Salida	AE8
RS232_RTS_IN	Entrada	AK8

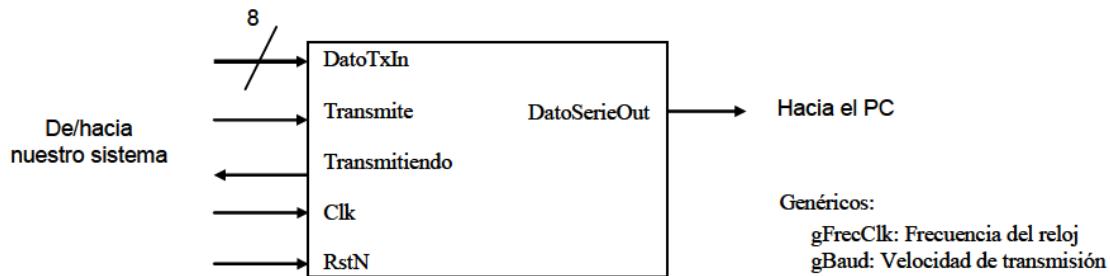
Existen distintas velocidades de transmisión de datos, que se definen en bits por segundo (bps) o baudios (921600, 460800, 230400, 115200, 57600, 38400, 19200, 9600, 4800, etc). En nuestro caso enviaremos datos a una velocidad de 9600 bps.

La línea serie permanece a nivel alto ('1') mientras no se envíen datos. Cuando el transmisor va a empezar la transmisión, lo hace enviando un bit de inicio que está a nivel bajo ('0'). Posteriormente se envían consecutivamente (en serie) los bits del dato empezando por el menos significativo. Después del último bit de dato, se envía un bit a '1' para indicar el final de la transmisión (el llamado 'bit de parada'). Este protocolo puede variar en la forma en la que se envían los datos. Existe también la posibilidad de enviar un bit de paridad y/o dos de parada. En la siguiente figura se muestra el cronograma de un envío RS232 con 8 datos. En este caso se usa un bit de paridad y otro de parada.



Una vez conocido el funcionamiento del protocolo de comunicaciones que vamos a utilizar para enviar y recibir datos de la placa, pasamos a su diseño.

### i. Diseño del transmisor Serie



Los puertos de la izquierda son los que se relacionan con nuestro sistema (el que hayamos implementado en la FPGA) y el puerto de la derecha (DatoSerieOut), envía el dato serie al PC. Vamos a usar también dos genéricos que harán que el circuito se pueda adaptar fácilmente a distintas frecuencias del reloj, y así poder implementar el diseño en otra placa, con otras velocidades de transmisión.

## 1. Especificaciones de los puertos

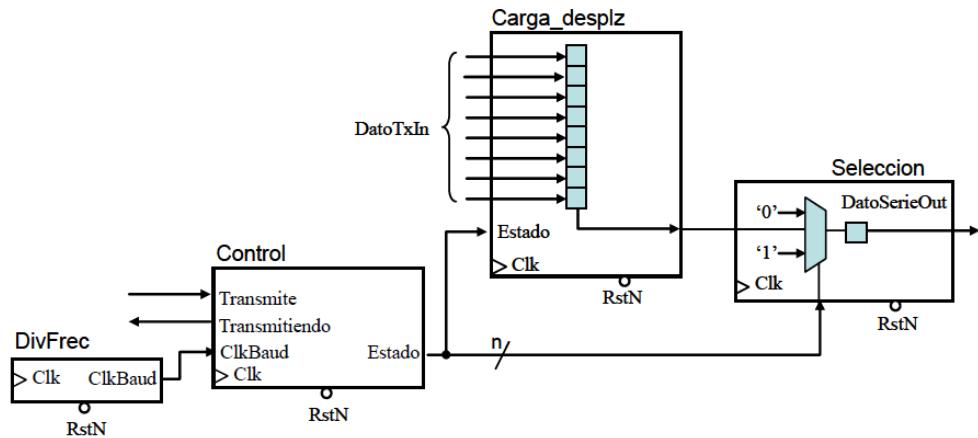
Señal	Nº bits	Tipo	Descripción
RstN	1	Entrada	Señal de reset asincrónico (activo a '0')
Clk	1	Entrada	Señal de reloj de la placa, en principio de 100 MHz, pero configurable por gFrecClk
Transmite	1	Entrada	Señal del sistema que ordena al módulo la transmisión del dato que se encuentra en DatoTxIn.
DatoTxIn	8	Entrada	El dato a enviar. Se proporciona de manera simultánea cuando Transmite = '1'
Transmitiendo	1	Salida	Señal del sistema que indica que en ese instante se está transmitiendo un dato.
DatoSerieOut	1	Salida	Trama de datos que se envía al PC y que sigue el protocolo RS232

Con estas especificaciones tenemos información suficiente para hacer el transmisor:

Lo dividiremos en cuatro bloques principales:

- DivFrec: Un divisor de frecuencia. Dividirá la frecuencia de reloj tantas veces como indique gFrecClk.

- Control: Una máquina de estados finitos.
- Carga\_desplaz: Un registro de carga en paralelo.
- Selección: Un multiplexor que selecciona la señal de salida según el estado actual. Este multiplexor termina en un biestable para evitar pulsos no deseados, ya que su salida (DatoSerieOut) es la salida del circuito.



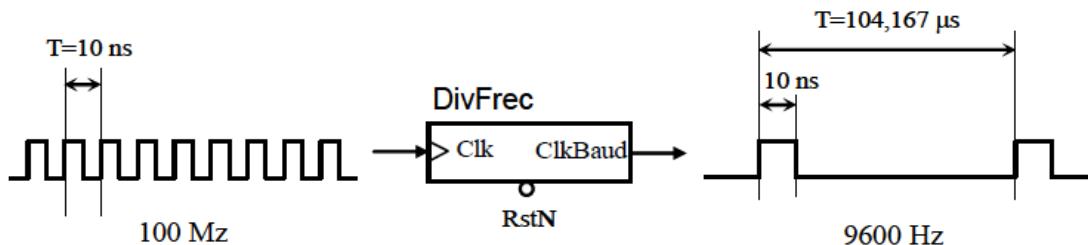
## 2. Divisor de frecuencia

Queremos diseñar un divisor de frecuencia genérico, es decir, que si queremos que cambiar la frecuencia de salida del divisor, no tengamos más que cambiar una constante, sin tener que rehacer el diseño.

En nuestro diseño usaremos dos genéricos:

- gFrecClk: indica la frecuencia de reloj de la placa. Usando este genérico podremos implementar nuestro módulo de entrada/salida en otra placa que tenga un reloj diferente. En nuestro caso, la frecuencia de reloj de la Virtex II Pro es de 100 MHz.
- gBaud: indica la velocidad de transmisión de la unidad de entrada/salida. En nuestro caso, como ya dijimos anteriormente, usaremos 9600 bps.

A partir del reloj de la placa de 100 MHz (Clk), queremos proporcionar una señal con frecuencia de 9600 Hz (ClkBaud). Este reloj tendrá por tanto un periodo de 104,167 µs, y estará a '1' durante un solo ciclo de reloj, estando el resto de tiempo a '0'.

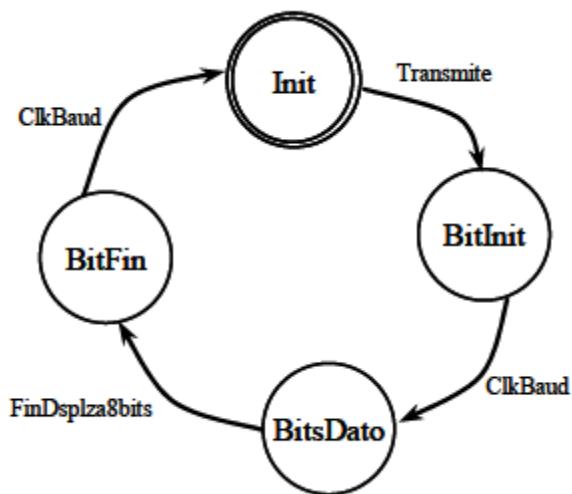


Para diseñar el divisor de frecuencia se divide la frecuencia de entrada entre la frecuencia de salida ( $100\text{ MHz}/9.6\text{ KHz} = 10416,67\text{ Hz}$ ) y el número resultante nos dará el contador que se necesitará para obtener la frecuencia de salida.

### 3. Circuito de control

El circuito de control es el encargado de dirigir al resto de bloques. Si el circuito no se encuentra enviando ningún dato, cuando se recibe la indicación de transmitir (Transmite = '1'), el circuito comienza la secuencia de envío. Esta secuencia de envío será gestionada mediante una máquina de estados (FSM).

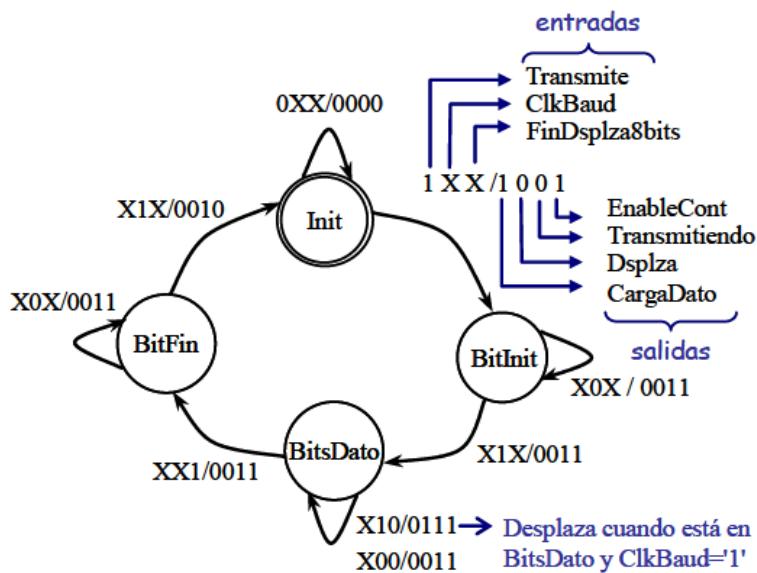
#### 3.1. Estados de la FSM



- eInit: Es el estado inicial. El sistema está en reposo esperando la orden de transmitir. Cuando la señal Transmite se ponga a '1', se pasará a enviar el bit de inicio, pasando para ello al siguiente estado (eBitInit). En ese momento se dará la orden de cargar el dato (DatoTxIn) en el registro de desplazamiento. También tendremos que sincronizar el contador del divisor de frecuencia; para esto tenemos dos opciones: o inicializar el contador a 0 en este estado, o hacer que en el estado inicial no cuente, y en el resto se habilite el contador. Nosotros implementaremos esta última opción.
- eBitInit: En este estado se está enviando el bit de inicio. Se saldrá de este estado al recibir un pulso de ClkBaud, que nos dirá que debemos pasar a enviar los bits de dato. El siguiente estado es eBitsData.

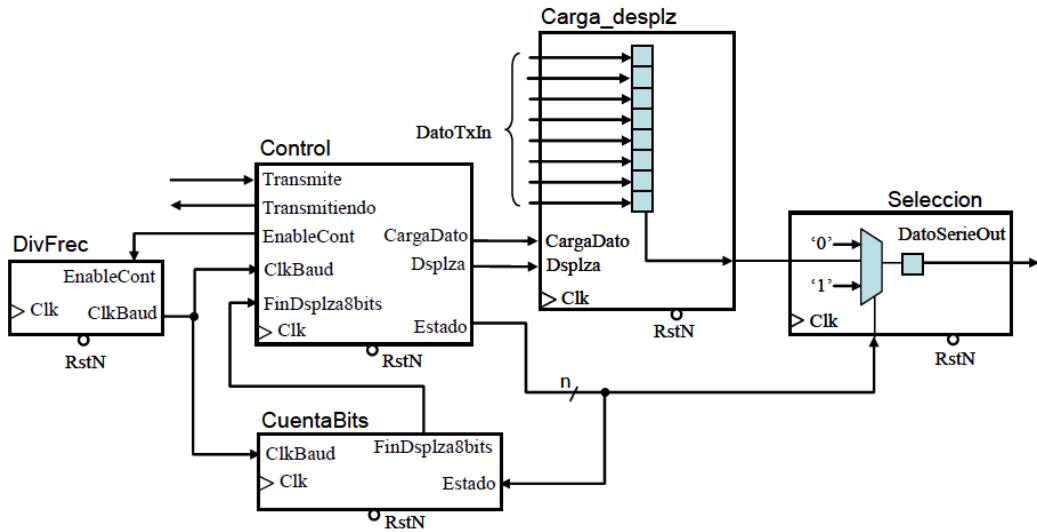
- eBitsData: Este estado se encarga de enviar los 8 bits del dato. Utilizando un contador, llevaremos la cuenta del número de bits que se han enviado. Cuando se hayan enviado los 8 bits –es decir, cuando hayan llegado 8 pulsos de ClkBaud- se activará la señal FinDsplza8bits que hará que cambiemos al siguiente estado (eBitFin).
- eBitFin: Este estado envía el bit de fin. Al llegar el siguiente pulso de ClkBaud, cambiaremos al estado inicial elInit.

Si incluimos las entradas y las salidas en la máquina de estados, nuestro diagrama quedará de la siguiente manera.



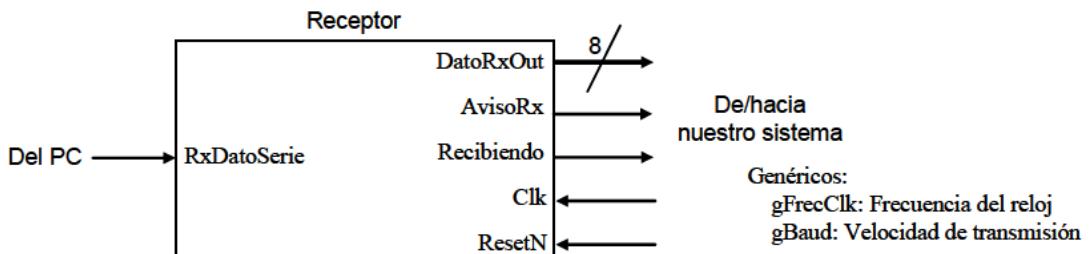
Vemos que se trata de una máquina de Mealy, pues las salidas no dependen solamente del estado sino también de las entradas.

Si incluimos las señales de la unidad de control en nuestro diagrama de bloques, tendremos el siguiente:



[Ver código transmisor serie Tx\\_Serie](#)

## ii. Diseño del receptor serie



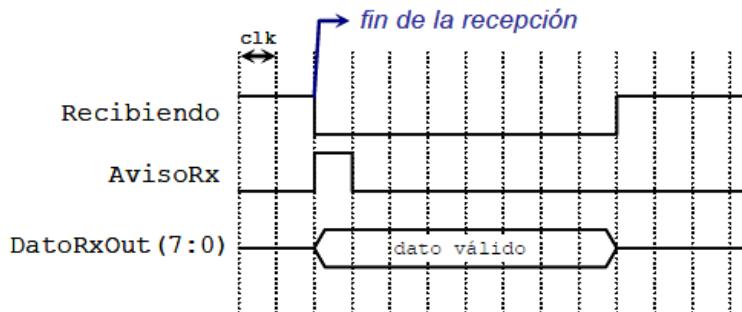
El puerto de la izquierda (RxDataSerie) es el que se encarga de recibir el dato del PC. Los puertos de la derecha se relacionan con el sistema de la FPGA. Análogamente al transmisor serie, vamos a definir los genéricos gFreqClk y gBaud.

### 1. Especificaciones de los puertos

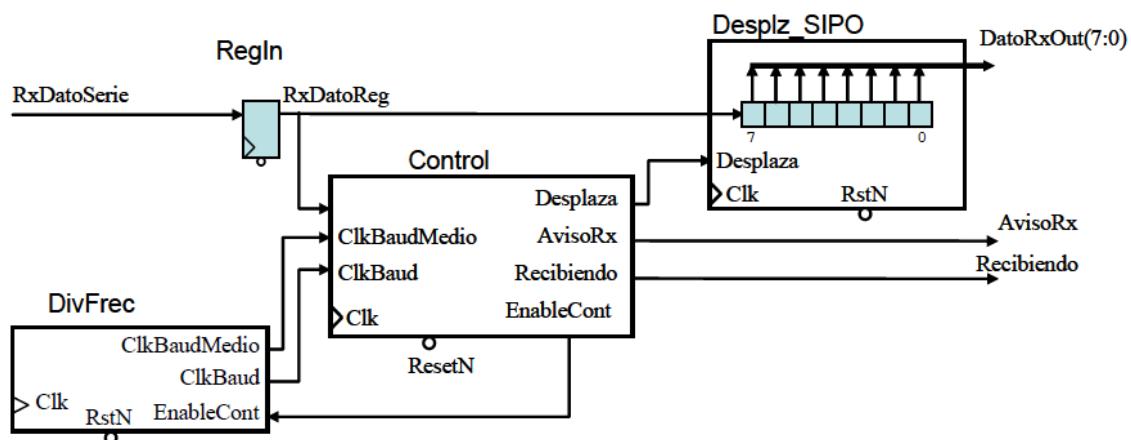
Señal	Nº bits	Tipo	Descripción
RstN	1	Entrada	Señal de reset asíncrono (activo a '0')
Clk	1	Entrada	Señal de reloj de la placa, en principio de 100 MHz, pero configurable por gFreqClk
RxDatoserie	1	Entrada	Trama que recibe del PC, que sigue el protocolo

			RS232
DatoRxOut	8	Salida	El dato que se ha recibido. Este dato sólo es válido desde que AvisoRx vale '1' y mientras recibiendo sea '0'.
AvisoRx	1	Salida	Aviso de que se ha recibido un nuevo dato y que está disponible en DatoRxOut. El aviso se dará poniendo la señal a '1' durante un ciclo de reloj.
Recibiendo	1	Salida	Indica que el receptor se encuentra recibiendo un dato y por lo tanto el valor de DatorxOut no es válido.

En el siguiente cronograma se muestra un ejemplo del final en la recepción de un dato.



El diseño del receptor se puede hacer de manera muy parecida al del transmisor serie.

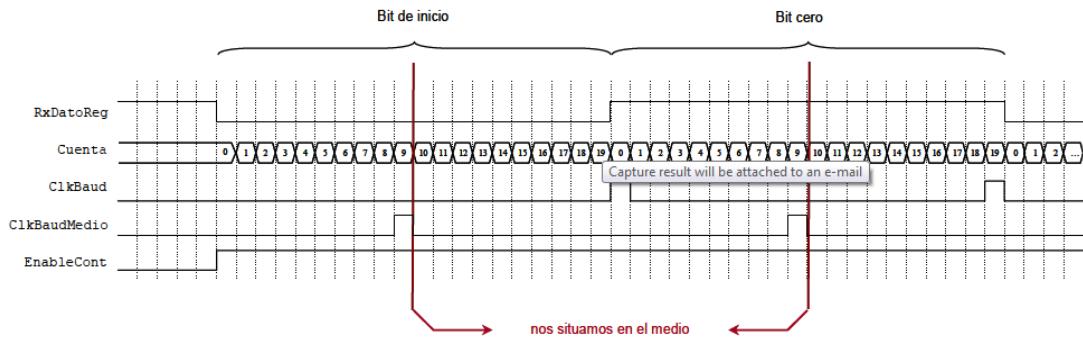


Se puede ver que también tenemos un diseño dividido en bloques.

## 2. Divisor de frecuencia

Para el receptor, el divisor de frecuencia debe sincronizarse con la trama que se recibe porque el receptor no es el que dirige la transmisión. El divisor de frecuencia se mantiene inactivo y a

'0' hasta que EnableCont se pone a '1'. El divisor de frecuencia tiene dos salidas: Clkbaud y ClkBaudMedio. ClkBaud marca la transición de estado mientras que ClkBaudMedio marca el punto medio de cada bit de la transmisión. Con esto nos aseguramos que el dato está estable en este punto ya que es el punto intermedio y por tanto el bit recibido es el correcto, evitando evaluar el bit cerca de las transiciones donde se puede tomar el valor del bit anterior o siguiente.



### 3. Registro de desplazamiento

El registro de desplazamiento Serial In/Parallel Out (Desplz\_SIPO) es un registro al que se le van cargando los datos en serie y los devuelve en paralelo. Es justo el registro opuesto al que teníamos en el transmisor (en el que cargábamos en paralelo y leímos uno a uno). Como el primer bit que se recibe es el 0, si la carga serie se hace por el bit más significativo del registro y se hacen desplazar los bits hacia la derecha, en el último desplazamiento el bit 0 recibido estará en el bit menos significativo del registro, estando así todos los bits ordenados. La carga y el desplazamiento se realizan bajo la orden de la señal Desplaza que sale del bloque de control.

### 4. Registro de entrada

Como la entrada de la comunicación serie RxDatoserie es asíncrona, se almacena en un registro RxDataReg para evitar pulsos y entradas no deseadas.

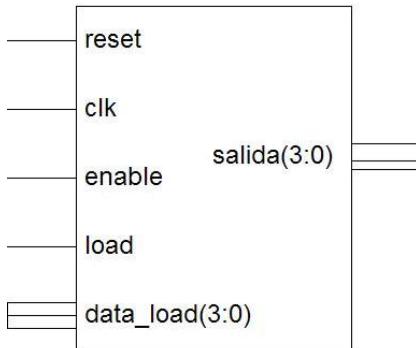
### 5. Circuito de control

El circuito de control es similar al explicado para el transmisor serie.

[Ver código receptor serie Rx\\_serie](#)

## 2. Circuito inicial de prueba

Una vez diseñados el emisor y el receptor serie ya podíamos enviar y recibir datos de la FPGA. Lo primero que hicimos fue diseñar un contador muy simple para probar nuestro módulo de entrada/salida. Se trataba de un contador módulo 16 con carga en paralelo que funcionaba a 1 Hz, es decir contaba cada 1 s. Para ello utilizamos una señal Cuenta que se iría incrementando en una unidad cada en cada pulso de reloj. Al llegar a 100.000 (número de ciclos que se ejecutan en un segundo) la salida se incrementaría en una unidad. Cuando la salida fuera 15 (todo 1's) en el siguiente ciclo se resetearía a 0. El contador también tendría una entrada de enable y otra de reset asíncrona. A continuación se muestra un diagrama del contador inicial.



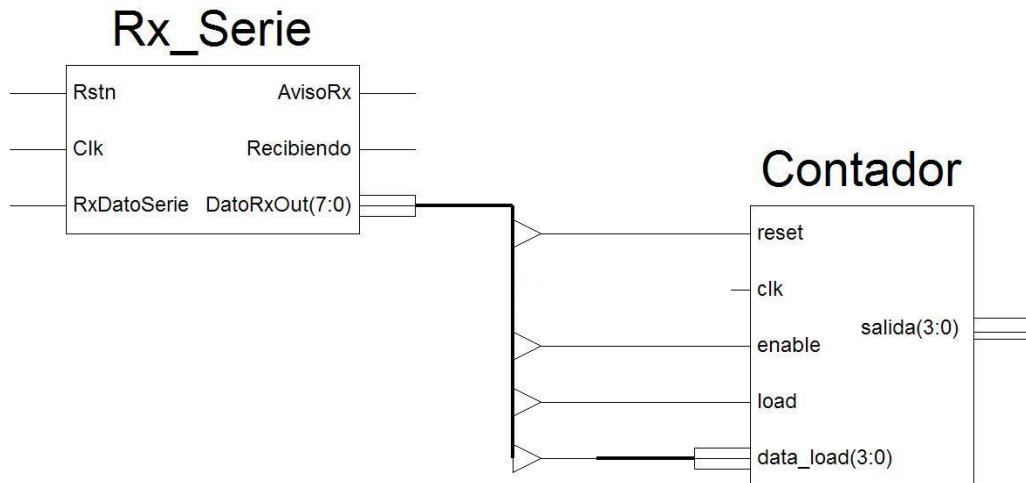
#### [Ver código contador](#)

Nuestra idea inicial fue la de codificar y decodificar los datos de entrada/salida de tal forma que cada combinación distinta de entradas se interpretaba de forma distinta para las entradas de nuestro circuito, en este caso el contador. La interpretación se hacía de la siguiente manera:

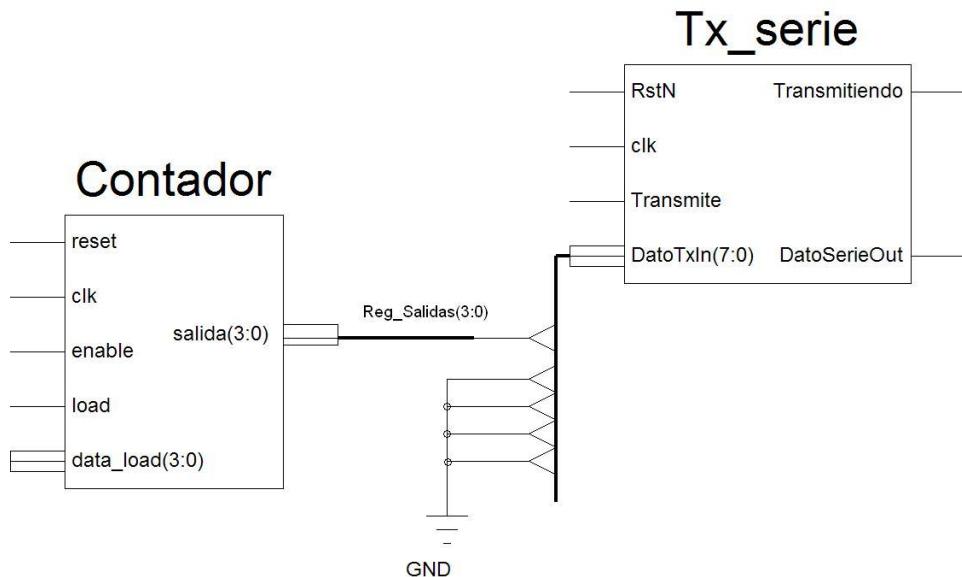
Byte recibido	Carácter ASCII	Entradas al contador			
		Enable	Reset	Load	Data_Load
0110 0101	e	1	0	0	xxxx
0111 0010	r	0	1	0	xxxx
0011 0000	0	0	0	1	0000
0011 0001	1	0	0	1	0001
.....	....	...	..	..	..
0011 1111	?	0	0	1	1111

Pero esto tiene un claro problema. Varias entradas del contador no pueden estar simultáneamente a '1'. En este caso, con cada una de las combinaciones sólo está a '1' una de las entradas de un bit. La forma de arreglar esto es haciendo corresponder a cada bit de entradas un bit de cada byte recibido en la FPGA. Así por ejemplo haríamos corresponder al

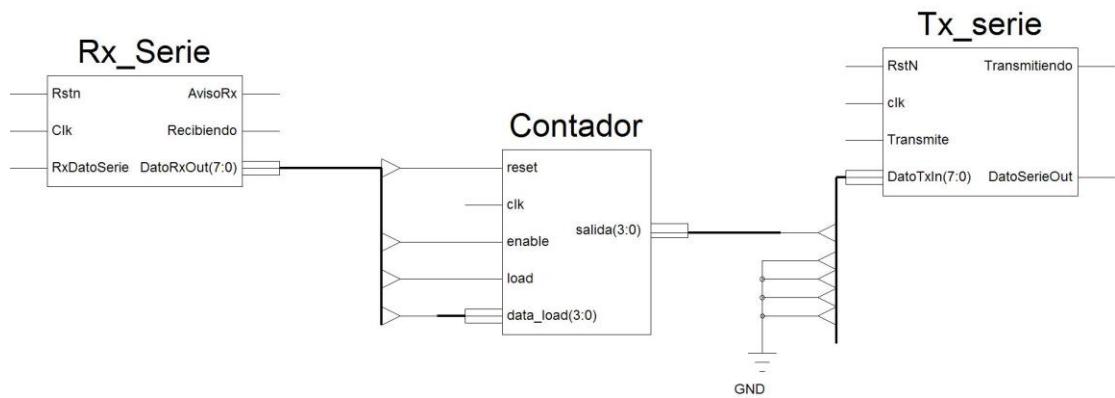
enable el bit cero (el menos significativo) de la transmisión, al reset el bit uno y así sucesivamente. En caso de existir menos de 8 entradas el resto de bits simplemente se ignorarían.



De forma análoga, conectaríamos las salidas del contador al transmisor serie de tal forma que las pudiéramos leer en el PC.



Nuestro circuito completo quedaría de la siguiente manera:



Nótese que la única entrada del circuito que no se hace corresponder con ninguno de los bits de recepción es la entrada de reloj. Esta entrada como podíamos imaginar es especial y le asignaremos la propia entrada de reloj de la placa.

Ahora no tendríamos más que generar un .bit con este circuito y cargarlo en la FPGA. Pero, ¿qué ocurre si queremos conectar cualquier otro circuito a nuestro módulo de entrada/salida? ¿Tenemos que volver a reescribir el código vhdl que haga que se conecten las entradas y las salidas con el módulo? ¿No se podría hacer esto de forma automática? La respuesta es sí.

### 3. Generación automática de fichero top vhdl

#### 3.1. Lector de entidades de un fichero vhdl

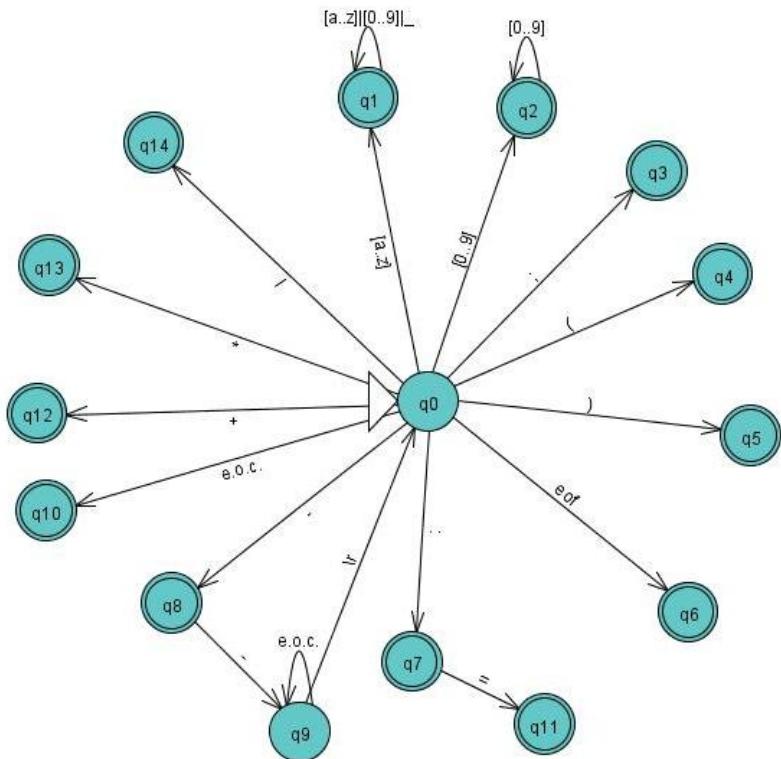
La idea es poder leer cualquier circuito, descrito mediante una entidad en vhdl, y conectarlo automáticamente a nuestro módulo de entrada/salida. Para ello deberemos leer la parte del fichero que nos interese (en este caso sólo nos interesa la parte de la entidad) y cargarlo en memoria, para a partir de esa entidad crear automáticamente un fichero vhdl –que será el módulo top del circuito- en el que estén conectadas las entradas al receptor y las salidas al emisor.

Tendremos que poner en marcha técnicas de procesadores de lenguajes que nos permitan leer de fichero la entidad, y comprobar que está correctamente definida, tanto léxica como sintácticamente.

Es importante remarcar que en este proceso no se analiza el funcionamiento de un circuito descrito en vhdl sino únicamente sus entradas y salidas a efectos de elaborar un modelo a partir de esa entidad que nos permita posteriormente ejecutar el circuito, ofreciéndole las entradas necesarias y leyendo las salidas pertinentes.

### 3.1.1. Analizador Léxico

En esta fase nos interesa leer cada uno de los pequeños elementos de los que se compone el fichero vhdl. Desde palabras reservadas hasta elementos tales como un punto y coma. Hay que saber separarlos para ofrecérselos al analizador sintáctico (que explicaremos más adelante). Este analizador está regido por un autómata finito, el cual se caracteriza por el siguiente dirigrama.



### 3.1.2. Gramática de una entidad

```
Entidad = Cabecera entity identificador is Generic Puertos end identificador
Generic = generic (Variables);
Variables = Variable RVariables;
Variable = identificador : InOut integer := entero
RVariables = ; Variable RVariables
RVariables = λ
Puertos = port(Señales);
Señales = Señal RSeñales
Señal = identificador : InOut Tipo
RSeñales = ; Señal RSeñales
RSeñales = lambda
InOut = in
InOut = out
Tipo = std_logic
Tipo = std_logic_vector(Exp downto Exp)
Exp = Term RExp
RExp = + Exp | - Exp | lambda
Term = Fac RTem
RTerm = * Term | / Term | lambda
Fac = ( E ) | nat | identificador
Cabeceara = *
```

Estaremos leyendo sin procesar hasta encontrar la palabra ‘ENTITY’. A partir de ahí se empezará a procesar. Este analizador es también capaz de reconocer genéricos, evaluarlos y sustituir sus valores donde corresponda.

### 3.1.3. Entradas especiales

Una vez procesado el fichero, tendremos en memoria cargada una entidad. De esta forma podremos hacer una asignación de cada uno de los bits recibidos en la FPGA a cada una de las entradas de la entidad automáticamente. Sin embargo, como dijimos anteriormente, hay una entrada especial que es la de reloj la cual no queremos que sea asignada a ningún bit de entrada serie, sino que queremos asignarla a la propia entrada de reloj de la placa. Por eso, cuando nuestro lector de ficheros encuentre una entrada que contenga ‘clk’ o ‘reloj’, la marcará con un atributo especial que indique que se trata de la entrada de reloj.

### 3.1.4. Errores

Puede ocurrir que la entidad no esté bien definida, en cuyo caso se indicará mediante un mensaje indicativo la línea en la que se ha producido el error, con una breve explicación del mismo. También puede ser que la entidad definida tenga un número de entradas mayor de las permitidas para este protocolo de comunicaciones (en este caso 8), por lo que también se indicará mediante un mensaje, no permitiendo la generación del fichero top vhdl.

Más adelante, en el manual de la aplicación, se explicará la forma en la que se muestran estos mensajes y la manera en la que se presenta al usuario la entidad leída.

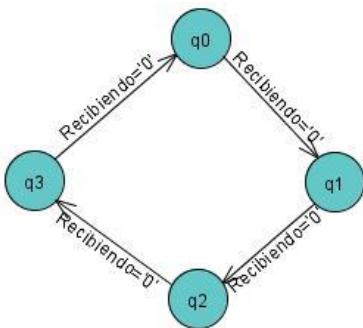
## 4. Ampliación a 32 bits de entrada/salida

Anteriormente hemos visto que podíamos leer y escribir datos fácilmente mediante el puerto serie RS232 utilizando como ejemplo un contador muy simple. Éste, tenía 7 entradas (sin contar la entrada clk), y 4 salidas. Nos sobraban pues, una entrada y cinco salidas para asignar al módulo de entrada/salida. Sin embargo esto es muy poco escalable. A la hora de querer probar otros circuitos, vimos que estábamos muy limitados por el número de entradas y salidas. Por tanto tomamos la decisión de ampliar la posibilidad de leer y escribir hasta 32 bits.

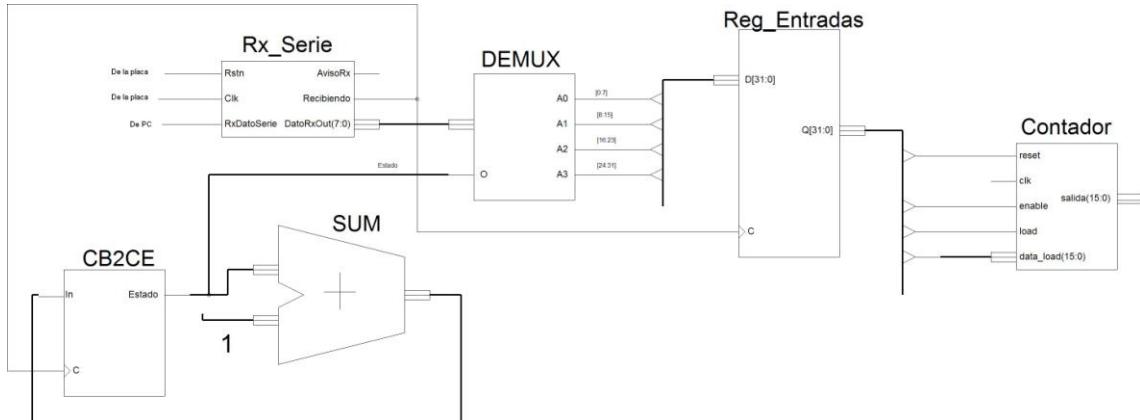
Como vimos anteriormente, el protocolo RS232 sólo es capaz de mandar 8 bits cada vez que envía o recibe. Por tanto vimos que no podíamos modificar el módulo de entrada/salida que habíamos diseñado. Tendríamos que adaptar nuestro módulo top que se generaba automáticamente a este nuevo modelo de 32 bits.

### 4.1. Recepción de datos

Para poder recibir 32 bits deberemos utilizar una máquina de estados, en este caso cuatro, en cada uno de los cuales se reciba un byte.



Cada vez que haya un flanco de bajada de la señal Recibiendo significará que se acaba de recibir un dato satisfactoriamente. Cuando esto suceda y, dependiendo del estado en el que nos encontremos se asignará el dato leído a las entradas correspondientes de un registro al que llamaremos Reg\_entradas y que será el encargado de ofrecer las entradas al circuito. A continuación se muestra un diagrama del circuito resultante utilizando como circuito general un contador con carga en paralelo de 16 bits y salida de 32 bits (mod 4294967296).

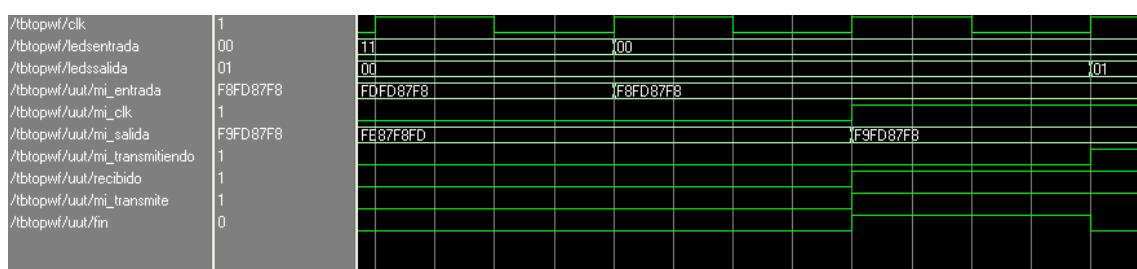


En el cuarto ciclo de lectura (cuando se hayan recibido cuatro pulsos de bajada de Recibiendo) se activará una señal, llamada *fin\_recepcion*, para indicar que la recepción ha concluido. Esta señal tiene una doble función. En primer lugar servirá como reloj del circuito general. La motivación de esto es que no queremos que se ejecute un ciclo de reloj si las entradas aún no están listas. Por otro lado, generará un flanco que durará un solo ciclo y que hará que se active la señal de transmisión.

#### 4.1. Transmisión de datos

Para la transmisión serie de 32 bits también hemos utilizado una máquina de estados. Esta máquina está compuesta también por cuatro estados. En este caso hacemos el mismo proceso pero a la inversa. Vamos a tener que prestar especial atención a algo que no se producía en la recepción de datos. Al recibir sabemos por el bit de inicio que la recepción ha comenzado. Esto nos da tiempo para hacer la asignación correcta de las entradas correspondientes para que estén listas justo cuando la recepción de cada byte termina. Ahora tenemos que tener lista la asignación de salidas antes de que se produzca la transmisión. Por esto introduciremos dos procesos para el caso de la transmisión de datos. Uno que se encargue de cambiar de estado a la máquina de estados de la transmisión de 32 bits y otro que, dependiendo del estado y del flanco positivo del reloj (para asegurar que está preparado antes de que la transmisión comience), haga las asignaciones oportunas.

#### 4.2. Cronograma de ejemplo



En este cronograma se puede observar cómo se realiza la entrada/salida. En este caso se ha simulado para un circuito muy simple, el cual suma 1 a los 8 bits más significativos.

La secuencia es la siguiente. El receptor termina de recibir su último byte (vemos cómo pasa del estado 11 al 10 y la entrada se queda como F8FD87F8). A continuación en el siguiente ciclo de reloj, la señal *recibido* se activa, para indicar que ha terminado la transmisión. Esta señal será la misma que la señal de reloj del circuito. Por tanto la señal hará que cambie la salida; se puede observar que la salida pasa a valer F9FD87B que es justo los 8 bits más significativos más uno. Además a la vez que se terminaba la recepción, se enviaba un flanco de fin que hace que la transmisión comience. Cuando la transmisión de ese dato termine, se generará otro flanco de ‘fin’ que hará que se deje de transmitir. De esta forma conseguimos que se transmitan 4 bytes y sólo 4.

[Ver fichero generado automáticamente para 32 bits](#)

## 5. Fichero de restricciones de usuario (User Constraints File)

Un fichero de restricciones de usuario (UCF) es un fichero que contiene restricciones de localización y temporización. Este fichero es leído por el proceso NGDBUILD (detallado más adelante) durante el proceso de traducción. La finalidad de este fichero es asignar nuestras entradas y salidas del circuito top a localizaciones físicas de pines.

Las entradas de nuestro circuito principal van a ser:

Clk  
Reset  
Entrada\_Serie

Las salidas serán:

Salida\_serie

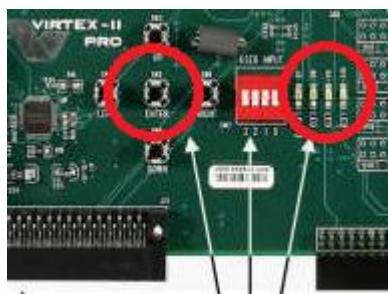
Además para visualizar los estados de la máquina de estados del módulo de entrada/salida de 32 bits, añadimos (de forma opcional) la visualización en los leds de dichos estados.

Nuestro fichero constraints.ucf tendrá el siguiente contenido:

```
NET "clk" LOC = "AJ15";      # Reloj del Sistema  
NET "salida_serie" LOC = "AE7"; #Datos puerto serie RS232_TX_DATA  
#NET "enable" LOC="AD11";     #Switch 1  
NET "reset" loc="AG5";       #Botón enter el del centro  
NET "entrada_serie" LOC = "AJ8"; #Datos entrada puerto serie RS232_RX_DATA  
NET "ledsEntrada<0>" LOC = "AC4";  
NET "ledsEntrada<1>" LOC = "AC3";  
NET "ledsSalida<0>" LOC = "AA6";  
NET "ledsSalida<1>" LOC = "AA5";
```

De esta forma hemos conectado la entrada de reloj al reloj generado por la placa. La entrada salida\_serie a los pines de entrada y salida del puerto serie respectivamente. El reset al botón central de la FPGA (sólo resetea el módulo del puerto serie). Los leds de la derecha serán los correspondientes al estado de la Entrada Serie y los de la izquierda los correspondientes al estado de la Salida Serie.

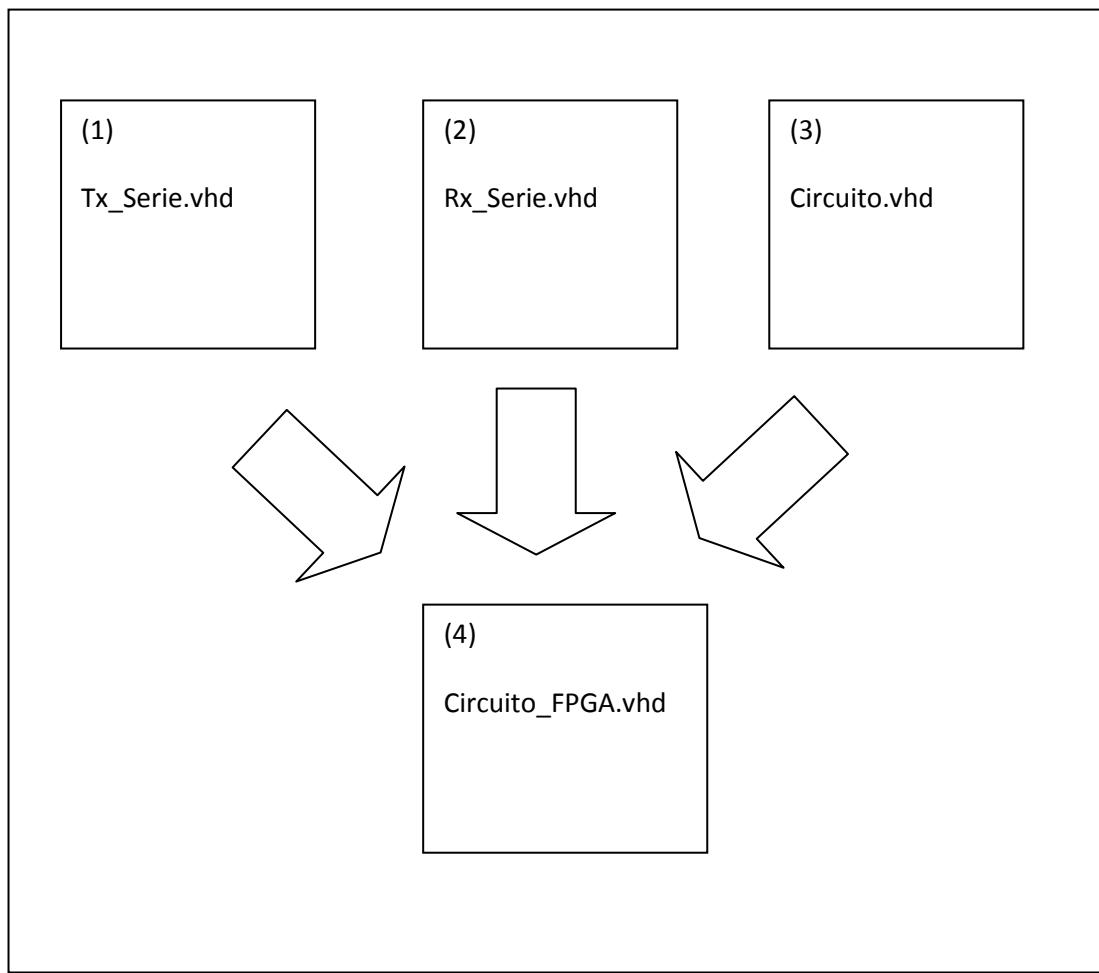
La única forma que tenemos de interactuar con el módulo de entrada/salida es mediante los mecanismos que nos ofrece la FPGA. Por tanto, la forma en la que podemos resetear el módulo es pulsando el botón central (ENTER) de los botones de la FPGA. En principio no necesitaremos pulsar ese botón. Para ver que la entrada/salida está funcionando correctamente es suficiente con fijarse en si todos los leds están encendidos. Esto indicará que el módulo está sincronizado. En caso de desincronizarse puede volver al estado correcto pulsando reset.



## 6. Generación del fichero BIT

Como hemos comentado anteriormente, creábamos automáticamente un fichero vhd que serviría como módulo top para nuestro diseño que conectara cualquier circuito de (como máximo) 32 entradas a nuestro módulo de entrada/salida.

A continuación se ve gráficamente lo que hacemos exactamente:



- (1) Fichero vhd de transmisión serie
- (2) Fichero vhd de recepción serie
- (3) Circuito introducido por el usuario (en el ejemplo anterior, un contador). Luego explicaremos que también pueden ser varios ficheros.
- (4) Circuito vhd del módulo top generado automáticamente.

La motivación principal de todo esto es que el usuario seleccione un fichero vhd que quiera probar en la FPGA, y de forma transparente se genere un fichero .bit con la entrada/salida ya conectadas de tal forma que sólo tenga que indicar las entradas que quiera introducir al circuito. Si solamente se generaran los ficheros vhd y el usuario los tuviera que introducir manualmente en la herramienta Xilinx para generar el fichero .bit, sería todo mucho más incómodo, si bien también se puede utilizar esta metodología de trabajo. La idea es automatizar también este proceso.

Para ellos deberemos utilizar los procesos que tiene Xilinx para sintetizar, traducir, mapear, “Place and Route” y generar .bit. Intentaremos integrarlos en la medida de lo posible con nuestra aplicación.

### **6.1. Pasos a seguir para la generación de un fichero .bit**

#### **6.1.2. Crear un fichero de proyecto**

El fichero de proyecto (.prj) lo utiliza Xilinx para saber dónde están todos los ficheros vhd que van a ser incluidos en la generación del .bit (en concreto es el fichero de entrada al proceso de síntesis). En nuestro caso tendremos que introducir los ya conocidos del módulo de entrada/salida serie más los introducidos por el usuario más el fichero top generado automáticamente. A continuación mostramos un ejemplo del fichero que se debe generar:

```
vhdl work "Tx_serie.vhd"  
vhdl work "Rx_serie.vhd"  
vhdl work "Circuito_FPGA.vhd"  
vhdl work "E:\Universidad 2009-2010\PFC\Nessy2.0\test\FFE_FPGA.vhd"  
vhdl work "E:\Universidad 2009-2010\PFC\Nessy2.0\test\FFE_TOP.vhd"
```

Los tres primeros que se escriben son fijos y correspondientes a la entrada/salida y los dos últimos son los introducidos por el usuario.

#### **6.1.3. Borrar los archivos anteriores**

A efectos de evitar conflictos de ficheros de proyecto, antes de generar cualquier otro fichero, borraremos los anteriores existentes excepto los vhd correspondientes a entrada/salida.

#### **6.1.4. Comandos Xilinx para creación del .bit**

A continuación se ejecutarán todos los comandos necesarios para la creación del .bit. Esto se ha implementado con ficheros por lotes de comandos (.bat). Cada uno de los ficheros bat, hará una llamada al proceso correspondiente de Xilinx basándose en la configuración de fichero para determinar la ruta donde se encuentra instalado el ISE Xilinx. Todos los ficheros bat se encuentran en una carpeta llamada comandosXilinx. Al ejecutar estos comandos se abrirá una pantalla del símbolo del sistema (Windows) la cual, al finalizar el proceso habrá que cerrar manualmente. La razón de esto es que si ha algún error ha ocurrido, éste pueda ser visualizado correctamente por el usuario.

Las llamadas a los comandos se harán a través de un fichero bat general llamado COMPILE.bat que será el encargado de gestionar todas las llamadas y hacer los cambios de ruta oportunos entre comando y comando. Así pues al comienzo, guardará la ruta actual desde la que se arranca el comando general. De esta forma, aunque para llamar a los comandos se cambie de unidad de disco, siempre se volverá al directorio original para hacer la siguiente llamada al comando de Xilinx (ubicada siempre en la carpeta /comandosXilinx). De ahí que entre comando y comando se observen las siguientes líneas:

```
cd %pwd%
%pwd:~0,2%
```

##### **a. Sintetizar**

Para la síntesis se utiliza el comando XST de Xilinx. Durante la síntesis se analiza el código HDL y se intentan inferir bloques de diseño específicos tales como multiplexores, RAMs, sumadores y restadores para los cuales pueda crear implementaciones eficientes. También se reconocen las posibles máquinas de estados (FSM).

Utilizaremos un fichero script como guía para la síntesis. Este fichero será el que contenga realmente las opciones del comando.

De esta manera escribiremos en un fichero llamado sint.txt lo siguiente:

```
echo run %cd%\..\\IOSerie\Circuito_FPGA.prj -ifmt mixed -top Circuito_FPGA -ofn
%cd%\..\\IOSerie\Circuito_FPGA.ngc -ofmt NGC -p xc2vp30-7ff896 -opt_mode Speed -opt_level
1 > sint.txt
```

(donde %cd% es el directorio actual)

A continuación haremos la llamada al comando XST pasándole el fichero que acabamos de generar.

```
C:\Xilinx10.1\ISE\bin\nt\xst.exe -ifn "%cd%\sint.txt"
```

Donde (C:\Xilinx10.1) sea la ruta que está determinada en el fichero de configuración para Home Xilinx.

Explicación del comando:

- ifmt** Indica el formato de los ficheros de entrada. Para permitir varios tipos de ficheros nosotros permitiremos entrada mezclada (mixed) aunque podríamos poner solamente vhdl.
- top** Especifica el nombre del módulo top. En nuestro caso siempre va a ser Circuito\_FPGA
- ofn** Especifica el nombre del fichero de salida.
- ofmt** Indica el formato del fichero de salida. En este caso queremos que sea NGC.
- p** Dispositivo de destino. Introducimos los parámetros para nuestra Virtex II-Pro
- opt\_mode** Define la estrategia de optimización en la síntesis. Al poner Speed como en este caso, lo que hacemos es intentar reducir el número de niveles lógicos y así incrementar la frecuencia. También podríamos definir que se optimice el área de tal forma que los circuitos se ajusten mejor al área disponible.
- opt\_level** Define el nivel de optimización de la síntesis. Nosotros utilizamos el nivel 1 que es nivel por defecto, para que no se consuma demasiado tiempo en el proceso de síntesis, puesto que nuestros circuitos no van a ser nunca especialmente complejos.

Ficheros de entrada: Circuito\_FPGA.prj

Ficheros de salida: Circuito\_PGA.ngc

### **b. Traducir**

El proceso de traducción se lleva a cabo con el comando NGDBUILD. Este comando realiza todos los pasos necesarios para leer un fichero en formato ngc (el obtenido mediante la síntesis con XST) en el que se encuentra la netlist del proyecto, y crear un fichero ngd que describe el diseño lógico en términos de puertas lógicas, decodificadores, flip-flops y RAMs. El fichero ngd puede ser mapeado a la familia de FPGAs que se elija.

El comando NGDBUILD necesita que haya creada una variable de entorno llamada XILINX (el nombre es obligado) en la que se encuentra una librería dinámica que necesita NGDBuild para ejecutarse. Por ello, lo primero que haremos será crear esta variable de entorno:

```
set XILINX=C:\Xilinx10.1\ISE\
```

Tras ello, haremos la llamada al comando del a siguiente manera:

```
C:\Xilinx10.1\ISE\bin\nt\unwrapped\ngdbuild.exe -uc %ioserie%\constraints.ucf -intstyle xflow
-dd _ngo -nt timestamp -p xc2vp30-ff896-7 %ioserie%\Circuito_FPGA.ngc
%ioserie%\Circuito_FPGA.ngd
```

Explicación del comando:

**-uc** Indicamos que estamos introduciendo un fichero de restricciones. Este fichero lo especificaremos más adelante.

**-dd** Indica el directorio en el que ubicaremos los ficheros de salida.

**-nt** Indica la forma en la que se tratan los timestamps al llamar al comando NGDBUILD. Un timestamp es la información relativa a la fecha y la hora en la que un archivo fue creado. Las opciones pueden ser timestamp|on|off. En nuestro caso está en la opción timestamp, de tal forma que se ordena al comando que chequee de forma normal el timestamp y actualice los ficheros ngo de acuerdo con esta información.

**-p** Especifica la parte en la cual se implementa el diseño. Puede especificar sólo una arquitectura, una parte completa (dispositivo, paquete y velocidad) o una especificación parcial (por ejemplo paquete y dispositivo solamente). En nuestro caso marcamos las tres opciones siendo xc2vp30 el dispositivo, ff896 el paquete y 7 la velocidad.

A continuación se indican al comando los ficheros de entrada y de salida.

Ficheros de entrada: Circuito\_PGA.ngc, constraints.ucf

Ficheros de salida: Circuito\_PGA.ngd

### c. Mapear

Después de la traducción se ejecuta el proceso de mapeo sobre el diseño. Este proceso utiliza el fichero NGD (Native Generic Database) creado en la traducción y mapea el diseño lógico a la FPGA tras comprobar unas reglas de diseño determinadas. Los resultados se escriben en un fichero NCD (Native Circuit Description) que será usado más adelante para el proceso de Place and Route.

Para el comando de mapeo se hace la siguiente llamada:

```
C:\Xilinx10.1\ISE\bin\nt\map -intstyle xflow -p xc2vp30-ff896-7 -cm area -pr off -k 4 -c 100 -tx off -o %ioserie%\Circuito_FPGA_map.ncd %ioserie%\Circuito_FPGA.ngd %ioserie%\Circuito_FPGA.pcf
```

Explicación del comando

**-p** Para indicar dispositivo, paquete y velocidad

**-cm** Especifica el criterio utilizado en la fase de cobertura del mapeo. En esta fase, el proceso asigna la lógica a los generadores de funciones de los CLBs, es decir a los LUTs. Se pueden usar las opciones *area*, *speed* y *balanced*. Nosotros utilizamos *area* para que dé prioridad a la reducción del número de LUTs y por tanto también del número de CLBs.

**-pr** Especifica que los flip-flops o latches pueden estar empaquetados en registros de entrada salida. En nuestro caso esta opción está desactivada.

**-c** Pack Slices. Esta opción determina el grado en el cual las slices utilizan empaquetado no relativo cuando el diseño está mapeado. El valor por defecto es 100 que es el que utilizaremos.

**-o** Especifica el nombre del fichero de salida para el diseño.

Ficheros entrada: Circuito\_FPGA.ngd

Ficheros salida: Circuito\_FPGA.ncd, Circuito\_FPGA\_map.ncd, Circuito\_FPGA.pcf

#### **d. Place and Route**

Este proceso (PAR) es el encargado de colocar y enrutar lo que hay en el fichero NCD generado mediante el proceso de mapeo. Al hacerlo, genera un fichero de salida NCD que será el que lea el proceso encargado de generar el fichero BIT (proceso BitGen). El place and route puede ser dirigido por temporización (timing driven) o por tablas de coste (cost-based). Nosotros lo haremos de esta segunda manera; de esta forma se realiza utilizando varias tablas de coste (cost tables) que asignan pesos a factores relevantes tales como restricciones, longitud de conexión y recursos disponibles.

La llamada al comando PAR se hace de la siguiente forma:

```
C:\Xilinx10.1\ISE\bin\nt\par -w -intstyle ise -ol std -t 1 %ioserie%\Circuito_FPGA_map.ncd  
%ioserie%\Circuito_FPGA.ncd %ioserie%\Circuito_FPGA.pcf
```

**-w** Esta opción sirve para sobrescribir el fichero NCD en caso de que éste existiera.

**-ol** Overall Effort Level. Para hacer que el proceso sea más o menos rápido. Nosotros ponemos la opción std que es la opción por defecto (la más rápida).

**-t** Especifica la tabla de costes. Usamos el valor por defecto que es 1.

#### **e. Bitgen**

Este comando genera un bitstream. Después de que el diseño está completamente enrutado mediante el proceso de Place and Route, configuramos el dispositivo mediante el fichero generado mediante BitGen (fichero BIT). El fichero BIT contendrá la información de configuración del fichero NCD. El fichero NCD define la lógica interna y las interconexiones de la FPGA. Este fichero se descargará en las celdas de memoria de la FPGA de la forma que explicaremos más adelante.

Al igual que en el caso de NGDBuild, este comando BITGEN necesita que la variable de entorno XILINX esté definida, por lo que la definimos de la siguiente forma:

```
SET XILINX=C:\Xilinx10.1\ISE
```

La llamada al comando la haremos de la siguiente forma:

```
C:\Xilinx10.1\ISE\bin\nt\unwrapped\bitgen.exe -intstyle ise -w -g ActiveReconfig:Yes -g Persist:yes -g DebugBitstream:No -g Binary:no -g CRC:Enable -g ConfigRate:4 -g CClkPin:PullUp -g MOPin:PullUp -g M1Pin:PullUp -g M2Pin:PullUp -g ProgPin:PullUp -g DonePin:PullUp -g PowerdownPin:PullUp -g TckPin:PullUp -g TdiPin:PullUp -g TdoPin:PullNone -g TmsPin:PullUp -g UnusedPin:PullDown -g UserID:0xFFFFFFFF -g DCMShutdown:Disable -g DisableBandgap:No -g DCIUpdateMode:AsRequired -g StartUpClk:JtagClk -g DONE_cycle:4 -g GTS_cycle:5 -g GWE_cycle:6 -g LCK_cycle:NoWait -g Match_cycle:Auto -g Security:None -g DonePipe:No -g DriveDone:No -g Encrypt:No %ioserie%\Circuito_FPGA.ncd
```

Como vemos tiene una gran cantidad de opciones. De todas ellas, destacaremos una que será importante a la hora de reconfigurar nuestro fichero .bit.

**-g ActiveReconfig:Yes** Se requiere para reconfiguración parcial, lo cual significa que el dispositivo se mantiene operativo mientras el nuevo bitstream parcial está siendo cargado. De esta forma sólo se cargarán en la FPGA las frames que difieran entre el archivo que se está cargando y el que está en la FPGA ya cargado.

## 7. Carga del fichero BIT en la FPGA

Una vez tengamos el fichero BIT generado, la idea es también poder cargarlo a través de nuestra propia aplicación. Esta funcionalidad nos la ofrece el programa IMPACT, una de las utilidades del Xilinx ISE. Trataremos pues de integrar este programa en nuestra aplicación en la medida de lo posible.

Sabemos que la aplicación IMPACT se basa en la llamada a comandos. No tendremos más que conocer qué comandos se usan cuando se carga cualquier fichero BIT.

*Nota: los siguientes comandos están adaptados para una Virtex II-Pro cargada mediante Paralell Cable IV.*

Lo que haremos será crear un fichero llamado carga.txt que tenga todos los comandos que queramos ejecutar con el IMPACT. Después ejecutaremos el IMPACT en modo batch de manera que ejecute los comandos que le pasamos por parámetro:

```
C:\Xilinx82\bin\nt\impact.exe –batch carga.txt
```

Donde carga.txt tendrá como contenido:

```
setMode -bs
setCable -port auto
Identify
identifyMPM
assignFile -p 3 -file "D:\PFC\Nessy2.0\IOSerie\circuito_fpga.bit"
Program -p 3
exit
```

Donde la ruta "D:\PFC\Nessy2.0\IOSerie\circuito\_fpga.bit" será la ruta que introduzca el fichero bit que quiere cargar.

## 8. Comunicación de nuestra aplicación con el puerto RS232

En este punto ya sabemos que podemos comunicarnos con la FPGA a través del puerto serie. Para realizar las pruebas utilizábamos el hyperterminal. Con él sólo teníamos que configurar los parámetros de bits por segundo, bits de parada y bits de paridad. Podíamos introducir desde teclado cualquier carácter que se transmitiría automáticamente al puerto serie. De esta manera para mandar 32 bits teníamos que pulsar 4 teclas, ya que se enviaba 1 byte por cada tecla pulsada. Pero tenemos que encontrar una manera de comunicar también nuestra aplicación (implementada en Java) con el puerto serie.

Para ello, teníamos que introducir una librería que manejara el envío y recepción de datos con el puerto serie. En [www.giovynet.com/serialport.htm](http://www.giovynet.com/serialport.htm) encontramos unas librerías que permitían trabajar con el puerto RS232.

Nota: en este apartado hablamos desde el punto de vista del PC (recibir = recibir el PC).

Tiene una interfaz muy sencilla de usar. Podemos enviar un byte de datos que se transmitirá por el puerto serie siguiendo el protocolo RS232. A la hora de recibir datos (hacia el PC) es algo más complicado. Esta interfaz utiliza un buffer en el que va colocando todos los bytes que va recibiendo. Puede ocurrir que la FPGA mande muchos datos pero nosotros sólo leamos por ejemplo 4 bytes de tal forma que el resto se quedarán en el buffer y serían leídos la próxima vez que ejecutemos una orden de recepción de datos. Es por ello importante que consumamos siempre todos los datos que se reciban de tal forma que no quede nunca "basura" en el buffer. Este problema está perfectamente contemplado con la forma en la que tenemos de leer y escribir los datos.

La forma en la que hay que utilizarlo es haciendo que envíe 4 bytes a la FPGA (ejecutando 4 órdenes de envío) y a continuación, como sabemos que va a recibir como resultado otros 4 bytes desde la FPGA, ejecutaremos 4 veces la orden de recepción de datos, quedando así el buffer vacío. De esta forma siempre serán datos útiles y no atrasados.

También podría ocurrir lo contrario, es decir, que queramos leer datos del puerto serie y no haya ningún dato depositado en el buffer. Esto puede ocurrir, por ejemplo, en el caso en el que estemos leyendo las salidas de una ejecución y en un instante determinado se pulse el botón de reset del módulo serie (botón central). Esto dejaría al módulo en un estado en el que no sería capaz de leer ni escribir datos en el puerto serie. Para evitar que la instrucción de recepción de datos se quede bloqueada, la interfaz está provista de un timeout, tras el cual, si no se ha conseguido leer nada se pasa a la siguiente instrucción y se recibe el byte 0. Esta es la razón por la cual, si pulsamos reset mientras estamos ejecutando un circuito, observaremos que en la salida sólo obtenemos ceros, y que además éstos se obtienen de una forma más

lenta que las salidas obtenidas de forma habitual (retardo producido por la espera hasta el timeout).

Para el envío y recepción vamos a trabajar con cadenas de 0s y 1s (representando los bits de las entradas y las salidas). Sin embargo la interfaz sólo es capaz de enviar y recibir datos en formato entero. Para ello deberemos codificar y decodificar los datos enviados y recibidos de formato binario a entero y viceversa.

Para pasar de entero a cadena binaria utilizaremos el algoritmo de la división por dos y para pasar de cadena binaria a entero simplemente sumaremos los pesos.

## 9. Ejecución de un circuito

Una vez conocidos los pasos previos, la ejecución de un circuito es simple. Para ejecutar un circuito tendremos que ofrecer al mismo, una serie de entradas en forma de cadenas de 0's y 1's que coincidan en tamaño con el número de bits de entrada que tenga la entidad que queramos ejecutar.

Estas entradas pueden ofrecerse o bien mediante fichero, o bien escribiéndolas directamente en la aplicación (en el manual se explicará el lugar exacto). Si hemos comprendido el diseño y comportamiento de la entrada/salida, será fácil ver que cada una de las distintas entradas se corresponderá exactamente a un ciclo de ejecución.

Ejemplo: Imaginemos la siguiente entidad

```
component CONTADOR
    Port(
        RESET: in STD_LOGIC;
        CLK: in STD_LOGIC;
        ENABLE: in STD_LOGIC;
        LOAD: in STD_LOGIC;
        DATA_LOAD: in STD_LOGIC_VECTOR(15 downto 0);
        SALIDA: out STD_LOGIC_VECTOR(15 downto 0)
    );

```

Vemos que tiene 20 bits de entrada y 15 de salida. De los 20 bits de entrada debemos descartar la entrada de reloj ya que esa no se transmitirá por el puerto serie sino que se conectaría a la señal de fin de recepción (tal y como explicamos anteriormente). Por tanto tenemos que tener en cuenta 19 bits de entrada y 15 de salida de datos. Pues bien, la entrada

menos significativa se asignará a la primera señal de entrada declarada de la entidad y así sucesivamente. Lo mismo para las salidas.

La entrada (19 bits):

0000 0000 0000 0000 0 1 0

Se interpreta como:

```
RESET <= 0  
ENABLE <= 1  
LOAD <= 0  
DATA_LOAD(0) <= 0  
DATA_LOAD(1) <= 0  
:  
:  
:  
DATA_LOAD(15) <= 0
```

Si la salida (16 bits) fuera:

0001 0100 000 1011

Es sencillo intuir que corresponderá a:

```
SALIDA(0) <= 1  
SALIDA(1) <= 1  
SALIDA(2) <= 0  
:  
:  
:  
SALIDA(15) <= 0
```

Antes de cada ejecución y para asegurar que el circuito arranca en un estado correcto, enviaremos una señal de activo al circuito.

## 10. Proceso de reconfiguración parcial

Hasta ahora hemos hablado de aspectos que eran fundamentales para el desarrollo de la aplicación y sin los cuales la funcionalidad se vería muy afectada. Al comienzo de esta lectura hablábamos de la posibilidad de insertar errores en la FPGA de forma que emulásemos así la posibilidad de que una partícula solar incidiera en la superficie de la FPGA modificando su contenido.

### **10.1. Reconfiguración en FPGAs de Xilinx**

La mínima sección de la FPGA que se puede cambiar está definida por un frame. Un frame es una unidad de la memoria SRAM de configuración. Tiene un bit de anchura y abarca todo el alto de la FPGA. En el caso de la Virtex-II Pro tenemos 32 bits por cada frame, y un total de 36194 frames.

### **10.2. Tipos de configuración de una FPGA**

La tecnología permite diferentes formas de cambiar la configuración de los bloques y conexiones.

- **Reconfiguración total, en tiempo de compilación o estática:** Cada vez que se realiza una nueva configuración, toda la FPGA se actualiza. Se tiene que detener la ejecución de la FPGA, configurarlo todo y volver a ponerlo en marcha.
- **Reconfiguración parcial:** Se puede modificar una parte de la configuración mientras la otra sigue realizando la computación de forma no interrumpida. En el fichero de configuración se especifican las direcciones (de los frames) para que se reconfigure. Con esto se puede cargar nuevas configuraciones en áreas de la FPGA sin necesidad de cambiar el contexto.

### **10.3. Aplicación de reconfiguración**

En este punto tenemos que coordinarnos con otro compañero que nos ofrecerá una aplicación capaz de modificar un solo bit de un fichero de configuración (fichero .BIT). Esta aplicación solo necesita conocer el bit que se quiere modificar, el frame que se desea modificar y el bit que se desea modificar dentro de ese frame.

La aplicación generará dos nuevos ficheros de configuración a partir del original. El primero habrá modificado un bit concreto de un frame concreto (ambos pasados como parámetro). Ese fichero de configuración se cargará modificando, por tanto, tan solo un frame de la FPGA, lo cual será equivalente a haber modificado tan solo un bit. El segundo fichero que se genera es un fichero de configuración para restaurar el frame modificado. De esta forma dejaremos el circuito como estaba originalmente.

### **10.4. Descripción del proceso**

El proceso que ejecutará la aplicación para la emulación de inyección de errores será el siguiente.

En primer lugar se cargará una entidad seguida de su fichero .BIT correspondiente. A continuación se pedirá al usuario un fichero para cargar el banco de pruebas. Tras cargarlo, y sabiendo que en la FPGA está cargado el fichero .BIT correcto (sin modificar), se generará una salida especial, llamada salida Golden, la cual será con la que comparemos en el resto de ejecuciones. Este paso sólo lo realizaremos una vez, al comienzo del proceso.

A continuación entraremos en un bucle, en el cual se irá iterando para cada bit y para cada frame distinto. En cada una de las iteraciones se ejecutará la aplicación de reconfiguración aplicada siempre al mismo fichero .BIT original (introducido por el usuario) en el que iremos cambiando los parámetros de frame y bit. Cada vez que se ejecute la aplicación se generarán

dos ficheros de configuración: fichero\_modif.bit y fichero\_modifRestore.bit. Cargaremos el primer fichero en la FPGA, lo que es equivalente a que una partícula solar modificara el valor del bit de la LUT que estamos modificando. Una vez injectado el error (cargado el fichero de configuración) volveremos a ejecutar el circuito en la FPGA, con las mismas entradas con las que habíamos generado la salida Golden. Compararemos las nuevas salidas obtenidas con la salida Golden para ver si el error ha incidido en la ejecución del circuito.

Ahora tenemos que devolver a la FPGA a su “estado anterior”. Es decir, tenemos que restaurar el bit que acabamos de modificar. Para ello ordenaremos la carga del segundo fichero que se había generado: fichero\_modifRestore.bit.

Con el circuito restaurado, ya podemos volver a ejecutar una iteración más del bucle.

El algoritmo que se ejecuta es el siguiente:

1. Cargar entidad
2. Cargar .BIT (fichero.bit) correspondiente a esa entidad
3. Cargar Test Bench
4. Generar Salida Golden
5. Para cada frame f y para cada bit b
  - a. Ejecutar **Virtex\_II\_Partial\_Reconfiguration -i fichero.bit fichero\_modif -f f -b b**
  - b. Cargar fichero\_modif.bit
  - c. Ejecutar y comparar con Salida Golden
  - d. Cargar fichero\_modifRestore.bit
  - e. Volver a a

### 3. Manual de Usuario

En esta sección, buscamos dar una guía al usuario para facilitar el manejo de la aplicación y su correcto uso. Comenzaremos hablando de los requisitos del sistema necesarios para que se pueda ejecutar con éxito.

#### 3.1. Requisitos del sistema.

##### 3.1.1. Requisitos software

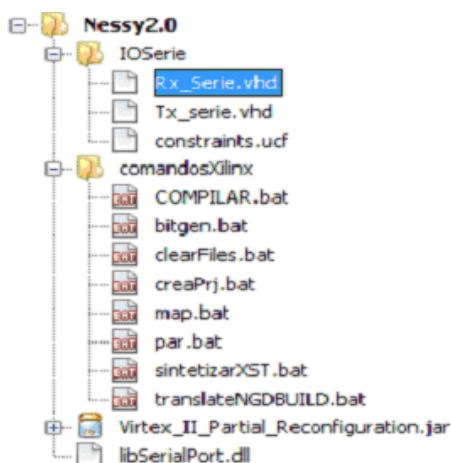
- Windows XP (la aplicación no se ejecuta para sistemas basados en 64 bits)
- La aplicación Xilinx ISE debe estar instalada
- La aplicación Impact debe estar instalada

##### 3.1.2. Requisitos hardware

- Máquina con puerto serie (RS232) habilitado
- Máquina con puerto paralelo (Parallel Cable IV) habilitado
- Máquina con puerto PS2
- FPGA Virtex-II Pro (REV 04)

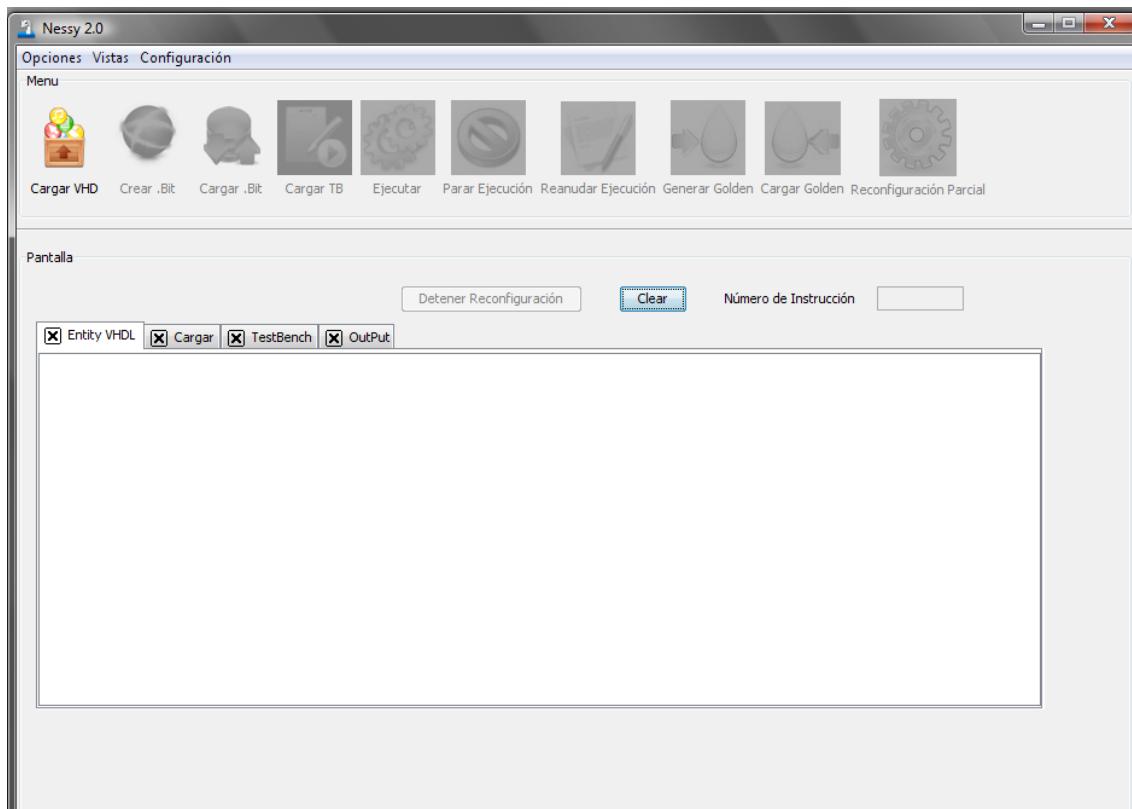
#### 3.2. Requisitos de nuestra aplicación.

Para ejecutar nuestra aplicación es necesaria la siguiente estructura de carpetas con componentes que se detallan dentro de cada una.



### 3.2. Ventana Principal.

Es la ventana principal de la aplicación. Desde ella se pueden realizar todas las acciones, si son permitidas en ese momento, que permite ejecutar la aplicación. También se visualizan distintas salidas que hemos ido provocando con nuestra ejecución.

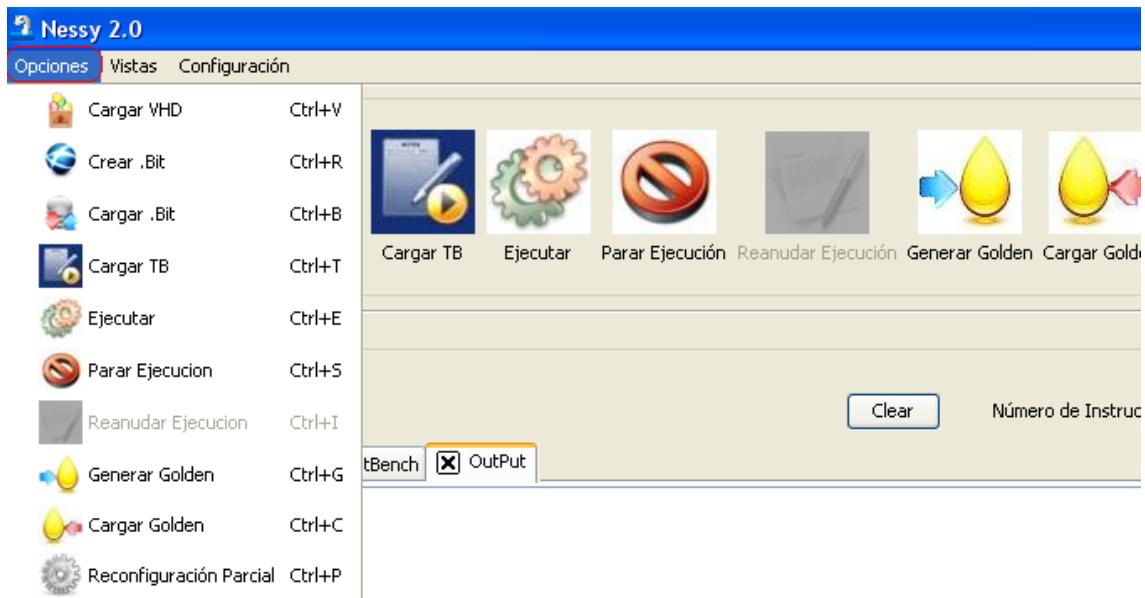


#### 3.2.1. Barra de menús

##### Opciones:

Se muestran un menú con toda la barra de botones que se muestra en la ventana. Si un elemento de la barra de botones se encuentra deshabilitado, en este menú aparece de forma análoga.

El efecto de pulsar cualquier acción desde el menú de opciones o desde la barra de botones tiene el mismo comportamiento.



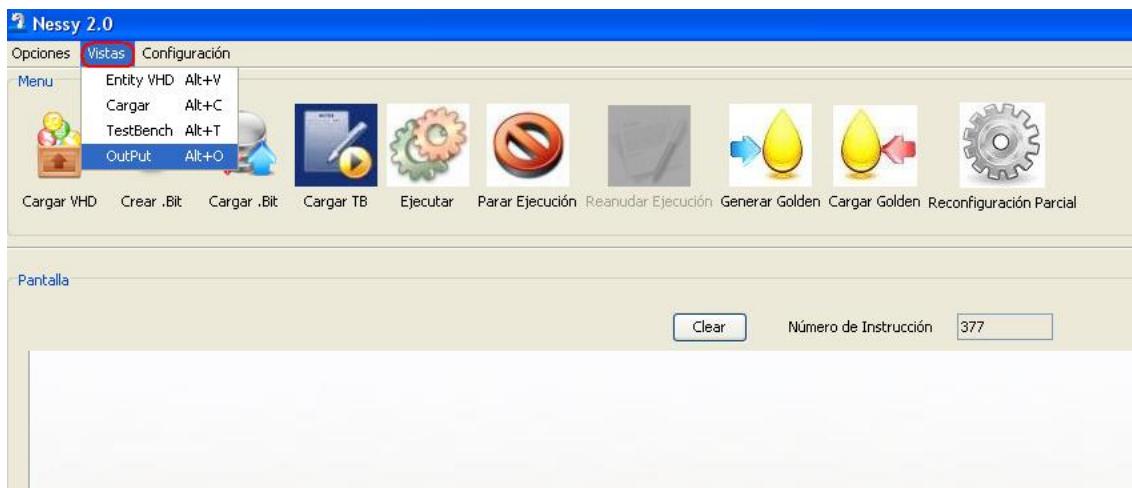
## Vistas:

Desde este menú se puede seleccionar de entre la vistas disponibles cual es la que deseamos ver. Si la vista se encontrara cerrada se abriría y se seleccionaría, si por el contrario se encuentra abierta, se pone como la vista seleccionada.

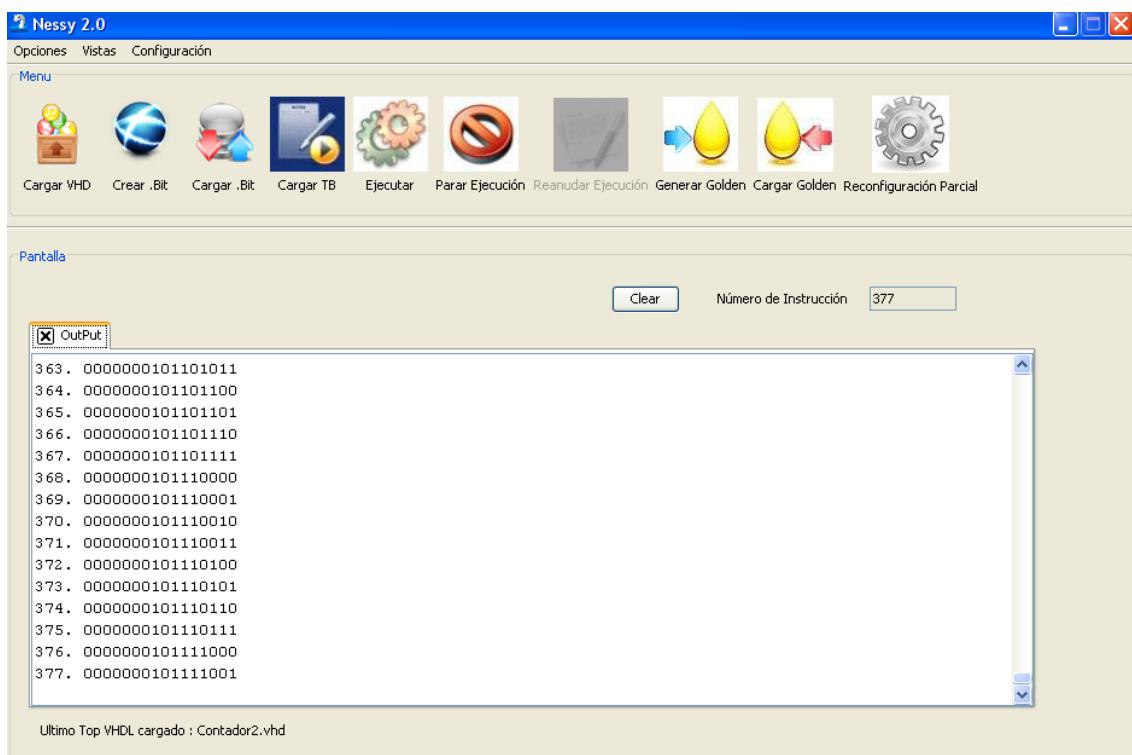
Tenemos 4 vistas disponibles que son:

1. **Entity VHDL** : En ella mostramos la entidad VHDL del archivo top con el que estamos trabajando.
2. **Cargar** : Mostramos la salida generada al cargar un archivo .bit en nuestra aplicación
3. **TestBench** : Mostramos el banco de pruebas, de forma editable, para que el usuario pueda modificarlo. Se ejecutará este banco de pruebas si previamente seleccionamos la opción de Cargar Test Bench en Pantalla.
4. **Output** : Se muestra la salida generada por la última ejecución.

Ej:



Al pulsar en Output aparece la nueva vista, en este caso no había ninguna vista abierta.

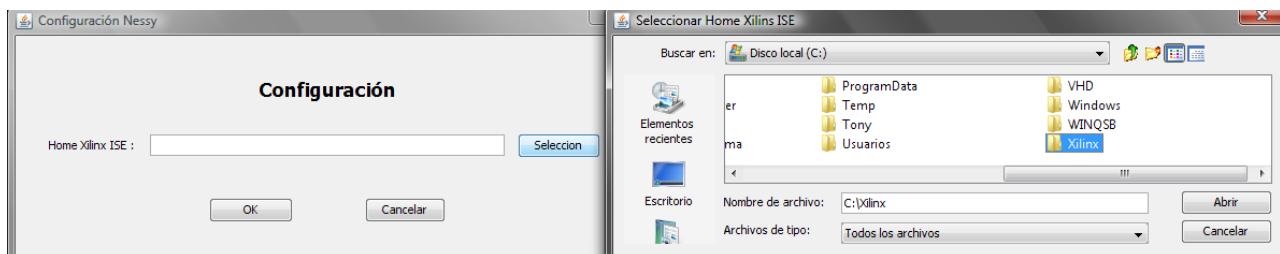


## Configuración:

En este menú podremos establecer la carpeta donde se encuentra la aplicación Xilinx instalada. Es importante que al seleccionarla sea la carpeta raíz de la herramienta Xilinx porque si no, no funcionará correctamente la aplicación. Tenemos dos opciones de configuración:

## Configurar Nessy :

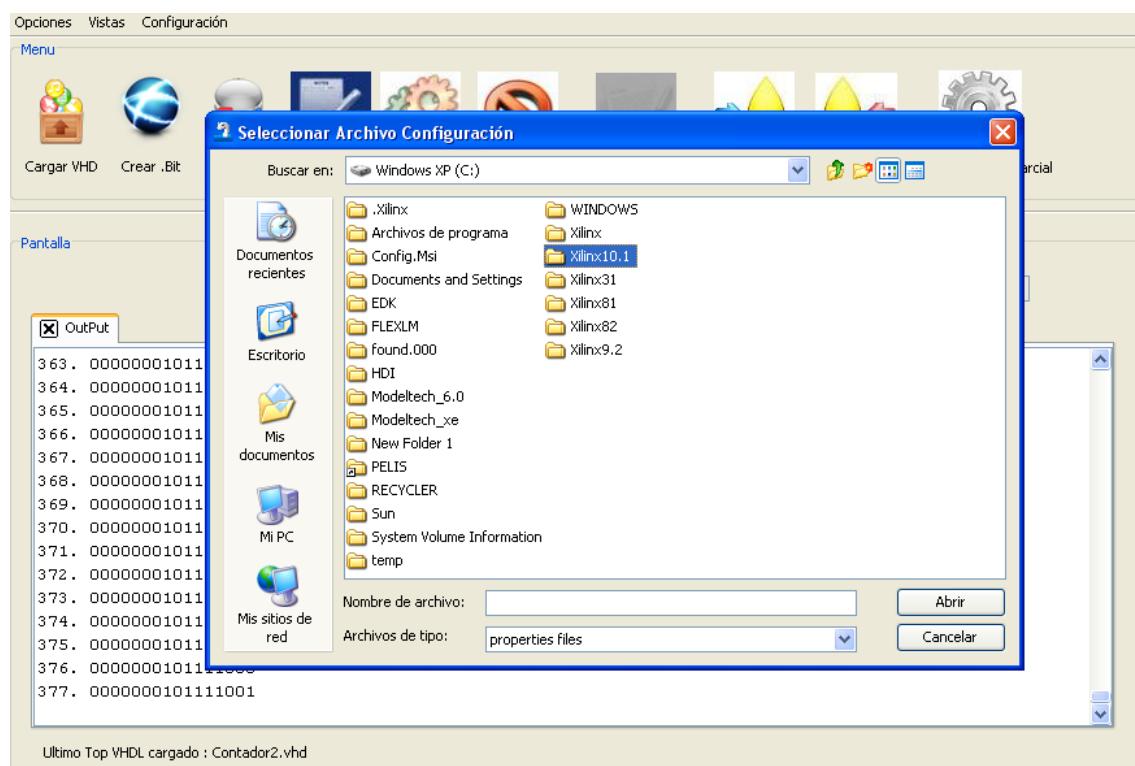
Se abre una ventana emergente, en cual podremos seleccionar el fichero raíz donde encuentra la aplicación Xilinx instalada. Deberemos seleccionarla dando al botón de selección



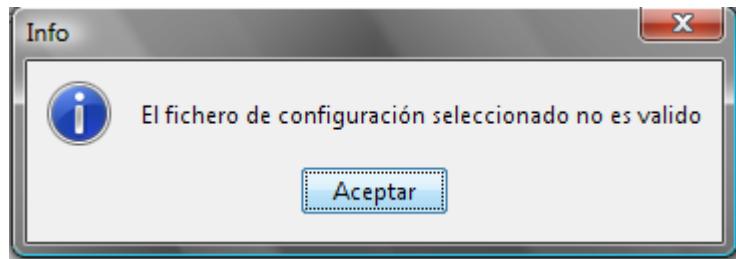
y pulsando sobre la carpeta elegida.

Si tuviéramos ya una dirección de Xilinx establecida esta aparecería al abrir la ventana ya escrita por defecto. Una vez elegida la nueva dirección bastaría con pulsar OK para guardarla.

## Cargar Fichero de Ejecución:



Se abre una ventana para seleccionar el fichero .properties para establecer las propiedades de nuestra aplicación. Si seleccionara un fichero que no tuviera el campo HomeXilinx daría el siguiente aviso.

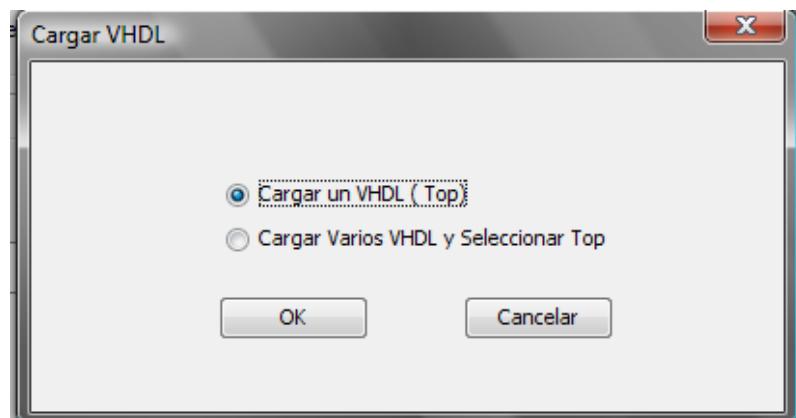


### 3.2.2. Barra de botones:

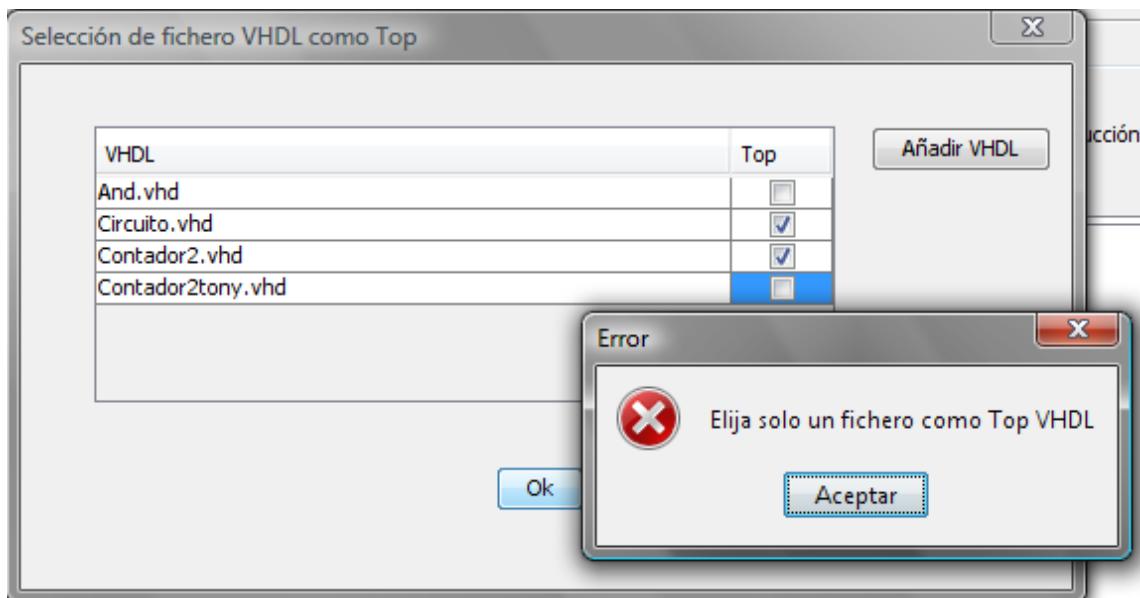
Desde ella se controlan los procesos más importantes de la herramienta. Al comenzar la aplicación solo estará activo el primer botón, porque para trabajar correctamente es necesario tener bien definida la entidad con la que deseamos trabajar.



Desde aquí cargamos los ficheros de tipo VHDL del proyecto con el que queramos trabajar. Al seleccionar esta opción se abrirá una ventana emergente. Si nuestra aplicación solo contiene un archivo VHDL deberemos seleccionar Cargar un VHDL y este será el archivo Top.

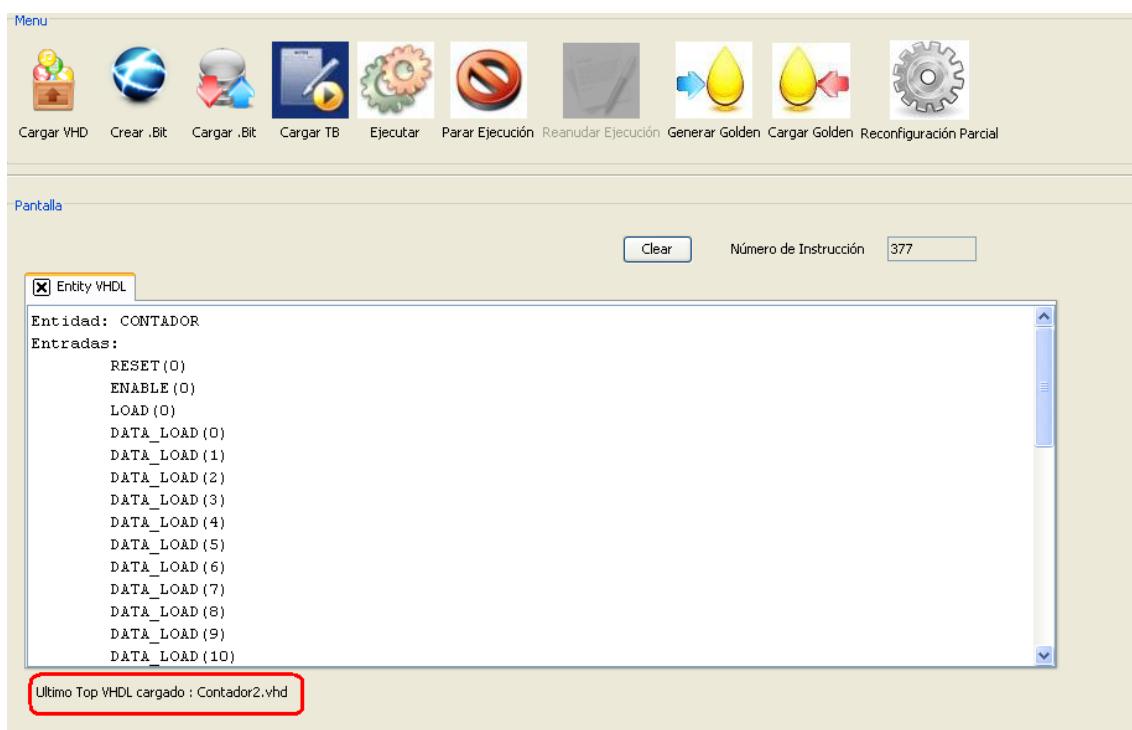


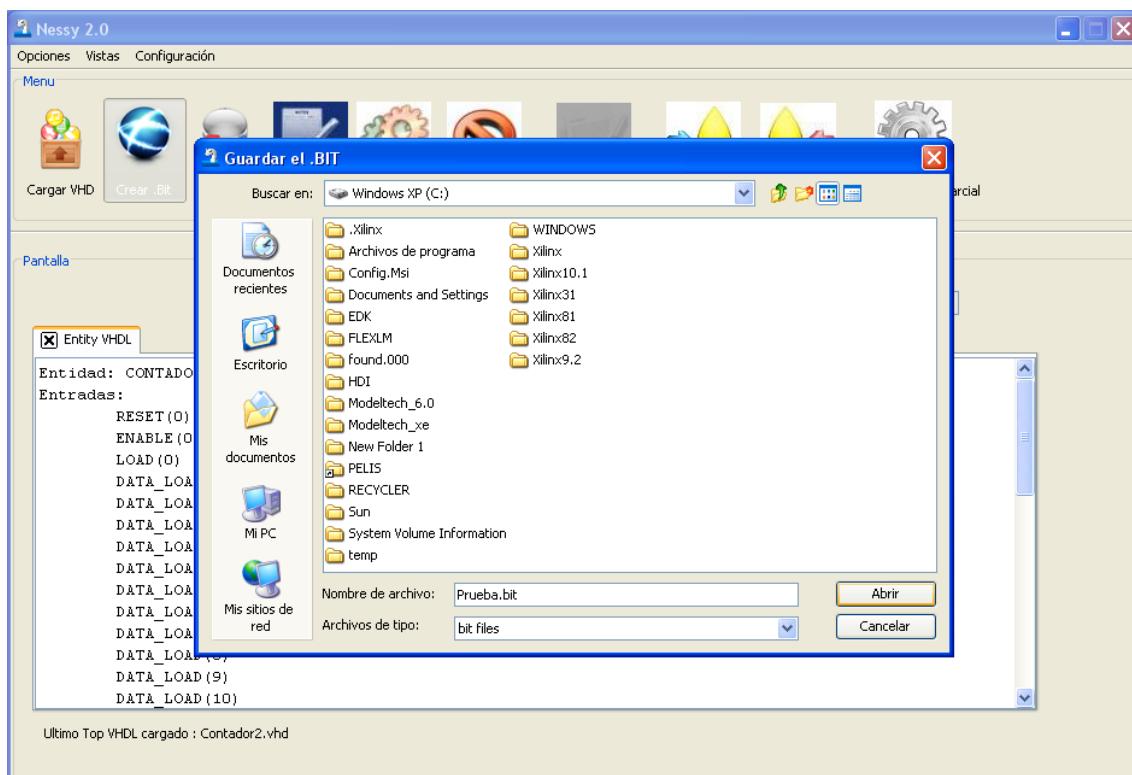
Si por el contrario, queremos seleccionar varios archivos utilizaremos la otra opción. En ella podremos añadir todos los ficheros VHDL que deseemos y de carpetas distintas. Podemos añadir cuantas veces deseemos. Es importante tener seleccionado un archivo como Top, y solamente uno, si no, no podremos continuar porque solo puede haber un fichero Top.



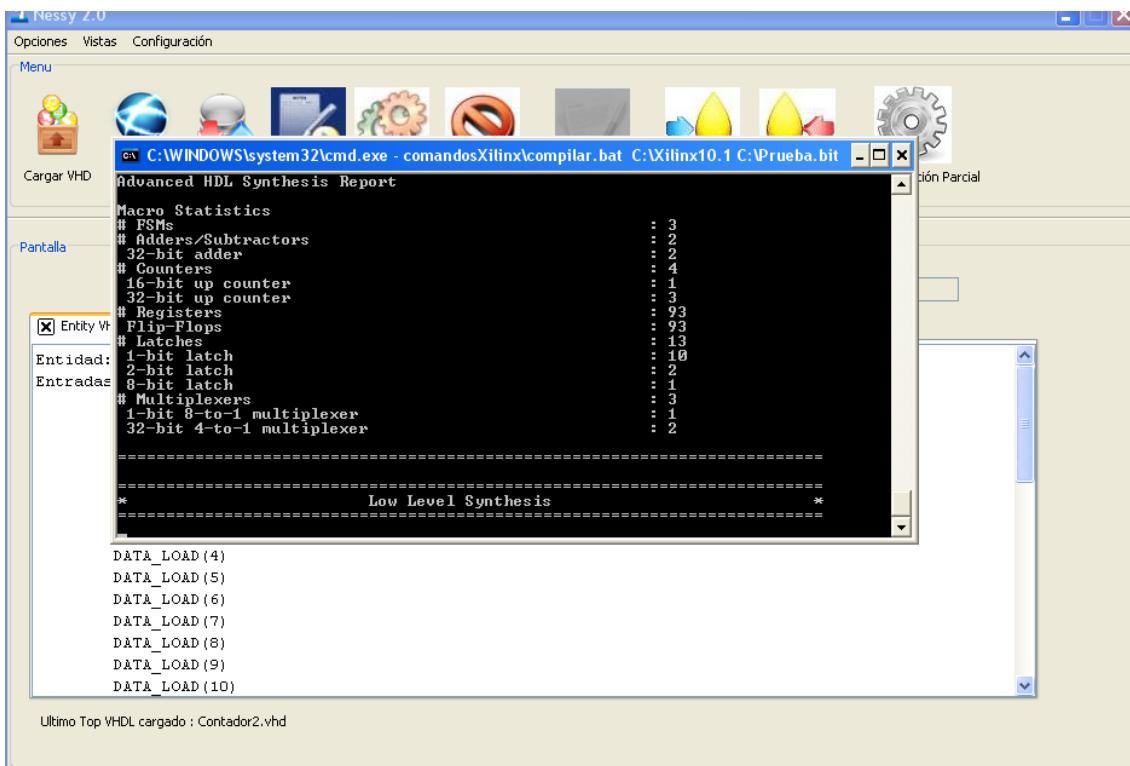
Una vez seleccionado los ficheros necesarios se cargará el archivo Top en la Vista Entity VHDL, para la cómoda visualización de la entidad principal.

El último archivo Top cargado correctamente se mostrará en la parte inferior de la ventana.





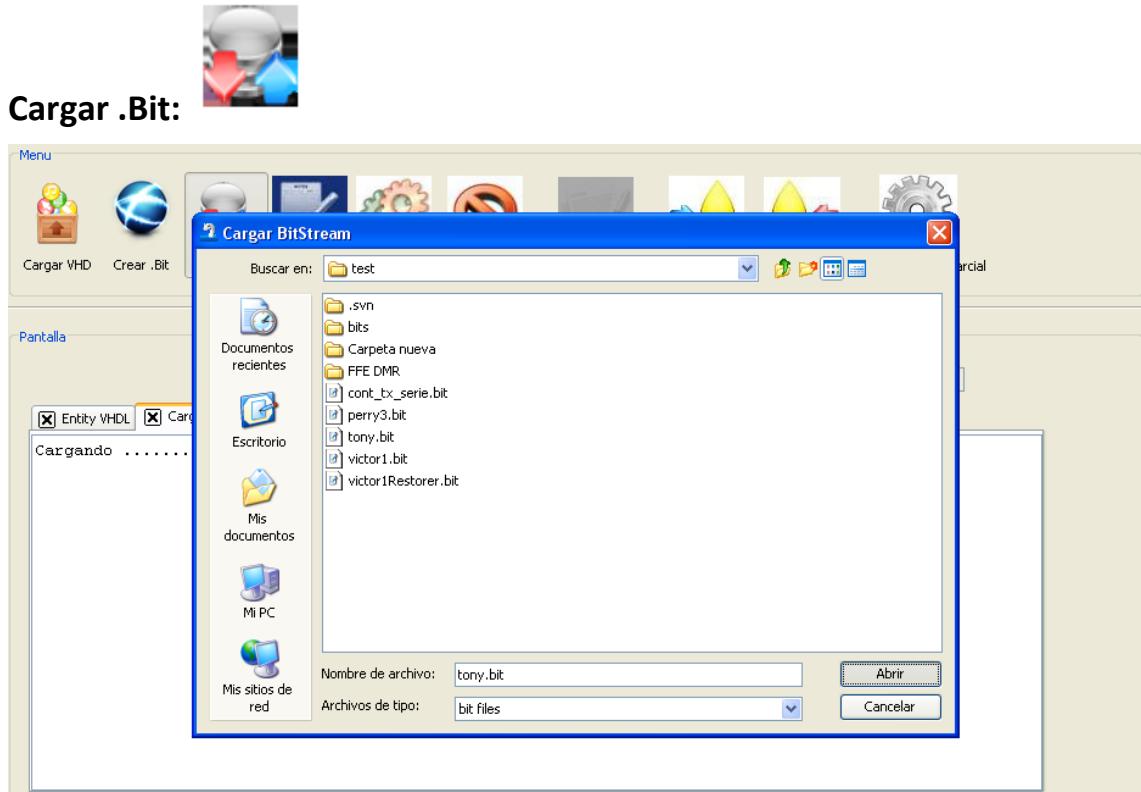
Este botón aparece deshabilitado hasta que no hemos pulsado Cargar VHDL y hemos definido la entidad correctamente.



El primer paso al pulsar este botón es escoger donde deseamos guardar el fichero .Bit y el nombre que tendrá el fichero. Se generará a partir de la entidad que hemos definido y siempre que no aparezca ningún error a la hora de generar el nuevo archivo.bit.

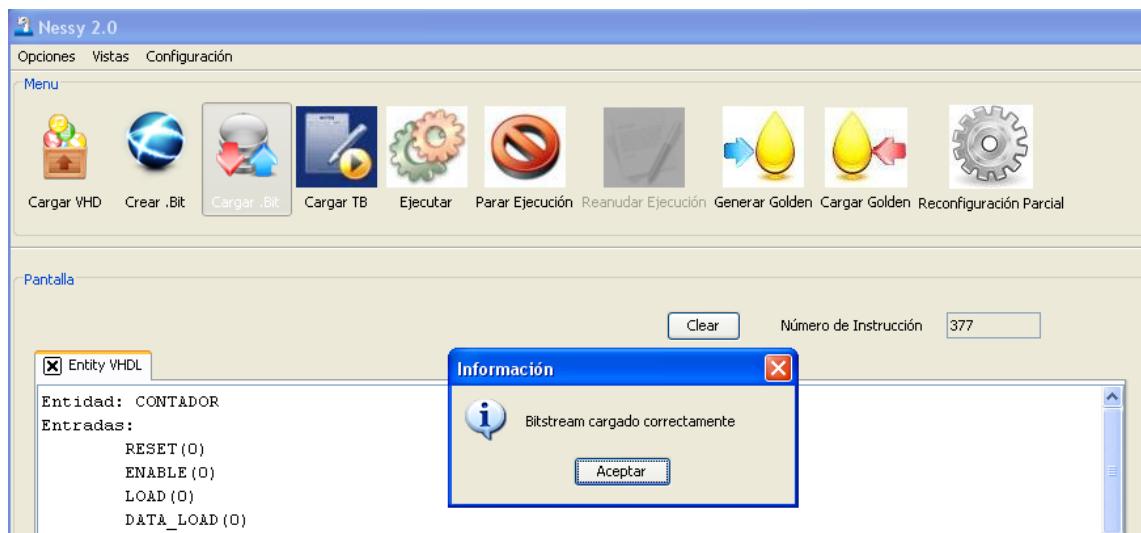
Este proceso es lento porque hace una acción similar a crear el .bit desde Xilinx. Se abrirá una consola de comandos que no se cerrará por si se desea consultar alguna etapa de la creación.

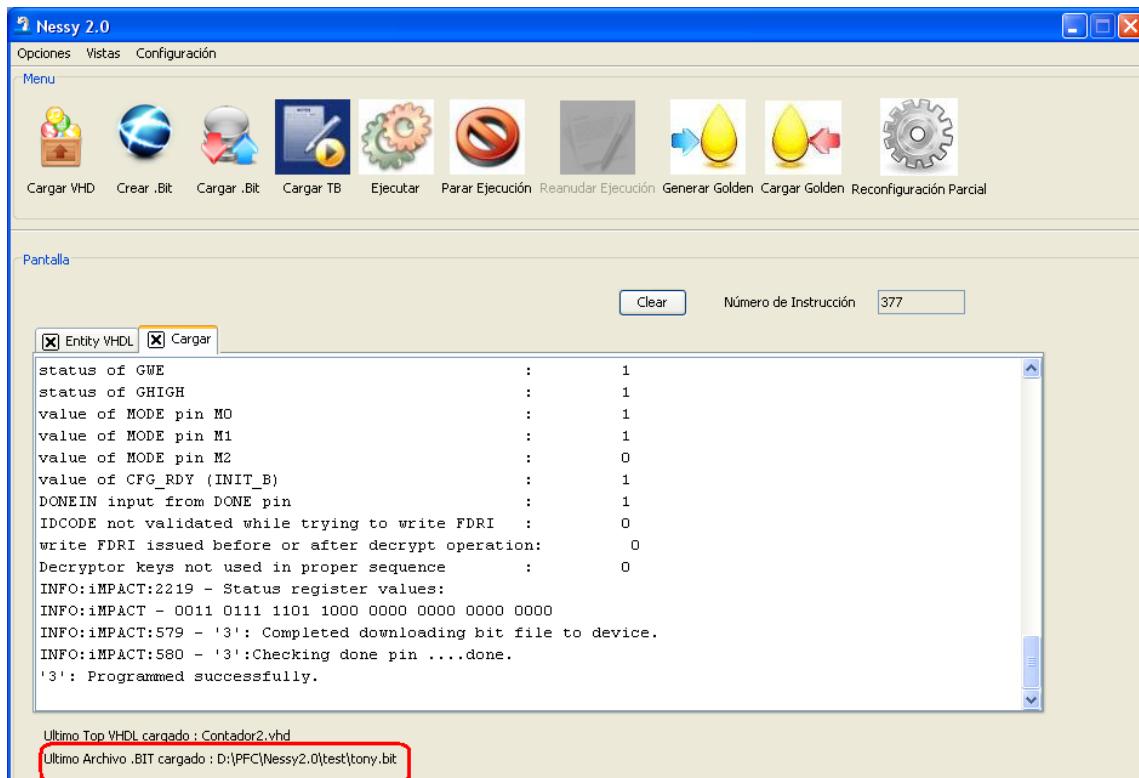
Internamente generamos un .bit creando un proyecto como haríamos en Xilinx añadiéndole los archivos Rx.vhd, Tx.vhd y un archivo que será TOP de todos. Con estos nuevos archivos proporcionamos una comunicación correcta con la FPGA siguiendo el protocolo del puerto serie. No importa el circuito que se declare como Top en nuestra herramienta siempre que no sobrepase las 32 entradas que tenemos ahora, que es límite superior de entradas que podemos declarar.



Desde esta opción podemos cargar el archivo .bit que elijamos en la ventana que se abrirá al pulsar este botón. Esta opción está disponible desde que tenemos cargados correctamente los ficheros VHDL, porque podemos estar en el caso que deseemos cargar un archivo .bit ya generado sin tener que crearlo de nuevo.

La salida que genere este proceso se mostrará por la vista Cargar y podremos ver si hemos tenido éxito en la carga del fichero en la FPGA.





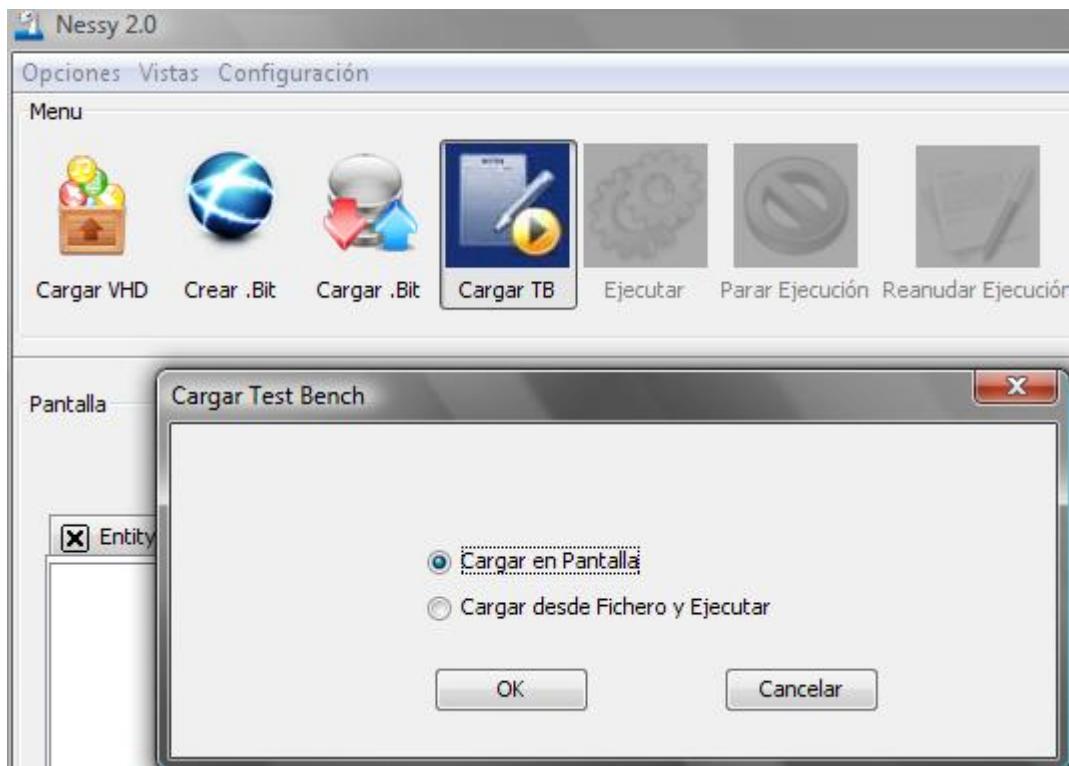
Aquí observamos que se ha actualizado el último archivo .bit cargado.



Cargar TB:

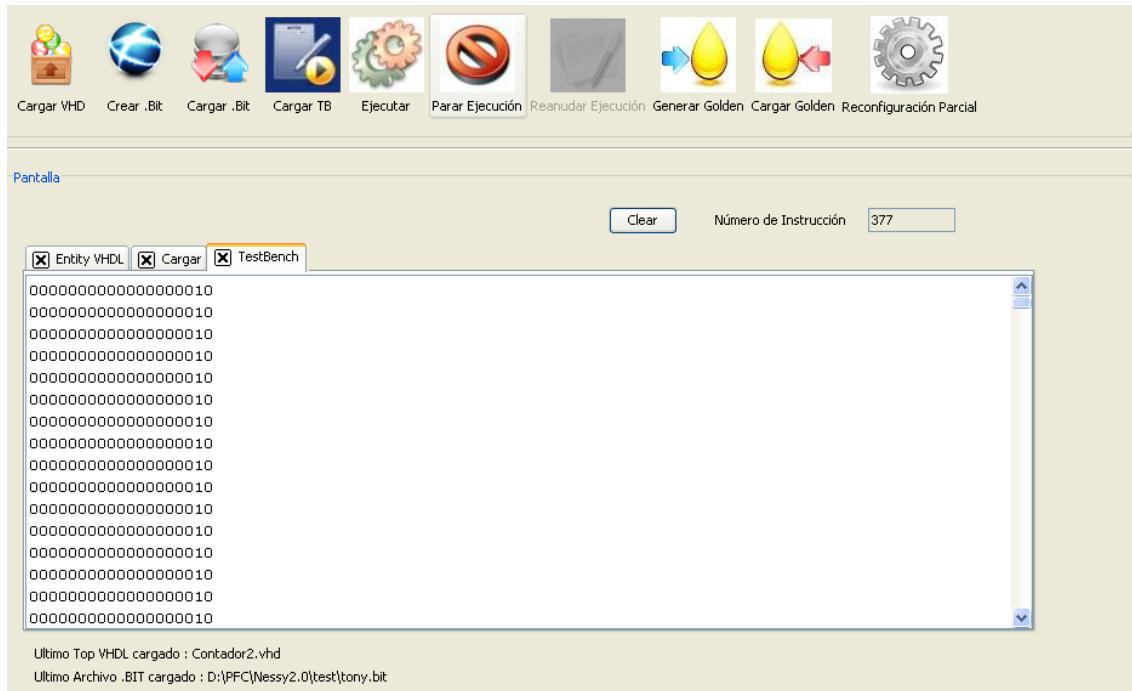
En este caso también esta opción de trabajo esta habilitada desde que tenemos cargados correctamente los ficheros VHDL, porque puede que ya tengamos cargado en la FPGA el fichero .bit con el que deseamos trabajar.

Se nos ofrecen dos opciones de trabajo:

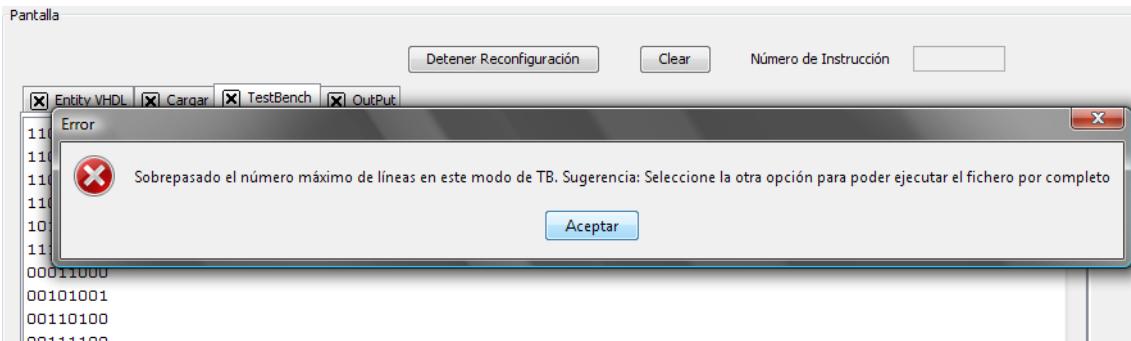


El último archivo .bit cargado correctamente también aparece en la parte inferior de la ventana.

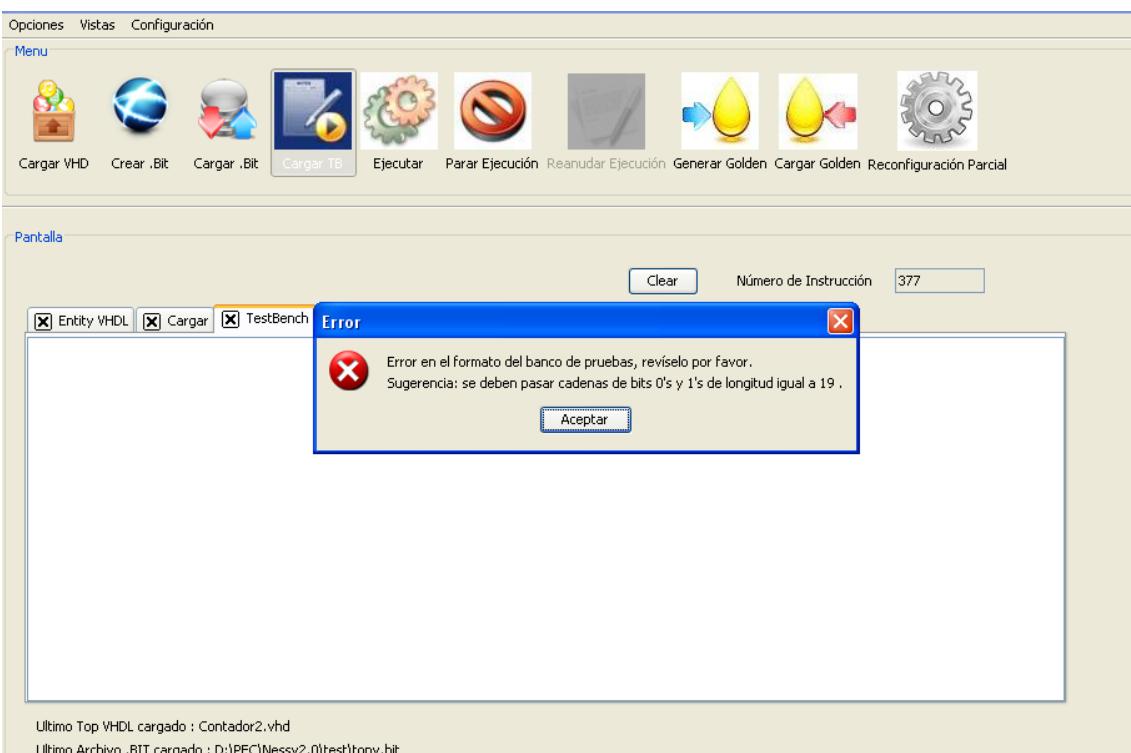
**Cargar en Pantalla:** Nos cargará en la vista Test Bench el archivo con extensión .txt que seleccionemos con el repertorio de instrucciones que deseemos mandar a la FPGA. Esta vista es editable por lo que se puede modificar las instrucciones cuantas veces deseemos.



Si el número de instrucciones supera las 250.000, no se podrá ejecutar esta opción por lo que deberemos seleccionar Cargar desde Fichero y Ejecutar.



**Cargar desde Fichero y Ejecutar:** Con esta opción al seleccionar el archivo de banco de pruebas elegido, comenzará la ejecución siempre que el formato de las instrucciones que contenga ese fichero esté en concordancia con la entidad Top que hayamos elegido, en caso contrario mostrará el correspondiente error.





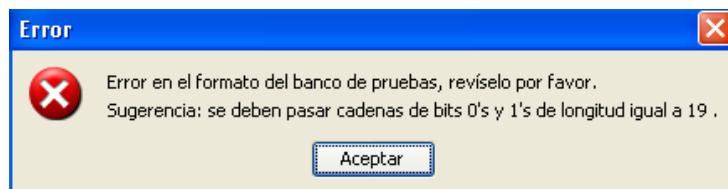
## Ejecutar:

Esta opción está habilitada, en el momento que hemos definido el banco de pruebas con el que queremos trabajar. Al pulsarlo se comienzan a enviar las instrucciones que tengamos en la vista del TestBench si hemos elegido cargar las instrucciones en pantalla, o las cogerá del fichero que le hallamos definido.

Es importante que el formato de las instrucciones sean cadenas de números binarios, con una longitud igual a la suma de las entradas del archivo Top que hemos definido al cargar la Entidad.

La primera entrada que tenemos definida se corresponde con el bit menos significativo de la cadena y la última entrada con el bit más significativo de la entrada. Esto es importante tenerlo en cuenta cuando definamos las instrucciones que queremos ejecutar.

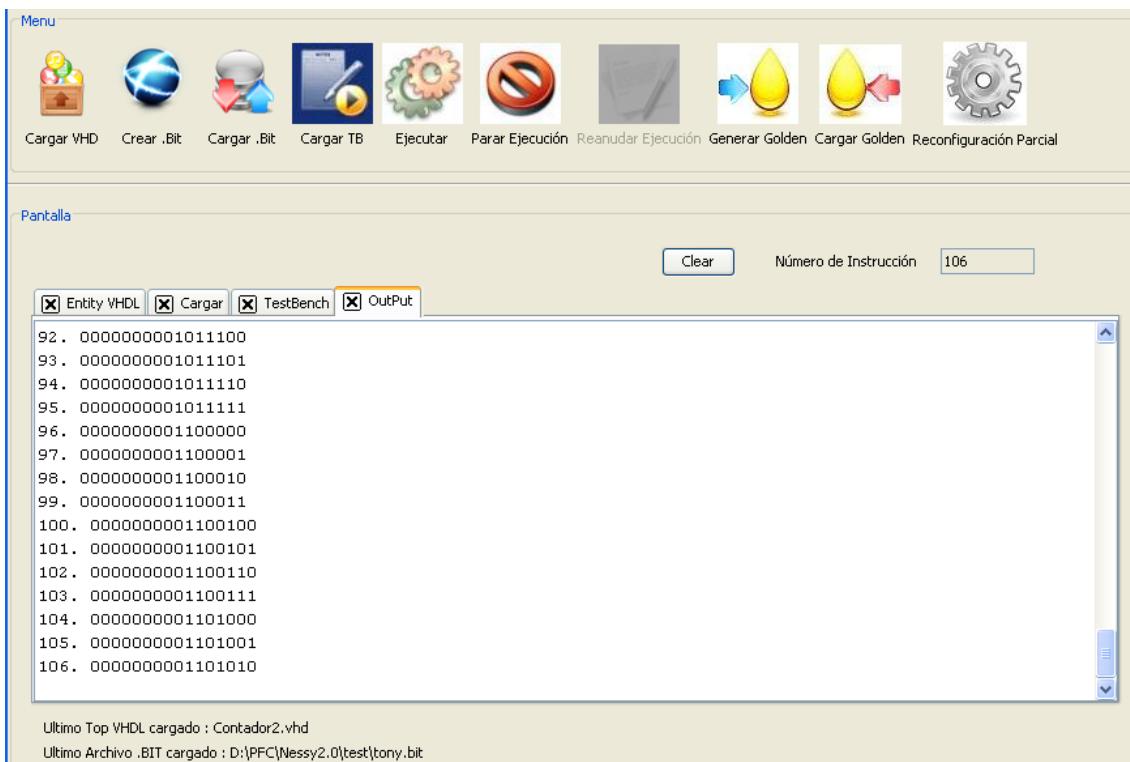
Si alguna de las cadenas contiene un carácter que no sea un número binario o la cadena tenga una longitud distintas del número de entradas, saltará un aviso y no se ejecutará nada.



La salida que compara con Golden que fichero de

la salida de la ejecución actual coincide con la Golden como si no se mostrará un mensaje de información. La salida que se genera al pulsar este botón se guarda en un archivo de texto llamado Salida.txt dentro de la carpeta de salidas.

se genera se la última salida se guarda en un texto. Tanto si



Durante la ejecución se puede observar el número de instrucción por el que va el proceso. Al pulsar ejecutar, y mientras dure la ejecución se habilita el botón Parar Ejecución para detener el proceso.

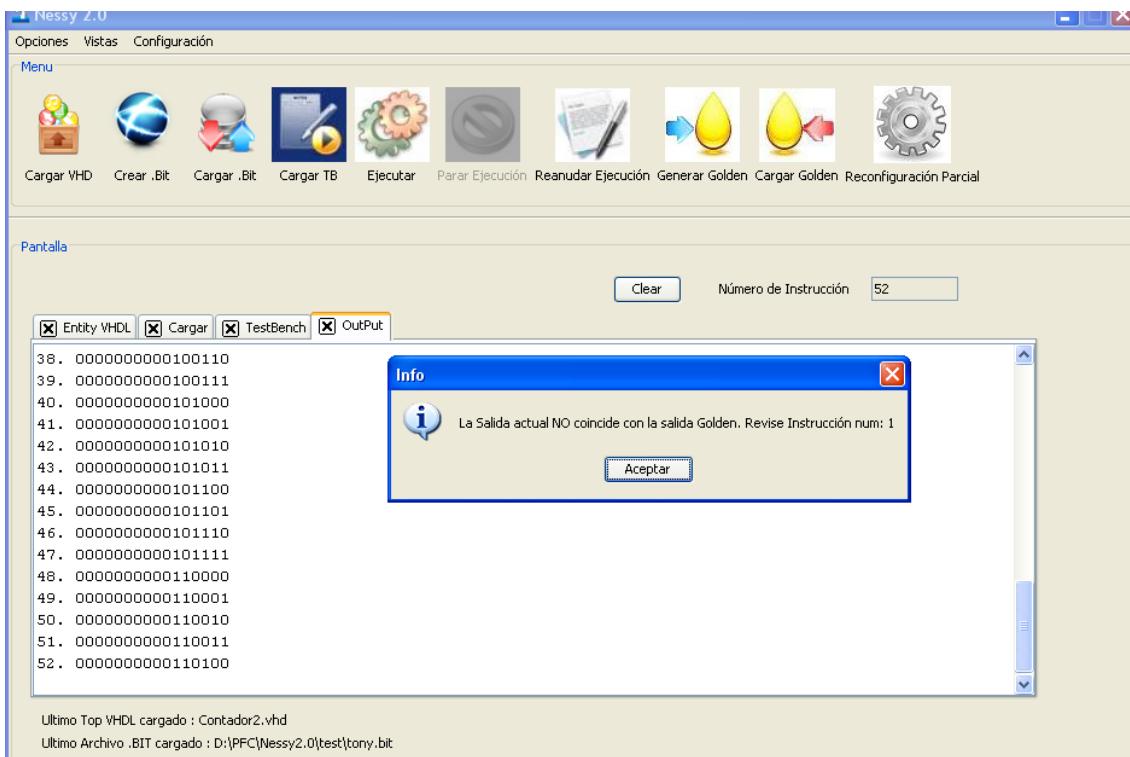


### Parar Ejecución:

Este botón se habilita mientras dura la ejecución de un banco de pruebas. Se detiene el envío de instrucciones y se deja de recibir. Podemos observar en la instrucción que nos hemos quedado.

El parar la ejecución no afecta en absoluto a la salida que se produzca, al menos que se modifique el archivo .bit cargado en la placa o el fichero con las instrucciones que se están enviando a la FPGA.

Si la salida actual que se está generando no coincide con la salida Golden al pulsar este botón nos alertaría ya de este hecho, indicándonos también la instrucción que causó la diferencia.



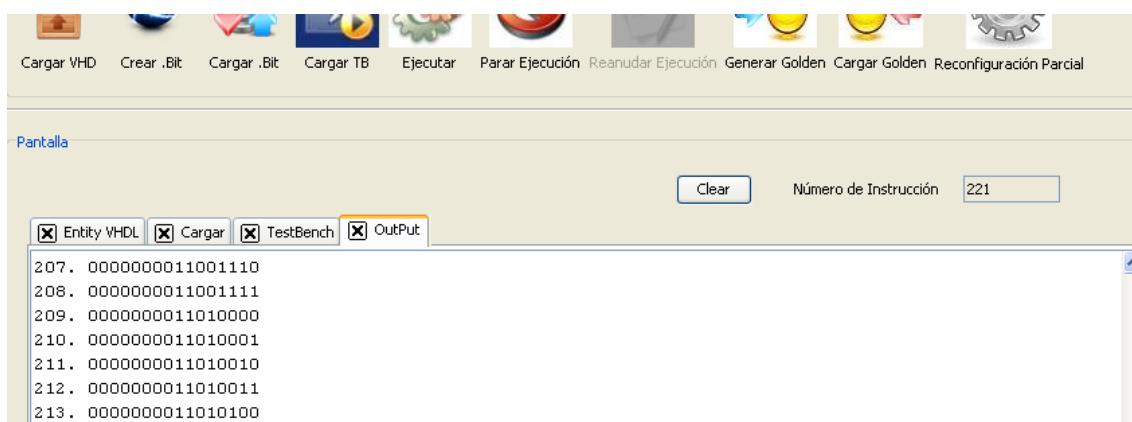
Al pulsar este botón se deshabilita el propio botón y habilita el botón de reanudar ejecución. Vuelve estar habilitado al volver al pulsar ejecución, o al pulsar reanudar ejecución.



### Reanudar Ejecución:

Este botón esta habilitado cuando hemos parado la ejecución, y permite continuar la ejecución en el punto donde se estaba antes de pulsar parar ejecución.

Este botón solo esta habilitado mientras dura la ejecución y después de haber parado la ejecución. El efecto al pulsarlo es que habilita parar ejecución y prosigue el envío de instrucciones a la FPGA.





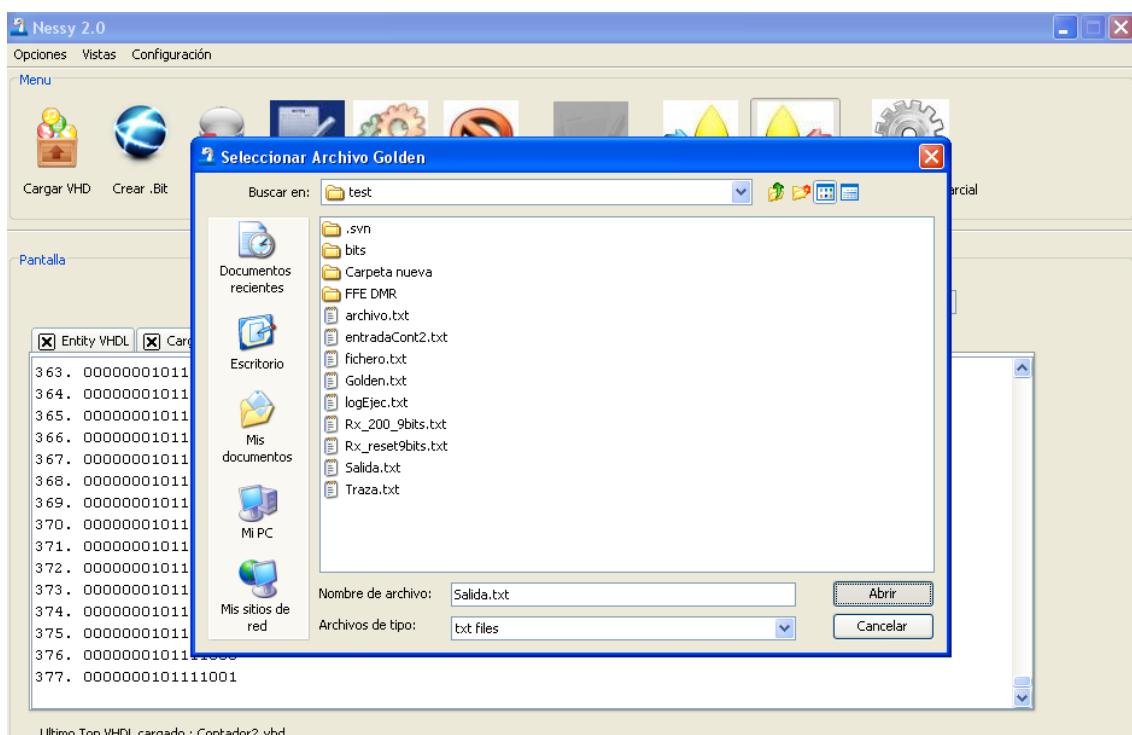
### Generar Golden:

Este botón queda habilitado después de que tengamos definido el conjunto de instrucciones del banco de pruebas que deseamos enviar a la placa. Su funcionalidad es la de una ejecución pero guarda la salida en Golden.txt y será el fichero de referencia para comparar el resto de ejecuciones y ver si ha habido algún error entre las distintas ejecuciones.



### Cargar Golden:

Se encuentra habilitado al definir el cargar el banco de pruebas. Al pulsar este botón se abrirá una ventana para elegir el fichero que a partir de ese momento será nuestra salida Golden con la que comparemos todas las ejecuciones.

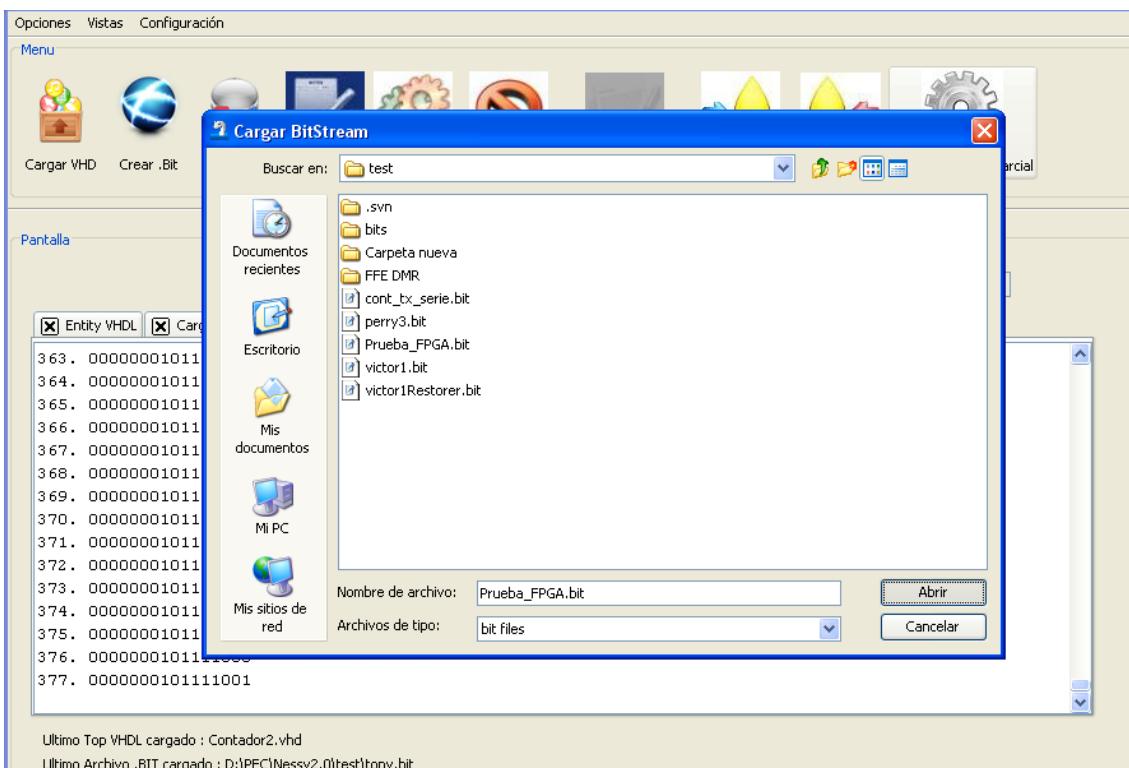


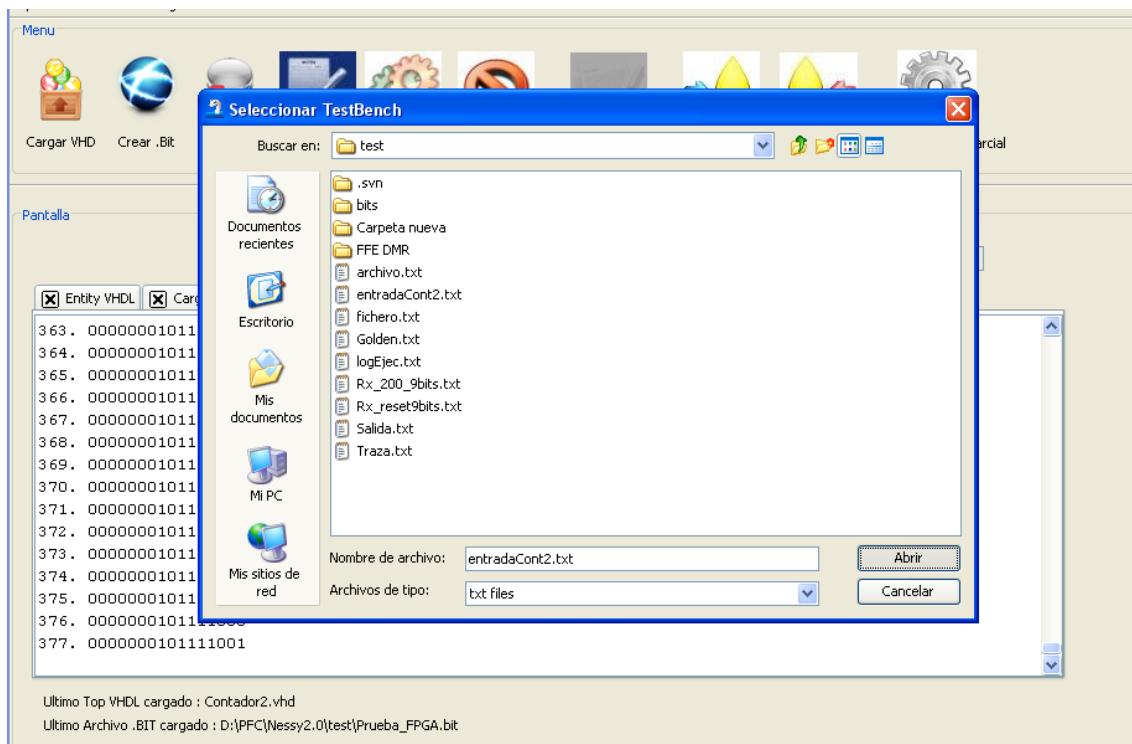


## Reconfiguración Parcial:

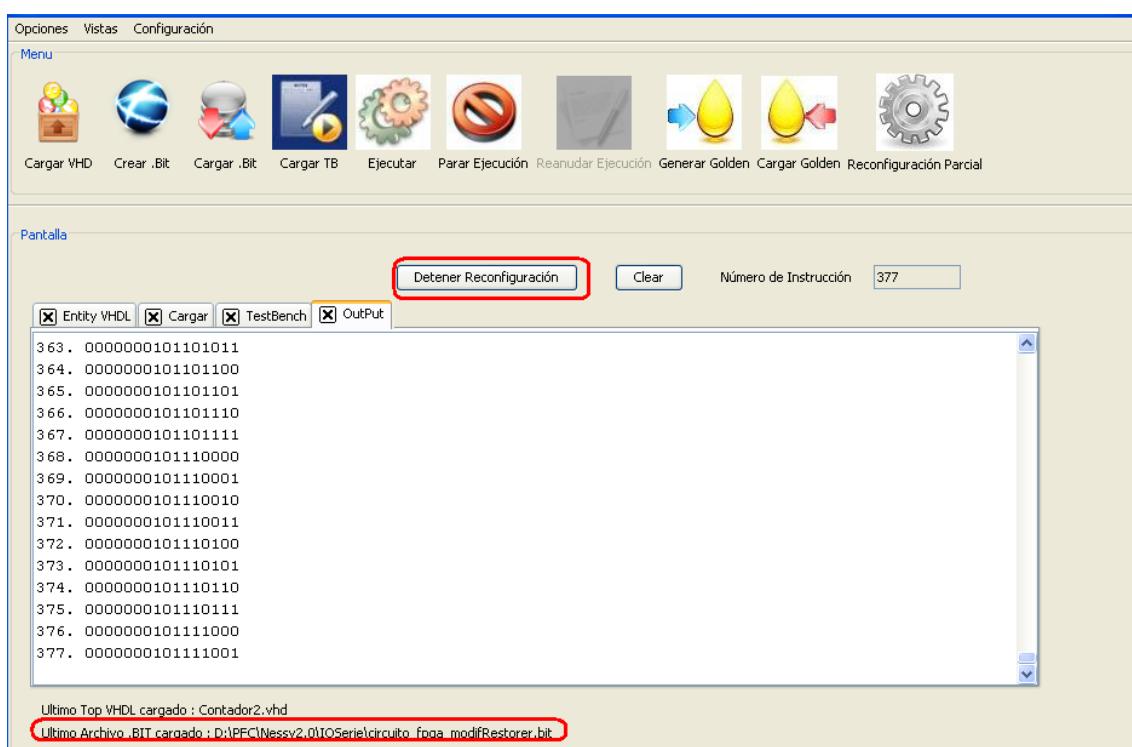
Este botón se encuentra habilitado al tener definida la entidad con la que vamos a trabajar. La funcionalidad es ir modificando bit a bit el archivo .bit que elijamos al principio para ir viendo como afecta la modificación de ese bit en la salida.

Al pulsar el botón elegiremos el archivo .bit que vamos a tomar como referencia y sobre el que haremos las modificaciones. Posteriormente tendremos que seleccionar el fichero de banco de pruebas con el repertorio de instrucciones que se van a ejecutar.





En un primer momento se hará una ejecución sin modificar el archivo para generar la salida Golden, y así, cuando vayamos modificando el archivo cargado en la FPGA ver como ha afectado a la ejecución.



La salida que genera este proceso se guarda en un fichero logEjec.txt donde se puede revisar en que punto la ejecución fue distinta porque el bit modificado afectó a la salida.

Es importante tener en cuenta que como es proceso largo no se muestran ventanas emergentes con el resultado de la comparación entre la salida Golden y la última ejecución que se ha producido por comodidad para el usuario.

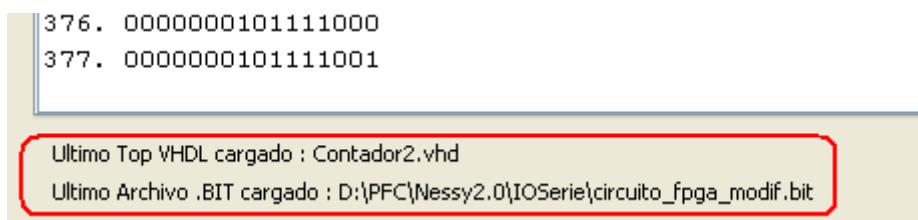
Al pulsar este botón se habilita un botón (Detener Reconfiguración) debajo de la barra de botones que permite abortar la reconfiguración parcial, por si el usuario decide terminar el proceso.

#### Más detalles a tener en cuenta:

En la primera ejecución de la aplicación se abrirá la ventana para definir la ruta HomeXilinx y hasta que no la definamos no podremos acceder a la ventana principal.

El botón de Clear situado debajo de la barra de botones sirve para limpiar el contenido de la vista que actual.

El último .bit y el último archivo Top utilizados durante el uso de la aplicación se detallan en la parte inferior de la ventana principal.



#### Generación de un Test Bench correcto:

Se trata ya sea cargado en pantalla o desde fichero de cadenas binarias. Tienen que tener la misma longitud que el número de entradas que tengamos definidas en el archivo VHDL que hemos cargado como TOP.

La última entrada declarada en la entidad TOP se corresponde con el bit más significativo, el siguiente bit se correspondería con la penúltima entrada declarada, y así sucesivamente hasta que llegamos a la primera entrada declarada que sería el bit menos significativo.

Ejemplo :

```
entity MiEntidadInventada is
    Port ( a : in STD_LOGIC;
            b : in STD_LOGIC;
            c : in STD_LOGIC;
            z : out STD_LOGIC);
end MiEntidadInventada;
```

Aquí solo tenemos tres entradas, deberíamos generar cadenas de longitud 3.

Para a = 0, b = 1, c = 1 deberíamos enviar 110

Para a = 1, b = 1, c = 0 deberíamos enviar 011

### **Teclas Rápidas:**

Para facilitar la interfaz con el usuario, existe la siguiente combinación de teclas rápidas:

#### **Opciones:**

- Cargar VHD → Control + V
- Crear .Bit → Control + R
- Cargar .Bit → Control + B
- Cargar TB → Control + T
- Ejecutar → Control + E
- Parar Ejecución → Control + S
- Reanudar Ejecución → Control + I
- Generar Golden → Control + G
- Cargar Golden → Control + C
- Reconfiguración Parcial → Control + P

#### **Vistas:**

- Entity VHD → Alt + V
- Cargar → Alt + C
- TestBench → Alt + T
- Output → Alt + O

## 4. Código fuente

### 4.1. Ficheros VHDL

#### RX\_Serie

```
-----  
-----  
-- Company:  
-- Engineer:  
--  
-- Create Date:      18:31:36 11/18/2009  
-- Design Name:     Rx_Serie - Behavioral  
-- Project Name:  
-- Target Devices:  
-- Tool versions:  
-- Description:  
--  
-- Dependencies:  
--  
-- Revision:  
-- Revision 0.01 - File Created  
-- Additional Comments:  
--  
-----  
-----  
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.STD_LOGIC_ARITH.ALL;  
use IEEE.STD_LOGIC_UNSIGNED.ALL;  
  
---- Uncomment the following library declaration if instantiating  
---- any Xilinx primitives in this code.  
--library UNISIM;  
--use UNISIM.VComponents.all;  
  
entity Rx_Serie is  
  generic (  
    gFrecClk : integer :=100000000; --31125; --100MHz  
    gBaud : integer :=9600--3 --9600bps  
  );  
  Port ( Rstn : in STD_LOGIC;  
          Clk : in STD_LOGIC;  
          RxDataSerie : in STD_LOGIC;  
          DataRxOut : out STD_LOGIC_VECTOR (7 downto 0);  
          AvisoRx : out STD_LOGIC;  
          Recibiendo : out STD_LOGIC);  
end Rx_Serie;  
  
-----  
-----  
architecture Behavioral of Rx_Serie is
```

```

-- Tipo nuevo para los estados --
type t_Estado is (eInit,eBitInit,eBitsData,eBitFin);
constant cFinCuenta : natural := (gFrecClk/gBaud)-1;
constant cFinCuenta_Medio : natural := cFinCuenta / 2 ;
-- Señales

signal Cuenta,CuentaBits: integer;
signal ClkBaud,EnableCont,Dsplza,FinDspaza8bits,ClkBaudMedio :
std_logic;

signal Estado : t_Estado;
signal Rx_Registro: STD_LOGIC_VECTOR (7 downto 0);
signal MiDatoTxIn:STD_LOGIC_VECTOR (7 downto 0);
signal SalidaSeleccion:std_logic;
signal RxDataReg : std_logic;

begin
-----
-----  

-- Proceso encargado de dividir la frecuencia del reloj de la  

placa y  

-- generar ClkBaud

P_DivFrec: Process (RstN, Clk)
begin
    if RstN = '0' then
        Cuenta <= 0;
        ClkBaud <= '0';
        ClkBaudMedio <= '0';
        --Rx_Registro<="11111111";----?¿?
    elsif EnableCont = '1' then
        if Clk'event and Clk='1' then
            if Cuenta = cFinCuenta then
                Cuenta <= 0;
                ClkBaud <= '1';
                ClkBaudMedio <= '0';
            elsif Cuenta = cFinCuenta_Medio then
                ClkBaudMedio <= '1';
                Cuenta <= Cuenta + 1;
                ClkBaud <= '0';
            else
                Cuenta <= Cuenta + 1;
                ClkBaud <= '0';
                ClkBaudMedio <= '0';
            end if;
        end if;
    end if;
end process;
-----  

-----  

-- Proceso encargado de contar los 8 bits de datos.

P_CuentaBits: Process (RstN, Clk)
begin
    if RstN = '0' then

```

```

        CuentaBits <= 0;
elsif Clk'event and Clk='1' then
    if Estado = eBitsDato then
        if ClkBaud = '1' then
            if CuentaBits = 7 then
                CuentaBits <= 0;
            else
                CuentaBits <= CuentaBits + 1;
            end if;
        end if;
    else
        CuentaBits <= 0;
    end if;
end if;
end process;

-----
-----

-- Proceso que controla las señales internas necesarias para el
-- registro de desplazamiento y para el contador de los 8 bits de
datos

POut: Process (Estado, RxDataReg, ClkBaudMedio,ClkBaud) --
RxDataSerie
begin
    EnableCont <= '0';
    Dsplza <= '0';
    --recibiendo <= '1';
    case Estado is
        when eInit =>
            EnableCont <= '0';
            Recibiendo <= '0';
            AvisoRx <= '0';
            Dsplza <= '0';
            if RxDataReg = '0' then
                AvisoRx <= '1';
                EnableCont <= '1';
            end if;
        when eBitInit =>
            EnableCont <= '1';
            Recibiendo <= '1';
        when eBitsDato =>
            EnableCont <= '1';
            Recibiendo <= '1';
            if ClkBaudMedio = '1' then
                Dsplza <= '1';
            end if;
        when eBitFin =>
            Recibiendo <= '1';
            if ClkBaud = '1' then
                EnableCont <= '0';
            else
                EnableCont <= '1';
            end if;
    end case;
end process;
-----
```

```

-- Proceso de la máquina de estados

P_Control_FSM: Process (RstN, Clk)
begin
    if RstN = '0' then
        Estado <= eInit;
    elsif Clk'event and Clk='1' then
        case Estado is
            when eInit =>
                if RxDataReg = '0' then
                    Estado <= eBitInit;
                end if;
            when eBitInit =>
                if ClkBaud = '1' then
                    Estado <= eBitsDato;
                end if;
            when eBitsDato =>
                if FinDsplza8bits = '1' then
                    Estado <= eBitFin;
                end if;
            when eBitFin =>
                if ClkBaud = '1' then
                    Estado <= eInit;
                end if;
        end case;
    end if;
end process;

-----
-----

-- Proceso que se encarga de ir poniendo en SalReg el bit del
registro
-- de desplazamiento concecreto.

recibe_desplazamiento: process(Dsplza)
begin
    if Estado = eBitsDato and Dsplza = '1' then
        Rx_Registro(CuentaBits) <= RxDataReg ;
    end if;
end process;

-----
-----

-- Proceso que funciona como un biestable con la señal de entrada
-- RxDataReg seleccionando un 1 cuando no hay nada en la línea.

biestable: process(Clk,RstN)
begin
    if (RstN = '0') then
        RxDataReg <= '1';
    elsif (Clk'event and Clk = '1') then
        RxDataReg <= RxDataSerie;
    end if;
end process;

DatoRxOut <= Rx_Registro;
FinDsplza8bits <= '1' when CuentaBits=7 and ClkBaud = '1' else '0';

end Behavioral;

```

## Tx\_Serie

```
-----  
-- Company:  
-- Engineer:  
--  
-- Create Date:      19:27:16 11/05/2009  
-- Design Name:     Tx_serie - Behavioral  
-- Project Name:  
-- Target Devices:  
-- Tool versions:  
-- Description:  
--  
-- Dependencies:  
--  
-- Revision:  
-- Revision 0.01 - File Created  
-- Additional Comments:  
--  
-----  
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.STD_LOGIC_ARITH.ALL;  
use IEEE.STD_LOGIC_UNSIGNED.ALL;  
  
---- Uncomment the following library declaration if instantiating  
---- any Xilinx primitives in this code.  
--library UNISIM;  
--use UNISIM.VComponents.all;  
  
-----  
entity Tx_serie is  
--  
  generic (  
    gFrecClk : integer := 100000000; --31125;-- --  
    100MHz  
    gBaud : integer := 9600 --9600bps  
  );  
  Port ( RstN : in STD_LOGIC;  
         clk : in STD_LOGIC;  
         Transmite : in STD_LOGIC;  
         DatoTxIn : in STD_LOGIC_VECTOR (7 downto 0);  
         Transmitiendo : out STD_LOGIC;  
         DatoSerieOut : out STD_LOGIC);  
  end Tx_serie;  
  
-----  
architecture Behavioral of Tx_serie is
```

```

-- Tipo nuevo para los estados --
type t_Estado is (eBitInit,eBitsData,eBitFin,eInit);
constant cFinCuenta : natural := (gFrecClk/gBaud)-1;

-- Señales

signal Cuenta,CuentaBits: integer;
signal ClkBaud,EnableCont,Dsplza,FinDspaza8bits,CargaDatos : std_logic;
signal Estado : t_Estado;
signal Registro: STD_LOGIC_VECTOR (7 downto 0);
signal SalReg: std_logic;
signal MiDatosTxIn:STD_LOGIC_VECTOR (7 downto 0);
signal SalidaSeleccion:std_logic;

begin
-----
-- Proceso encargado de dividir la frecuencia del reloj de la
placa y
-- generar ClkBaud

P_DivFrec: Process (RstN, Clk)
begin
    if RstN = '0' then
        Cuenta <= 0;
        ClkBaud <= '0';
    elsif Clk'event and Clk='1' then
        if EnableCont = '1' then
            if Cuenta = cFinCuenta then
                Cuenta <= 0;
                ClkBaud <= '1';
            else
                Cuenta <= Cuenta + 1;
                ClkBaud <= '0';
            end if;
        end if;
    end if;
end process;
-----
-- Proceso encargado de contar los 8 bits de datos.

P_CuentaBits: Process (RstN, Clk)
begin
    if RstN = '0' then
        CuentaBits <= 0;
    elsif Clk'event and Clk='1' then
        if Estado = eBitsData then
            if ClkBaud = '1' then
                if CuentaBits = 7 then
                    CuentaBits <= 0;
                else
                    CuentaBits <= CuentaBits + 1;
                end if;
            end if;
        end if;
    end if;

```

```

                end if;
            else
                CuentaBits <= 0;
            end if;
        end if;
    end process;

-----
-----
```

-- Proceso que controla las señales internas necesarias para el registro de desplazamiento y para el contador de los 8 bits de datos

```

POut: Process (Estado, Transmite, ClkBaud)
begin
    Dsplza <= '0';
    CargaDato <= '0';
    EnableCont <= '1';

    case Estado is
        when eInit =>
            EnableCont <= '0';
            Transmitiendo <= '0';
            if Transmite = '1' then
                CargaDato <= '1';
                EnableCont <= '1';
            end if;
        when eBitInit => Transmitiendo <= '1';
        when eBitsDato => Transmitiendo <= '1';
        --Transmitiendo <= '0';
            if ClkBaud = '1' then
                Dsplza <= '1';
            end if;
        when eBitFin =>
            Transmitiendo <= '1';
            if ClkBaud = '1' then
                EnableCont <= '0';
            end if;
    end case;
end process;
```

```

-----
-----
```

-- Proceso de la máquina de estados

```

P_Control_FSM: Process (RstN, Clk)
begin
    if RstN = '0' then
        Estado <= eInit;
    elsif Clk'event and Clk='1' then
        case Estado is
            when eInit =>
                if Transmite = '1' then
                    Estado <= eBitInit;
                end if;
            when eBitInit =>
                if ClkBaud = '1' then
                    Estado <= eBitsDato;
```

```

                end if;
when eBitsData =>
    if FinDsplza8bits = '1' then
        Estado <= eBitFin;
    end if;
when eBitFin =>
    if ClkBaud = '1' then
        Estado <= eInit;
    end if;
end case;
end if;
end process;
-----
-----

-- Proceso que se encarga de ir poniendo en SalReg el bit del
registro
-- de desplazamiento concecreto.

carga_desplazamiento: process(Dsplza,CargaDatos)
begin
    if CargaDatos='1' then
        Registro <= DatoTxIn;
    else
        SalReg <= Registro(CuentaBits);
    end if;
end process;

-----
-- Proceso encargado de elegir la salida en función del Estado
en el que
-- estemos

seleccion: process(Estado)
begin
    if Estado = eInit then
        SalidaSeleccion <= '1';
    elsif Estado = eBitInit then
        SalidaSeleccion <= '0';
    elsif Estado = eBitsData then
        SalidaSeleccion <= SalReg;
    elsif Estado = eBitFin then
        SalidaSeleccion <= '1';
    end if;
    -- end if;
end process;

DatoSerieOut <= SalidaSeleccion;
FinDsplza8bits <= '1' when CuentaBits=7 and ClkBaud = '1' else
'0';

end Behavioral;

```

### Contador inicial de prueba

-- Company:

```

-- Engineer:
--
-- Create Date:      16:56:23 10/20/2009
-- Design Name:
-- Module Name:     Contador - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--

-----
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

-- Entidad del contador Ppal. Aqui se genera el pulso de transmite que
-- usa
-- Tx_serie para transmitir un nuevo dato.

-----
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

-- Entidad del contador Ppal. Aqui se genera el pulso de transmite que
-- usa
-- Tx_serie para transmitir un nuevo dato.

-----
entity Contador is
    Generic(a : in integer := 3;
            b : in integer := 5);
    Port ( reset : in STD_LOGIC;
           clk : in STD_LOGIC;
           enable : in STD_LOGIC;
           load : in STD_LOGIC;
           data_load : in STD_LOGIC_VECTOR (3 downto 0);
           cambiando : out std_logic;

```

```

        salida : out STD_LOGIC_VECTOR (3 downto 0));
end Contador;
-----
-----

architecture Behavioral of Contador is

-- Señales auxiliares --

signal mienable, mireset, micambiando,miload: std_logic;
--signal aux: std_logic_vector(31 downto 0);
signal aux: integer;
signal misalida,midata_load: std_logic_vector(3 downto 0);
--signal a:std_logic_vector(24 downto 0);

begin

process (clk,reset,enable,load)
begin
if miload='1' then
    misalida <= midata_load;
    micambiando <= '1';
    aux<=0;
else
    if mienable='1' then
        if mireset='1' then
            reset -> inicialización
                aux<=0;
                misalida <= "0000";
                micambiando <= '1';
            elsif(clk'event and clk='1') then      -- flanco
de reloj ascendente
                if(aux=100000000) then           -- cada
100.000.000 ciclos (a 1 sg.)
                    aux<=0;                  -- vuelvo a
comenzar
                    micambiando <= '1';
                    if misalida = "1111" then
                        misalida <= "0000";
                    else
                        misalida <= misalida +1;
                    end if;
                else
                    aux<=aux+1;               --
cuento
                    micambiando <= '0';
                end if;
            end if;
        end if;
    end if;
    -- saco la salida
end process;

-- Asignación de señales --
-- Como en la placa los botones están negados, negamos el reset.
salida <= misalida;

```

```

        mireset <= not reset;
        mienable <= enable;
        cambiando <= micambiando;
        miload <= load;
        midata_load<=data_load;

end Behavioral;

```

---

## Circuito\_FPGA.vhd (generado automáticamente)

---

```

--Descripción:
--  Este fichero ha sido generado automáticamente por la aplicación
Nessy2.0
--  Se trata del fichero que describe la entidad top generada para
cualquier circuito que quiera ser ejecutado en la FPGA
--
--Especificaciones:
--  Circuito a ejecutar:
--    Num. Entradas: 19
--    Num. Salidas: 16
--Autor:
--  Carlos Sanchez-Vellisco Sanchez
--  Facultad de Informatica. Universidad Complutense de Madrid
--Fecha:
--  Thu Jun 10 23:02:35 CEST 2010
--

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--Descripción de la entidad
entity Circuito_FPGA is
Port ( clk : in STD_LOGIC;
reset : in STD_LOGIC;
salida_serie : out STD_LOGIC;
entrada_serie : in STD_LOGIC;
ledsEntrada : out std_logic_vector(1 downto 0);
ledsSalida : out std_logic_vector(1 downto 0));
end Circuito_FPGA;

architecture Behavioral of Circuito_FPGA is

```

```

--Transmisor serie
component Tx_serie
    Port ( RstN : in STD_LOGIC;
            clk : in STD_LOGIC;
            Transmite : in STD_LOGIC;
            DatoTxIn : in STD_LOGIC_VECTOR (7 downto 0);
            Transmitiendo : out STD_LOGIC;
            DatoSerieOut : out STD_LOGIC);
end component;

--Receptor Serie
component Rx_Serie
    Port ( Rstn : in STD_LOGIC;
            Clk : in STD_LOGIC;
            RxDataSerie : in STD_LOGIC;
            DatoRxOut : out STD_LOGIC_VECTOR (7 downto 0);
            AvisoRx : out STD_LOGIC;
            Recibiendo : out STD_LOGIC);
end component;

--Entidad que se quiere ejecutar
component CONTADOR
    Port(
        RESET: in STD_LOGIC;
        CLK: in STD_LOGIC;
        ENABLE: in STD_LOGIC;
        LOAD: in STD_LOGIC;
        DATA_LOAD: in STD_LOGIC_VECTOR(15 downto 0);

        SALIDA: out STD_LOGIC_VECTOR(15 downto 0)
    );
end component;

--Señales del circuito principal

--Entradas
signal mi_RESET: STD_LOGIC;
signal mi_CLK: STD_LOGIC;
signal mi_ENABLE: STD_LOGIC;
signal mi_LOAD: STD_LOGIC;
signal mi_DATA_LOAD: STD_LOGIC_VECTOR(15 downto 0);

--Salidas
signal mi_SALIDA: STD_LOGIC_VECTOR(15 downto 0);

--Señales para la Recepción Serie
signal mi_resetserie:std_logic;
signal mi_transmite:std_logic;
signal mi_datotxin:std_logic_vector(7 downto 0);
signal mi_datorxout:std_logic_vector(7 downto 0);
signal mi_avisoRx:std_logic;
signal mi_recibiendo:std_logic;

--Señales para la Transmisión Serie
signal mi_transmitiendo:std_logic;

```

```

signal mi_datoserieout:std_logic;
signal mi_rxdatoserie:std_logic;

--Señales intermedias para la entrada y la salida. Se conectarán a las
entradas y las salidas del circuito principal
signal Reg_entradas: std_logic_vector(31 downto 0);
signal Reg_salidas: std_logic_vector(31 downto 0);

--Señales para los estados del emisor/receptor de 32 bits
signal estadoEnt: integer;
signal estadoSal: integer;

signal ledsEnt:std_logic_vector(1 downto 0);
signal ledsSal:std_logic_vector(1 downto 0);

--Señales necesarias para la correcta entrada/salida con 32 bits
signal fin_recepcion : std_logic;
signal recibido,transmitido,biest_recibido, biest_transmitido:
std_logic;
signal frecibido,ftransmitido,fin: std_logic; --flancos de fin

begin

--Asignación de señales a los componentes de la entrada/salida
f: Tx_serie port
map(mi_resetserie,clk,mi_transmite,mi_datotxin,mi_transmitiendo,mi_dat
oserieout);
R: Rx_serie port
map(mi_resetserie,clk,mi_rxdatoserie,mi_datorxout,mi_avisorx,mi_recibi
endo);

--Asignación de señales al componente del circuito principal
U: CONTADOR port
map(mi_RESET,mi_CLK,mi_ENABLE,mi_LOAD,mi_DATA_LOAD,mi_SALIDA);

--Proceso encargado de la correcta recepción de datos (32 bits)
--Cada vez que se reciba un byte, se irá asignando a las entradas
desde las menos significativas a las más significativas
process(mi_recibiendo,mi_resetserie)
begin
    if mi_resetserie = '0' then
        estadoEnt <= 0;
        fin_recepcion <= '0';
    elsif mi_recibiendo'event and mi_recibiendo = '0' then
        fin_recepcion <= '0';
        if estadoEnt = 0 then
            Reg_entradas(7 downto 0) <= mi_datorxout;
            estadoEnt <= 1;
        elsif estadoEnt = 1 then
            Reg_entradas(15 downto 8) <= mi_datorxout;
            estadoEnt <= 2;
        elsif estadoEnt = 2 then
            Reg_entradas(23 downto 16) <= mi_datorxout;

```

```

        estadoEnt <= 3;
    elsif estadoEnt = 3 then
        Reg_entradas(31 downto 24) <= mi_datorxout;
        estadoEnt <= 0;
        fin_recepcion <= '1'; --fin de la recepción
    else
        Reg_entradas(7 downto 0) <= mi_datorxout;
        estadoEnt <= 1;
    end if;
end if;
end process;

--Proceso encargado de que la salida correspondiente del circuito esté conectada a la salida serie antes de que la transmisión se produzca
process(estadoSal, clk, mi_resetserie)
begin
    if mi_resetserie = '0' then
        mi_datotxin <= Reg_salidas(7 downto 0);
    elsif clk'event and clk = '1' then
        if estadoSal = 0 then
            mi_datotxin <= Reg_salidas(7 downto 0);
        elsif estadoSal = 1 then
            mi_datotxin <= Reg_salidas(15 downto 8);
        elsif estadoSal = 2 then
            mi_datotxin <= Reg_salidas(23 downto 16);
        elsif estadoSal = 3 then
            mi_datotxin <= Reg_salidas(31 downto 24);
        end if;
    end if;
end process;

--Proceso encargado de cambiar de estado cada vez que se comienza a transmitir un byte
process(mi_transmitiendo, mi_resetserie)
begin
    if mi_resetserie = '0' then
        estadoSal <= 0;
        transmitido <= '0';
    elsif mi_transmitiendo'event and mi_transmitiendo = '1' then
        if estadoSal = 3 then
            transmitido <= '1';
            estadoSal <= 0;
        else
            estadoSal <= estadoSal+1;
            transmitido <= '0';
        end if;
    end if;
end process;

--Proceso encargado de registrar que la recepción ha terminado. La salida del biestable ('recibido') se usará como reloj del circuito principal
process(clk,fin_recepcion)
begin
    if clk'event and clk='1' then
        if fin_recepcion = '1' then

```

```

        recibido <= '1'; --flanco positivo que hará funcionar
el circuito principal
    else
        recibido <= '0';
    end if;
end if;
end process;

--Proceso encargado de registrar el fin de la recepción y de la
transmisión
process(clk)
begin
    if clk'event and clk='1' then
        biest_recibido <= recibido;
        biest_transmitido <= transmitido;
    end if;
end process;

--Proceso que indica que, o bien ha terminado una recepción de datos,
o bien ha terminado una transmisión
--Cuando se detecte un flanco positivo, se negará la señal que hace
que se transmita, de tal forma que si se estaba transitando,
--se deje de transmitir y si no se estaba transmitiendo se comience
una nueva transmisión
process(fin)
begin
    if fin'event and fin = '1' then
        if mi_transmite = '0' then
            mi_transmite <= '1';
        else
            mi_transmite <= '0';
        end if;
    end if;
end process;

--Este proceso es prescindible. Sólo a efectos de visualizar estado de
entrada
process(estadоОnt)
begin
    if estadоОnt = 0 then
        ledsEnt <= "00";
    elsif estadоОnt = 1 then
        ledsEnt <= "01";
    elsif estadоОnt = 2 then
        ledsEnt <= "10";
    elsif estadоОnt = 3 then
        ledsEnt <= "11";
    end if;
end process;

--Este proceso es prescindible. Sólo a efectos de visualizar estado de
salida
process(estadоВal)
begin
    if estadоВal = 0 then
        ledsSal <= "00";

```

```

        elsif estadoSal = 1 then
            ledsSal <= "01";
        elsif estadoSal = 2 then
            ledsSal <= "10";
        elsif estadoSal = 3 then
            ledsSal <= "11";
        end if;
    end process;

ledsEntrada <= ledsEnt;
ledsSalida <= ledsSal;

--El reloj del circuito principal será el flanco que indique el fin de
la recepción
mi_clk <= recibido;

--Asignación de las señales del circuito general
mi_resetserie <= reset;
salida_serie <= mi_datoserieout;
mi_rxdatoserie <= entrada_serie;

--Asignación de las señales necesarias para la transmisión correcta
frecibido <= not biest_recibido and recibido; --flanco que indica fin
de recepcion
ftransmitido <= not biest_transmitido and transmitido; --flanco que
indica fin de transmision
fin <= frecibido or ftransmitido; --flanco que indica fin de envio o
transmision

--Asignación de las señales intermedias de entrada/salida a las del
circuito principal
mi_RESET <= Reg_entradas(0);
mi_ENABLE <= Reg_entradas(1);
mi_LOAD <= Reg_entradas(2);
mi_DATA_LOAD(0) <= Reg_entradas(3);
mi_DATA_LOAD(1) <= Reg_entradas(4);
mi_DATA_LOAD(2) <= Reg_entradas(5);
mi_DATA_LOAD(3) <= Reg_entradas(6);
mi_DATA_LOAD(4) <= Reg_entradas(7);
mi_DATA_LOAD(5) <= Reg_entradas(8);
mi_DATA_LOAD(6) <= Reg_entradas(9);
mi_DATA_LOAD(7) <= Reg_entradas(10);
mi_DATA_LOAD(8) <= Reg_entradas(11);
mi_DATA_LOAD(9) <= Reg_entradas(12);
mi_DATA_LOAD(10) <= Reg_entradas(13);
mi_DATA_LOAD(11) <= Reg_entradas(14);
mi_DATA_LOAD(12) <= Reg_entradas(15);
mi_DATA_LOAD(13) <= Reg_entradas(16);
mi_DATA_LOAD(14) <= Reg_entradas(17);
mi_DATA_LOAD(15) <= Reg_entradas(18);

Reg_salidas(0) <= mi_SALIDA(0);
Reg_salidas(1) <= mi_SALIDA(1);
Reg_salidas(2) <= mi_SALIDA(2);
Reg_salidas(3) <= mi_SALIDA(3);

```

```
Reg_salidas(4) <= mi_SALIDA(4);
Reg_salidas(5) <= mi_SALIDA(5);
Reg_salidas(6) <= mi_SALIDA(6);
Reg_salidas(7) <= mi_SALIDA(7);
Reg_salidas(8) <= mi_SALIDA(8);
Reg_salidas(9) <= mi_SALIDA(9);
Reg_salidas(10) <= mi_SALIDA(10);
Reg_salidas(11) <= mi_SALIDA(11);
Reg_salidas(12) <= mi_SALIDA(12);
Reg_salidas(13) <= mi_SALIDA(13);
Reg_salidas(14) <= mi_SALIDA(14);
Reg_salidas(15) <= mi_SALIDA(15);

end Behavioral;
```

## 5. Documentación del código

### Packages

[compiladorEntidad](#)

[generadorVHDL](#)

[IOFPGA](#)

[nessy20](#)

### Package compiladorEntidad

#### Class Summary

<a href="#"><u>Entidad</u></a>	Clase que representa una entidad vhdl.
<a href="#"><u>Entrada</u></a>	Clase que representa la entrada de una entidad
<a href="#"><u>Errores</u></a>	Clase para almacenar los errores de compilación que se puedan ir produciendo
<a href="#"><u>EvaluadorExps</u></a>	Clase que contiene métodos para evaluar una expresión aritmética
<a href="#"><u>LexicoEntidad</u></a>	Clase que analiza léxicamente un fichero en el que está contenida una entidad en lenguaje VHDL
<a href="#"><u>Pila&lt;T&gt;</u></a>	Representa una pila de elementos genéricos.
<a href="#"><u>Puerto</u></a>	Representa el puerto de una entidad el cual puede ser tanto una entrada como una salida
<a href="#"><u>Salida</u></a>	Representa la salida de una entidad.
<a href="#"><u>SintacticoEntidad</u></a>	Clase que analiza sintácticamente una entidad definida en VHDL.
<a href="#"><u>Token</u></a>	Clase para representar un token del fichero que contiene una entidad descrita en VHDL

## Class Entidad

```
java.lang.Object  
└ compiladorEntidad.Entidad
```

---

```
public class Entidad  
extends java.lang.Object
```

Clase que representa una entidad vhdl.

---

## Constructor Summary

### Entidad()

Crea una entidad vacía.

## Method Summary

void	<a href="#"><u>anadeEntrada(Entrada e)</u></a> Añade el argumento e de la clase Entrada al array de entradas de la entidad.
void	<a href="#"><u>anadeSalida(Salida s)</u></a> Añade el argumento s de la clase Salida al array de salidas de la entidad.
int	<a href="#"><u>getBitsEntrada()</u></a> Getter para consultar el número de bits de entrada de la Entidad
int	<a href="#"><u>getBitsSalida()</u></a> Getter para consultar el número de bits de salida de la Entidad.
<a href="#"><u>Entrada</u></a>	<a href="#"><u>getEntrada(int i)</u></a> Getter que devuelve la entrada i del ArrayList de las entradas.
java.lang.String	<a href="#"><u>getNombre()</u></a>
java.lang.String	<a href="#"><u>getNombreReset()</u></a>

	Método para consultar el nombre de la entrada de reset.
int	<u><a href="#">getNumEntradas()</a></u> Getter para consultar el número de entradas de la Entidad.
int	<u><a href="#">getNumSalidas()</a></u> Getter para consultar el número de salidas de la Entidad.
int	<u><a href="#">getPosicionClk()</a></u> Devuelve la posición relativa de las entradas en la que se encuentra el clk.
int	<u><a href="#">getPosicionReset()</a></u> Devuelve la posición relativa de las entradas en la que se encuentra el reset.
Salida	<u><a href="#">getSalida(int i)</a></u> Getter que devuelve la salida i del ArrayList de las entradas.
void	<u><a href="#">muestra()</a></u> Para mostrar por pantalla.
void	<u><a href="#">setBitsEntrada(int bitsEntrada)</a></u> Setter para establecer el número de bits de entrada de la Entidad.
void	<u><a href="#">setBitsSalida(int bitsSalida)</a></u> Setter para establecer el número de bits de salida de la Entidad.
void	<u><a href="#">setNombre(java.lang.String nombre)</a></u> Cambia el nombre de la entidad.
java.lang.String	<u><a href="#">toString()</a></u> Devuelve una cadena con la descripción de las entradas y salidas de la entidad.

#### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

## Constructor Detail

### Entidad

```
public Entidad()  
    Crea una entidad vacía.
```

## Method Detail

### getNombre

```
public java.lang.String getNombre()  
    Returns:
```

Devuelve el nombre de la entidad.

---

### setNombre

```
public void setNombre(java.lang.String nombre)  
    Cambia el nombre de la entidad.
```

#### Parameters:

nombre - Nuevo nombre de la entidad

---

### getNumEntradas

```
public int getNumEntradas()  
    Getter para consultar el número de entradas de la Entidad.
```

#### Returns:

El número de Entradas de la entidad.

---

### getNumSalidas

```
public int getNumSalidas()  
    Getter para consultar el número de salidas de la Entidad.
```

#### Returns:

El número de Salidas de la entidad.

---

### getBitsEntrada

```
public int getBitsEntrada()  
    Getter para consultar el número de bits de entrada de la Entidad
```

**Returns:**

El número de bits de entrada de la entidad.

---

**setBitsEntrada**

```
public void setBitsEntrada(int bitsEntrada)  
    Setter para establecer el número de bits de entrada de la Entidad.
```

**Parameters:**

bitsEntrada -

---

**getBitsSalida**

```
public int getBitsSalida()  
    Getter para consultar el número de bits de salida de la Entidad.
```

**Returns:**

El número de bits de salida de la entidad.

---

**setBitsSalida**

```
public void setBitsSalida(int bitsSalida)  
    Setter para establecer el número de bits de salida de la Entidad.
```

**Parameters:**

bitsSalida - Número de bits de salida

---

**getEntrada**

```
public Entrada getEntrada(int i)  
    Getter que devuelve la entrada i del ArrayList de las entradas.
```

**Parameters:**

i - Posición de la Entrada que queremos devolver.

**Returns:**

La entrada i del ArrayList de Entradas.

---

**getSalida**

```
public Salida getSalida(int i)  
    Getter que devuelve la salida i del ArrayList de las entradas.
```

**Parameters:**

i - Posición de la Salida que queremos devolver.

**Returns:**

La entrada i del ArrayList de salidas.

---

**getPosicionReset**

public int **getPosicionReset()**

Devuelve la posición relativa de las entradas en la que se encuentra el reset. Esto será utilizado para poder enviar al circuito la señal de reset aislándola de todas las demás

**Returns:**

Posición que ocupa el reset dentro de las entradas

---

**getPosicionClk**

public int **getPosicionClk()**

Devuelve la posición relativa de las entradas en la que se encuentra el clk. Esto será utilizado para poder hacer la asignación adecuada, por tratarse esta de una entrada especial.

**Returns:**

Posición que ocupa el clk dentro de las entradas

---

**anadeEntrada**

public void **anadeEntrada(Entrada e)**

Añade el argumento e de la clase Entrada al array de entradas de la entidad. Diferencia entre dos tipos de entradas especiales. Si el nombre de la entrada es CLK, CLOCK O RELOJ, la entrada será marcada como entrada de reloj. De la misma forma, si el nombre de la entrada es RST o RESET se marcará como una entrada especial de reset.

**Parameters:**

e - Entrada a añadir a la entidad

---

**anadeSalida**

public void **anadeSalida(Salida s)**

Añade el argumento s de la clase Salida al array de salidas de la entidad.

**Parameters:**

s - Salida a añadir a la entidad

---

### **muestra**

```
public void muestra()  
    Para mostrar por pantalla. Sólo de prueba
```

---

### **getNombreReset**

```
public java.lang.String getNombreReset()  
    Método para consultar el nombre de la entrada de reset.
```

#### **Returns:**

Devuelve el Nombre de la Entrada de reset.

---

### **toString**

```
public java.lang.String toString()  
    Devuelve una cadena con la descripción de las entradas y salidas de la entidad.
```

#### **Overrides:**

toString in class java.lang.Object

#### **Returns:**

Cadena con las entradas y salidas de la Entidad.

## **Class Entrada**

```
java.lang.Object  
└ compiladorEntidad.Puerto  
    └ compiladorEntidad.Entrada
```

---

```
public class Entrada  
extends Puerto
```

Clase que representa la entrada de una entidad

---

## **Constructor Summary**

<a href="#">Entrada()</a>	
---------------------------	--

## Method Summary

boolean	<a href="#"><u>getEsReloj()</u></a> Método de consulta para saber si la Entrada es el reloj
boolean	<a href="#"><u>getEsReset()</u></a> Getter de para saber si la entrada se trata del reset.
void	<a href="#"><u>ponerComoReloj()</u></a> Método que establece la Entrada como reloj.
void	<a href="#"><u>ponerComoReset(boolean valor)</u></a> Establece la entrada como Reset o le quita la propiedad.
void	<a href="#"><u>quitarComoReloj()</u></a> Método para quitar a la Entrada la propiedad de que es el reloj.

### Methods inherited from class compiladorEntidad.[Puerto](#)

[getNombre](#), [getNumBits](#), [setNombre](#), [setNumBits](#)

### Methods inherited from class java.lang.Object

[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [toString](#), [wait](#), [wait](#), [wait](#)

## Constructor Detail

### Entrada

public [Entrada\(\)](#)

## Method Detail

### [getEsReloj](#)

public boolean [getEsReloj\(\)](#)  
Método de consulta para saber si la Entrada es el reloj

**Returns:**

---

Cierto o Falso dependiendo si se trata o no del reloj.

---

### **ponerComoReloj**

```
public void ponerComoReloj()
```

Método que establece la Entrada como reloj.

---

### **quitarComoReloj**

```
public void quitarComoReloj()
```

Método para quitar a la Entrada la propiedad de que es el reloj.

---

### **getEsReset**

```
public boolean getEsReset()
```

Getter de para saber si la entrada se trata del reset.

#### **Returns:**

Boolean si es cierto que la entrada es un reset.

---

### **ponerComoReset**

```
public void ponerComoReset(boolean valor)
```

Establece la entrada como Reset o le quita la propiedad.

#### **Parameters:**

valor - Nuevo valor para el parámetro esReset.

## **Class Errores**

```
java.lang.Object
```

```
└ compiladorEntidad.Errores
```

---

```
public class Errores  
extends java.lang.Object
```

Clase para almacenar los errores de compilación que se puedan ir produciendo

## Constructor Summary

### [Errores \(\)](#)

Constructor de la Clase.

## Method Summary

	void <a href="#">error</a> (java.lang.String error) Añade un nuevo error al arrayList errores, donde se encuentran todos los fallos.
java.util.ArrayList<java.lang.String>	<a href="#">getErrores ()</a> Devuelve todos los errores que tiene la clase.

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
--

## Constructor Detail

### Errores

public [Errores \(\)](#)

Constructor de la Clase. Inicializa el atributo errores.

## Method Detail

### error

public void [error](#)(java.lang.String error)

Añade un nuevo error al arrayList errores, donde se encuentran todos los fallos.

**Parameters:**

`error` - Nuevo error a insertar en la clase.

---

### **getErrores**

```
public java.util.ArrayList<java.lang.String> getErrores()  
Devuelve todos los errores que tiene la clase.
```

#### **Returns:**

ArrayList con todos los errores de la clase.

## **Class EvaluadorExps**

```
java.lang.Object  
└ compiladorEntidad.EvaluadorExps
```

---

```
public class EvaluadorExps  
extends java.lang.Object
```

Clase que contiene métodos para evaluar una expresión aritmética

---

## **Constructor Summary**

[EvaluadorExps \(\)](#)

## **Method Summary**

static int	<a href="#"><u>aplicar</u></a> (int x, int op1, int op2) Reduce una Expresión y devuelve el resultado
static boolean	<a href="#"><u>esMenorIg</u></a> (int t1, int t2) Método de consulta de preferencia de operadores.
static boolean	<a href="#"><u>esOperador</u></a> (int t) Método que devuelve un booleano dependiendo si la entrada es un Operador
static boolean	<a href="#"><u>esOperando</u></a> (int t) Método que devuelve un booleano dependiendo si la entrada es un

	Operando
--	----------

#### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait

## Constructor Detail

### EvaluadorExps

public EvaluadorExps()

## Method Detail

### esOperador

public static boolean esOperador(int t)

Método que devuelve un booleano dependiendo si la entrada es un Operador

#### Parameters:

t - Entero a consultar.

#### Returns:

Booleano que indica si la entrada es un Operador o no.

---

### esOperando

public static boolean esOperando(int t)

Método que devuelve un booleano dependiendo si la entrada es un Operando

#### Parameters:

t - Entero a consultar.

#### Returns:

Booleano que indica si la entrada es un Operando o no.

---

### esMenorIg

public static boolean esMenorIg(int t1,

```
        int t2)  
Método de consulta de preferencia de operadores.
```

**Parameters:**

t1 - Primer Operador.

t2 - Segundo Operador.

**Returns:**

Bololean Devuelve true si el primer operador tiene menor o igual preferencia que el segundo operador.

---

**aplicar**

```
public static int aplicar(int x,  
                         int op1,  
                         int op2)
```

Reduce una Expresión y devuelve el resultado

**Parameters:**

x - Operador a aplicar.

op1 - Primer operando.

op2 - Segundo operando.

**Returns:**

Resultado de la operación.

**Class LexicoEntidad**

```
java.lang.Object  
└ compiladorEntidad.LexicoEntidad
```

---

```
public class LexicoEntidad  
extends java.lang.Object
```

Clase que analiza léxicamente un fichero en el que está contenida una entidad en lenguaje VHDL

---

**Field Summary**

static int	<b><u>ABRE_PARENTESIS</u></b> Código del elemento ABRE_PARENTESIS -> 11
static int	<b><u>ASIG_GENERIC</u></b> Código del elemento ASIG_GENERIC -> 19
static int	<b><u>CIERRA_PARENTESIS</u></b> Código del elemento CIERRA_PARENTESIS -> 12
static int	<b><u>DIV</u></b> Código del elemento DIV -> 23
static int	<b><u>DOS_PUNTOS</u></b> Código del elemento DOS_PUNTOS -> 10
static int	<b><u>DWNTO</u></b> Código de la palabra reservada DWNTO -> 7
static int	<b><u>END</u></b> Código de la palabra reservada END -> 8
static int	<b><u>ENTERO</u></b> Código del elemento ENTERO -> 13
static int	<b><u>ENTITY</u></b> Código de la palabra reservada Entidad -> 0
static int	<b><u>EOF</u></b> Código del elemento EOF -> 15
static int	<b><u>GENERIC</u></b> Código del elemento GENERIC -> 17
static int	<b><u>IDENTIFICADOR</u></b> Código del elemento IDENTIFICADOR -> 14
static int	<b><u>IN</u></b> Código de la palabra reservada IN -> 5
static int	<b><u>INTEGER</u></b> Código del elemento INTEGER -> 18
static int	<b><u>IS</u></b> Código de la palabra reservada IS -> 1

static int	<u>MULT</u> Código del elemento MULT -> 22
static int	<u>OTRO</u> Código del elemento OTRO -> 16
static int	<u>OUT</u> Código de la palabra reservada OUT -> 6
static int	<u>PORT</u> Código de la palabra reservada Port -> 2
static int	<u>PUNTO Y COMA</u> Código del elemento PUNTO_Y_COMA -> 9
static int	<u>RESTA</u> Código del elemento RESTA -> 21
static int	<u>STD_LOGIC</u> Código de la palabra reservada STD_LOGIC -> 3
static int	<u>STD_LOGIC_VECTOR</u> Código de la palabra reservada STD_LOGIC_VECTOR -> 4
static int	<u>SUMA</u> Código del elemento SUMA -> 20

## Constructor Summary

[LexicoEntidad](#)(java.lang.String fichero, [Errores](#) errores)

Constructor de la clase.

## Method Summary

void	<u>cerrar()</u> Cierra el Buffer de lectura del fichero.
int	<u>getNumLinea()</u> Getter que devuelve el número de línea por la que va el

	Analizador.
java.lang.Character	<b><a href="#">getUltimoCharLeido()</a></b> Método de Consulta sobre el último carácter leído.
<a href="#">Token</a>	<b><a href="#">iniciar()</a></b> Inicia el análisis, llama a la función sigToken()
<a href="#">Token</a>	<b><a href="#">sigToken()</a></b> Método que gestiona el autómata finito que ocntrola el Analizador Léxico
void	<b><a href="#">transita(int sigEstado)</a></b> Cambia el estado actual del analizador léxico por el que se le pasa por parámetro.

#### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait

## Field Detail

### ENTITY

public static final int ENTITY  
Código de la palabra reservada Entidad -> 0

**See Also:**

[Constant Field Values](#)

### IS

public static final int IS  
Código de la palabra reservada IS -> 1

**See Also:**

[Constant Field Values](#)

**PORT**

```
public static final int PORT  
Código de la palabra reservada Port -> 2
```

**See Also:**

[Constant Field Values](#)

---

**STD\_LOGIC**

```
public static final int STD_LOGIC  
Código de la palabra reservada STD_LOGIC -> 3
```

**See Also:**

[Constant Field Values](#)

---

**STD\_LOGIC\_VECTOR**

```
public static final int STD_LOGIC_VECTOR  
Código de la palabra reservada STD_LOGIC_VECTOR -> 4
```

**See Also:**

[Constant Field Values](#)

---

**IN**

```
public static final int IN  
Código de la palabra reservada IN -> 5
```

**See Also:**

[Constant Field Values](#)

---

**OUT**

```
public static final int OUT  
Código de la palabra reservada OUT -> 6
```

**See Also:**

[Constant Field Values](#)

---

**DOWNT0**

```
public static final int DOWNT0  
Código de la palabra reservada DOWNT0 -> 7
```

**See Also:**

[Constant Field Values](#)

---

## **END**

```
public static final int END  
Código de la palabra reservada END -> 8
```

**See Also:**

[Constant Field Values](#)

---

## **PUNTO\_Y\_COMA**

```
public static final int PUNTO_Y_COMA  
Código del elemento PUNTO_Y_COMA -> 9
```

**See Also:**

[Constant Field Values](#)

---

## **DOS\_PUNTOS**

```
public static final int DOS_PUNTOS  
Código del elemento DOS_PUNTOS -> 10
```

**See Also:**

[Constant Field Values](#)

---

## **ABRE\_PARENTESIS**

```
public static final int ABRE_PARENTESIS  
Código del elemento ABRE_PARENTESIS -> 11
```

**See Also:**

[Constant Field Values](#)

---

## **CIERRA\_PARENTESIS**

```
public static final int CIERRA_PARENTESIS  
Código del elemento CIERRA_PARENTESIS -> 12
```

**See Also:**

[Constant Field Values](#)

---

## **ENTERO**

```
public static final int ENTERO  
    Código del elemento ENTERO -> 13
```

**See Also:**

[Constant Field Values](#)

---

## **IDENTIFICADOR**

```
public static final int IDENTIFICADOR  
    Código del elemento IDENTIFICADOR -> 14
```

**See Also:**

[Constant Field Values](#)

---

## **EOF**

```
public static final int EOF  
    Código del elemento EOF -> 15
```

**See Also:**

[Constant Field Values](#)

---

## **OTRO**

```
public static final int OTRO  
    Código del elemento OTRO -> 16
```

**See Also:**

[Constant Field Values](#)

---

## **GENERIC**

```
public static final int GENERIC  
    Código del elemento GENERIC -> 17
```

**See Also:**

[Constant Field Values](#)

---

**INTEGER**

```
public static final int INTEGER  
Código del elemento INTEGER -> 18
```

**See Also:**

[Constant Field Values](#)

---

**ASIG\_GENERIC**

```
public static final int ASIG_GENERIC  
Código del elemento ASIG_GENERIC -> 19
```

**See Also:**

[Constant Field Values](#)

---

**SUMA**

```
public static final int SUMA  
Código del elemento SUMA -> 20
```

**See Also:**

[Constant Field Values](#)

---

**RESTA**

```
public static final int RESTA  
Código del elemento RESTA -> 21
```

**See Also:**

[Constant Field Values](#)

---

**MULT**

```
public static final int MULT  
Código del elemento MULT -> 22
```

**See Also:**

[Constant Field Values](#)

---

**DIV**

```
public static final int DIV  
Código del elemento DIV -> 23
```

**See Also:**

[Constant Field Values](#)

## Constructor Detail

### **LexicoEntidad**

```
public LexicoEntidad(java.lang.String fichero,  
                     Errores errores)  
    throws java.io.FileNotFoundException,  
           java.io.IOException
```

Constructor de la clase.

#### **Parameters:**

fichero - Código a analizar.

errores - Errores que se han detectado

#### **Throws:**

java.io.FileNotFoundException

java.io.IOException

## Method Detail

### **getNumLinea**

```
public int getNumLinea()
```

Getter que devuelve el número de línea por la que va el Analizador.

#### **Returns:**

Integer Número de línea.

---

### **iniciar**

```
public Token iniciar()  
    throws java.io.IOException  
Inicia el análisis, llama a la función sigToken()
```

#### **Returns:**

El primer Token.

#### **Throws:**

java.io.IOException

---

**cerrar**

```
public void cerrar()  
    throws java.io.IOException  
Cierra el Buffer de lectura del fichero.
```

**Throws:**

java.io.IOException

---

**sigToken**

```
public Token sigToken()  
    throws java.io.IOException  
Método que gestiona el autómata finito que ocntrola el Analizador Léxico
```

**Returns:**

El siguiente Token a leer.

**Throws:**

java.io.IOException

---

**transita**

```
public void transita(int sigEstado)  
    throws java.io.IOException  
Cambia el estado actual del analizador léxico por el que se le pasa por parámetro.  
Además lee el siguiente carácter para dejarlo preparado para ser analizado.
```

**Parameters:**

sigEstado - Siguiente estado al que va el analizador.

**Throws:**

java.io.IOException

---

**getUltimoCharLeido**

```
public java.lang.Character getUltimoCharLeido()  
Método de Consulta sobre el último carácter leído.
```

**Returns:**

El último carácter leído.

## Class Pila<T>

```
java.lang.Object
└ compiladorEntidad.Pila<T>
```

### Type Parameters:

T -

---

```
public class Pila<T>
extends java.lang.Object
```

Representa una pila de elementos genéricos. Utilizada para la evaluación de expresiones.

---

## Constructor Summary

### [Pila \(\)](#)

Método constructor de la clase.

## Method Summary

void	<a href="#">apilar (T t)</a>	Apilar un nuevo elemento en la pila.
T	<a href="#">desapilar ()</a>	Desapila un elemento de la pila
boolean	<a href="#">esVacia ()</a>	Método para saber si es pila vacía.
T	<a href="#">getCima ()</a>	Consulta la cima de la pila
int	<a href="#">numElems ()</a>	Método que consulta el número de elementos de la pila.

### Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll,
toString, wait, wait, wait
```

## Constructor Detail

### Pila

```
public Pila()
```

Método constructor de la clase. Crea una pila vacía.

## Method Detail

### esVacia

```
public boolean esVacia()
```

Método para saber si es pila vacía.

#### Returns:

Boolean, true si es pila vacía, false en caso contrario.

---

### apilar

```
public void apilar(T t)
```

Apilar un nuevo elemento en la pila.

#### Parameters:

t - Elemento a apilar.

---

### desapilar

```
public T desapilar()
```

Desapila un elemento de la pila

#### Returns:

Devuelve el elemento desapilado.

---

### getCima

```
public T getCima()
```

Consulta la cima de la pila

#### Returns:

El elemento de la cima de la Pila

---

### **numElems**

```
public int numElems()
```

Método que consulta el número de elementos de la pila.

#### **Returns:**

El número de elementos de la Pila.

## **Class Puerto**

```
java.lang.Object  
└ compiladorEntidad.Puerto
```

#### **Direct Known Subclasses:**

[Entrada](#), [Salida](#)

---

```
public abstract class Puerto  
extends java.lang.Object
```

Representa el puerto de una entidad el cual puede ser tanto una entrada como una salida

---

## **Constructor Summary**

<a href="#">Puerto ()</a>	
---------------------------	--

## **Method Summary**

java.lang.String	<a href="#">getNombre ()</a> Método que obtiene el nombre del Puerto.
------------------	--

int	<a href="#">getNumBits ()</a> Consulta el número de bits que utiliza el puerto.
-----	--

void	<a href="#">setNombre (java.lang.String nombre)</a> Establece el nombre del Puerto.
------	--

void	<b><a href="#">setNumBits</a></b> (int numBits) Establece el número de bits del Puerto
------	---

#### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait

## Constructor Detail

### Puerto

public **Puerto**()

## Method Detail

### getNombre

public java.lang.String **getNombre**()

Método que obtiene el nombre del Puerto.

#### Returns:

El nombre del Puerto.

---

### setNombre

public void **setNombre**(java.lang.String nombre)

Establece el nombre del Puerto.

#### Parameters:

nombre - Nuevo nombre del puerto.

---

### getNumBits

public int **getNumBits**()

Consulta el número de bits que utiliza el puerto.

#### Returns:

Número de bits del puerto.

### **setNumBits**

```
public void setNumBits(int numBits)  
    Establece el número de bits del Puerto
```

#### **Parameters:**

numBits - Nuevo número de bits del Puerto.

## **Class Salida**

```
java.lang.Object  
└ compiladorEntidad.Puerto  
    └ compiladorEntidad.Salida
```

---

```
public class Salida  
extends Puerto
```

Representa la salida de una entidad. Hereda de la clase Puerto.

---

## **Constructor Summary**

[Salida\(\)](#)

## **Method Summary**

### **Methods inherited from class compiladorEntidad.[Puerto](#)**

[getNombre](#), [getNumBits](#), [setNombre](#), [setNumBits](#)

### **Methods inherited from class java.lang.Object**

[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#),  
[toString](#), [wait](#), [wait](#), [wait](#)

## Constructor Detail

### Salida

```
public Salida()
```

## Class SintacticoEntidad

```
java.lang.Object  
└ compiladorEntidad.SintacticoEntidad
```

---

```
public class SintacticoEntidad  
extends java.lang.Object
```

Clase que analiza sintácticamente una entidad definida en VHDL. Además mientras la analiza, almacena los datos leídos en nuestra estructura de datos de la entidad. La forma en la que funciona este analizador sintáctico está determinada en la gramática que se especifica en la memoria.

---

## Field Summary

static int	<a href="#"><u>MAX ENTRADAS</u></a>
	Número de entradas máximo.
static int	<a href="#"><u>MAX SALIDAS</u></a>
	Número de salidas máximo.

## Constructor Summary

```
SintacticoEntidad(java.lang.String fichero, Errores errores)  
Constructor de la clase
```

## Method Summary

void	<a href="#"><u>Cabecera</u></a> ()
	Lee del fichero mientras no encuentra la palabra

		'Entity'.
	void	<a href="#"><b>cerrar()</b></a> Función para cerrar el fichero que estamos analizando.
	void	<a href="#"><b>empareja(int tk)</b></a> Intenta emparejar el token actual del analizador con el del fichero.
	boolean	<a href="#"><b>Entidad()</b></a> Analiza la entidad del fichero.
	void	<a href="#"><b>errorSint()</b></a> Añade el error al atributo errores y lanza la excepción indicando el error.
	int	<a href="#"><b>evaluar()</b></a> Evalúa una expresión y devuelve el resultado.
	int	<a href="#"><b>Exp()</b></a> Analiza una expresión aritmética.
	boolean	<a href="#"><b>Generic()</b></a> Analiza la parte de un componente genérico.
	<a href="#"><b>Entidad</b></a>	<a href="#"><b>getEntidad()</b></a> Getter para consultar la Entidad.
	void	<a href="#"><b>inicia()</b></a> Inicia el análisis sintáctico.
java.util.ArrayList< <a href="#"><b>Token</b></a> >		<a href="#"><b>pasarAPostFija()</b></a> Transforma una expresión a PostFija para poder evaluarla.
	boolean	<a href="#"><b>Puertos()</b></a> Analiza la parte correspondiente a los puertos de entrada y salida
	boolean	<a href="#"><b>RSenales()</b></a> Analiza el resto de señales.

	boolean	<a href="#"><b>RVariables ()</b></a> Analiza el resto de variables.
	boolean	<a href="#"><b>Senal ()</b></a> Analiza si está bien declarada una señal.
	boolean	<a href="#"><b>Senales ()</b></a> Analiza la parte donde se declaran todas las señales.
	int	<a href="#"><b>Tipo ()</b></a> Analiza la parte correspondiente al tipo de una señal.
	boolean	<a href="#"><b>Variable ()</b></a> Analiza la declaración de una variable y la inserta en la tabla de símbolos.
	boolean	<a href="#"><b>Variables ()</b></a> Analiza la parte de declaración de variables dentro de un genérico.

#### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait

## Field Detail

### MAX\_ENTRADAS

public static final int **MAX\_ENTRADAS**  
Número de entradas máximo. En nuestra Aplicación son 32

**See Also:**

[Constant Field Values](#)

### MAX\_SALIDAS

public static final int **MAX\_SALIDAS**  
Número de salidas máximo. En nuestra Aplicación son 32

**See Also:**

[Constant Field Values](#)

## Constructor Detail

### SintacticoEntidad

```
public SintacticoEntidad(java.lang.String fichero,  
                         Errores errores)  
    throws java.io.IOException
```

Constructor de la clase

#### Parameters:

fichero - Ruta del fichero que vamos a analizar sintácticamente.

errores - ArrayList de errores encontrados

#### Throws:

java.io.IOException

## Method Detail

### getEntidad

```
public Entidad getEntidad()  
    Getter para consultar la Entidad.
```

#### Returns:

Devuelve el Objeto de la clase Entidad

---

### inicia

```
public void inicia()  
    throws java.io.IOException  
Inicia el análisis sintáctico.
```

#### Throws:

java.io.IOException

---

### cerrar

```
public void cerrar()  
    Función para cerrar el fichero que estamos analizando.
```

---

### **Cabecera**

```
public void Cabecera()
           throws java.io.IOException
```

Lee del fichero mientras no encuentra la palabra 'Entity'. Con esto descartamos todos los comentarios iniciales además de la inclusión de librerías además de cualquier otra cosa que no nos interese del fichero.

#### **Throws:**

```
java.io.IOException
```

---

### **Entidad**

```
public boolean Entidad()
           throws java.lang.Exception
```

Analiza la entidad del fichero.

#### **Returns:**

Devuelve True si ha habido algún error, si no devuelve Falso.

#### **Throws:**

```
java.lang.Exception
```

---

### **Generic**

```
public boolean Generic()
           throws java.lang.Exception
```

Analiza la parte de un componente genérico.

#### **Returns:**

Cierto si ha habido algún error o falso si está correcto.

#### **Throws:**

```
java.lang.Exception
```

---

### **Variables**

```
public boolean Variables()
           throws java.lang.Exception
```

Analiza la parte de declaración de variables dentro de un genérico.

#### **Returns:**

Cierto si ha habido algún error o falso si está correcto.

**Throws:**

```
java.lang.Exception
```

---

**Variable**

```
public boolean Variable()
    throws java.lang.Exception
Analiza la declaración de una variable y la inserta en la tabla de símbolos.
```

**Returns:**

Cierto si ha habido algún error o falso si está correcto.

**Throws:**

```
java.lang.Exception
```

---

**RVariables**

```
public boolean RVariables()
    throws java.lang.Exception
Analiza el resto de variables.
```

**Returns:**

Cierto si ha habido algún error o falso si está correcto.

**Throws:**

```
java.lang.Exception
```

---

**Puertos**

```
public boolean Puertos()
    throws java.lang.Exception
Analiza la parte correspondiente a los puertos de entrada y salida
```

**Returns:**

Cierto si ha habido algún error o falso si está correcto.

**Throws:**

```
java.lang.Exception
```

---

**Senales**

```
public boolean Senales()
    throws java.lang.Exception
```

Analiza la parte donde se declaran todas las señales.

**Returns:**

Cierto si ha habido algún error o falso si está correcto.

**Throws:**

java.lang.Exception

---

**Senal**

```
public boolean Senal()
    throws java.lang.Exception
```

Analiza si está bien declarada una señal. Además la añade a la entidad. Si el número de entradas o de salidas es excedido se mostrará el error al final del análisis.

**Returns:**

Cierto si ha habido algún error o falso si está correcto.

**Throws:**

java.lang.Exception

---

**RSenales**

```
public boolean RSenales()
    throws java.lang.Exception
```

Analiza el resto de señales.

**Returns:**

Cierto si ha habido algún error o falso si está correcto.

**Throws:**

java.lang.Exception

---

**Tipo**

```
public int Tipo()
    throws java.lang.Exception
```

Analiza la parte correspondiente al tipo de una señal. Ésta puede ser STD\_LOGIC o STD\_LOGIC\_VECTOR. En función de ello, devuelve el tamaño de esa entrada o salida.

**Returns:**

devuelve el tamaño de una entrada o salida.

**Throws:**

```
java.lang.Exception
```

---

### **Exp**

```
public int Exp()
    throws java.lang.Exception
```

Analiza una expresión aritmética.

#### **Returns:**

Devuelve el valor de la expresión.

#### **Throws:**

```
java.lang.Exception
```

---

### **pasarAPostFija**

```
public java.util.ArrayList<Token> pasarAPostFija()
    throws java.lang.Exception
```

Transforma una expresión a PostFija para poder evaluarla.

#### **Returns:**

Devuelve el ArrayList con la expresión transformada a PostFija.

#### **Throws:**

```
java.lang.Exception
```

---

### **evaluar**

```
public int evaluar()
    throws java.lang.Exception
```

Evalúa una expresión y devuelve el resultado.

#### **Returns:**

El resultado de la expresión.

#### **Throws:**

```
java.lang.Exception
```

---

### **empareja**

```
public void empareja(int tk)
    throws java.lang.Exception
```

Intenta emparejar el token actual del analizador con el del fichero. Además actualiza el siguiente token que se debe leer, pidiéndoselo al analizador léxico.

**Parameters:**

tk - Token a emparejar

**Throws:**

java.lang.Exception

---

**errorSint**

```
public void errorSint()  
    throws java.lang.Exception
```

Añade el error al atributo errores y lanza la excepción indicando el error.

**Throws:**

java.lang.Exception

---

**Class Token**

```
java.lang.Object  
└ compiladorEntidad.Token
```

---

```
public class Token  
extends java.lang.Object
```

Clase para representar un token del fichero que contiene una entidad descrita en VHDL

---

---

## Constructor Summary

<code>Token(int codigo, java.lang.String lexema, int numLinea)</code> Constructor de la clase.
---

<code>Token (Token otro)</code> Contructora que crea una copia de un Token a partir de otro.
---

---

## Method Summary

int	<code>getCodigo()</code> Obtiene el código de un Token
-----	---

java.lang.String	<a href="#"><b>getLexema()</b></a> Obtiene la cadena del String.
int	<a href="#"><b>getNumLinea()</b></a> Obtiene el número de línea donde se encuentra el Token.

#### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait

## Constructor Detail

### Token

```
public Token(int codigo,  
           java.lang.String lexema,  
           int numLinea)
```

Constructor de la clase.

#### Parameters:

codigo - Código del Token.

lexema - String del Token

numLinea - Entero que indica la línea donde se encuentra el token dentro del fichero.

### Token

```
public Token(Token otro)
```

Contructora que crea una copia de un Token a partir de otro.

#### Parameters:

otro - Token a copiar.

## Method Detail

### getCodigo

```
public int getCodigo()  
Obtiene el código de un Token
```

**Returns:**

Entero que codifica el Token.

---

**getLexema**

public java.lang.String **getLexema()**

Obtiene la cadena del String.

**Returns:**

El String del Token asociado.

---

**getNumLinea**

public int **getNumLinea()**

Obtiene el número de línea donde se encuentra el Token.

**Returns:**

El entero con la línea donde se encuentra el Token.

## Package generadorVHDL

### Class Summary

<a href="#"><u>GeneraVhdl</u></a>	Fichero encargado de generar automáticamente el VHDL de circuito top (Circuito_FPGA.vhd) que será el que conecte el circuito que queramos ejecutar en la FPGA con nuestro módulo de entrada/salida.
-----------------------------------	---

### Class GeneraVhdl

java.lang.Object  
└ **generadorVHDL.GeneraVhdl**

---

public class **GeneraVhdl**  
extends java.lang.Object

Fichero encargado de generar automáticamente el VHDL de circuito top (Circuito\_FPGA.vhd) que será el que conecte el circuito que queramos ejecutar en la FPGA con nuestro módulo de entrada/salida. Además también contendrá los procesos necesarios para leer y escribir 32 bits desde/hacia la FPGA. esta clase está estructurada en diferentes métodos cada uno de los cuales escriben un bloque concreto del fichero.

## Constructor Summary

[\*\*GeneraVhdl\*\*](#)(java.lang.String fichero, [Entidad](#) entidad, [Errores](#) errores)

Constructor de la clase

## Method Summary

boolean	<a href="#"><b>abrir</b></a> ()	Abre el fichero para escribir el nuevo archivo vhdl.
boolean	<a href="#"><b>cerrar</b></a> ()	Cierra el fichero VHDL que estamos construyendo
void	<a href="#"><b>crearFichero</b></a> ()	Crea el nuevo fichero VHDL a partir de la entidad que tiene la clase.

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

## Constructor Detail

### GeneraVhdl

```
public GeneraVhdl(java.lang.String fichero,  
                  Entidad entidad,  
                  Errores errores)  
throws java.io.IOException
```

Constructor de la clase

#### Parameters:

fichero - Sobre el que vamos a generar el nuevo vhdl.

entidad - Entidad con la que estamos trabajando y a partir de la cual se genera el vhdl.

errores - Objeto de la clase Errores para ir almacenando los errores encontrados.

**Throws:**

java.io.IOException

## Method Detail

### abrir

public boolean **abrir()**

Abre el fichero para escribir el nuevo archivo vhdl.

**Returns:**

Cierto si no ha habido ningún error, falso en caso contrario.

---

### cerrar

public boolean **cerrar()**

Cierra el fichero VHDL que estamos construyendo

**Returns:**

Cierto si no ha habido ningún error, falso en caso contrario.

---

### crearFichero

public void **crearFichero()**

Crea el nuevo fichero VHDL a partir de la entidad que tiene la clase.

## Package IOFPGA

### Class Summary

<a href="#"><u>Ejecucion</u></a>	Clase encargada de hacer que se ejecute correctamente el circuito cargado en la FPGA.
<a href="#"><u>Reconfiguracion_Parcial</u></a>	Clase que contiene el proceso en el que se incluye el algoritmo para la reconfiguración parcial de la FPGA.

## Class Ejecucion

```
java.lang.Object
└ java.lang.Thread
    └ IOFPGA.Ejecucion
```

All Implemented Interfaces:

```
java.lang.Runnable
```

---

```
public class Ejecucion
extends java.lang.Thread
```

Clase encargada de hacer que se ejecute correctamente el circuito cargado en la FPGA. Contiene los métodos necesarios para el envío y la recepción de datos en dicho dispositivo. La ejecución se hará a través de un Test Bench o Banco de Pruebas que podrá leerse bien desde fichero, o bien desde la pantalla de la aplicación (pestaña TestBench). Hereda de la clase Thread puesto que en ocasiones queremos que la ejecución se realice en paralelo con la escritura en la interfaz gráfica, operaciones que no pueden ser realizadas a la vez si no es en hilos diferentes.

---

## Nested Class Summary

**Nested classes/interfaces inherited from class java.lang.Thread**

```
java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler
```

## Field Summary

**Fields inherited from class java.lang.Thread**

```
MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY
```

## Constructor Summary

```
Ejecucion(javax.swing.JTextField lj_jtf, Entidad e, app.Com ac_com,
javax.swing.JTextArea ata_textarea, boolean comparar,
java.lang.String nombreSalida, java.lang.String nombreTrazza,
boolean ab_reconfiguracionParcial)
```

Crea un objeto de la clase Ejecución y escribe el resultado en un JTextField .

```
Ejecucion(javax.swing.JTextField lj_jtf, Entidad e, app.Com ac_com,
javax.swing.JTextArea ata_textarea, java.io.BufferedReader l_br,
boolean comparar, java.lang.String nombreSalida,
java.lang.String nombreTrazza, boolean ab_reconfiguracionParcial)
```

Crea un objeto de la clase Ejecución y escribe el resultado en un JTextField.

## Method Summary

boolean	<a href="#"><b>convierteCadenas</b></a> ()
	Comprueba que una serie de cadenas tiene el formato correcto, es decir, tiene el número de bits adecuado y sólo está compuesta por 0's y 1's
void	<a href="#"><b>CopiarSalida</b></a> ()
	Función para copiar un fichero en otro fichero.
void	<a href="#"><b>ejecuta</b></a> ()
	Procedimiento que ejecuta el hilo.
void	<a href="#"><b>enviaReset</b></a> ()
	Realiza el envío de reset a la entidad, mandando para ello una cadena con todo ceros excepto un uno en la posición correspondiente al reset
void	<a href="#"><b>escribeEnLog</b></a> (java.lang.String s)
	Escribe la cadena s en el fichero de log en el que se introducen todos los detalles de ejecución para el caso del proceso de reconfiguración.
boolean	<a href="#"><b>formatoCorrectoFicheroTB</b></a> (java.lang.String ficheroTB)
	Comprueba si un fichero de Test Bench tiene el formato correcto, es decir, si sus cadenas tienen el tamaño correcto correspondiente con las entradas de la entidad a ejecutar, y si se trata sólamente de cadenas de 0's y 1's
boolean	<a href="#"><b>getejecutando</b></a> ()
	Método que consulta el estado de la ejecución del hilo.
void	<a href="#"><b>pararrecepcionfpga</b></a> ()
	Método que termina la ejecución de un hilo.

	void <a href="#"><u>run</u></a> () Método que comienza a ejecutar el hilo.
	void <a href="#"><u>setCadena</u></a> (java.lang.String as_cadenaaejecutar) Asigna la cadena que se quiere ejecutar para el caso de ejecución desde la pantalla de la GUI.
	void <a href="#"><u>setFileLogEjec</u></a> (java.io.FileWriter wr) Establece el fichero de escritura para el log de ejecucion
	void <a href="#"><u>setSetwait</u></a> (boolean setwait) Indica si el hilo que se está ejecutando tiene que pararse o reanudarse, según el argumento que recibe.
	int <a href="#"><u>traduceString</u></a> (java.lang.String s) Convierte a entero una una cadena en formato binario

#### Methods inherited from class java.lang.Thread

```
activeCount, checkAccess, countStackFrames, currentThread, destroy,
dumpStack, enumerate, getAllStackTraces, getContextClassLoader,
getDefaultUncaughtExceptionHandler, getId, getName, getPriority,
getStackTrace, getState, getThreadGroup, getUncaughtExceptionHandler,
holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted,
join, join, join, resume, setContextClassLoader, setDaemon,
setDefaultUncaughtExceptionHandler, setName, setPriority,
setUncaughtExceptionHandler, sleep, sleep, start, stop, stop, suspend,
toString, yield
```

#### Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait,
wait, wait
```

## Constructor Detail

### Ejecucion

```
public Ejecucion(javax.swing.JTextField lj_jtf,
                  Entidad e,
                  app.Com ac_com,
```

```
javax.swing.JTextArea ata_textarea,
boolean comparar,
java.lang.String nombreSalida,
java.lang.String nombreTraza,
boolean ab_reconfiguracionParcial)
```

Crea un objeto de la clase Ejecución y escribe el resultado en un JTextField . Este constructor está definido para el caso en el que el test bench es leído desde la pantalla de la GUI

**Parameters:**

lj\_jtf - JTextField en el que se escribirá la salida que reciba de la FPGA al ejecutar las instrucciones.

e - Entidad (top) sobre la que vamos a ejecutar.

ac\_com - Puerto con el que nos comunicamos con la FPGA.

ata\_textarea - TextArea de la aplicación del cual se recogerán las instrucciones a ejecutar.

comparar - Booleano que indica si hay que comparar los ficheros de traza y el de salida, para detectar diferencias.

nombreSalida - Fichero en el que se guardará el resultado de la ejecución.

nombreTraza - Fichero de traza con el que se podrá comparar la salida que está generando este objeto.

ab\_reconfiguracionParcial - Boolean que indica si estamos ejecutando con la opción de reconfiguración parcial. Si es falso se ejecutará el hilo directamente y no mostrará mensajes emergentes.

---

## Ejecucion

```
public Ejecucion(javax.swing.JTextField lj_jtf,
                  Entidad e,
                  app.Com ac_com,
                  javax.swing.JTextArea ata_textarea,
                  java.io.BufferedReader l_br,
                  boolean comparar,
                  java.lang.String nombreSalida,
                  java.lang.String nombreTraza,
                  boolean ab_reconfiguracionParcial)
```

Crea un objeto de la clase Ejecución y escribe el resultado en un JTextField. Este constructor está definido para el caso en el que el test bench es leído desde fichero

**Parameters:**

lj\_jtf - JTextField en el que se escribirá la salida que reciba de la FPGA al ejecutar las instrucciones.

e - Entidad top sobre la que vamos a ejecutar.

ac\_com - Puerto con el que nos comunicamos con la FPGA.

ata\_textarea - TextArea de la aplicación del cual se podrían haber recogido las instrucciones (en este caso no se utiliza).

l\_br - (BufferedReader) del que se cogerán las instrucciones a ejecutar.

comparar - Booleano que indica si hay que comparar los ficheros de traza y el de salida, para detectar diferencias.

nombreSalida - Fichero en el que se guardará el resultado de la ejecución.

nombreTraza - Fichero de traza con el que se podrá comparar la salida que está generando este objeto.

ab\_reconfiguracionParcial - Boolean que indica si estamos ejecutando con la opción de reconfiguración parcial. Si es falso se ejecutará el hilo directamente y no mostrará mensajes emergentes.

## Method Detail

### **setCadena**

```
public void setCadena(java.lang.String as_cadenaaejecutar)  
Asigna la cadena que se quiere ejecutar para el caso de ejecución desde la pantalla de la GUI. Todas las cadenas a ejecutar estarán contenidas en as_cadenaaejecutar
```

#### **Parameters:**

as\_cadenaaejecutar - Cadena a ejecutar.

---

### **traduceString**

```
public int traduceString(java.lang.String s)  
Convierte a entero una una cadena en formato binario
```

#### **Parameters:**

s - Cadena a traducir.

#### **Returns:**

Entero equivalente al String introducido o -1 si el formato no es correcto (no son 0's y 1's)

---

**setSetwait**

```
public void setSetwait(boolean setwait)
```

Indica si el hilo que se está ejecutando tiene que pararse o reanudarse, según el argumento que recibe.

**Parameters:**

setwait - boolean, valor true que indica que el hilo tiene que continuar, false para pararlo.

---

**setFileLogEjec**

```
public void setFileLogEjec(java.io.FileWriter wr)
```

Establece el fichero de escritura para el log de ejecución

**Parameters:**

wr - El fichero de escritura

---

**convierteCadenas**

```
public boolean convierteCadenas()
```

Comprueba que una serie de cadenas tiene el formato correcto, es decir, tiene el número de bits adecuado y sólo está compuesta por 0's y 1's

**Returns:**

boolean si todo es correcto.

---

**run**

```
public void run()
```

Método que comienza a ejecutar el hilo.

**Specified by:**

run in interface java.lang.Runnable

**Overrides:**

run in class java.lang.Thread

---

**getejecutando**

```
public boolean getejecutando()
```

Método que consulta el estado de la ejecución del hilo.

**Returns:**

Boolean estado de la ejecución. Ciento si está ejecutando, falso en caso contrario.

---

### **pararreceptionfpga**

```
public void pararreceptionfpga()
```

Método que termina la ejecución de un hilo.

---

### **ejecuta**

```
public void ejecuta()
```

Procedimiento que ejecuta el hilo.

---

### **CopiarSalida**

```
public void CopiarSalida()
```

```
throws java.io.IOException
```

Función para copiar un fichero en otro fichero. En nuestro caso copiamos el archivo de escritura en el de comparar con traza.

#### **Throws:**

```
java.io.IOException
```

---

### **formatoCorrectoFicheroTB**

```
public boolean formatoCorrectoFicheroTB(java.lang.String ficheroTB)
```

Comprueba si un fichero de Test Bench tiene el formato correcto, es decir, si sus cadenas tienen el tamaño correcto correspondiente con las entradas de la entidad a ejecutar, y si se trata sólamente de cadenas de 0's y 1's

#### **Parameters:**

ficheroTB - Fichero de Test Bench a analizar

#### **Returns:**

true si el formato es correcto y false en caso contrario

---

### **enviaReset**

```
public void enviaReset()
```

Realiza el envío de reset a la entidad, mandando para ello una cadena con todo ceros excepto un uno en la posición correspondiente al reset

---

### **escribeEnLog**

```
public void escribeEnLog(java.lang.String s)
```

Escribe la cadena s en el fichero de log en el que se introducen todos los detalles de ejecución para el caso del proceso de reconfiguración. Se controla que el fichero esté inicializado para el caso de ejecuciones en las que no se escribe en el log (ejecuciones simples).

#### **Parameters:**

s - La cadena que se desea escribir en el fichero de log.

## **Class Reconfiguracion\_Parcial**

```
java.lang.Object  
└ java.lang.Thread  
  └ IOFPGA.Reconfiguracion_Parcial
```

#### **All Implemented Interfaces:**

java.lang.Runnable

---

```
public class Reconfiguracion_Parcial  
extends java.lang.Thread
```

Clase que contiene el proceso en el que se incluye el algoritmo para la reconfiguración parcial de la FPGA. Necesita ser creado en una nueva clase que herede de un hilo, de tal forma que la ejecución pueda ser parada en cualquier momento desde la interfaz gráfica de usuario.

---

## **Nested Class Summary**

### **Nested classes/interfaces inherited from class java.lang.Thread**

```
java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler
```

## **Field Summary**

## Fields inherited from class java.lang.Thread

MAX\_PRIORITY, MIN\_PRIORITY, NORM\_PRIORITY

## Constructor Summary

[Reconfiguracion\\_Parcial](#) ([GUIPrincipal](#) in\_gui, java.lang.String rutaBit)

Inicia el hilo de la reconfiguración parcial

## Method Summary

void	<a href="#">pararreconfiguracionparcial</a> () Detiene el proceso de reconfiguración parcial sucesiva
boolean	<a href="#">reconfiguracionParcial</a> () Método que contiene el algoritmo que ejecuta la reconfiguración parcial sucesiva. En primer lugar se cargará una entidad seguida de su fichero .BIT correspondiente.
void	<a href="#">run</a> ()

## Methods inherited from class java.lang.Thread

activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getAllStackTraces, getContextClassLoader, getDefaultUncaughtExceptionHandler, getId, getName, getPriority, getStackTrace, getState, getThreadGroup, getUncaughtExceptionHandler, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setDefaultUncaughtExceptionHandler, setName, setPriority, setUncaughtExceptionHandler, sleep, sleep, start, stop, stop, suspend, toString, yield

## Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait,

```
wait, wait
```

## Constructor Detail

### Reconfiguracion\_Parcial

```
public Reconfiguracion_Parcial(GUIPrincipal in_gui,  
                               java.lang.String rutaBit)
```

Inicia el hilo de la reconfiguración parcial

#### Parameters:

in\_gui - GUI de entrada a copiar

rutaBit - Ruta del .BIT a modificar

## Method Detail

### reconfiguracionParcial

```
public boolean reconfiguracionParcial()
```

Método que contiene el algoritmo que ejecuta la reconfiguración parcial sucesiva. En primer lugar se cargará una entidad seguida de su fichero .BIT correspondiente. A continuación se pedirá al usuario un fichero para cargar el banco de pruebas. Tras cargarlo, y sabiendo que en la FPGA está cargado el fichero .BIT correcto (sin modificar), se generará una salida especial, llamada salida Golden, la cual será con la que comparemos en el resto de ejecuciones. Este paso sólo lo realizaremos una vez, al comienzo del proceso. A continuación entraremos en un bucle, en el cual se irá iterando para cada bit y para cada frame distinto. En cada una de las iteraciones se ejecutará la aplicación de reconfiguración aplicada siempre al mismo fichero .BIT original (introducido por el usuario) en el que iremos cambiando los parámetros de frame y bit. Cada vez que se ejecute la aplicación se generarán dos ficheros de configuración: fichero\_modif.bit y fichero\_modifRestore.bit. Cargaremos el primer fichero en la FPGA, lo que es equivalente a que una partícula solar modificara el valor del bit de la LUT que estamos modificando. Una vez injectado el error (cargado el fichero de configuración) volveremos a ejecutar el circuito en la FPGA, con las mismas entradas con las que habíamos generado la salida Golden. Compararemos las nuevas salidas obtenidas con la salida Golden para ver si el error ha incidido en la ejecución del circuito. Después tendremos que devolver a la FPGA a su “estado anterior”. Es decir, tenemos que restaurar el bit que acabamos de modificar. Para ello ordenaremos la carga del segundo fichero que se había generado: fichero\_modifRestore.bit.

#### Returns:

true si la ejecución ha sido correcta y false en caso contrario

---

**run**

```
public void run()
```

**Specified by:**

run in interface java.lang.Runnable

**Overrides:**

run in class java.lang.Thread

---

**pararreconfiguracionparcial**

```
public void pararreconfiguracionparcial()
```

Detiene el proceso de reconfiguración parcial sucesiva

## Package nessy20

### Class Summary

<a href="#"><u>CargaBit</u></a>	Clase para que contiene la funcionalidad necesaria para cargar un archivo .BIT con la aplicación IMPACT
<a href="#"><u>Filtro</u></a>	Clase que sirve para crear un filtro de extensiones a la hora de mostrar un selector de ficheros
<a href="#"><u>GUICargaTB</u></a>	Interfaz gráfica para elegir de dónde se carga un test bench
<a href="#"><u>GUICargaVHDL</u></a>	Interfaz gráfica para elegir si se carga un sólo fichero VHD o varios
<a href="#"><u>GUIConfig</u></a>	Interfaz gráfica para la configuración de la ubicación de Xilinx Created on 04-jun-2010, 16:02:42
<a href="#"><u>GUIPrincipal</u></a>	Frame principal de la aplicación
<a href="#"><u>GUISeleccionTop</u></a>	Clase para la elección de TOP entre varios VHD
<a href="#"><u>JTabbedPaneWithCloseIcon</u></a>	A JTabbedPane which has a close ('X') icon on each tab.
<a href="#"><u>Main</u></a>	Clase principal.

<b>Selección</b>	Clase donde tenemos los tipos de selecciones que se hacen en la aplicación.
------------------	---

## Class CargaBit

```
java.lang.Object
└ nessy20.CargaBit
```

---

```
public class CargaBit
extends java.lang.Object
```

Clase para que contiene la funcionalidad necesaria para cargar un archivo .BIT con la aplicación IMPACT

---

## Method Summary

boolean	<a href="#"><b>cargar</b></a> (boolean escribirEnPantalla) Procedimiento que carga un archivo .bit en la FPGA.
boolean	<a href="#"><b>existeFichero</b></a> (java.lang.String ruta) Comprueba si existe o no un fichero.

### Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll,
toString, wait, wait, wait
```

## Method Detail

### **existeFichero**

```
public boolean existeFichero(java.lang.String ruta)
Comprueba si existe o no un fichero.
```

#### Parameters:

ruta - Cadena de la ruta del fichero

**Returns:**

Boolean, cierto si existe el fichero falso en caso contrario.

---

**cargar**

```
public boolean cargar(boolean escribirEnPantalla)
    throws java.io.FileNotFoundException
Procedimiento que carga un archivo .bit en la FPGA.
```

**Parameters:**

escribirEnPantalla - Para no escribir la salida en la pantalla en caso de que estemos en el proceso de reconfiguración.

**Returns:**

Cierto si se ha conseguido cargar correctamente, falso en caso contrario.

**Throws:**

java.io.FileNotFoundException

**Class Filtro**

```
java.lang.Object
└ javax.swing.filechooser.FileFilter
    └ nessy20.Filtro
```

---

```
public class Filtro
extends javax.swing.filechooser.FileFilter
```

Clase que sirve para crear un filtro de extensiones a la hora de mostrar un selector de ficheros

---

## Constructor Summary

[Filtro](#)(java.lang.String ext)

Constructor de la clase.

[Filtro](#)(java.lang.String[] exts, java.lang.String descr)

Constructor de la clase.

## Method Summary

boolean	<a href="#">accept(java.io.File f)</a>
java.lang.String	<a href="#">getDescription()</a>

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait

## Constructor Detail

### Filtro

public **Filtro**(java.lang.String ext)

Constructor de la clase.

#### Parameters:

ext - Extensión de ficheros que deseamos que aparezcan.

---

### Filtro

public **Filtro**(java.lang.String[] exts,  
                  java.lang.String descr)

Constructor de la clase.

#### Parameters:

exts - Extensiones que deseamos que aparezcan.

descr - Extensión de una de ellas.

## Method Detail

### accept

public boolean **accept**(java.io.File f)

#### Specified by:

```
accept in class javax.swing.filechooser.FileFilter
```

---

### **getDescription**

```
public java.lang.String getDescription()
```

**Specified by:**

```
getDescription in class javax.swing.filechooser.FileFilter
```

## **Class GUICargaTB**

```
java.lang.Object
└ java.awt.Component
    └ java.awt.Container
        └ java.awt.Window
            └ java.awt.Dialog
                └ javax.swing.JDialog
                    └ nessy20.GUICargaTB
```

### **All Implemented Interfaces:**

```
java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable,
javax.accessibility.Accessible, javax.swing.RootPaneContainer,
javax.swing.WindowConstants
```

---

```
public class GUICargaTB
extends javax.swing.JDialog
```

Interfaz gráfica para elegir de dónde se carga un test bench

### **See Also:**

[Serialized Form](#)

---

## **Nested Class Summary**

### **Nested classes/interfaces inherited from class javax.swing.JDialog**

```
javax.swing.JDialog.AccessibleJDialog
```

#### **Nested classes/interfaces inherited from class java.awt.Dialog**

```
java.awt.Dialog.AccessibleAWTDialog,  
java.awt.Dialog.ModalExclusionType, java.awt.Dialog.ModalityType
```

#### **Nested classes/interfaces inherited from class java.awt.Window**

```
java.awt.Window.AccessibleAWTWindow
```

#### **Nested classes/interfaces inherited from class java.awt.Container**

```
java.awt.Container.AccessibleAWTContainer
```

#### **Nested classes/interfaces inherited from class java.awt.Component**

```
java.awt.Component.AccessibleAWTComponent,  
java.awt.Component.BaselineResizeBehavior,  
java.awt.Component.BltBufferStrategy,  
java.awt.Component.FlipBufferStrategy
```

## **Field Summary**

#### **Fields inherited from class javax.swing.JDialog**

```
accessibleContext, rootPane, rootPaneCheckingEnabled
```

#### **Fields inherited from class java.awt.Dialog**

```
DEFAULT_MODALITY_TYPE
```

#### **Fields inherited from class java.awt.Component**

```
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT,  
TOP_ALIGNMENT
```

#### Fields inherited from interface javax.swing.WindowConstants

```
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, EXIT_ON_CLOSE, HIDE_ON_CLOSE
```

#### Fields inherited from interface java.awt.image.ImageObserver

```
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH
```

## Constructor Summary

[GUICargaTB](#)(javax.swing.JFrame jf, boolean bol, [Seleccion](#) sel)

Constructor del form GUICargaTB.

## Method Summary

#### Methods inherited from class javax.swing.JDialog

```
addImpl, createRootPane, dialogInit, getAccessibleContext,  
getContentPane, getDefaultCloseOperation, getGlassPane, getGraphics,  
getJMenuBar, getLayeredPane, getRootPane, getTransferHandler,  
isDefaultLookAndFeelDecorated, isRootPaneCheckingEnabled, paramString,  
processWindowEvent, remove, repaint, setContentPane,  
setDefaultCloseOperation, setDefaultLookAndFeelDecorated,  
setGlassPane, setJMenuBar, setLayeredPane, setLayout, setRootPane,  
setRootPaneCheckingEnabled, setTransferHandler, update
```

#### Methods inherited from class java.awt.Dialog

```
addNotify, getModalityType, getTitle, hide, isModal, isResizable,  
isUndecorated, setModal, setModalityType, setResizable, setTitle,
```

```
setUndecorated, setVisible, show, toBack
```

#### Methods inherited from class java.awt.Window

```
addPropertyChangeListener, addPropertyChangeListener,
addWindowFocusListener, addWindowListener, addWindowStateListener,
applyResourceBundle, applyResourceBundle, createBufferStrategy,
createBufferStrategy, dispose, getBufferStrategy,
getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner,
getFocusTraversalKeys, getGraphicsConfiguration, getIconImages,
getInputContext, getListeners, getLocale, getModalExclusionType,
getMostRecentFocusOwner, getOwnedWindows, getOwner,
getOwnerlessWindows, getToolkit, getWarningString,
getWindowFocusListeners, getWindowListeners, getWindows,
getWindowStateListeners, isActive, isAlwaysOnTop,
isAlwaysOnTopSupported, isFocusableWindow, isFocusCycleRoot,
isFocused, isLocationByPlatform, isShowing, pack, paint, postEvent,
processEvent, processWindowFocusEvent, processWindowStateEvent,
removeNotify, removeWindowFocusListener, removeWindowListener,
removeWindowStateListener, reshape, setAlwaysOnTop, setBounds,
setBounds, setCursor, setFocusableWindowState, setFocusCycleRoot,
setIconImage, setIconImages, setLocationByPlatform,
setLocationRelativeTo, setMinimumSize, setModalExclusionType, setSize,
setSize, toFront
```

#### Methods inherited from class java.awt.Container

```
add, add, add, add, add, addContainerListener,
applyComponentOrientation, areFocusTraversalKeysSet, countComponents,
deliverEvent, doLayout, findComponentAt, findComponentAt,
getAlignmentX, getAlignmentY, getComponent, getComponentAt,
getComponentAt, getComponentCount, getComponents, getComponentZOrder,
getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout,
getMaximumSize, getMinimumSize, getMousePosition, getPreferredSize,
insets, invalidate, isAncestorOf, isFocusCycleRoot,
isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout,
list, list, locate, minimumSize, paintComponents, preferredSize,
print, printComponents, processContainerEvent, remove, removeAll,
removeContainerListener, setComponentZOrder, setFocusTraversalKeys,
setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setFont,
transferFocusBackward, transferFocusDownCycle, validate, validateTree
```

#### Methods inherited from class java.awt.Component

```
action, add, addComponentListener, addFocusListener,
addHierarchyBoundsListener, addHierarchyListener,
addInputMethodListener, addKeyListener, addMouseListener,
addMouseMotionListener, addMouseWheelListener, bounds, checkImage,
checkImage, coalesceEvents, contains, contains, createImage,
createImage, createVolatileImage, createVolatileImage, disable,
disableEvents, dispatchEvent, enable, enable, enableEvents,
enableInputMethods, firePropertyChange, firePropertyChange,
firePropertyChange, firePropertyChange, firePropertyChange,
firePropertyChange, firePropertyChange, firePropertyChange,
firePropertyChange, getBackground, getBaseline,
getBaselineResizeBehavior, getBounds, getBounds, getColorModel,
getComponentListeners, getComponentOrientation, getCursor,
getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled,
getFont, getFontMetrics, getForeground, getHeight,
getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint,
getInputMethodListeners, getInputMethodRequests, getKeyListeners,
getLocation, getLocation, getLocationOnScreen, getMouseListeners,
getMouseMotionListeners, getMousePosition, getMouseWheelListeners,
getName, getParent, getPeer, getPropertyChangeListeners,
getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth,
getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside,
isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered,
isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet,
isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet,
isOpaque, isPreferredSizeSet, isValid, isVisible, keyDown, keyUp,
list, list, list, location, lostFocus, mouseDown, mouseDrag,
mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll,
prepareImage, prepareImage, printAll, processComponentEvent,
processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent,
processInputMethodEvent, processKeyEvent, processMouseEvent,
processMouseMotionEvent, processMouseWheelEvent, remove,
removeComponentListener, removeFocusListener,
removeHierarchyBoundsListener, removeHierarchyListener,
removeInputMethodListener, removeKeyListener, removeMouseListener,
removeMouseMotionListener, removeMouseWheelListener,
removePropertyChangeListener, removePropertyChangeListener, repaint,
repaint, repaint, requestFocus, requestFocus, requestFocusInWindow,
requestFocusInWindow, resize, resize, setBackground,
setComponentOrientation, setDropTarget, setEnabled, setFocusable,
setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint,
setLocale, setLocation, setLocation, setMaximumSize, setName,
setPreferredSize, show, size, toString, transferFocus,
transferFocusUpCycle
```

#### Methods inherited from class `java.lang.Object`

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait,
```

```
wait, wait
```

## Constructor Detail

### **GUICargaTB**

```
public GUICargaTB(javax.swing.JFrame jf,  
                   boolean bol,  
                   Seleccion sel)  
Constructor del form GUICargaTB.
```

#### **Parameters:**

`jf` - Frame padre encargado de la llamada.

`sel` - Tipo de Selección a elegir.

`bol` - Indica si es modal o no.

### **Class GUICargaVHDL**

```
java.lang.Object  
└ java.awt.Component  
    └ java.awt.Container  
        └ java.awt.Window  
            └ java.awt.Dialog  
                └ javax.swing.JDialog  
                    └ nessy20.GUICargaVHDL
```

#### **All Implemented Interfaces:**

`java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable,  
javax.accessibility.Accessible, javax.swing.RootPaneContainer,  
javax.swing.WindowConstants`

---

```
public class GUICargaVHDL  
extends javax.swing.JDialog
```

Interfaz gráfica para elegir si se carga un sólo fichero VHD o varios

#### **See Also:**

[Serialized Form](#)

---

## Nested Class Summary

### Nested classes/interfaces inherited from class javax.swing.JDialog

javax.swing.JDialog.AccessibleJDialog

### Nested classes/interfaces inherited from class java.awt.Dialog

java.awt.Dialog.AccessibleAWTDialog,  
java.awt.Dialog.ModalExclusionType, java.awt.Dialog.ModalityType

### Nested classes/interfaces inherited from class java.awt.Window

java.awt.Window.AccessibleAWTWindow

### Nested classes/interfaces inherited from class java.awt.Container

java.awt.Container.AccessibleAWTContainer

### Nested classes/interfaces inherited from class java.awt.Component

java.awt.Component.AccessibleAWTComponent,  
java.awt.Component.BaselineResizeBehavior,  
java.awt.Component.BlitBufferStrategy,  
java.awt.Component.FlipBufferStrategy

## Field Summary

### Fields inherited from class javax.swing.JDialog

```
accessibleContext, rootPane, rootPaneCheckingEnabled
```

#### Fields inherited from class java.awt.Dialog

```
DEFAULT_MODALITY_TYPE
```

#### Fields inherited from class java.awt.Component

```
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT,  
TOP_ALIGNMENT
```

#### Fields inherited from interface javax.swing.WindowConstants

```
DISPOSE_ON_CLOSE, DO NOTHING ON CLOSE, EXIT_ON_CLOSE, HIDE_ON_CLOSE
```

#### Fields inherited from interface java.awt.image.ImageObserver

```
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH
```

## Constructor Summary

[GUICargaVHDL](#)(javax.swing.JFrame jf, boolean bol, [Seleccion](#) sel)

Constructor del form GUICargaVHDL.

## Method Summary

#### Methods inherited from class javax.swing.JDialog

```
addImpl, createRootPane, dialogInit, getAccessibleContext,  
getContentPane, getDefaultCloseOperation, getGlassPane, getGraphics,  
getJMenuBar, getLayeredPane, getRootPane, getTransferHandler,
```

```
isDefaultLookAndFeelDecorated, isRootPaneCheckingEnabled, paramString,
processWindowEvent, remove, repaint, setContentPane,
setDefaultCloseOperation, setDefaultLookAndFeelDecorated,
setGlassPane, setJMenuBar, setLayeredPane, setLayout, setRootPane,
setRootPaneCheckingEnabled, setTransferHandler, update
```

#### Methods inherited from class java.awt.Dialog

```
addNotify, getModalityType, getTitle, hide, isModal, isResizable,
isUndecorated, setModal, setModalityType, setResizable, setTitle,
setUndecorated, setVisible, show, toBack
```

#### Methods inherited from class java.awt.Window

```
addPropertyChangeListener, addPropertyChangeListener,
addWindowFocusListener, addWindowListener, addWindowStateListener,
applyResourceBundle, applyResourceBundle, createBufferStrategy,
createBufferStrategy, dispose, getBufferStrategy,
getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner,
getFocusTraversalKeys, getGraphicsConfiguration, getIconImages,
getInputContext, getListeners, getLocale, getModalExclusionType,
getMostRecentFocusOwner, getOwnedWindows, getOwner,
getOwnerlessWindows, getToolkit, getWarningString,
getWindowFocusListeners, getWindowListeners, getWindows,
getWindowStateListeners, isActive, isAlwaysOnTop,
isAlwaysOnTopSupported, isFocusableWindow, isFocusCycleRoot,
isFocused, isLocationByPlatform, isShowing, pack, paint, postEvent,
processEvent, processWindowFocusEvent, processWindowStateEvent,
removeNotify, removeWindowFocusListener, removeWindowListener,
removeWindowStateListener, reshape, setAlwaysOnTop, setBounds,
setBounds, setCursor, setFocusableWindowState, setFocusCycleRoot,
setIconImage, setIconImages, setLocationByPlatform,
setLocationRelativeTo, setMinimumSize, setModalExclusionType, setSize,
setSize, toFront
```

#### Methods inherited from class java.awt.Container

```
add, add, add, add, add, addContainerListener,
applyComponentOrientation, areFocusTraversalKeysSet, countComponents,
deliverEvent, doLayout, findComponentAt, findComponentAt,
getAlignmentX, getAlignmentY, getComponent, getComponentAt,
getComponentAt, getComponentCount, getComponents, getComponentZOrder,
getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout,
```

```
getMaximumSize, getMinimumSize, getMousePosition, getPreferredSize,
insets, invalidate, isAncestorOf, isFocusCycleRoot,
isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout,
list, list, locate, minimumSize, paintComponents, preferredSize,
print, printComponents, processContainerEvent, remove, removeAll,
removeContainerListener, setComponentZOrder, setFocusTraversalKeys,
setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setFont,
transferFocusBackward, transferFocusDownCycle, validate, validateTree
```

#### Methods inherited from class java.awt.Component

```
action, add, addComponentListener, addFocusListener,
addHierarchyBoundsListener, addHierarchyListener,
addInputMethodListener, addKeyListener, addMouseListener,
addMouseMotionListener, addMouseWheelListener, bounds, checkImage,
checkImage, coalesceEvents, contains, contains, createImage,
createImage, createVolatileImage, createVolatileImage, disable,
disableEvents, dispatchEvent, enable, enable, enableEvents,
enableInputMethods, firePropertyChange, firePropertyChange,
firePropertyChange, firePropertyChange, firePropertyChange,
firePropertyChange, firePropertyChange, firePropertyChange,
firePropertyChange, firePropertyChange, firePropertyChange,
firePropertyChange, getBackground, getBaseline,
getBaselineResizeBehavior, getBounds, getBounds, getColorModel,
getComponentListeners, getComponentOrientation, getCursor,
getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled,
getFont, getFontMetrics, getForeground, getHeight,
getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint,
getInputMethodListeners, getInputMethodRequests, getKeyListeners,
getLocation, getLocation, getLocationOnScreen, getMouseListeners,
getMouseMotionListeners, getMousePosition, getMouseWheelListeners,
getName, getParent, getPeer, getPropertyChangeListeners,
getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth,
getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside,
isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered,
isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet,
isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet,
isOpaque, isPreferredSizeSet, isValid, isVisible, keyDown, keyUp,
list, list, list, location, lostFocus, mouseDown, mouseDrag,
mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll,
prepareImage, prepareImage, printAll, processComponentEvent,
processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent,
processInputMethodEvent, processKeyEvent, processMouseEvent,
processMouseMotionEvent, processMouseWheelEvent, remove,
removeComponentListener, removeFocusListener,
removeHierarchyBoundsListener, removeHierarchyListener,
removeInputMethodListener, removeKeyListener, removeMouseListener,
removeMouseMotionListener, removeMouseWheelListener,
removePropertyChangeListener, removePropertyChangeListener, repaint,
repaint, repaint, requestFocus, requestFocus, requestFocusInWindow,
```

```
requestFocusInWindow, resize, resize, setBackground,
setComponentOrientation, setDropTarget, setEnabled, setFocusable,
setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint,
setLocale, setLocation, setLocation, setMaximumSize, setName,
setPreferredSize, show, size, toString, transferFocus,
transferFocusUpCycle
```

#### Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait,
wait, wait
```

## Constructor Detail

### GUICargaVHDL

```
public GUICargaVHDL(javax.swing.JFrame jf,
                      boolean bol,
                      Seleccion sel)
```

Constructor del form GUICargaVHDL.

#### Parameters:

`jf` - Frame padre encargado de la llamada.

`sel` - Tipo de Seleccion a elegir.

`bol` - Indica si es modal o no.

### Class GUIConfig

```
java.lang.Object
└ java.awt.Component
  └ java.awt.Container
    └ java.awt.Window
      └ java.awt.Dialog
        └ javax.swing.JDialog
          └ nessy20.GUIConfig
```

#### All Implemented Interfaces:

```
java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable,
javax.accessibility.Accessible, javax.swing.RootPaneContainer,
javax.swing.WindowConstants
```

---

```
public class GUIConfig
```

```
extends javax.swing.JDialog
```

Interfaz gráfica para la configuración de la ubicación de Xilinx Created on 04-jun-2010, 16:02:42

**See Also:**

[Serialized Form](#)

---

## Nested Class Summary

### Nested classes/interfaces inherited from class javax.swing.JDialog

```
javax.swing.JDialog.AccessibleJDialog
```

### Nested classes/interfaces inherited from class java.awt.Dialog

```
java.awt.Dialog.AccessibleAWTDialog,  
java.awt.Dialog.ModalExclusionType, java.awt.Dialog.ModalityType
```

### Nested classes/interfaces inherited from class java.awt.Window

```
java.awt.Window.AccessibleAWTWindow
```

### Nested classes/interfaces inherited from class java.awt.Container

```
java.awt.Container.AccessibleAWTContainer
```

### Nested classes/interfaces inherited from class java.awt.Component

```
java.awt.Component.AccessibleAWTComponent,  
java.awt.Component.BaselineResizeBehavior,  
java.awt.Component.BltBufferStrategy,  
java.awt.Component.FlipBufferStrategy
```

## Field Summary

### Fields inherited from class javax.swing.JDialog

accessibleContext, rootPane, rootPaneCheckingEnabled

### Fields inherited from class java.awt.Dialog

DEFAULT\_MODALITY\_TYPE

### Fields inherited from class java.awt.Component

BOTTOM\_ALIGNMENT, CENTER\_ALIGNMENT, LEFT\_ALIGNMENT, RIGHT\_ALIGNMENT,  
TOP\_ALIGNMENT

### Fields inherited from interface javax.swing.WindowConstants

DISPOSE\_ON\_CLOSE, DO NOTHING ON CLOSE, EXIT\_ON\_CLOSE, HIDE\_ON\_CLOSE

### Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

## Constructor Summary

[GUIConfig](#)(java.awt.Frame parent, boolean modal,  
java.lang.String ruta)

Creates new form GUIConfig

## Method Summary

### Methods inherited from class javax.swing.JDialog

```
addImpl, createRootPane, dialogInit, getAccessibleContext,  
getContentPane, getDefaultCloseOperation, getGlassPane, getGraphics,  
getJMenuBar, getLayeredPane, getRootPane, getTransferHandler,  
isDefaultLookAndFeelDecorated, isRootPaneCheckingEnabled, paramString,  
processWindowEvent, remove, repaint, setContentPane,  
setDefaultCloseOperation, setDefaultCloseOperation,  
setGlassPane, setJMenuBar, setLayeredPane, setLayout, setRootPane,  
setRootPaneCheckingEnabled, setTransferHandler, update
```

### Methods inherited from class java.awt.Dialog

```
addNotify, getModalityType, getTitle, hide, isModal, isResizable,  
isUndecorated, setModal, setModalityType, setResizable, setTitle,  
setUndecorated, setVisible, show, toBack
```

### Methods inherited from class java.awt.Window

```
addPropertyChangeListener, addPropertyChangeListener,  
addWindowFocusListener, addWindowListener, addWindowStateListener,  
applyResourceBundle, applyResourceBundle, createBufferStrategy,  
createBufferStrategy, dispose, getBufferStrategy,  
getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner,  
getFocusTraversalKeys, getGraphicsConfiguration, getIconImages,  
getInputContext, getListeners, getLocale, getModalExclusionType,  
getMostRecentFocusOwner, getOwnedWindows, getOwner,  
getOwnerlessWindows, getToolkit, getWarningString,  
getWindowFocusListeners, getWindowListeners, getWindows,  
getWindowStateListeners, isActive, isAlwaysOnTop,  
isAlwaysOnTopSupported, isFocusableWindow, isFocusCycleRoot,  
isFocused, isLocationByPlatform, isShowing, pack, paint, postEvent,  
processEvent, processWindowFocusEvent, processWindowStateEvent,  
removeNotify, removeWindowFocusListener, removeWindowListener,  
removeWindowStateListener, reshape, setAlwaysOnTop, setBounds,  
setBounds, setCursor, setFocusableWindowState, setFocusCycleRoot,  
setIconImage, setIconImages, setLocationByPlatform,  
setLocationRelativeTo, setMinimumSize, setModalExclusionType, setSize,  
setSize, toFront
```

#### **Methods inherited from class java.awt.Container**

```
add, add, add, add, add, addContainerListener,  
applyComponentOrientation, areFocusTraversalKeysSet, countComponents,  
deliverEvent, doLayout, findComponentAt, findComponentAt,  
getAlignmentX, getAlignmentY, getComponent, getComponentAt,  
getComponentAt, getComponentCount, getComponents, getComponentZOrder,  
getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout,  
getMaximumSize, getMinimumSize, getMousePosition, getPreferredSize,  
insets, invalidate, isAncestorOf, isFocusCycleRoot,  
isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout,  
list, list, locate, minimumSize, paintComponents, preferredSize,  
print, printComponents, processContainerEvent, remove, removeAll,  
removeContainerListener, setComponentZOrder, setFocusTraversalKeys,  
setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setFont,  
transferFocusBackward, transferFocusDownCycle, validate, validateTree
```

#### **Methods inherited from class java.awt.Component**

```
action, add, addComponentListener, addFocusListener,  
addHierarchyBoundsListener, addHierarchyListener,  
addInputMethodListener, addKeyListener, addMouseListener,  
addMouseMotionListener, addMouseWheelListener, bounds, checkImage,  
checkImage, coalesceEvents, contains, contains, createImage,  
createImage, createVolatileImage, createVolatileImage, disable,  
disableEvents, dispatchEvent, enable, enable, enableEvents,  
enableInputMethods, firePropertyChange, firePropertyChange,  
firePropertyChange, firePropertyChange, firePropertyChange,  
firePropertyChange, firePropertyChange, firePropertyChange,  
firePropertyChange, firePropertyChange, firePropertyChange,  
firePropertyChange, getBackground, getBaseline,  
getBaselineResizeBehavior, getBounds, getBounds, getColorModel,  
getComponentListeners, getComponentOrientation, getCursor,  
getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled,  
getFont, getFontMetrics, getForeground, getHeight,  
getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint,  
getInputMethodListeners, getInputMethodRequests, getKeyListeners,  
getLocation, getLocation, getLocationOnScreen, getMouseListeners,  
getMouseMotionListeners, getMousePosition, getMouseWheelListeners,  
getName, getParent, getPeer, getPropertyChangeListeners,  
getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth,  
getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside,  
isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered,  
isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet,  
isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet,  
isOpaque, isPreferredSizeSet, isValid, isVisible, keyDown, keyUp,  
list, list, list, location, lostFocus, mouseDown, mouseDrag,  
mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll,  
prepareImage, prepareImage, printAll, processComponentEvent,
```

```
processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent,
processInputMethodEvent, processKeyEvent, processMouseEvent,
processMouseMotionEvent, processMouseWheelEvent, remove,
removeComponentListener, removeFocusListener,
removeHierarchyBoundsListener, removeHierarchyListener,
removeInputMethodListener, removeKeyListener, removeMouseListener,
removeMouseMotionListener, removeMouseWheelListener,
removePropertyChangeListener, removePropertyChangeListener, repaint,
repaint, repaint, requestFocus, requestFocus, requestFocusInWindow,
requestFocusInWindow, resize, resize, setBackground,
setComponentOrientation, setDropTarget, setEnabled, setFocusable,
setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint,
setLocale, setLocation, setLocation, setMaximumSize, setName,
setPreferredSize, show, size, toString, transferFocus,
transferFocusUpCycle
```

#### Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait,
wait, wait
```

## Constructor Detail

### GUIConfig

```
public GUIConfig(java.awt.Frame parent,
                  boolean modal,
                  java.lang.String ruta)
```

Creates new form GUIConfig

## Class GUIPrincipal

```
java.lang.Object
  ↘java.awt.Component
    ↘java.awt.Container
      ↘java.awt.Window
        ↘java.awt.Frame
          ↘javax.swing.JFrame
            ↘nessy20.GUIPrincipal
```

### All Implemented Interfaces:

```
java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable,
javax.accessibility.Accessible, javax.swing.RootPaneContainer,
javax.swing.WindowConstants
```

---

```
public class GUIPrincipal
extends javax.swing.JFrame
```

Frame principal de la aplicación

**See Also:**

[Serialized Form](#)

---

## Nested Class Summary

### Nested classes/interfaces inherited from class javax.swing.JFrame

```
javax.swing.JFrame.AccessibleJFrame
```

### Nested classes/interfaces inherited from class java.awt.Frame

```
java.awt.Frame.AccessibleAWTFrame
```

### Nested classes/interfaces inherited from class java.awt.Window

```
java.awt.Window.AccessibleAWTWindow
```

### Nested classes/interfaces inherited from class java.awt.Container

```
java.awt.Container.AccessibleAWTContainer
```

### Nested classes/interfaces inherited from class java.awt.Component

```
java.awt.Component.AccessibleAWTComponent,
java.awt.Component.BaselineResizeBehavior,
java.awt.Component.BltBufferStrategy,
```

```
java.awt.Component.FlipBufferStrategy
```

## Field Summary

### Fields inherited from class javax.swing.JFrame

```
accessibleContext, EXIT_ON_CLOSE, rootPane, rootPaneCheckingEnabled
```

### Fields inherited from class java.awt.Frame

```
CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR,  
ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT,  
MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL,  
NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR,  
TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR
```

### Fields inherited from class java.awt.Component

```
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT,  
TOP_ALIGNMENT
```

### Fields inherited from interface javax.swing.WindowConstants

```
DISPOSE_ON_CLOSE, DO NOTHING ON CLOSE, HIDE ON CLOSE
```

### Fields inherited from interface java.awt.image.ImageObserver

```
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH
```

## Constructor Summary

### GUIPrincipal()

Constructor de la clase.

## Method Summary

boolean	<a href="#"><u>cargarBit</u></a> (java.lang.String fichero_bit, , boolean ab_mostrar_mensajes)  Carga un fichero .BIT utilizando la interfaz de carga.
boolean	<a href="#"><u>cargarBitConChooser</u></a> ()  Función para cargar el archivo .bit.
boolean	<a href="#"><u>compilarEntidad</u></a> ()  Compila la entidad y genera el archivo VHDL a partir de ella de tal forma que se interconecte con el módulo de entrada/salida.
void	<a href="#"><u>ejec</u></a> (boolean lb_reconfiguracionParcial )  Realiza la ejecución de un circuito.
void	<a href="#"><u>escribirEnPantalla</u></a> (java.lang.String str)  Escribe una cadena que se le pasa como argumento en el text Area correspondiente al proceso de cargar un archivo .bit en la FPGA
boolean	<a href="#"><u>generarGolden</u></a> ()  Genera el archivo Golden.txt que será el fichero con el que compararemos nuestras salidas.
app.Com	<a href="#"><u>getCom1</u></a> ()  Obtiene el interfaz de puerto serie
Entidad	<a href="#"><u>getEntidad</u></a> ()  Devuelve la Entidad con la que se está trabajado
java.lang.String	<a href="#"><u>getFichero_bit</u></a> ()  Devuelve la ruta en la que se encuentra el fichero .BIT elegido por el usuario

java.util.ArrayList<java.io.Fi le>	<b><u>getFiles()</u></b> Devuelve el Array de ficheros Vhdl Cargados en la aplicación.
boolean	<b><u>inicializarPuertoSerie()</u></b> Función que inicializa el puerto serie necesario para la comunicación con la FPGA.
void	<b><u>seleccionaPanel</u></b> (javax.swing.JPanel panel) Selecciona uno de las cuatro pestañas según lo pasado por parámetro
boolean	<b><u>SeleccionTBModifFichero()</u></b> Abre un selector de ficheros para seleccionar un test bench y a continuación genera una salida golden a partir de él.
void	<b><u>setCerradoTop</u></b> (boolean cerradoTop) Establece el campo cerradoTop, con el valor del argumento de la función.
void	<b><u>setFwLog</u></b> (java.io.FileWriter fw) Establece el fichero de escritura para el fichero de log del proceso de reconfiguracion
void	<b><u>setNumeroInst</u></b> (int inst) Actualiza el numero de instrucción que se está ejecutando.
void	<b><u>setTop</u></b> (int top) Establece el fichero Top

#### Methods inherited from class javax.swing.JFrame

```
addImpl, createRootPane, frameInit, getAccessibleContext,
getContentPane, getDefaultCloseOperation, getGlassPane, getGraphics,
getJMenuBar, getLayeredPane, getRootPane, getTransferHandler,
isDefaultLookAndFeelDecorated, isRootPaneCheckingEnabled, paramString,
processWindowEvent, remove, repaint, setContentPane,
setDefaultCloseOperation, setDefaultLookAndFeelDecorated,
setGlassPane, setIconImage, setJMenuBar, setLayeredPane, setLayout,
setRootPane, setRootPaneCheckingEnabled, setTransferHandler, update
```

#### **Methods inherited from class java.awt.Frame**

```
addNotify, getCursorType, getExtendedState, getFrames, getIconImage,  
getMaximizedBounds, getMenuBar, getState, getTitle, isResizable,  
isUndecorated, remove, removeNotify, setCursor, setExtendedState,  
setMaximizedBounds, setMenuBar, setResizable, setState, setTitle,  
setUndecorated
```

#### **Methods inherited from class java.awt.Window**

```
addPropertyChangeListener, addPropertyChangeListener,  
addWindowFocusListener, addWindowListener, addWindowStateListener,  
applyResourceBundle, applyResourceBundle, createBufferStrategy,  
createBufferStrategy, dispose, getBufferStrategy,  
getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner,  
getFocusTraversalKeys, getGraphicsConfiguration, getIconImages,  
getInputContext, getListeners, getLocale, getModalExclusionType,  
getMostRecentFocusOwner, getOwnedWindows, getOwner,  
getOwnerlessWindows, getToolkit, getWarningString,  
getWindowFocusListeners, getWindowListeners, getWindows,  
getWindowStateListeners, hide, isActive, isAlwaysOnTop,  
isAlwaysOnTopSupported, isFocusableWindow, isFocusCycleRoot,  
isFocused, isLocationByPlatform, isShowing, pack, paint, postEvent,  
processEvent, processWindowFocusEvent, processWindowStateEvent,  
removeWindowFocusListener, removeWindowListener,  
removeWindowStateListener, reshape, setAlwaysOnTop, setBounds,  
setBounds, setCursor, setFocusableWindowState, setFocusCycleRoot,  
setIconImages, setLocationByPlatform, setLocationRelativeTo,  
setMinimumSize, setModalExclusionType, setSize, setSize, setVisible,  
show, toBack, toFront
```

#### **Methods inherited from class java.awt.Container**

```
add, add, add, add, add, addContainerListener,  
applyComponentOrientation, areFocusTraversalKeysSet, countComponents,  
deliverEvent, doLayout, findComponentAt, findComponentAt,  
getAlignmentX, getAlignmentY, getComponent, getComponentAt,  
getComponentAt, getComponentCount, getComponents, getComponentZOrder,  
getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout,  
getMaximumSize, getMinimumSize, getMousePosition, getPreferredSize,  
insets, invalidate, isAncestorOf, isFocusCycleRoot,  
isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout,  
list, list, locate, minimumSize, paintComponents, preferredSize,  
print, printComponents, processContainerEvent, remove, removeAll,
```

```
removeContainerListener, setComponentZOrder, setFocusTraversalKeys,
setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setFont,
transferFocusBackward, transferFocusDownCycle, validate, validateTree
```

### Methods inherited from class java.awt.Component

```
action, add, addComponentListener, addFocusListener,
addHierarchyBoundsListener, addHierarchyListener,
addInputMethodListener, addKeyListener, addMouseListener,
addMouseMotionListener, addMouseWheelListener, bounds, checkImage,
checkImage, coalesceEvents, contains, contains, createImage,
createImage, createVolatileImage, createVolatileImage, disable,
disableEvents, dispatchEvent, enable, enable, enableEvents,
enableInputMethods, firePropertyChange, firePropertyChange,
firePropertyChange, firePropertyChange, firePropertyChange,
firePropertyChange, firePropertyChange, firePropertyChange,
firePropertyChange, getBackground, getBaseline,
getBaselineResizeBehavior, getBounds, getBounds, getColorModel,
getComponentListeners, getComponentOrientation, getCursor,
getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled,
getFont, getFontMetrics, getForeground, getHeight,
getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint,
getInputMethodListeners, getInputMethodRequests, getKeyListeners,
getLocation, getLocation, getLocationOnScreen, getMouseListeners,
getMouseMotionListeners, getMousePosition, getMouseWheelListeners,
getName, getParent, getPeer, getPropertyChangeListeners,
getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth,
getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside,
isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered,
isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet,
isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet,
isOpaque, isPreferredSizeSet, isValid, isVisible, keyDown, keyUp,
list, list, list, location, lostFocus, mouseDown, mouseDrag,
mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll,
prepareImage, prepareImage, printAll, processComponentEvent,
processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent,
processInputMethodEvent, processKeyEvent, processMouseEvent,
processMouseMotionEvent, processMouseWheelEvent,
removeComponentListener, removeFocusListener,
removeHierarchyBoundsListener, removeHierarchyListener,
removeInputMethodListener, removeKeyListener, removeMouseListener,
removeMouseMotionListener, removeMouseWheelListener,
removePropertyChangeListener, removePropertyChangeListener, repaint,
repaint, repaint, requestFocus, requestFocus, requestFocusInWindow,
requestFocusInWindow, resize, resize, setBackground,
setComponentOrientation, setDropTarget, setEnabled, setFocusable,
setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint,
setLocale, setLocation, setLocation, setMaximumSize, setName,
setPreferredSize, show, size, toString, transferFocus,
```

```
transferFocusUpCycle
```

#### Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait,  
wait, wait
```

#### Methods inherited from interface java.awt.MenuContainer

```
getFont, postEvent
```

## Constructor Detail

### **GUIPrincipal**

```
public GUIPrincipal()
```

Constructor de la clase. Los botones reanudar y parar ejecución se ponen a no visibles.

## Method Detail

### **getFiles**

```
public java.util.ArrayList<java.io.File> getFiles()
```

Devuelve el Array de ficheros Vhdl Cargados en la aplicación.

#### Returns:

Valor del atributo ficherosVHD

---

### **setTop**

```
public void setTop(int top)
```

Establece el fichero Top

#### Parameters:

top - Índice del archivo TOP.

---

### **getEntidad**

```
public Entidad getEntidad()
```

Devuelve la Entidad con la que se está trabajado

**Returns:**

Entidad con la que se está trabajando.

---

**setCerradoTop**

```
public void setCerradoTop(boolean cerradoTop)
```

Establece el campo cerradoTop, con el valor del argumento de la función.

**Parameters:**

cerradoTop - Nuevo valor de tipo boolean de cerradoTop.

---

**generarGolden**

```
public boolean generarGolden()
```

Genera el archivo Golden.txt que será el fichero con el que compararemos nuestras salidas.

**Returns:**

Cierto si todo ha sido correcto, falso si ha habido algún error.

---

**cargarBitConChooser**

```
public boolean cargarBitConChooser()
```

Función para cargar el archivo .bit. Abre un JFileChooser para elegir el archivo a cargar.

Tras la elección intenta cargar el fichero de configuración en la FPGA.

**Returns:**

Cierto si todo ha sido correcto, falso si ha habido algún error.

---

**getFichero\_bit**

```
public java.lang.String getFichero_bit()
```

Devuelve la ruta en la que se encuentra el fichero .BIT elegido por el usuario

**Returns:**

El valor del atributo fichero\_bit

---

**compilarEntidad**

```
public boolean compilarEntidad()
```

Compila la entidad y genera el archivo VHDL a partir de ella de tal forma que se interconecte con el módulo de entrada/salida.

**Returns:**

Cierto si todo ha sido correcto, falso si ha habido algún error.

---

### **inicializarPuertoSerie**

```
public boolean inicializarPuertoSerie()
```

Función que inicializa el puerto serie necesario para la comunicación con la FPGA.

Comprueba que esté libre el puerto y que la maquina sobre la que estamos ejecutando tenga el puerto COM1.

**Returns:**

Cierto si todo ha sido correcto, falso si ha habido algún error.

---

### **ejec**

```
public void ejec(boolean lb_reconfiguracionParcial)
```

Realiza la ejecución de un circuito. Para ello lee las entradas del banco de pruebas, las cuales pueden ser ofrecidas desde la pantalla de la aplicación o directamente desde fichero. A continuación comprueba que el formato de las entradas es correcto (están compuestas por 0's y 1's de la misma longitud que el número de entradas que la entidad cargada). Si la ejecución actual no pertenece a uno de los pasos de la reconfiguración parcial, entonces por cada ejecución se lanzará un nuevo hilo, de tal forma que podamos visualizar la salida a medida que se va produciendo. Además este hilo podrá ser detenido en cualquier momento y reanudado más adelante. Si por el contrario sí pertenece a la reconfiguración parcial, no queremos que nada más ocurra mientras se está ejecutando, para evitar así que por ejemplo se cargue el próximo .BIT mientras se está aun ejecutando con el anterior. Por lo tanto no se lanzará un nuevo hilo sino que se hará de forma secuencial.

**Parameters:**

lb\_reconfiguracionParcial - Indica si pertenece a un paso de la reconfiguración parcial

---

### **setFwLog**

```
public void setFwLog(java.io.FileWriter fw)
```

Establece el fichero de escritura para el fichero de log del proceso de reconfiguracion

**Parameters:**

fw -

---

**escribirEnPantalla**

```
public void escribirEnPantalla(java.lang.String str)
    Escribe una cadena que se le pasa como argumento en el text Area correspondiente al
    proceso de cargar un archivo .bit en la FPGA
```

**Parameters:**

str - Cadena a escribir.

---

**cargarBit**

```
public boolean cargarBit(java.lang.String fichero_bit,
                        boolean ab_mostrar_mensajes)
    Carga un fichero .BIT utilizando la interfaz de carga. Realiza 6 intentos de carga por
    hubiera algún error ajeno a la aplicación que impidiera su carga. Si no estamos en el
    proceso de reconfiguración parcial, se mostrará un mensaje indicando que la carga del
    bitstream se ha producido correctamente.
```

**Parameters:**

fichero\_bit - El fichero a cargar

ab\_mostrar\_mensajes - Indica si hay que mostrar mensajes de información de lo ocurrido.

**Returns:**

true si ha sido correcta la ejecución y false en caso contrario

---

**setNumeroInst**

```
public void setNumeroInst(int inst)
    Actualiza el numero de instrucción que se está ejecutando.
```

**Parameters:**

inst - Número de instrucción actual.

---

**seleccionaPanel**

```
public void seleccionaPanel(javax.swing.JPanel panel)
    Selecciona uno de las cuatro pestañas según lo pasado por parámetro
```

**Parameters:**

panel - Panel al que se quiere cambiar

---

### **SeleccionTBModifFichero**

```
public boolean SeleccionTBModifFichero()  
    Abre un selector de ficheros para seleccionar un test bench y a continuación genera  
    una salida golden a partir de él. Utilizado para el comienzo del proceso de  
    reconfiguración parcial
```

**Returns:**

true si ha sido correcto y false en caso contrario

---

### **getCom1**

```
public app.Com getCom1()  
    Obtiene el interfaz de puerto serie
```

**Returns:**

El valor del atributo com1

---

## **Class GUISeleccionTop**

```
java.lang.Object  
└ java.awt.Component  
    └ java.awt.Container  
        └ java.awt.Window  
            └ java.awt.Dialog  
                └ javax.swing.JDialog  
                    └ nessy20.GUISeleccionTop
```

**All Implemented Interfaces:**

```
java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable,  
javax.accessibility.Accessible, javax.swing.RootPaneContainer,  
javax.swing.WindowConstants
```

---

```
public class GUISeleccionTop  
extends javax.swing.JDialog
```

Clase para la elección de TOP entre varios VHD

**See Also:**

[Serialized Form](#)

---

## Nested Class Summary

### Nested classes/interfaces inherited from class javax.swing.JDialog

javax.swing.JDialog.AccessibleJDialog

### Nested classes/interfaces inherited from class java.awt.Dialog

java.awt.Dialog.AccessibleAWTDialog,  
java.awt.Dialog.ModalExclusionType, java.awt.Dialog.ModalityType

### Nested classes/interfaces inherited from class java.awt.Window

java.awt.Window.AccessibleAWTWindow

### Nested classes/interfaces inherited from class java.awt.Container

java.awt.Container.AccessibleAWTContainer

### Nested classes/interfaces inherited from class java.awt.Component

java.awt.Component.AccessibleAWTComponent,  
java.awt.Component.BaselineResizeBehavior,  
java.awt.Component.BlitBufferStrategy,  
java.awt.Component.FlipBufferStrategy

## Field Summary

### Fields inherited from class javax.swing.JDialog

```
accessibleContext, rootPane, rootPaneCheckingEnabled
```

#### Fields inherited from class java.awt.Dialog

```
DEFAULT_MODALITY_TYPE
```

#### Fields inherited from class java.awt.Component

```
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT,  
TOP_ALIGNMENT
```

#### Fields inherited from interface javax.swing.WindowConstants

```
DISPOSE_ON_CLOSE, DO NOTHING ON CLOSE, EXIT_ON_CLOSE, HIDE_ON_CLOSE
```

#### Fields inherited from interface java.awt.image.ImageObserver

```
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH
```

## Constructor Summary

```
GUISeleccionTop(java.awt.Frame parent, boolean modal,  
java.util.ArrayList<java.lang.String> fich)
```

Constructor de form GUISeleccionTop

## Method Summary

#### Methods inherited from class javax.swing.JDialog

```
addImpl, createRootPane, dialogInit, getAccessibleContext,  
getContentPane, getDefaultCloseOperation, getGlassPane, getGraphics,
```

```
getJMenuBar, getLayeredPane, getRootPane, getTransferHandler,  
isDefaultLookAndFeelDecorated, isRootPaneCheckingEnabled, paramString,  
processWindowEvent, remove, repaint, setContentPane,  
setDefaultCloseOperation, setDefaultLookAndFeelDecorated,  
setGlassPane, setJMenuBar, setLayeredPane, setLayout, setRootPane,  
setRootPaneCheckingEnabled, setTransferHandler, update
```

#### Methods inherited from class java.awt.Dialog

```
addNotify, getModalityType, getTitle, hide, isModal, isResizable,  
isUndecorated, setModal, setModalityType, setResizable, setTitle,  
setUndecorated, setVisible, show, toBack
```

#### Methods inherited from class java.awt.Window

```
addPropertyChangeListener, addPropertyChangeListener,  
addWindowFocusListener, addWindowListener, addWindowStateListener,  
applyResourceBundle, applyResourceBundle, createBufferStrategy,  
createBufferStrategy, dispose, getBufferStrategy,  
getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner,  
getFocusTraversalKeys, getGraphicsConfiguration, getIconImages,  
getInputContext, getListeners, getLocale, getModalExclusionType,  
getMostRecentFocusOwner, getOwnedWindows, getOwner,  
getOwnerlessWindows, getToolkit, getWarningString,  
getWindowFocusListeners, getWindowListeners, getWindows,  
getWindowStateListeners, isActive, isAlwaysOnTop,  
isAlwaysOnTopSupported, isFocusableWindow, isFocusCycleRoot,  
isFocused, isLocationByPlatform, isShowing, pack, paint, postEvent,  
processEvent, processWindowFocusEvent, processWindowStateEvent,  
removeNotify, removeWindowFocusListener, removeWindowListener,  
removeWindowStateListener, reshape, setAlwaysOnTop, setBounds,  
setBounds, setCursor, setFocusableWindowState, setFocusCycleRoot,  
setIconImage, setIconImages, setLocationByPlatform,  
setLocationRelativeTo, setMinimumSize, setModalExclusionType, setSize,  
setSize, toFront
```

#### Methods inherited from class java.awt.Container

```
add, add, add, add, add, addContainerListener,  
applyComponentOrientation, areFocusTraversalKeysSet, countComponents,  
deliverEvent, doLayout, findComponentAt, findComponentAt,  
getAlignmentX, getAlignmentY, getComponent, getComponentAt,  
getComponentAt, getComponentCount, getComponents, getComponentZOrder,
```

```
getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout,
getMaximumSize, getMinimumSize, getMousePosition, getPreferredSize,
insets, invalidate, isAncestorOf, isFocusCycleRoot,
isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout,
list, list, locate, minimumSize, paintComponents, preferredSize,
print, printComponents, processContainerEvent, remove, removeAll,
removeContainerListener, setComponentZOrder, setFocusTraversalKeys,
setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setFont,
transferFocusBackward, transferFocusDownCycle, validate, validateTree
```

#### Methods inherited from class java.awt.Component

```
action, add, addComponentListener, addFocusListener,
addHierarchyBoundsListener, addHierarchyListener,
addInputMethodListener, addKeyListener, addMouseListener,
addMouseMotionListener, addMouseWheelListener, bounds, checkImage,
checkImage, coalesceEvents, contains, contains, createImage,
createImage, createVolatileImage, createVolatileImage, disable,
disableEvents, dispatchEvent, enable, enable, enableEvents,
enableInputMethods, firePropertyChange, firePropertyChange,
firePropertyChange, firePropertyChange, firePropertyChange,
firePropertyChange, firePropertyChange, firePropertyChange,
firePropertyChange, firePropertyChange, firePropertyChange,
firePropertyChange, getBackground, getBaseline,
getBaselineResizeBehavior, getBounds, getBounds, getColorModel,
getComponentListeners, getComponentOrientation, getCursor,
getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled,
getFont, getFontMetrics, getForeground, getHeight,
getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint,
getInputMethodListeners, getInputMethodRequests, getKeyListeners,
getLocation, getLocation, getLocationOnScreen, getMouseListeners,
getMouseMotionListeners, getMousePosition, getMouseWheelListeners,
getName, getParent, getPeer, getPropertyChangeListeners,
getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth,
getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside,
isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered,
isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet,
isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet,
isOpaque, isPreferredSizeSet, isValid, isVisible, keyDown, keyUp,
list, list, list, location, lostFocus, mouseDown, mouseDrag,
mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll,
prepareImage, prepareImage, printAll, processComponentEvent,
processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent,
processInputMethodEvent, processKeyEvent, processMouseEvent,
processMouseMotionEvent, processMouseWheelEvent, remove,
removeComponentListener, removeFocusListener,
removeHierarchyBoundsListener, removeHierarchyListener,
removeInputMethodListener, removeKeyListener, removeMouseListener,
removeMouseMotionListener, removeMouseWheelListener,
removePropertyChangeListener, removePropertyChangeListener, repaint,
```

```
repaint, repaint, requestFocus, requestFocus, requestFocusInWindow,  
requestFocusInWindow, resize, resize, setBackground,  
setComponentOrientation, setDropTarget, setEnabled, setFocusable,  
setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint,  
setLocale, setLocation, setLocation, setMaximumSize, setName,  
setPreferredSize, show, size, toString, transferFocus,  
transferFocusUpCycle
```

#### Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait,  
wait, wait
```

## Constructor Detail

### GUISeleccionTop

```
public GUISeleccionTop(java.awt.Frame parent,  
                      boolean modal,  
                      java.util.ArrayList<java.lang.String> fich)  
Constructor de form GUISeleccionTop
```

#### Parameters:

parent - Frame padre encargado de la llamada.

modal - Indica si es modal o no.

fich - ArrayList de ficheros seleccionados.

## Class Main

```
java.lang.Object  
└ nessy20.Main
```

---

```
public class Main  
extends java.lang.Object
```

Clase principal. Lanza la interfaz gráfica

## Constructor Summary

[Main \(\)](#)

## Method Summary

static void [main](#)(java.lang.String[] args)  
Ejecuta la aplicación.

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

## Constructor Detail

### Main

public [Main \(\)](#)

## Method Detail

### main

public static void [main](#)(java.lang.String[] args)  
Ejecuta la aplicación.

#### Parameters:

args - the command line arguments

## Class Selection

java.lang.Object  
└ **nessy20.Seleccion**

```
public class Seleccion  
extends java.lang.Object
```

Clase donde tenemos los tipos de selecciones que se hacen en la aplicación. Ya sea al elegir el tipo de carga de VHDL's o del TestBench

---

## Field Summary

<a href="#">SeleccionCargaVHD</a>	<a href="#"><b>seleccion</b></a>
	Seleccion de VHD
<a href="#">SeleccionTB</a>	<a href="#"><b>selTB</b></a>
	Seleccion de carga de banco de pruebas

## Constructor Summary

[Seleccion \(\)](#)

## Method Summary

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

## Field Detail

### **seleccion**

```
public SeleccionCargaVHD seleccion  
Seleccion de VHD
```

---

```
selTB  
public SeleccionTB selTB  
    Seleccion de carga de banco de pruebas
```

## Constructor Detail

### Seleccion

```
public Seleccion\(\)
```

## Enum SeleccionCargaVHD

```
java.lang.Object  
└ java.lang.Enum<SeleccionCargaVHD>  
    └ nessy20.SeleccionCargaVHD
```

### All Implemented Interfaces:

```
java.io.Serializable, java.lang.Comparable<SeleccionCargaVHD>
```

---

```
public enum SeleccionCargaVHD  
extends java.lang.Enum<SeleccionCargaVHD>
```

Opciones al seleccionar la Carga los VHDL's en nuestra aplicación.

---

## Enum Constant Summary

[NADA](#)

[SELECCION\\_VARIOS\\_VHDL](#)

[SELECCION\\_VHDL\\_TOP](#)

## Method Summary

static <a href="#">SeleccionCargaVHD</a> <a href="#"><u>valueOf</u></a> (java.lang.String name)	Returns the enum constant of this type with the
---	---

	specified name.
static <a href="#">SeleccionCargaVHD[]</a> <b>values()</b>	Returns an array containing the constants of this enum type, in the order they are declared.

#### Methods inherited from class java.lang.Enum

clone, compareTo, equals, finalize, getDeclaringClass, hashCode, name, ordinal, toString, valueOf

#### Methods inherited from class java.lang.Object

getClass, notify, notifyAll, wait, wait, wait

## Enum Constant Detail

### NADA

public static final [SeleccionCargaVHD](#) **NADA**

---

### SELECCION\_VHDL\_TOP

public static final [SeleccionCargaVHD](#) **SELECCION\_VHDL\_TOP**

---

### SELECCION\_VARIOS\_VHDL

public static final [SeleccionCargaVHD](#) **SELECCION\_VARIOS\_VHDL**

## Method Detail

### values

public static [SeleccionCargaVHD\[\]](#) **values()**

Returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants as follows:

```
for (SeleccionCargaVHD c : SeleccionCargaVHD.values())
    System.out.println(c);
```

#### Returns:

an array containing the constants of this enum type, in the order they are declared

---

### **valueOf**

```
public static SeleccionCargaVHD valueOf(java.lang.String name)
    Returns the enum constant of this type with the specified name. The string must
    match exactly an identifier used to declare an enum constant in this type. (Extraneous
    whitespace characters are not permitted.)
```

#### **Parameters:**

name - the name of the enum constant to be returned.

#### **Returns:**

the enum constant with the specified name

#### **Throws:**

java.lang.IllegalArgumentException - if this enum type has no constant with  
the specified name

java.lang.NullPointerException - if the argument is null

---

## **Enum SeleccionTB**

```
java.lang.Object
└ java.lang.Enum<SeleccionTB>
    └ nessy20.SeleccionTB
```

#### **All Implemented Interfaces:**

java.io.Serializable, java.lang.Comparable<[SeleccionTB](#)>

---

```
public enum SeleccionTB
extends java.lang.Enum<SeleccionTB>
```

Opciones al seleccionar la Carga de un Test Bench en nuestra aplicación.

---

## **Enum Constant Summary**

<a href="#"><b>CARGA_FICHERO</b></a>	
<a href="#"><b>CARGA_PANTALLA</b></a>	

[\*\*NADA\*\*](#)

## Method Summary

static <a href="#">SeleccionTB</a>	<a href="#"><b>valueOf</b></a> (java.lang.String name) Returns the enum constant of this type with the specified name.
static <a href="#">SeleccionTB</a> []	<a href="#"><b>values</b></a> () Returns an array containing the constants of this enum type, in the order they are declared.

### Methods inherited from class java.lang.Enum

clone, compareTo, equals, finalize, getDeclaringClass, hashCode, name, ordinal, toString, valueOf

### Methods inherited from class java.lang.Object

getClass, notify, notifyAll, wait, wait, wait

## Enum Constant Detail

**NADA**

public static final [SeleccionTB](#) **NADA**

---

**CARGA\_PANTALLA**

public static final [SeleccionTB](#) **CARGA\_PANTALLA**

---

**CARGA\_FICHERO**

public static final [SeleccionTB](#) **CARGA\_FICHERO**

## Method Detail

**values**

```
public static SeleccionTB[] values()
```

Returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants as follows:

```
for (SeleccionTB c : SeleccionTB.values())
    System.out.println(c);
```

**Returns:**

an array containing the constants of this enum type, in the order they are declared

---

**valueOf**

```
public static SeleccionTB valueOf(java.lang.String name)
```

Returns the enum constant of this type with the specified name. The string must match *exactly* an identifier used to declare an enum constant in this type. (Extraneous whitespace characters are not permitted.)

**Parameters:**

name - the name of the enum constant to be returned.

**Returns:**

the enum constant with the specified name

**Throws:**

java.lang.IllegalArgumentException - if this enum type has no constant with the specified name

java.lang.NullPointerException - if the argument is null

## 6. Conclusiones

### 6.1. Sobre los conocimientos adquiridos y requeridos

Al comenzar este curso, conocíamos unas pocas cosas sobre las FPGAs. Habíamos usado en cursos anteriores Spartan para simular circuitos secuenciales muy sencillos, para los cuales utilizábamos los botones, los switches y los leds para comunicarnos con ella. A medida que hemos ido profundizando en el desarrollo del proyecto hemos ido viendo todas las posibilidades que ofrecen este tipo de dispositivos.

El manejo de la entrada/salida para la comunicación con la FPGA nos ha permitido aplicar muchos de los conceptos aprendidos durante toda la carrera en relación con el hardware. En muchas ocasiones estos conceptos se quedan en la mera teoría, y sin embargo en esta ocasión hemos podido experimentar el verdadero uso y una aplicación concreta.

También hemos adquirido un profundo conocimiento de las aplicaciones de Xilinx. Anteriormente simplemente generábamos –por ejemplo- un bitfile sin saber lo que estaba haciendo internamente el programa. También hacíamos lo mismo a la hora de cargar un bitstream a la placa FPGA. Hemos tenido que aprender cómo funcionan todos estos procedimientos para poder llevar a cabo una aplicación más sencilla de ser utilizada.

Y no solamente hemos tenido que desarrollar conocimientos relacionados con el apartado hardware de la informática. Los conceptos adquiridos durante la carrera, relacionados con el software también nos han sido imprescindibles a la hora de desarrollar esta aplicación. Así por ejemplo hemos utilizado todos los conceptos relacionados con la programación orientada a objetos para toda la división de clases de la aplicación. También hemos tenido que utilizar técnicas de procesamiento de lenguajes para poder leer ficheros con un formato determinado (en este caso formato vhdl). Asimismo, también el uso de estructuras de datos para almacenar representaciones de información y evaluar, entre otras cosas, expresiones aritméticas.

## 6.2. Sobre la inserción de errores

Se ha llevado a cabo todo lo necesario para poder insterar errores en una FPGA. Para ello se carga un fichero en la FPGA, se ejecuta, se modifica un bit de su configuración y se compara. Esto ha de hacerse una gran cantidad de veces. Cada iteración no es baladí; tarda bastante tiempo ya que debe cargar 2 veces una frame y ejecutar. Además debe guardar los resultados en algún tipo de soporte (en este caso un fichero) para poder analizar los resultados posteriormente. Esto aumenta mucho el tiempo que requiere para ejecutar todo. Si tenemos en cuenta que esto se debe hacer  $36194 \text{ frames} \times 32 \text{ bits} = 1.158.208$  veces, veremos claramente que es imposible ejecutar esto en un tiempo razonable de tal forma que podamos obtener unos resultados adecuados.

## 7. Bibliografía

- [1] FPGA-Based System Design, Wayne Wolf
- [2]Gyovinet. <http://www.giovynet.com/serialport.html>
- [3] XST Guide User Guide, Xilinx
- [4] Digital integrated circuits. A design perspective, Jan M. Rabaey
- [5] Introducción a las FPGA, Alejandro Ehrenfeld G.

## **8. Agradecimientos**

Carlos Sánchez-Vellisco Sánchez: A mis padres y hermano. Por todo el apoyo, cariño y afecto recibidos.