



Universidad Autónoma de Querétaro

Facultad de Informática



# **INGENIERÍA DE SOFTWARE PARA DISPOSITIVOS DE HARDWARE RECONFIGURABLE**

## **TESIS**

Que como parte de los requisitos para obtener el grado de

## **MAESTRO EN INGENIERÍA DE SOFTWARE DISTRIBUIDO**

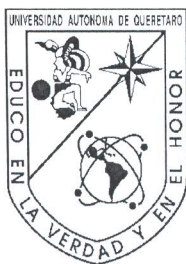
**Presenta**

Ernesto Arreola Olvera

**Director de Tesis**

Dr. Juan Manuel Ramos Arreguín

Querétaro, Qro. Septiembre de 2014



Universidad Autónoma de Querétaro  
Facultad de Informática  
Maestría

INGENIERIA DE SOFTWARE PARA DISPOSITIVOS DE  
HARDWARE RECONFIGURABLE

**TESIS**

Que como parte de los requisitos para obtener el grado de

Maestro en Ingeniería de Software Distribuido

**Presenta:**

Ernesto Arreola Olvera

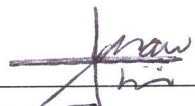
**Dirigido por:**

Dr. Juan Manuel Ramos Arreguín

SINODALES

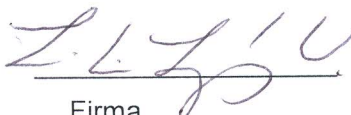
Dr. Juan Manuel Ramos Arreguín

Presidente

  
Firma


M.S.I. Lilia López Vallejo

Secretario

  
Firma

Dr. Jesús Carlos Pedraza Ortega

Vocal

  
Firma

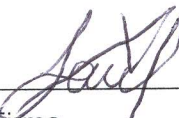
M. en C. Alberto Lamadrid Álvarez

Suplente

  
Firma

Dr. Saúl Tovar Arriaga

Suplente

  
Firma

Nombre y Firma  
Director de la Facultad

Nombre y Firma  
Director de Investigación y  
Posgrado

Centro Universitario  
Querétaro, Qro.  
Septiembre de 2014  
México

## RESUMEN

Esta investigación se enfoca en la importancia que tienen hoy en día ciertos dispositivos de hardware que proveen funcionalidad crítica a diversos procesos en nuestro mundo ordinario, ejemplos claros de ello son los motores de un avión, una máquina de diagnóstico ó tratamiento médico, dispositivos de vehículos que alcanzan altas velocidades y/o que ponen en riesgo la integridad humana en cualquier sentido. La investigación evalúa los aspectos críticos de estos dispositivos mediante el uso de las herramientas de software que los controlan, considerando que si cada uno de estos elementos de hardware se configura así mismo a través de una rutina ó condicional de software, el desempeño en conjunto de dichos componentes podrá ser no solamente controlado sino predecible y lo suficientemente robusto para considerar algún hecho extraordinario durante el uso de un sistema de estas características. El diseño experimental empleado fue a través del uso de herramientas de reconfiguración y simulación de componentes de hardware tales como VHDL, ISE Design Suite, Active HDL y FPGA por medio de algoritmos de configuración y análisis de circuitos digitales. Las variables consideradas en el desarrollo de las simulaciones para esta investigación son básicamente aquellas que activan y reconfiguran los dispositivos alternos involucrados en desarrollo de la simulación, así como todas aquellas variables que son utilizadas como los contenedores de la salida en el proceso, dichas salidas son la evidencia más directa de que las simulaciones fueron llevadas a buen término, así como los aspectos gráficos derivados de las salidas obtenidas en el proceso. Los procesos de simulación identifican distintos casos de prueba, casos de cobertura estructural en donde se busca encontrar de manera estresada toda la funcionalidad de los algoritmos involucrados, así como la cobertura funcional que implica el almacenamiento correcto de variables al final de la ejecución. La investigación generó una conciencia más profunda en relación con la importancia que implica el óptimo funcionamiento de software y el desempeño que debe cubrir al interactuar con componentes de hardware de funcionalidad crítica en cualquier tipo de aplicaciones.

**(Palabras clave:** Configware, Software, Reconfigurable)

## SUMMARY

This investigation focuses on the importance of today certain hardware devices that provide critical functionality to various processes in our ordinary world, clear examples of this are the engines of an aircraft, machinery or medical diagnostic devices, vehicles they reach high speeds and / or endanger human integrity in any way. The study evaluates the critical aspects of these devices by using software tools that control, considering that if each of these elements of hardware is configured through a software routine or conditional, the performance of these components may be controlled and predictable, and robust to consider some extraordinary event while using a system like this. The experimental design used was through the use of simulation tools and reconfiguration of hardware components such as VHDL, ISE Design Suite, Active HDL y FPGA and algorithms through setting and analysis of digital circuits. The variables considered in the development of simulations for this research are basically those that activate and reconfigure the alternate devices involved in development of the simulation as well as all those variables that are used as containers in the process output, these outputs are the most direct evidence that the simulations were carried to term, and the graphical aspects derived from the outputs obtained in the process. The simulation processes identify different test cases, structural coverage cases where attempts to find so stressed all the functionality of the algorithms involved, and functional coverage involves proper storage of variables at the end of execution. The investigation led to a deeper awareness regarding the importance involving the optimum operation and performance software to be covered by interacting with hardware components of critical functionality in any application.

**(Key words:** Configware, Software, Reconfigurable)

**A los alumnos de los posgrados de la  
Universidad Autónoma de Querétaro**

## **AGRADECIMIENTOS**

En la preparación de este material se recogieron las opiniones desinteresadas de los Directores y Coordinadores de Investigación y Posgrado de todas las Facultades de la Universidad Autónoma de Querétaro, así como de investigadores, académicos y personal administrativo de la misma.

En particular, la Dirección de Servicios Escolares y la Dirección de Investigación y Posgrado, agradecen al Dr. Juan Manuel Ramos Arreguín el haber revisado el texto y por sus atinados comentarios para mejorarlo.

# INDICE

I. INTRODUCCION.....	1
II. REVISION DE LITERATURA.....	5
III. METODOLOGIA.....	10
<b>Desarrollo de la Investigación</b> .....	11
<b>Herramientas</b> .....	12
<b>Introducción a VHDL</b> .....	19
<b>Simulación y Experimentación</b> .....	19
<b>Simulación de VHDL</b> .....	24
<b>Ciclo de simulación de VHDL</b> .....	24
<b>Simulación paralela de VHDL</b> .....	26
<b>Simulación de eventos discretos</b> .....	28
<b>Niveles de paralelismo/distribución</b> .....	29
<b>Simulación de procesos lógicos</b> .....	30
<b>Creación de diseño y simulación de un FPGA</b> .....	33
<b>Reconfiguración de dispositivos de hardware mediante VHDL</b> .....	51
IV. RESULTADOS Y DISCUSION .....	68

## INDICE DE FIGURAS

Figura	Página
1. Computadora de propósito general de alta velocidad.....	2
1.1 Ciclo de Vida de Software y su relación con la Configuración / Re- configuración de Hardware en la fase de implementación. ....	9
2. Diagrama de bloque de una implementación de “configware”.....	11
3. Niveles de acoplamiento de un sistema reconfigurable.....	14
4. Componentes del Compilador MATCH .....	15
5. Árbol de Sintaxis Abstracta.....	16
6. Acelerador reconfigurable como co-procesador .....	17
7. Ejecución de hilos en el tiempo sobre Software (sombreado) y sobre Hardware (en blanco).....	18
8. Descripción estructural de un sistema .....	23
9. Portada de la versión 9.1 de Active-HDL .....	34
10. Iniciando una nueva área de trabajo en Active-DHL.....	34
11. Asignación de un identificador a nueva área de trabajo.....	35
12. Creación de un diseño vacío en el área de trabajo .....	35
13. Selección del lenguaje de desarrollo para la prueba .....	36
14. Asignación de nombre para el diseño .....	37
15. Creación de un nuevo archivo de codificación.....	37
16. Generación de código VHDL en Active-HDL.....	39
17. Compilación de código VHDL en Active-HDL .....	40
18. Barra de resultados en la compilación.....	40
19. Inicialización de la simulación .....	41
20. Reporte de la inicialización de la simulación .....	41
21. Elementos de la entidad a ser simulada .....	42
22. Ejecución de simulación sin previa activación de un ambiente de prueba..	42
23. La Simulación fue satisfactoria aún a pesar de no tener activado un ambiente de prueba.....	43
24. Generación de una Waveform para la representación de señales.....	44
25. Vista de Waveform disponible para configurar el ambiente de prueba.....	45
26. Las señales son agregadas al ambiente de prueba.....	46
27. Inicialización de señales de entrada .....	47
28. Inicialización de señales con valores lógicos “0” y “1” .....	48
29. Barra de Simulación en una Waveform.....	49
30. Ejecución de la simulación en 1300 ns .....	49
31. Representación gráfica de las señales en Waveform.....	50
32. Cambio del valor de una de las señales de entrada para probar la lógica implementada .....	50
33. Gráfica del cambio de una de las señales de entrada que también afecta a la salida .....	51
34. Cabina de un avión .....	54
35. Declaración de entradas y salidas por cada elemento a simular .....	58
36. Definición de la arquitectura a ser ejecutada .....	59
37. Programación secuencial utilizada en VHDL .....	60
38. Proceso de compilación de código VHDL.....	61
39. Inicialización de la Simulación .....	62
40. Generación de una vista de Waveform para poder mostrar evidencia del proceso de Simulación.....	63



41. Señales de entrada y salida definidas en la entidad de la lógica.....	64
42. Inicialización de señales mediante la opción “Stimulators” .....	64
43. Ejecución de la funcionalidad de vuelo con una Simulación ejecutada a 3900 ns.....	65
44. Inserción de una falla en el motor 1 y la activación en tiempo real del motor 2.....	66
45. Inserción de una falla en uno de los controles principales.....	67

## I. INTRODUCCION

En 1959 Gerald Estrin un científico de la Universidad de California en los Ángeles, introduce el concepto de computación reconfigurable. El siguiente fragmento de una publicación en 1960 *The fix-plus machine* define el concepto del paradigma de computación reconfigurable:

*“Estudios pragmáticos sobre el problema predicen aumentos en la velocidad de cálculo en una variedad de tareas de cálculo cuando se ejecuta en configuraciones adecuadas orientadas hacia los problemas de equipo de estructura variable. La viabilidad económica del sistema está basada en la utilización esencialmente del mismo hardware en una variedad de propósitos especiales. Esta capacidad se consigue mediante la reestructuración programada ó física de una parte del hardware”* (Bobda, 2007).

Para implementar esta visión, Estrin diseñó un sistema de cómputo, The fix-plus machine. Igual que este, hoy existen muchos sistemas computacionales reconfigurables, The fix-plus machine consiste en tres elementos principales, tal y como se muestran en la figura 1:

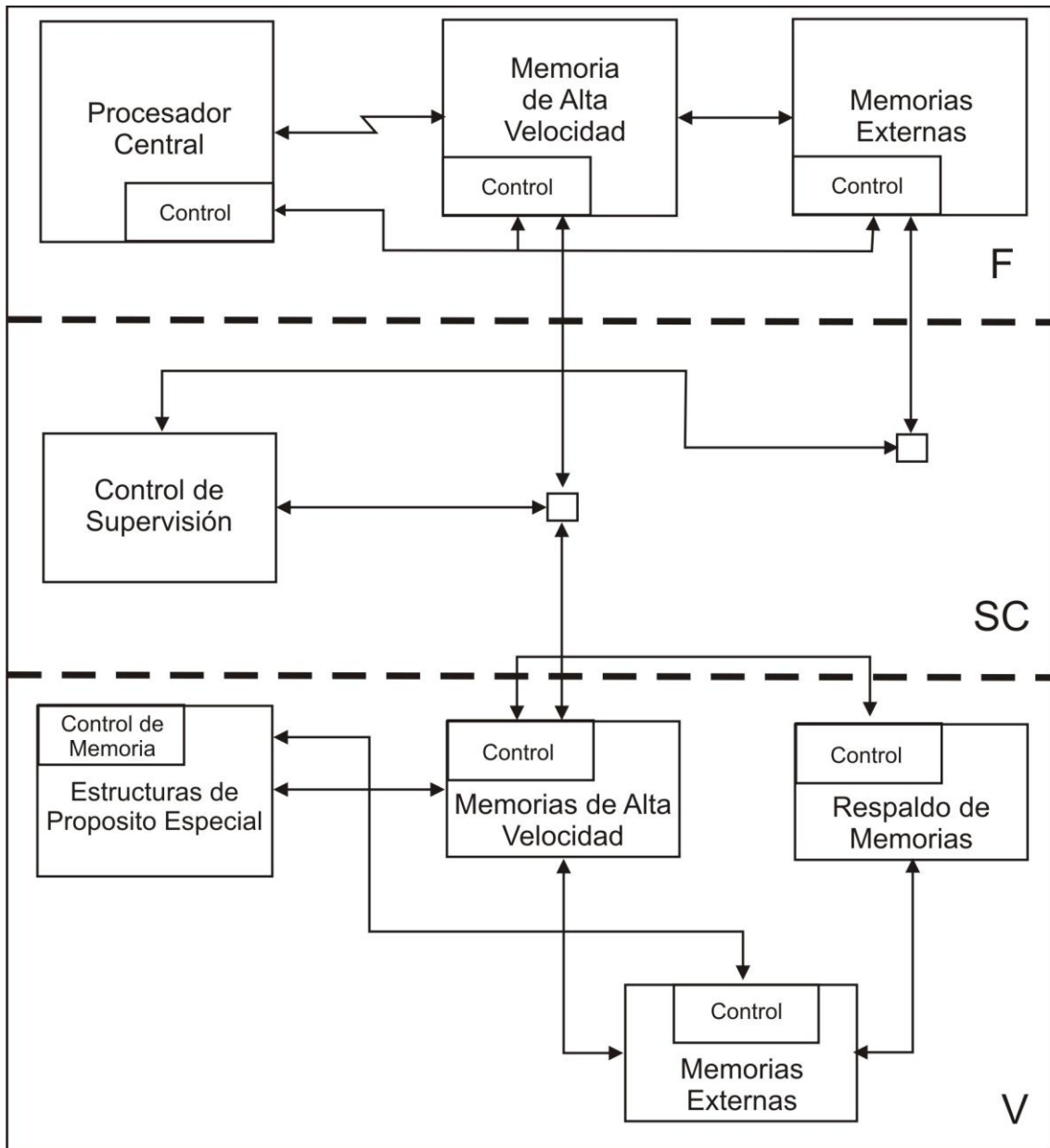


Figura 1. Computadora de propósito general de alta velocidad

1. Una computadora de propósito general de alta velocidad, llamada The fixed part (representada por la sección F de la figura 1) puede ser implementada en algún procesador de propósito general. En donde el procesador central, la memoria de alta velocidad y las memorias externas constituyen la parte de control del procesamiento y ejecución de las tareas en la computadora así como la potencialización de

recursos y soporte desde el punto de vista físico y de infraestructura para la computadora de propósito general.

2. El control de supervisión SC (representada en la sección SC de la figura 1) coordina las operaciones entre el módulo fix y la variable módulo. Es la sección encargada de asignar prioridades de procesamiento y administración de recursos, a través de ella se determina el método más conveniente de procesamiento y ejecución, identificadas a través de las solicitudes de memoria de alta velocidad y memorias externas que validarán su capacidad de soporte de acuerdo a la prioridad y demanda de ejecución de cada tarea,
3. Una variable parte V (representada en la sección V de la figura 1) consiste en varios tamaños de subestructuras de alta velocidad que pueden ser reorganizadas en configuraciones de propósito especial orientadas a problemas. La variable de la parte (V) es creada sobre un problema específico optimizado por unidades funcionales en una configuración básica (funciones trigonométricas, logarítmicas, exponenciales, raíces, aritmética compleja, hiperbólicas, operaciones de matrices).

Existen algunos métodos para la ejecución de algoritmos en la computación tradicional, uno de ellos es utilizar una aplicación específica del circuito integrado, o ASIC (Application Specific Integrated Circuits), para realizar las operaciones en el hardware. Debido a que estos ASICs están diseñados específicamente para realizar un cálculo determinado, que son muy rápidos y eficientes cuando ejecutan el cálculo exacto para el que fueron diseñados. Sin embargo, después de la fabricación del circuito no puede ser alterado. Los microprocesadores son una solución mucho más flexible, procesadores que ejecutan un conjunto de instrucciones para realizar un cálculo. Al cambiar las instrucciones del software, la funcionalidad del sistema se ve alterada sin cambiar el hardware. Sin embargo, el lado negativo de esta flexibilidad es que el rendimiento se resiente, y es muy inferior a la de un ASIC.

El procesador debe leer cada instrucción de memoria, determinar su significado, y sólo entonces ejecutarlo. El cómputo reconfigurable se destina a cubrir el vacío entre el hardware y software, consiguen un rendimiento potencial mucho más alto que el software, manteniendo al mismo tiempo un mayor nivel de flexibilidad que el hardware (Compton, 2007).

Los sistemas de hardware reconfigurable están asociados con la aparición de dispositivos lógicos programables (PLDs) en la década de 1990, rompiendo el punto de equilibrio entre flexibilidad y rendimiento. Pueden lograr un alto rendimiento con bajo nivel de ejecución y costo. El Hardware reconfigurable es programable por la reconfiguración de su estructura - un punto medio entre los enfoques de hardware y software. Un algoritmo estructuralmente programado en hardware reconfigurable es también conocido como configware (Becker, 2000). La computación reconfigurable alía el rendimiento de hardware basado en soluciones y flexibilidad de software, permitiendo la exploración del paralelismo inherente de algunas tareas computacionales. (Lopes, 2007).

La síntesis de circuitos lógicos en PLDs se realiza usando diseño de sistemas asistido por computadora, permitiendo el uso simultáneo de diferentes proyectos de interfaces. Ejemplos de lenguajes usados son interfaces gráficas (con esquemas), VHSIC lenguaje de descripción de hardware (VHDL), y el lenguaje de descripción de hardware Altera (AHDL) (Lopes, 2007). VHDL se ha convertido en el lenguaje estándar para la descripción de hardware. Actualmente hay varias plataformas comerciales de diseño asistido por ordenador para el diseño de sistemas de hardware reconfigurable. En general, proporcionan un ambiente de desarrollo integrado, permitiendo proyectos, simulación, prueba y documentación de circuitos digitales.

## II. REVISION DE LITERATURA

El objetivo del concepto de hardware reconfigurable es permitir una fácil y rápida adaptación de un proyecto para la evolución tecnológica continua, con el objetivo de mejorar la portabilidad y la capacidad de intercambio del sistema final. Por medio de dividir la estructura en pequeños bloques funcionales, con interfaces dedicadas muy específicas, la modularización del proyecto se hace eficaz. Como una consecuencia, el manejo e integración de un equipo de diseño multidisciplinario es facilitado, así como la adaptación de un bloque determinado a mantener el ritmo de la evolución de la tecnología.

Avances recientes en técnicas computacionales demuestran que los sistemas basados en hardware reconfigurable (o bien, simplemente configware) pueden ofrecer máquinas computacionales a medida para aplicaciones específicas, con órdenes de magnitud más rápidas que el software que se ejecuta en los procesadores regulares de uso común general.

Recientemente los lenguajes de alto nivel como MATLAB se han convertido populares en prototipos de algoritmos en ámbitos tales como procesamiento de señales e imagen. Muchas de estas aplicaciones cuyas subtareas tienen diversos requerimientos de ejecución a menudo emplean sistemas reconfigurables, distribuidos y heterogéneos. Estos sistemas consisten en un conjunto interconectado de recursos heterogéneos de procesamiento que proporcionan una variedad de capacidades de arquitectura (Banerjee, 2000). El objetivo de la MATCH (MATlab Compiler for Heterogeneous computing systems) un proyecto de compilador (Banerjee,1999) en la Universidad de Northwestern en Estados Unidos, facilita a los usuarios a desarrollar un código eficiente para sistemas distribuidos, heterogéneos y sistemas de computación reconfigurable. Con este fin estamos aplicando y evaluando el prototipo experimental de un sistema de software que tomará descripciones MATLAB de varias aplicaciones, y automáticamente las “mapeará” en un ambiente de computación distribuída que consiste en procesadores embebidos,

procesadores de señales digitales y puertas de arreglos autoprogramables construídas a partir de componentes comerciales.

Tradicionalmente, cuando se habla de arquitecturas paralelas reconfigurables, se hace referencia a aquellos sistemas donde la red de interconexión no es fija, sino que es definida en función del programa que se va a ejecutar. Es decir, la reconfigurabilidad de ambientes de procesamiento paralelo se refiere básicamente a la posibilidad de definir la topología de interconexión de los procesadores para cada aplicación. Dicha definición se refiere al número de procesadores y los enlaces entre ellos: de esta manera quedará identificada una topología particular.

Es posible que la definición hecha pueda realizarse de forma real, es decir, los procesadores se interconectan físicamente como lo indica la topología especificada; en este caso se estaría hablando de reconfiguración por hardware o reconfiguración por hardware programable (switching). Alternativamente, la definición topológica puede hacerse completamente por software, implementándose a través de una topología virtual. En este caso, la creación y la manipulación de dicha topología se hacen por software.

Bajo el esquema de topología virtual, la interconexión real de los procesadores sólo es conocida por el Sistema Manejador del Ambiente (SMA). El SMA debe determinar la topología sobre la cual se debe ejecutar una aplicación dada, para generarla virtualmente. Además, realizará todas las tareas de gestión sobre esa topología, de manera de optimizar el tiempo de ejecución de la aplicación.

Si la reconfiguración se realiza antes de iniciarse la ejecución del programa, se está hablando de una reconfiguración estática, de lo contrario, si ésta es modificable en el curso de la ejecución de un programa y/o de la operación del sistema, se habla de una reconfiguración dinámica.

Un nuevo modelo computacional, llamado arreglo lineal con un sistema reconfigurable de bus segmentado (LARPBS), ha sido propuesto como un

modelo computacional paralelo viable y eficaz basado en tecnología óptica actual, las operaciones de movimiento de datos básicos en el modelo incluyen la difusión, multidifusión, compresión, división, suma de prefijo binario y máximo hallazgo. El modelo LARPBS puede utilizarse como un modelo computacional paralelo nuevo y práctico para el diseño de algoritmos paralelos.

En arreglos de procesador con sistemas de bus reconfigurable, se proponen dos algoritmos de tiempo  $O(1)$  para calcular el cierre transitivo de un grafo. Uno está diseñado en una matriz de procesador  $n \times n \times n$  tridimensional con un sistema reconfigurable bus, y el otro está diseñado en una matriz de dos dimensiones  $n^2 \times n^2$  procesador con un sistema reconfigurable bus, donde  $n$  es el número de vértices en el gráfico. Utiliza los algoritmos de cierre transitivo de tiempo  $O(1)$ , muchos otros problemas de gráfico se resuelven en tiempo  $O(1)$ . Estos problemas incluyen reconocimiento de bipartitos gráficos y componentes conectados, puntos de articulación, componentes de doble conexión, puentes y árboles en grafos de expansión mínimos.

Actualmente el tema tiene distintas áreas de aplicación y derivado de ello podemos considerar su importancia, como es el caso de su uso en prototipos e implementación de sistemas neuromórficos (en donde la facultad de reconfiguración es especialmente relevante en este caso dado que cada paciente puede requerir distintas especificaciones de codificación y proyección de la información visual sobre los microelectródos de estimulación) (Ros, 2002), con esto se pueden definir arquitecturas de gran complejidad e implementar sistemas que requieran mecanismos de adaptación, plasticidad y aprendizaje a medio-largo plazo, así mismo, la aplicación del hardware reconfigurable al campo de la visión artificial está cada vez más extendida, para situaciones en las que el procesamiento en tiempo real es necesario, sin mencionar claro sistemas expertos más complejos como es el caso de investigación espacial o aeronáutica en donde el factor de reconfiguración de determinado hardware juega un papel fundamental.

Algunas de las ventajas de la utilización de hardware reconfigurable son (Ito, 2000):



1. Paralelismo Real, sin seguir el modelo de Von Newmann.
2. Desarrollo modular y jerárquico de un proyecto.
3. Proyecto de reducción de ciclo de tiempo, permitiendo metodologías de proyectos de arriba abajo y de abajo hacia arriba.
4. Disponibilidad de desarrollo de muchas interfaces y ambientes.
5. Disponibilidad de productos listos para usar las funciones de prueba (IP-core), reduciendo el ciclo del proyecto para funciones de alta complejidad.

Una motivación adicional para el uso de hardware reconfigurable en la aplicación de algoritmos es la amplia disponibilidad de dispositivos de alto rendimiento en el mercado. Por ejemplo, algunas de los más recientes FPGAs presentan características tales como 700 I/O pins, de impedancia controlada y líneas dedicadas para la operación en modo diferencial, especial en módulos internos multiplicadores, la modulación por ancho de pulso (PWM), registros dedicados a operaciones de alto rendimiento y memoria RAM de alta velocidad y capacidad de hasta 2 Mbits. Actualmente, los dispositivos comerciales con un máximo de cinco millones de células lógicas pueden ser encontrados, permitiendo la implementación de uno o más procesadores completos con memoria y periféricos en un solo chip.

## **APLICACIONES**

Las aplicaciones para este tipo de dispositivos principalmente se encuentran en modelos de funcionalidad de crítica tales como equipo de diagnóstico o tratamiento médico, motores de avión, dispositivos reguladores de velocidad en vehículos, etc. Como se muestra en la figura 1.1 podemos observar que la ingeniería de software comprende un ciclo completo de fases (ciclo de vida) para el desarrollo e implementación de alguna o muchas funcionalidades dictaminadas por requerimientos de alto nivel (necesidad de un cliente para realizar una tarea), luego serán convertidos en diseño, en código y dicha implementación tendrá control sobre los dispositivos de hardware que estén asociados a la tarea inicial que se concibió completar. El diseño de un

requerimiento y más tarde su implementación por medio de ingeniería de software tendrá que ser coherente y empatar en todos los aspectos de funcionalidad con el o los dispositivos de hardware que estén involucrados, miles de líneas de código son las que controlan la funcionalidad física de dispositivos tan grandes como motores de avión, controles de aeronavegabilidad, máquinas de diagnóstico y miles de aplicaciones de carácter crítico en nuestra vida cotidiana, a ello, se atribuye la importancia de esta relación conjunta y estrecha entre la ingeniería de software y la configuración de hardware.

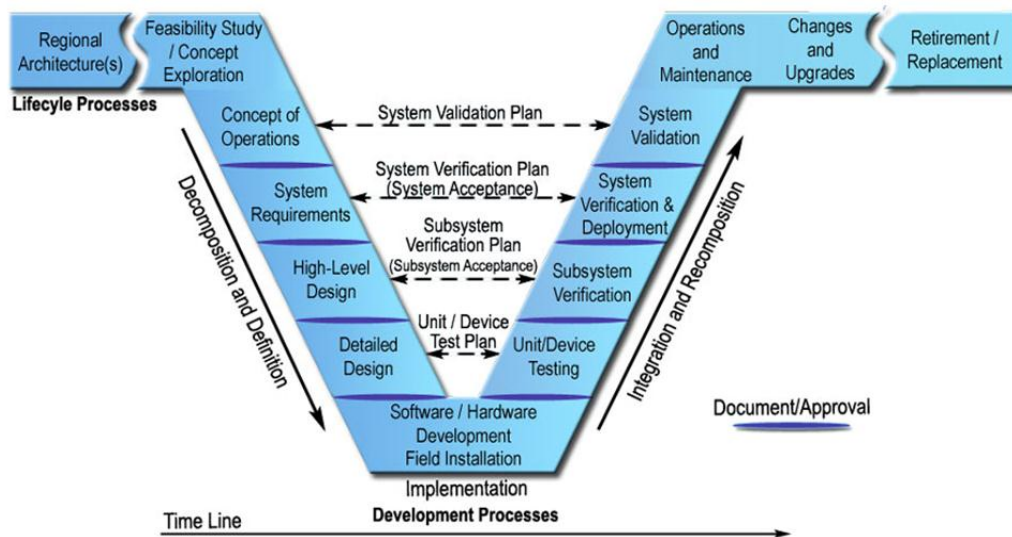


Figura 1.1 Ciclo de Vida de Software y su relación con la Configuración / Re-configuración de Hardware en la fase de implementación.

### **III. METODOLOGIA**

#### **Objetivo General**

Modelación y simulación de la reconfiguración de un dispositivo de hardware a través de algoritmos y eventualmente código fuente.

#### **Objetivo Específico**

Generar un caso de prueba de simulación donde se involucre hardware reconfigurable, y comprobar la correcta funcionalidad de estas técnicas.

Documentar el uso de hardware reconfigurable, de tal suerte que se puedan reutilizar los patrones de algoritmia e investigación para proyectos futuros.

#### **Hipótesis**

Existe una gran diversidad de dispositivos de Hardware que soportan funcionalidad crítica en muchos campos de la ciencia, dichos dispositivos presentan riesgos comunes de falla tanto en el aspecto físico como en el aspecto lógico o de control, en este sentido, es necesario encontrar alguna metodología adecuada para asegurar que la funcionalidad crítica emanada de estos dispositivos pueda ser mantenida en condiciones extraordinarias. La hipótesis tiene como base el hecho de que la reconfiguración automática de dispositivos de hardware ofrece la opción de cambiar la configuración inmediata y de manera automática a dispositivos de respaldo cuando algún daño es presentado en el control inicial, con miras a la preservación de funcionalidad en dispositivos de ejecución crítica. Sin embargo, los cambios también pueden ser desarrollados de manera manual, a través de la ejecución de comandos.

## Desarrollo de la Investigación

La Figura 2 representa un ejemplo de un diagrama de bloque de una implementación de “configware” (como en la simulación), y en general, es una manera de aprovechar las características de reconfiguración del hardware.

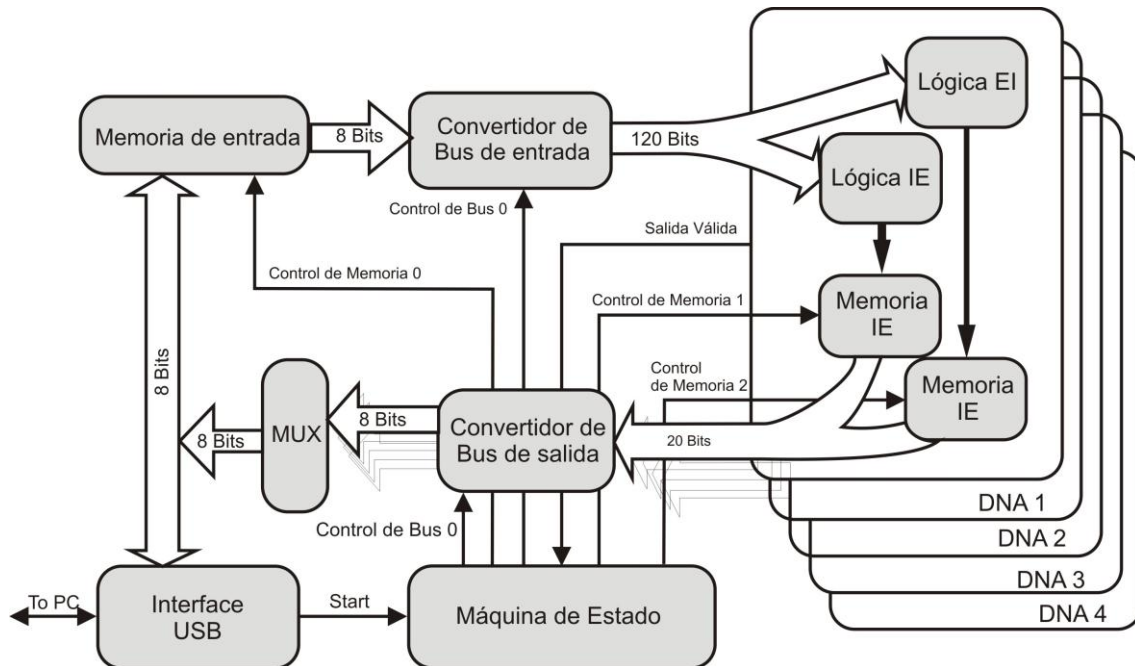


Figura 2. Diagrama de bloque de una implementación de “configware”

En la figura 2 se pueden destacar 2 estructuras básicas en la arquitectura: el controlador, implementado con una máquina de estados asíncrona y la ruta de datos, responsable de las entradas; y el procesamiento (bloques DNA) y las salidas de los datos. Las estructuras de salida de datos pueden ser:

1. USB Interface: USB (externa al FPGA) fue implementada para recibir y enviar datos a la PC.
2. Input Memory: Es usada para almacenamiento temporal de lo que se recibe a través de la PC.
3. Input bus converter: Este bloque agrupa 15 bytes sucesivos de la entrada de memoria para formar un bus de 120 bits para el bloque DNA.

4. DNA: Es el principal bloque del sistema donde las expresiones lógicas obtenidas en pasos anteriores fueron implementadas como circuitos lógicos. Hay cinco pares de DNAs, identificadas como DNA0, DNA1, ..., DNA4, representando cinco clasificaciones para EI y IE de acuerdo a sus procesos de validación.

Utilizando este tipo de esquemas, al detectar que un módulo ha dejado de funcionar, se da la orden de inhabilitar esa sección y redireccionar los datos hacia otro módulo funcional. Por ejemplo, si falla el módulo DNA1, entonces redireccionar los datos hacia el módulo DNA2.

La manera de realizar los cambios se desarrolla detectando fallas de comunicación entre módulos por una parte, y por otra, el usuario podrá llevar a cabo dichos cambios de manera manual.

## **Herramientas**

Utilizar herramientas como VHDL, Active HDL, ISE Design Suite, FPGAs y el método científico para sustentar de manera formal y objetiva cada una de las etapas de la investigación. Con bases firmes en la observación, la generación de hipótesis y la debida tela de juicio de cada una de ellas a base de probar con un método experimental todas las teorías que se generaron por medio de la observación.

Comprobar en cada paso de la investigación aquellas teorías que deriven de la observación y utilizar la simulación como herramienta indispensable para la comprobación de dichas conjeturas, tomar notas y realizar una bitácora que resuma teorías contra simulaciones prácticas que confirmen una correspondencia activa y bien relacionada de la especulación contra la comprobación.

1. Uso de VHDL para la experimentación, en donde este lenguaje será capaz de crear una reconfiguración por medio de la interacción y análisis de circuitos digitales, de tal suerte que los dispositivos involucrados puedan ser reconfigurados de manera automática cuando sea requerido.
2. Prueba de simulador.
3. Uso de Hardware reconfigurable a través de VHDL, Active HDL, ISE Design Suite y FPGA.

La Figura 3 representa los diferentes niveles de acoplamiento que existen en un sistema reconfigurable:

1. En primer lugar, el hardware reconfigurable se puede utilizar solamente para reconfigurar unidades funcionales dentro de un host procesador.
2. En segundo lugar, una unidad reconfigurable se puede utilizar como un coprocesador.
3. En tercer lugar, una unidad de procesamiento reconfigurable adjunta se comporta como si se tratara de un procesador adicional en un multiprocesador del sistema.
4. Por último, la forma más imprecisa de hardware reconfigurable es la de una unidad de procesamiento externa independiente.

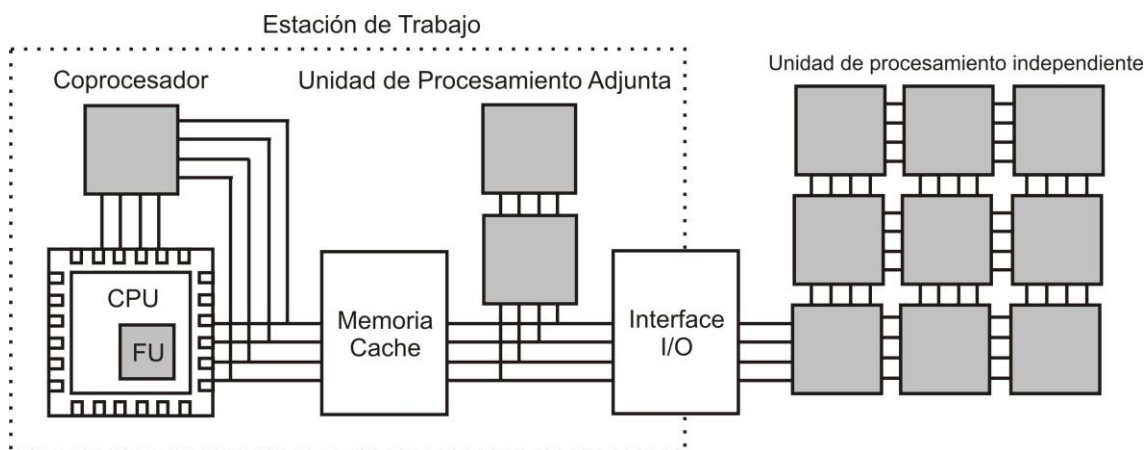


Figura 3. Niveles de acoplamiento de un sistema reconfigurable

MATLAB es básicamente una función orientada al lenguaje común, muchos de los programas de MATLAB pueden ser escritos usando funciones predefinidas. Estas funciones pueden ser funciones primitivas o aplicaciones específicas. En un programa secuencial de MATLAB estas funciones son normalmente implementadas como código secuencial corriendo en un procesador convencional. Sin embargo en el compilador MATCH, estas funciones necesitan ser implementadas en diferentes tipos de recursos (convencional y otros), y también algunos de estos necesitan ser implementaciones paralelas para tomar la mejor ventaja del paralelismo soportado por el registro de una máquina. MATLAB también soporta lenguajes constructores usando un programador que puede escribir convenciones de procedimiento, programas de estilo (o partes de él), el uso de loops, notación vectorial y similares. Nuestro compilador armoniza las necesidades y de forma automática traduce todas las partes del programa en caso secuencial o en código paralelo.

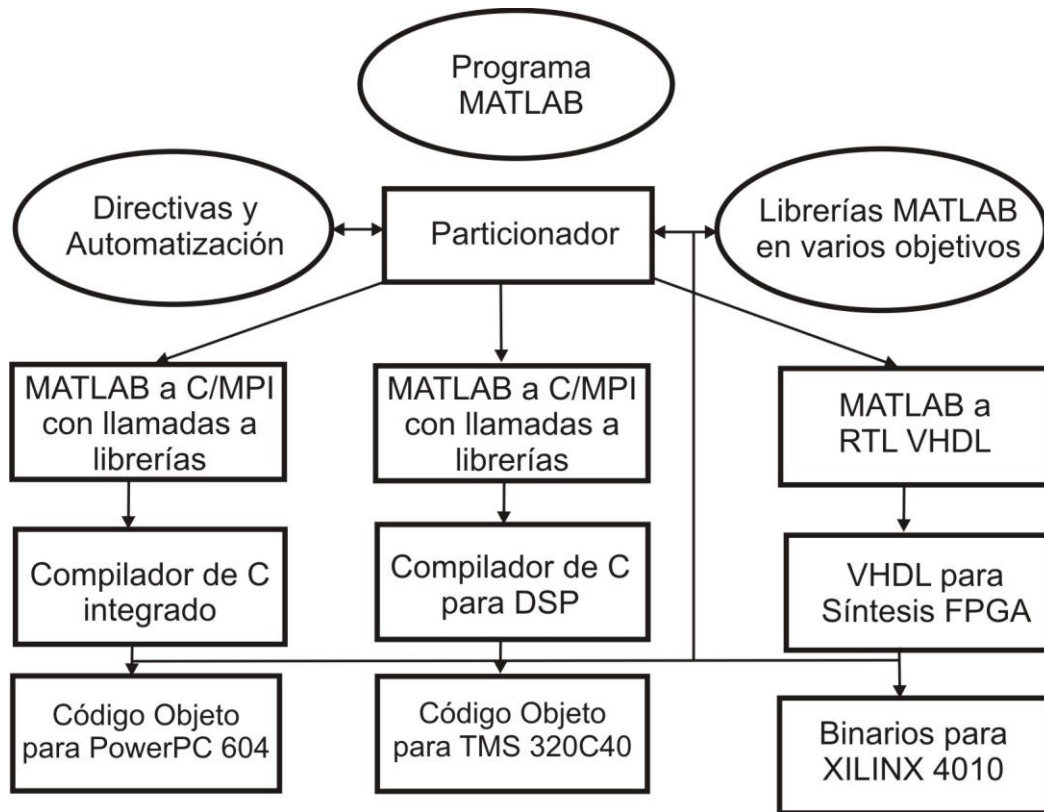


Figura 4. Componentes del Compilador MATCH

Debemos de generar los archivos AST para cada una de las partes del programa original de MATLAB, dichos archivos deben estar debidamente traducidos para procesadores correspondientes. Los archivos AST correspondientes a Host / DSP / procesadores embebidos se convierten en equivalentes de programas en lenguaje C. En los nodos correspondientes a los operadores y llamadas a funciones, la información comentada sobre los operandos y el operador o función se comprueban. Dependiendo de la información anotada, se añade una llamada a una función adecuada de C que lleva a cabo la tarea del operador/llamada a la función (de uso predefinido) en MATLAB. Por ejemplo para invocar la función FFT en el grupo de los procesadores DSP, el compilador genera una llamada para que encaje con la función contenedora (DSP...), en lugar del conjunto de llamadas reales necesarias para invocar el encaje en los procesadores DSP clúster. Esta envoltura contiene el mecanismo para invocar la función de los recursos respectivos. Por último, estos generan los programas y son compilados con el objetivo de generar el ejecutable / bit de configuración.



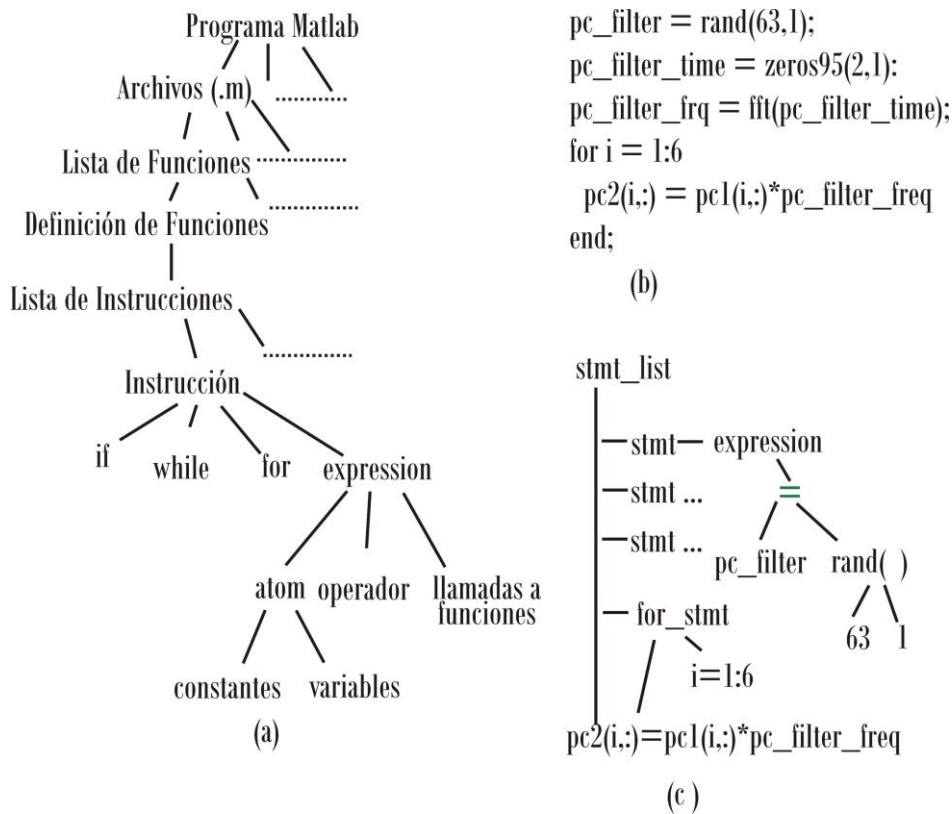
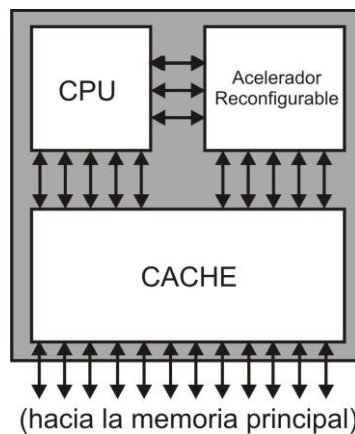


Figura 5. Árbol de Sintaxis Abstracta

El hardware reconfigurable está disponible para acelerar una variedad de aplicaciones tales como la secuenciación DNA, MPEG y procesamiento de datos por medio de satélite (Compton, 2002). El hardware es inherentemente más paralelo que un microprocesador y evita el esfuerzo de leer y decodificar instrucciones. La reconfigurabilidad permite al hardware acelerar diferentes aplicaciones y secciones de una aplicación única en diferentes ocasiones. Una variedad de sistemas con hardware reconfigurable han sido propuestos y construidos, incluidos Splash 2 (Buell, 1996), PAM (Bertin, 1993), RaPiD (Ebeling, 1996) y PipeRench (Goldstein, 2000), frecuentemente este hardware reconfigurable es usado como un coprocesador para propósito general (Hauser, 1997), (Miyamori, 1998), (Moshovos, 2000), (Wittig, 1996), como se muestra en la Figura 6 el control intensivo de partes de una aplicación típicamente ejecutada en software, mientras el cálculo intensivo de secciones son implementadas en hardware reconfigurable. Podemos referirnos a estas secciones como aplicación “kernel”, que ejecutan loops muy pesados y que pueden ser divididos o completamente calculados en paralelo. Una aplicación

puede tener muchos “kernels” y el hardware puede ser reconfigurado durante la ejecución para implementar cada uno de estos “kernels” como se necesite. Esta técnica es conocida como computación reconfigurable (RC). Sin embargo la mayor parte del trabajo se centra en la computación reconfigurable en el diseño de la propia lógica reconfigurable o su conexión a un procesador anfitrión. Hay una variedad de problemas por resolver antes de que el hardware reconfigurable pueda ser considerado dentro de las principales opciones como acelerados de uso general entre los dispositivos convencionales.



*Figura 6. Acelerador reconfigurable como co-procesador*

Los desarrolladores de aplicaciones deben ser capaces de llegar fácilmente al hardware reconfigurable. Varios grupos están investigando compiladores que puedan detectar automáticamente las secciones de una aplicación que debe ser acelerada en hardware (Allen, 2000), (Callahan, 2000) (Darnell, 2000). El objetivo es crear compiladores que tengan un programa escrito en un lenguaje de alto nivel como C++ o Java, y la salida de ambos un ejecutable binario y uno o más archivos de flujo de bits de configuración para el hardware reconfigurable específico. Actualmente esto depende de saber exactamente lo que el hardware está buscando, y en muchos casos la configuración no es compatible. Esto significa que para cualquier actualización del hardware reconfigurable la aplicación debe volverse a compilar. Debido a esto el soporte a este tipo de hardware se vuelve menos atractivo. Todas las investigaciones previas han asumido un solo hilo de ejecución, cuando la aplicación dada tiene toda la propiedad del

microprocesador predefinido y la lógica reconfigurable, esto puede ser una suposición válida para algunas aplicaciones y entornos específicos, esta no es válida para la computación de propósito más general e incluso algunos sistemas embebidos como PDAs y teléfonos celulares. El impulso hacia multiprocesadores chip (CMP) y multiprocesos simultáneos (SMT) agravan este problema. Varios subprocesos de varias aplicaciones (o incluso de una sola) pueden requerir simultáneamente la utilización de hardware reconfigurable.

La clave de usar Computación Reconfigurable en un entorno multi-threaded es la manera en como se utilizan los recursos de Hardware debido a que las aplicaciones están distribuidas con el software y el hardware de los kernels, con ello podemos decidir en tiempo de ejecución que hilo debe de utilizar el hardware y cual deberá utilizar el software. La Figura 7 muestra un ejemplo de tres hilos diferentes T0, T1 y T2 ejecutando un sistema. Las áreas sombreadas muestran uso de hardware, mientras que las secciones blancas muestran la ejecución de software. Varios subprocesos utilizan el CPU y el hardware reconfigurable al mismo tiempo. Idealmente todos los hilos podrían utilizar hardware reconfigurable donde sea que lo necesiten, sin embargo estos no siempre es posible debido a limitaciones en los recursos, el sistema no debe de confiar en los hilos para calendarizar su propio uso, invariablemente al menos uno de los hilos no asignará dispositivos, el sistema operativo es un selector natural como un árbitro independiente de reconfiguración de recursos (Brebner, 1996).

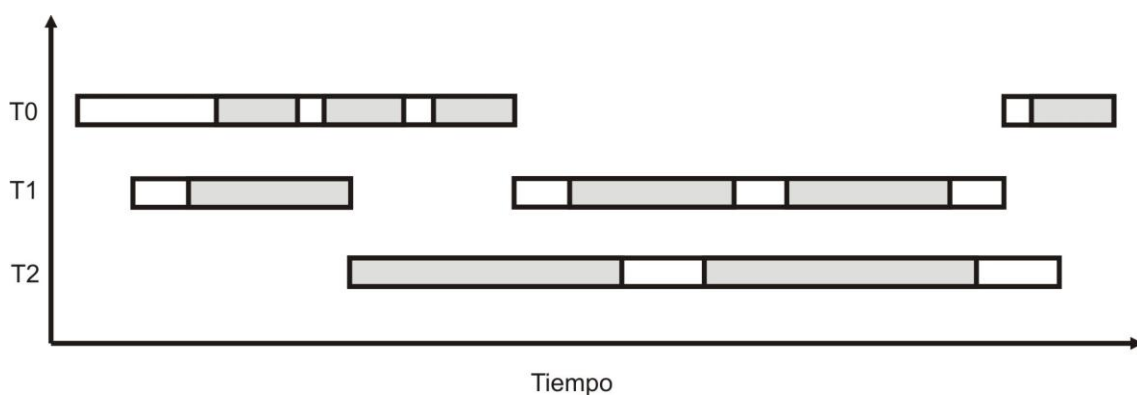


Figura 7. Ejecución de hilos en el tiempo sobre Software (sombreado) y sobre Hardware (en blanco)

## **Introducción a VHDL**

Entre los circuitos integrados que se utilizan en el diseño digital se encuentran según su escala de integración o número de compuertas, los SSI, MSI, LSI, VLSI, ULSI, GSI., algunos de ellos permiten ser programados por el usuario, entre estos están los PLD, los mismos se utilizan en muchas aplicaciones para reemplazar a los circuitos SSI y MSI.

Entre los dispositivos que se clasifican como PLD están: los PAL, PLA, GAL, CPLD y FPGA. Estos dispositivos para su programación requieren un software, un hardware representado en un prototipo, el cual se programa luego de realizar la síntesis del circuito usando lenguaje de descripción de hardware (HDL). Entre los lenguajes de descripción de hardware se encuentran Verilog y VHDL. VHDL es un acrónimo que resulta de la combinación de VHSIC y HDL, que significan Very High Speed Integrated Circuit (Circuito Integrado de muy Alta Velocidad) y Hardware Description Language (Lenguaje de Descripción de Hardware), respectivamente.

Se trata de un lenguaje diseñado a iniciativa del departamento de defensa de Estados Unidos, basado en el lenguaje de programación Ada. En 1986 el departamento de defensa de E.U. transfirió los derechos a IEEE, con la intención de que fuera más ampliamente aceptado por la industria, desde 1987, IEEE se ha encargado de la publicación y actualización del estándar VHDL (Active-HDL, 2013).

## **Simulación y Experimentación**

Una de las áreas de diseño de sistemas digitales más importante es la simulación. La simulación permite detectar numerosos errores en el diseño de un sistema digital antes de su construcción. Dado que la simulación de sistemas complejos es computacionalmente costosa, se han tratado de desarrollar técnicas para optimizar su ejecución; una de estas es la simulación en paralelo. El desarrollo de simuladores que se ejecuten de manera paralela

implica el uso de algunas técnicas de simulación de eventos discretos, sincronización de la ejecución, manejo distribuido del tiempo, etc.

Actualmente la computación reconfigurable investiga frecuentemente el uso de herramientas basadas en simuladores para proveer un desempeño estimado de sus diseños y métodos, sin embargo, debido a que el alcance de estas simulaciones es generalmente a alto nivel y limitada a la reconfiguración de hardware y posiblemente a un procesador central, los resultados presentados pueden ser desafortunadamente menos confiables particularmente con respecto a la memoria y el tiempo de comunicación entre hardware y procesador. Los sistemas de computación reconfigurable han venido a ser menos teóricos y están más cerca de extenderse, evaluaciones más confiables a través de los simuladores son necesarias para comparar los nuevos diseños de sistema, métodos de configuración, hardware calendarizando algoritmos y estructuras de hardware. Por tanto nosotros discutimos los problemas relacionados en la implementación de un simulador de sistema completo para computación reconfigurable, presentando técnicas de implementación y demostrando el valor de la simulación de computación reconfigurable de un sistema completo en un ejemplo de aplicación.

VHDL, VHSIC (Very High Speed Integrated Circuits) Hardware Description Language, en un HDL ampliamente aceptado por los fabricantes de dispositivos electrónicos. Más aún, VHDL es reconocido como un estándar por IEEE (IEEE estándar 1076) y el departamento de la defensa de los Estados Unidos. Además de los requerimientos mencionados de cualquier HDL, VHDL proporciona la capacidad de diseño jerárquico, lo que permite a los usuarios crear diseños en varios niveles de abstracción de una manera sencilla. Dependiendo del estilo del diseño y de la complejidad del circuito, los diseñadores pueden especificar sus diseños en cualquiera o todos los siguientes niveles de abstracción de diseño: nivel de sistema, transferencia de registros, lógico y de circuitos.

Una vez que se construye un modelo en VHDL, este puede ser simulado de acuerdo con la semántica descrita en el Manual de Referencia del Lenguaje

VHDL. Para que una simulación sea efectiva, la velocidad con la cual se realiza tiene que ser razonable desde la perspectiva del usuario. La velocidad de una simulación está determinada por la eficiencia del simulador utilizado, la velocidad de la computadora utilizada y la complejidad del modelo. Para modelos de sistemas complejos, con varios cientos de miles de compuertas, hasta las computadoras más veloces pueden ser rápidamente rebasadas por los requerimientos computacionales de la simulación.

Una de las alternativas para la simulación de sistemas complejos es explotar el paralelismo inherente que existe en el diseño de los sistemas y hacer la simulación distribuyendo las componentes del modelo en diferentes procesadores. Algunas estrategias para la simulación paralela son vectorización, aceleración por hardware y el uso de sistemas paralelos y/o distribuidos. Aunque cada una de esas alternativas tiene sus ventajas y desventajas, el uso de sistemas de multiprocesamiento es particularmente atractivo debido a que ellos pueden ser usados para sistemas de simulación de eventos discretos, en general, y para otras aplicaciones. Más aún, la tecnología en microprocesadores y en redes de computadoras permite agrupar estaciones de trabajo e interconectarlas mediante una red de interconexión con una velocidad aceptable a un costo razonable.

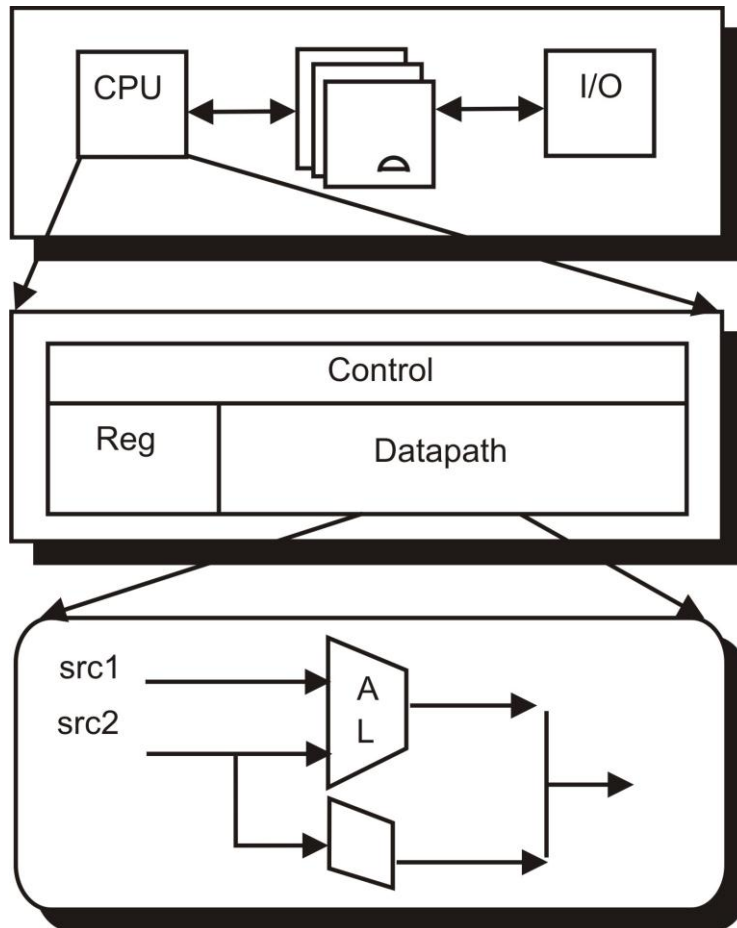
La construcción de simuladores distribuidos (paralelos) de eventos discretos presenta varios problemas que no aparecen en simulación secuencial. En primer lugar, es necesario hacer un particionamiento del sistema completo, Cada parte debe ser asignada a un procesador de manera que el trabajo de simulación quede balanceado entre todos los procesadores. Una vez que empieza el trabajo de simulación es necesario coordinar a todos los procesadores, de tal manera que eventos que ocurren en un procesador y que afectan el comportamiento de otro procesador deban ser comunicados en el momento adecuado. En un simulador de eventos discretos es necesario tener un reloj real y un reloj de simulación. En un simulador distribuido es necesario coordinar a todos los procesadores para que todos ellos mantengan un reloj global virtual. Más aún, los eventos, dado que ocurren en tiempo, deben

respetar la regla de causalidad; esto es, los eventos deben ser procesados en el orden secuencial de acuerdo al reloj global.

La descripción de un sistema mediante VHDL se puede hacer siguiendo dos metodologías: estructural y funcional. En la especificación estructural se modelan los componentes individuales del hardware, los cuales son interconectados para constituir la descripción del sistema original. La técnica de modelado funcional describe la relación entrada-salida que existe en un sistema digital. Esta forma de modelación permite aislar el sistema de los detalles de implementación de sus componentes, modelando solamente su comportamiento externo.

Por un lado, la modelación estructural permite identificar componentes independientes cuyo funcionamiento es quizá concurrente, sin embargo, la simulación de modelos estructurales es relativamente lenta. Por otro lado, la modelación funcional no toma en cuenta la estructura interna de un sistema pero puede acelerar la simulación del diseño ya que no requiere simular los detalles internos de los componentes.

VHDL permite modelar un sistema haciendo uso de ambas técnicas de representación. Combinando una modelación estructural inicial de un sistema con una modelación funcional de cada una de sus componentes es posible explotar el paralelismo en el diseño y hacer una simulación eficaz (diseño de un sistema en al menos dos niveles). En primer lugar, se debe dar una descripción estructural del sistema, identificando cada uno de sus componentes y las relaciones entre ellos mediante la interconexión de puertos de entrada y salida. En segundo lugar, cada uno de los componentes es descrita mediante una descripción funcional indicando la transformación de señales de entrada a señales de salida. La descripción estructural puede involucrar varios niveles de descripción del mismo tipo como se muestra en la figura 8.



*Figura 8. Descripción estructural de un sistema*

La descripción de un sistema a nivel estructural y a nivel funcional permite resolver los problemas de particionamiento y asignamiento de tareas para el proceso de simulación. Dado que se identifican cada uno de los componentes del sistema es posible estimar su complejidad individual para hacer la simulación. Esto permite hacer una distribución balanceada del trabajo de simulación. Por otra parte, las interconexiones entre puertos de entrada y salida establecen claramente las interacciones entre componentes diferentes; esto permite identificar las señales cuyos cambios producirán puntos de sincronización entre procesadores diferentes.



## **Simulación de VHDL**

La simulación de un modelo escrito en VHDL se divide en tres etapas: análisis, elaboración y simulación.

Durante la etapa del análisis, las descripciones en VHDL seleccionadas por el usuario son examinadas para verificar errores sintácticos. Después, las construcciones en VHDL se traducen en árboles abstractos los cuales se descomponen en unidades de diseño que constituyen la descripción original. Así, se obtiene una representación intermedia de VHDL (VIF VHDL Intermediate Format). Durante la etapa de elaboración, las unidades definidas por el usuario son ensambladas con las bibliotecas de componentes básicos especificados por el usuario. Las unidades son ligadas produciendo un conjunto de procesos interconectados por una red de señales. Estos son utilizados para simular el comportamiento funcional del sistema. Por último, la simulación es particionada en varias entidades que se ejecutan concurrentemente llamadas procesos. Un proceso interactúa con otros intercambiando eventos en forma de mensajes. Una colección de procesos es asignada a un proceso lógico (PL), el cuál no toma parte en la simulación, solamente sirve para agrupar procesos. Asignando diferentes PL a varios procesadores permite el paralelismo en la ejecución del simulador.

## **Ciclo de simulación de VHDL**

La simulación se refiere a la ejecución de las declaraciones concurrentes en el diseño, propagando los valores de las señales, actualizando las señales implícitas, etc. El VHDL LRM 1076-1993 establece claramente el algoritmo para la simulación de un diseño VHDL. Antes de iniciar la simulación, el kernel, que es el responsable del total de la operación debe comenzar la fase de inicialización.

La fase de inicialización consiste en los siguientes pasos:

- Se calcula el valor del manejador y el valor efectivo de las señales explícitas y se asigna a cada señal el valor efectivo.
- Cada señal implícita GUARD toma el valor del resultado de la evaluación de la correspondiente expresión asociada.
- Cada proceso no aplazado en el diseño es ejecutado hasta que es suspendido.
- Cada proceso aplazado en el diseño se mantiene en ejecución hasta que es suspendido.
- Se evalúa el tiempo en el siguiente ciclo de simulación.

El ciclo de simulación consiste de ejecutar repetidamente los siguientes pasos hasta que la simulación concluye:

- Se calcula el tiempo actual. Si se tiene todas las condiciones de terminación de la simulación, la simulación termina, de lo contrario los siguientes pasos son ejecutados.
- Las señales explícitas son actualizadas.
- Para cada proceso P sensitivo a una señal S, si un evento ocurre en S, P se reactiva.
- Cada proceso no aplazado que ha sido reactivado en este ciclo se mantiene en ejecución hasta que es suspendido.
- Se calcula el tiempo del siguiente ciclo de simulación.

- Si el siguiente ciclo no es un ciclo delta, cada proceso no aplazado que no ha sido ejecutado en ese ciclo se ejecuta hasta que se suspende.

### **Simulación paralela de VHDL**

El simulador paralelo explota el hecho que los procesos VHDL definidos por el usuario pueden ser ejecutados concurrentemente. Una partición está compuesta como un conjunto de procesos definidos por el usuario (PL) y es asignada a los procesadores disponibles. Una vez construidos, los PL son ejecutados en los procesadores. Una vez construidos, los PL son ejecutados en los procesadores. Cada PL procesa una serie de transacciones las cuales son generadas por él mismo o por otras PL. La simulación termina cuando ningún PL tiene transacciones por procesar.

Aún cuando es posible mejorar la eficiencia de la simulación ejecutando de manera concurrente a los PL, la implementación de simulación paralela requiere respetar el orden en que los eventos se realizan de acuerdo con un tiempo de simulación global. Así, la regla de causalidad establece que no se puede procesar ningún evento que haya ocurrido en un tiempo global menor al actual. Para garantizar la regla de causalidad, en ocasiones es necesario retrasar el estado del sistema en tiempo hasta un punto seguro, en donde se sabe no se viola ninguna relación de causalidad. Este proceso, llamado <<rollback>>, requiere deshacer todos los cambios que han ocurrido desde el último punto de verificación.

El elemento básico de simulación en VHDL es una señal. Una señal puede tomar valores diferentes en el tiempo. Los cambios en los valores de las señales provocan transacciones las cuales necesitan ser procesadas por todos aquellos procesos que utiliza el valor de la señal. Un manejador (driver) de una señal es el conjunto de todos los procesos que asignan un valor a una señal, estos valores son utilizados para definir el valor efectivo de la señal utilizando la función de resolución. Un valor puede ser asignado a una señal con un retardo,

el cual indica el manejador de la señal que asigne el valor después de cierto tiempo; a estos valores se les conoce como la forma de onda proyectada en la salida.

Una descripción típica de un PL incluye colas para entrada, para salida y para los estados. Las colas de entrada son la unión de todas las colas de transacciones. La cola de salida es utilizada para almacenar todas las transacciones enviadas a otros PL. Por otro lado, la cola de estado proporciona el almacenamiento del estado del sistema a intervalos regulares de tiempo.

La secuencia de eventos que, por medio de una declaración de asignación de señal, modifican el valor de una señal es conocida como propagación del valor de una señal. Ciertas señales pueden estar asociadas a dos valores durante el ciclo de simulación, el valor del manejador y el valor efectivo, el primero está definido como el valor de la señal que es utilizada como origen para otras señales. El valor efectivo de una señal es el valor obtenido por la evaluación de una referencia a una señal dentro de una expresión. En general, un asignamiento a una señal genera un evento con la información requerida y lo envía a todos los procesos que utilizan la señal, como se muestra en la Figura \*9. En caso de una señal compuesta, se envían los eventos correspondientes a cada uno de los componentes independientemente. El proceso que recibe tal evento actualiza la copia del manejador de la señal, ejecuta la resolución de la señal y la conversión de tipos para calcular el valor efectivo de la señal. Mientras tanto, el tiempo de recepción del evento es calculado agregando el valor de la expresión de retardo al tiempo actual del proceso. Simultáneamente, el simulador integra la lista de transacciones dentro de la cola de entrada de cada uno de los PL, agregado una cola de marcado para implementar la actualización de la forma de onda proyectada.

El fanout de una señal es el conjunto de procesos, después de la elaboración, que evalúan una referencia a esta señal dentro de una expresión, esto no incluye procesos que manejan la señal sin evaluar su referencia, por lo que, cada uno de estos procesos mantiene una copia separada de la señal; además

el nuevo valor de la señal debe ser notificado a todos los procesos cuando ocurre una asignación de señal.

La propagación del valor de una señal está compuesta por cuatro etapas: asignamiento de la señal, marcado, actualización del manejador y etapa de resolución. La forma de onda proyectada de la señal es modificada en la etapa de asignamiento de la señal, y el valor efectivo es calculado utilizando las funciones de resolución y las funciones de conversión de tipos en la etapa de resolución.

### **Simulación de eventos discretos**

Un simulador es la especificación de un sistema físico en términos de un conjunto de estados y eventos; la función básica que ejecuta es la de detectar la ocurrencia de eventos en el tiempo y reconocer sus efectos por medio de estados. En una simulación continua, los cambios de estado ocurren de manera continua en el tiempo, mientras que en una simulación discreta la ocurrencia de un evento es instantánea y fija a un punto seleccionado en el tiempo. Para el caso de VHDL se realiza una simulación por eventos discretos.

La simulación discreta más usual es manejada por tiempo en la cual el tiempo de simulación avanza en pasos (ticks) discretos de tamaño constante; esto es, la observación del sistema es discretizado por intervalos de tiempo unitarios. Otra forma de conducir la simulación es mediante eventos en la cual la observación del sistema simulado se realiza en el instante en que ocurre un evento. Un sistema de simulación de eventos discretos, SED, cuando es ejecutado secuencialmente, procesa iterativamente eventos en tiempo de simulación (conocido como tiempo virtual TV) manteniendo una lista de eventos (LEV) ordenados con respecto al tiempo, mientras un reloj global indica el tiempo actual y las variables de estado S definen un estado actual del sistema. Una máquina de simulación maneja la simulación tomando continuamente el primer evento de la lista de eventos, simulando el efecto del evento actualizando las variables de estado y/o programando nuevos eventos en el EVL, posiblemente también removiendo eventos obsoletos. Se mantiene la

ejecución hasta que se alcanza un tiempo definido como el final de la simulación, y no existen más eventos a procesar.

### **Niveles de paralelismo/distribución**

El proceso de simulación es una tarea compleja y de computación intensiva, y está limitada por la computadora utilizada y a la complejidad del sistema a simular. El uso de máquinas más potentes es costoso, pero si se usan computadoras pequeñas conectadas por una red, como una computadora distribuida, es posible simular diseños complejos.

Existen tres formas típicas de realizar una simulación paralela o distribuida de manera efectiva: a nivel de replicación, a nivel de subrutina o componente y a nivel de eventos. En el primer caso, se replica el sistema a simular en varios procesadores para simular más casos en la misma cantidad de tiempo. Este enfoque es adecuado para la simulación de sistemas con muchas instancias, sin embargo, no es muy útil para simular una sola instancia de un diseño complejo.

La simulación a nivel de componentes o subrutinas es particularmente adecuada cuando se requiere serializar los procesos de una misma instancia. En este caso, se pueden agregar actividades de simulación para distribuir tales como el procesamiento de eventos, la actualización del estado del sistema y la recolección de estadísticas.

En el nivel de eventos se intenta distribuir eventos entre varios procesadores para su ejecución concurrente; existen dos posibles maneras de lograr esto, mediante una LEV centralizado o descentralizado. En un esquema donde la LEV es una estructura de datos centralizada mantenida por un procesador maestro, la aceleración puede ser lograda distribuyendo eventos concurrentes en un conjunto de procesadores esclavos dedicados a ejecutar estos eventos. El procesador maestro se encarga de mantener la consistencia de la estructura de los eventos, prohibiendo la ejecución de aquellos eventos que potencialmente pueden producir violaciones de causalidad, debido al traslape

de sus efectos cuando son procesadores concurrentemente. Este esquema es particularmente apropiado para multiprocesadores de memoria compartida, donde la LEV puede ser implementada como una estructura de datos compartida accesada por todos los demás procesadores.

Por otro lado, la simulación a nivel de eventos mediante una LEV descentralizada, nos permite alcanzar el mayor grado de paralelismo, ya que los eventos que existen en puntos arbitrarios de espacio-tiempo pueden ser asignados a diferentes procesadores, de una manera regular o irregular. Actualmente se logra incrementar el paralelismo en estrategias que permiten la simulación concurrente de eventos con diferentes marcas de tiempo (timestamp). Estos requieren de protocolos para una sincronización local, lo cual puede incrementar el costo de comunicación dependiendo de la dispersión de eventos en el espacio y el tiempo del modelo de simulación.

### **Simulación de procesos lógicos**

Las estrategias de simulación con distribución a nivel de eventos tratan de dividir una tarea de simulación global en un conjunto de procesos lógicos (PL) comunicantes, para lograr la explotación del paralelismo inherente entre los respectivos componentes del modelo mediante la ejecución concurrente de estos procesos. La simulación de procesos lógicos es la cooperación de un arreglo de PL's interactuando, cada uno de los cuales simula un subespacio del espacio-tiempo conocido como región. La simulación de procesos lógicos incluye las siguientes actividades:

Un conjunto de PL's es diseñado para ejecutar eventos de manera síncrona o asíncrona en paralelo.

- Un Sistema de Comunicación (SC) permite intercambiar datos locales, además de sincronizar las actividades locales.
- Cada PL, es asignado a una región R, como parte del modelo de simulación, MS, El MS opera sobre un R en la modalidad de manejo por

evento, ejecutando ocurrencias locales y remotas de eventos, avanzando el reloj local (tiempo virtual local, TVL).

- Cada PL tiene acceso solamente a un subconjunto particionado estéticamente de variables de estado S que es diferente a las variables asignadas a otros PLs.
- Se procesan dos clases de eventos en cada PL: 1) eventos internos que sólo afectan a las variables de estado asignadas a este proceso lógico y 2) eventos externos que tienen efecto en variables de estado locales a otros PL's.
- La interfaz de comunicación (IC) se encarga de la propagación de los efectos causales de los eventos simulados por PL's remotos, y la incorporación de efectos causales a la simulación local producida por PL's remotos. Para esto existen dos métodos: conservadores y optimistas.

De acuerdo con la posición sobre el avance en la ejecución de eventos, las IC's para la simulación de procesos lógicos se pueden clasificar en conservadores u optimistas. Ambos están basados en el envío de mensajes que portan información de causalidad que ha sido creada por un PL y afecta a uno o más PL's. Por otro lado, la interfaz de comunicación es la responsable de prevenir la violación de causalidad de eventos de manera global. Para el protocolo conservador, la interfaz de comunicación activa a la máquina de simulación (MS) de manera que previene la ocurrencia de errores de causalidad (bloqueando la MS si existe un evento que pueda causar una violación de causalidad). Mientras que en el protocolo optimista la IC activa la MS para rehacer la simulación de un evento detectando si el procesamiento prematuro de eventos locales es inconsistente con las condiciones de causalidad producidas por otros PL's. En ambos protocolos los mensajes son llamados y captados por las IC de cada uno de los PL's.



Una plataforma de simulación de computación reconfigurable (RC) podría proveer un ambiente flexible para examinar nuevas ideas y comparar diferentes diseños antes de seleccionar el mejor o una combinación de ellos para implementación aún en un FPGA o en uno o más chips customizados. El diseño modular podría permitir simulación de diferentes diseños de hardware reconfigurable (RH) sin cambiar la funcionalidad predefinida. El uso de una plataforma de simulación modificable podría permitir a los diseñadores alterar el acoplamiento del Hardware Reconfigurable con otras partes del sistema, tales como el procesador y la memoria. Un simulador de sistema completo podría también permitir un cierto grado de realismo no disponible si se usan herramientas de estimación simple. Actualmente algunos investigadores tienden a utilizar simuladores y herramientas de estimación que sólo modelan el espacio de código del usuario e ignoran las actividades a nivel de sistema. Estas herramientas pueden también carecer de modelos de memoria realistas y pueden modelar ISAs secundarios tales como Alpha o PISA (Burger, 1997). En estos casos, puede ser complicado contextualizar los resultados en términos de la tecnología del procesador actual, y para ISAs impares, compilar nuevos puntos de referencia útiles puede ser un proceso difícil y doloroso.

La plataforma de simulación no pretende reemplazar el valor de implementación de un sistema en hardware. Sin embargo, el diseño de un sistema Reconfigurable es un proceso complejo. Dependiendo de las necesidades y los objetivos de los diseñadores, el sistema puede ser construido desde componentes off-the-shelf ó FPGAs comerciales. Diferentes diseños lógicos pueden ser acomodados por el modelado del nuevo diseño en un HDL y sintetizar a un FPGA.

Para diseños de acoplamiento de lógica reconfigurable con uno o más microprocesadores, hay un número limitado de opciones de microprocesador fácilmente disponibles para tarjetas FPGA comerciales y SOPC/Platform FPGA restringiendo las opciones del diseñador. Aunque diferentes procesadores son ofrecidos para usarse en FPGAs, estas tienden a tener capacidades similares de una a otra, generalmente caen dentro de categorías de procesadores embebidos las cuales no cumplen con los requerimientos de los diseñadores.

Una plataforma de simulación flexible y modificable permite modelar una variedad muy grande de diseños con inversiones muy pequeñas. El sistema puede ser modelado a varios grados de detalle, permitiendo al investigador realizar una rápida pero menos detallada simulación temprana en el desarrollo de una nueva idea, y una más sofisticada implementación.

### **Creación de diseño y simulación de un FPGA**

Un FPGA (*Field-Programmable Gate Array*) es un circuito integrado que dicho en términos llanos puede configurarse para llevar a cabo cualquier función lógica y hacer lo que a su dueño le plazca.

Claro que para conseguir eso el diseñador debe configurar el circuito, normalmente siguiendo la especificación de un lenguaje de descripción de hardware. Esto es algo así como hacer código en vez de electrónica digital. Considera que con un FPGA eres capaz, en teoría, de reinventar todo tipo de dispositivos de cómputo. Incluso hacerlos trabajar en un mismo espacio y de forma paralela.

Por supuesto, en la práctica la creación está limitada por la capacidades de cada tarjeta FPGA, y también por la plataforma. Para trabajar con un FPGA debes contar con un software especial creado por el fabricante.

Para efectos de simulación en este trabajo utilizaremos la versión 9.1 de Active-HDL que es un programa bajo ambiente Windows orientado al diseño y la simulación de FPGAs integrados. Se pueden usar los lenguajes Verilog o VHDL.



Figura 9. Portada de la versión 9.1 de Active-HDL

Comenzaremos creando una nueva área de trabajo en Active-HDL, seleccionando el las opciones de *File, New, Workspace* y asignando un nombre para su identificación, como se muestra en las figuras 10 y 11.

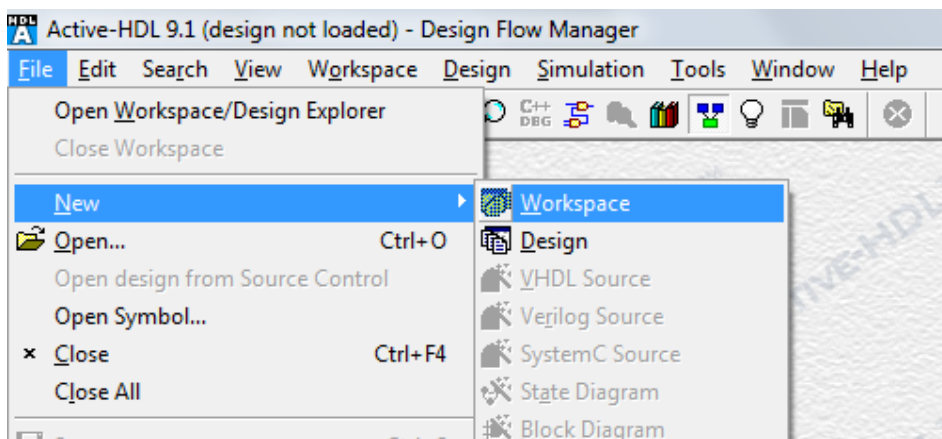


Figura 10. Iniciando una nueva área de trabajo en Active-DHL

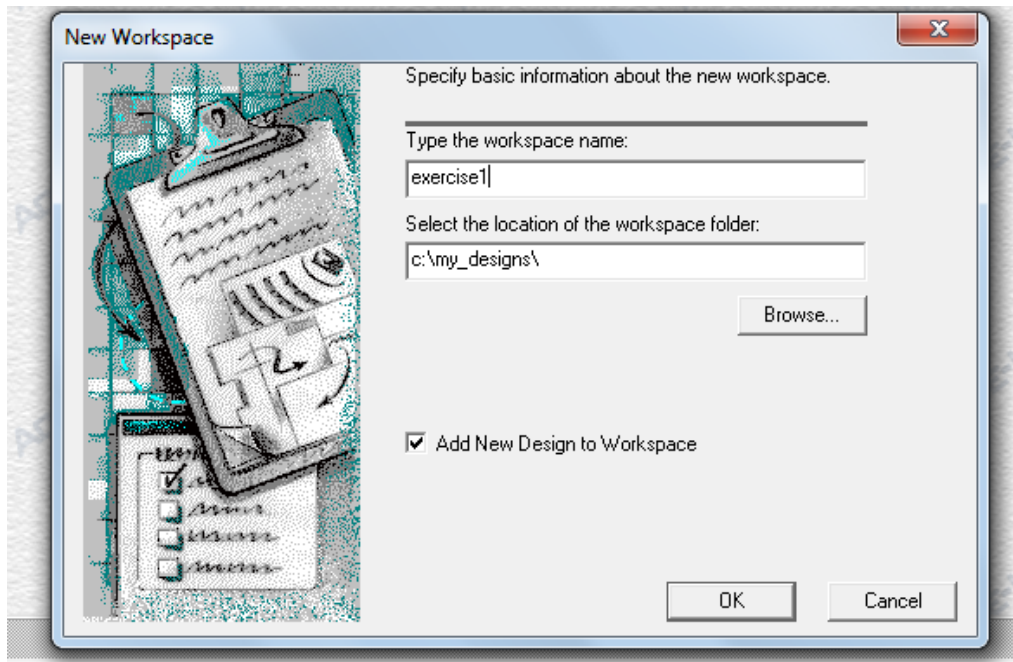


Figura 11. Asignación de un identificador a nueva área de trabajo

Inmediatamente después de la creación del área de trabajo se debe crear un diseño para comenzar a trabajar:

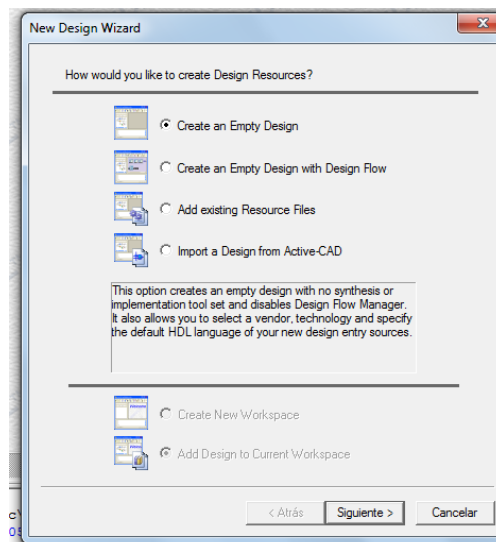


Figura 12. Creación de un diseño vacío en el área de trabajo

Se debe seleccionar el tipo de lenguaje que utilizaremos dentro de nuestro diseño y simulación, podemos elegir entre Verilog y VHDL, para nuestro caso particular seleccionaremos VHDL de acuerdo a toda la investigación que se ha venido desarrollando a lo largo de este documento:

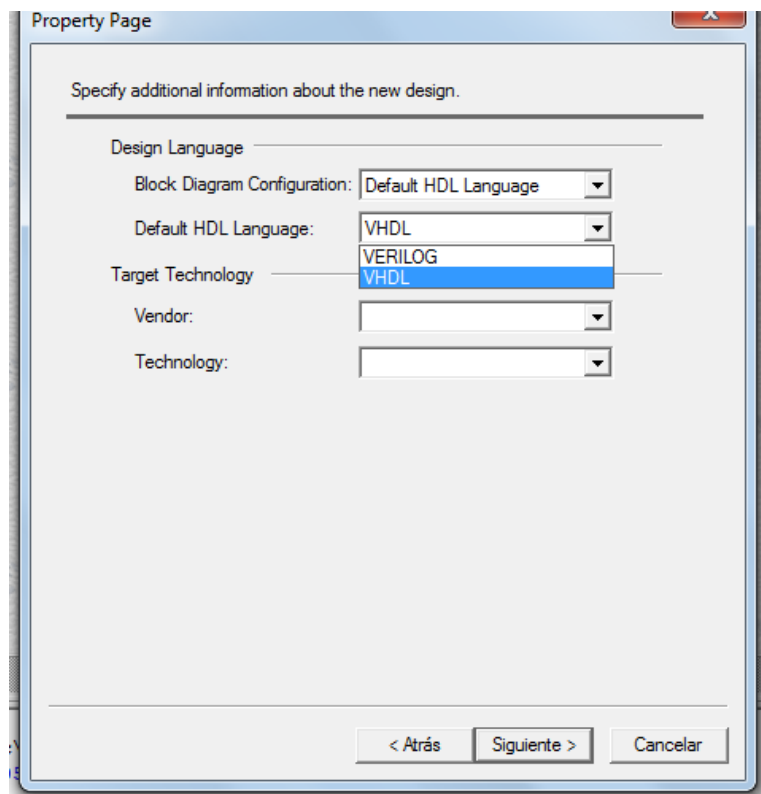


Figura 13. Selección del lenguaje de desarrollo para la prueba

A continuación se asignará el nombre del diseño que estaremos desarrollando con el lenguaje VHDL previamente seleccionado y en el área de trabajo definida como “exercise1”:

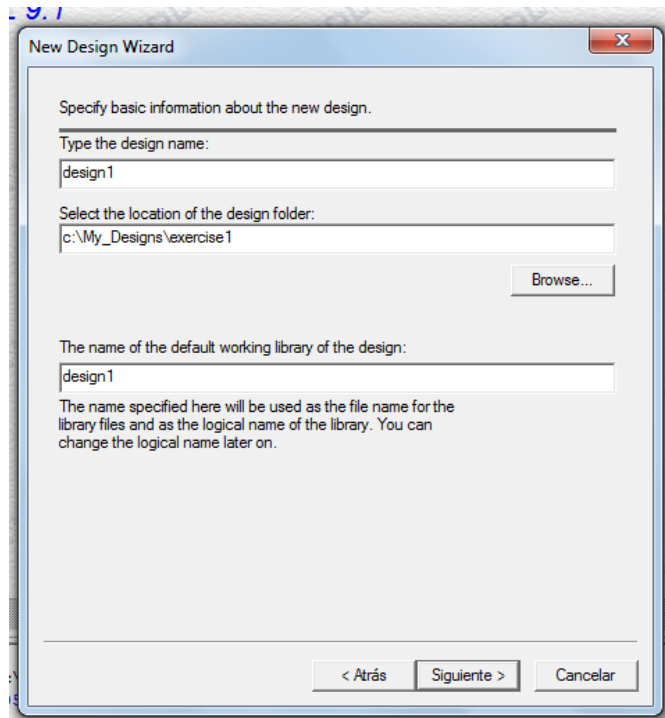


Figura 14. Asignación de nombre para el diseño

Una vez confirmado el nombre del diseño y de su ubicación, podemos agregar un nuevo archivo que añadirá funcionalidad a nuestro componente mediante codificación en VHDL, seleccionando también la opción de VHDL Source Code y asignándole un nombre (file1.vhd):

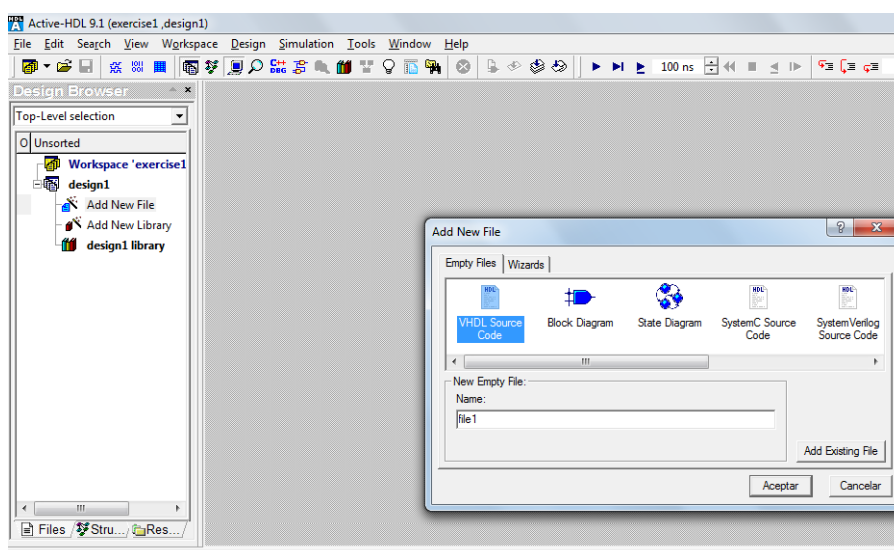


Figura 15. Creación de un nuevo archivo de codificación

El primer ejemplo que realizaremos será algo muy sencillo, simplemente crearemos una compuerta lógica con una funcionalidad AND programada por medio de VHDL y a la cual agregaremos 2 valores de entrada y uno de salida, teniendo en consideración las siguientes premisas:

Se debe declarar la librería IEEE por default (`library ieee`) debido a los derechos que este organismo tiene sobre la regulación y actualización de VHDL desde 1987, es un estándar en la programación de VHDL.

Se deberá crear una entidad que identifique el segmento de código en donde serán definidas las entradas y las salidas de la funcionalidad que se estará programando (`entity AND_gate is`).

Se define un puerto que contenga la estructura de la entidad que se está programando (entradas, salidas y sus tipos de dato para su operación).

Finalmente se define la lógica que se asignará a la entidad definida, es decir, se crea la funcionalidad que queremos probar. Esto se hace mediante una asignación de arquitectura (`architecture simple of AND_gate is`).

Por tanto nuestro primer segmento de código contiene lo siguiente:

```
library ieee;  
use ieee.std_logic_1164.all; --se debe declarar la librería IEEE por default  
  
entity AND_gate is --se crea una entidad  
    port (  
        A : in std_logic; --se declara un puerto con 2 entradas y una  
salida  
        B : in std_logic;  
        Y : out std_logic  
    );  
end entity AND_gate;
```

--lógica del circuito que se declaró y se define mediante una arquitectura, simple es el nombre de arquitectura que uso para AND\_gate

architecture simple of AND\_gate is

begin

Y <= A and B;

end architecture simple;

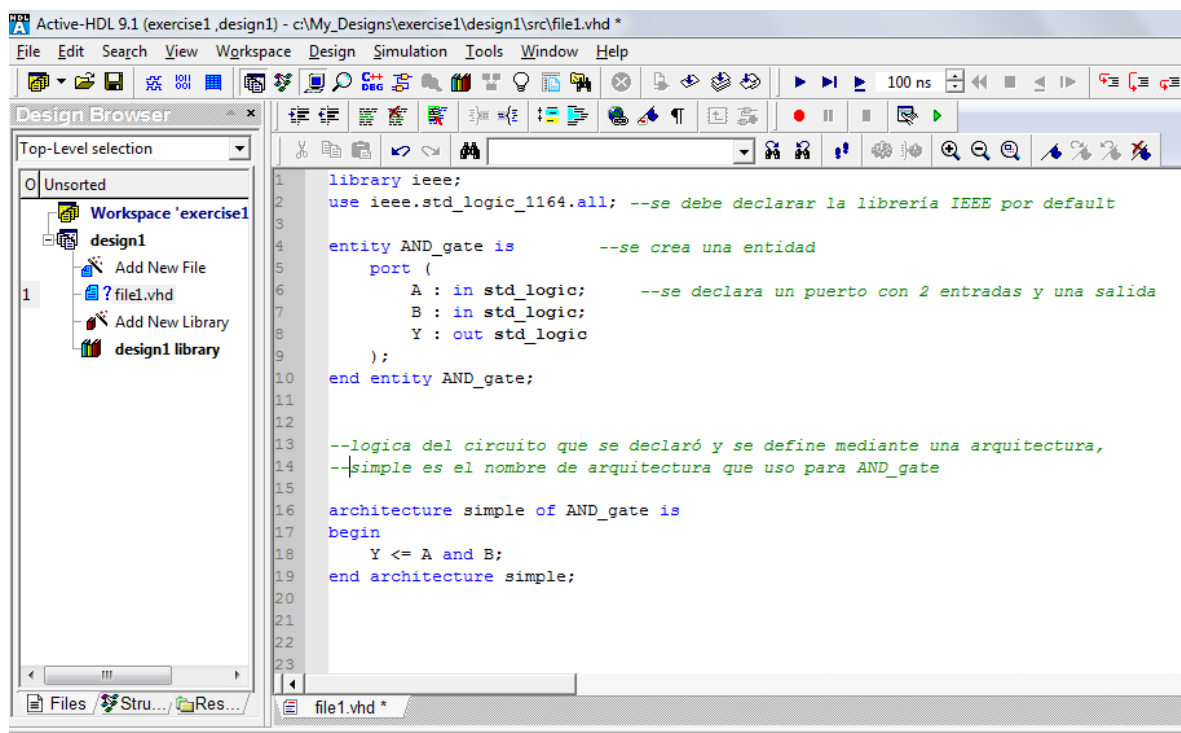


Figura 16. Generación de código VHDL en Active-HDL

Como cualquier código convencional, es necesario saber si está libre de errores para su ejecución y por tanto se requiere de una compilación seleccionando las opciones *Design*, *Compile all* como se muestra en la figura 17:



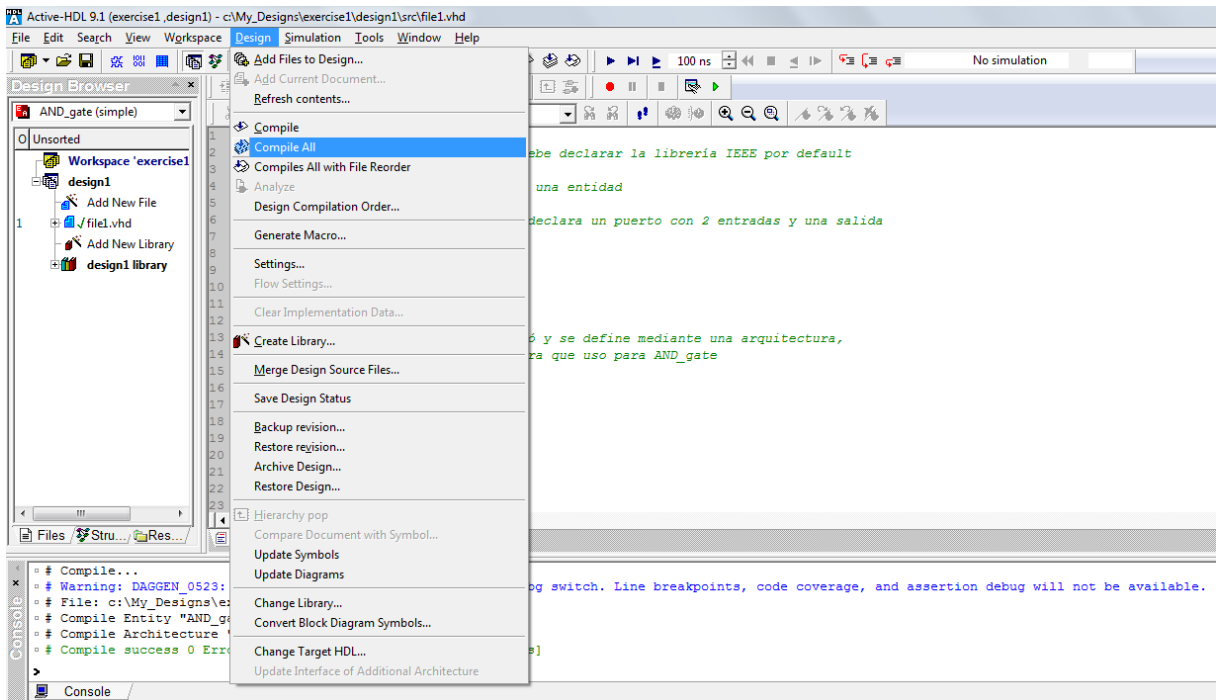


Figura 17. Compilación de código VHDL en Active-HDL

A través de la barra de resultados es como sabemos si nuestro código tiene algún tipo de violación previo a su ejecución:

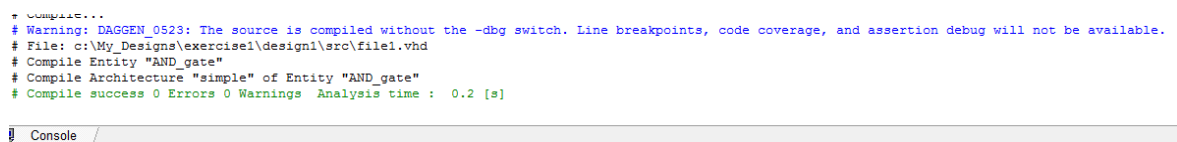


Figura 18. Barra de resultados en la compilación

Una vez que hemos confirmado que nuestro código está libre de cualquier anomalía, podemos continuar con la preparación para la simulación en donde se deberán establecer los parámetros, condiciones y ambiente de prueba para poder validar la funcionalidad del código implementado al interior de nuestro componente. Para poder iniciar el proceso de simulación dentro de Active-HDL, debemos de ir a las opciones Simulation, Initialize Simulation para poder adecuar la aplicación al ambiente de simulación.

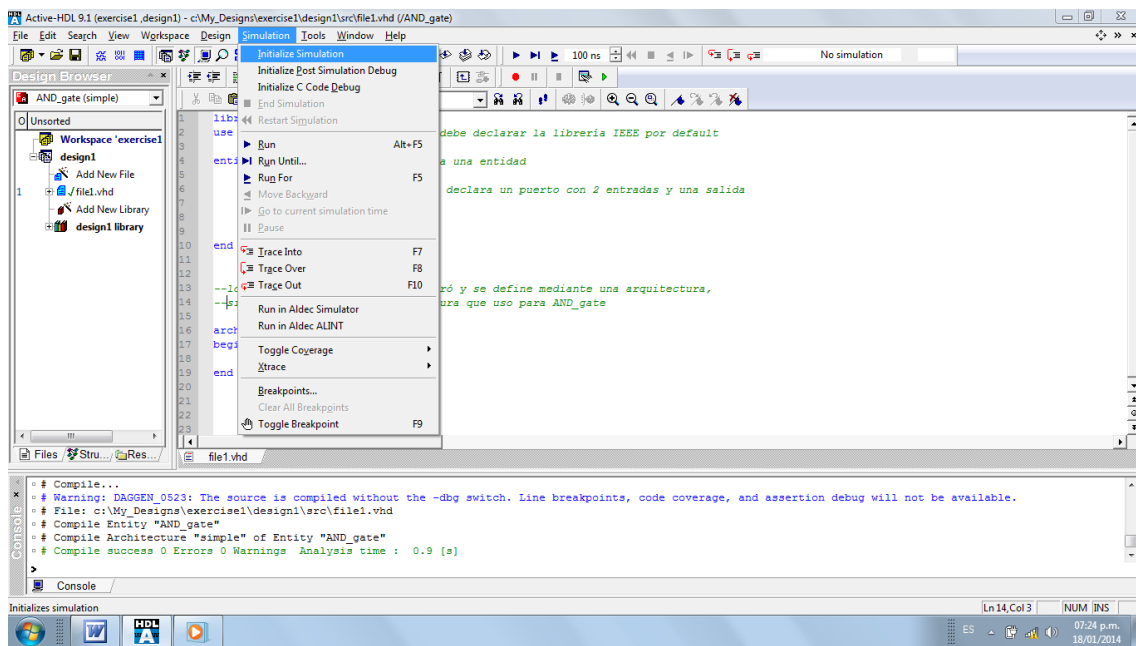


Figura 19. Inicialización de la simulación

Dentro de la barra de resultados podemos ver la confirmación de la inicialización de la simulación y en el extremo izquierdo del área de trabajo, también podemos observar el listado de los elementos que fueron declarados dentro del código para interactuar en la funcionalidad que se programó:

```

▫ # KERNEL: Kernel process initialization done.
▫ # Allocation: Simulator allocated 5748 kB (elbread=1023 elab2=4633 kernel=90 sdf=0)
▫ # KERNEL: ASDB file was created in location c:\My_Designs\exercise1\design1\src\wave.asdb
▫ # 07:26 p.m., sábado, 18 de enero de 2014
▫ # Simulation has been initialized
▫ # Selected Top-Level: AND_gate (simple)
>
Console

```

Figura 20. Reporte de la inicialización de la simulación

Name	Value
A	U
B	U
Y	U

Figura 21. Elementos de la entidad a ser simulada

Con el proceso de inicialización concluido, es posible correr una simulación sin haber configurado todo el ambiente de prueba, mediante las opciones de *Simulation, Run*

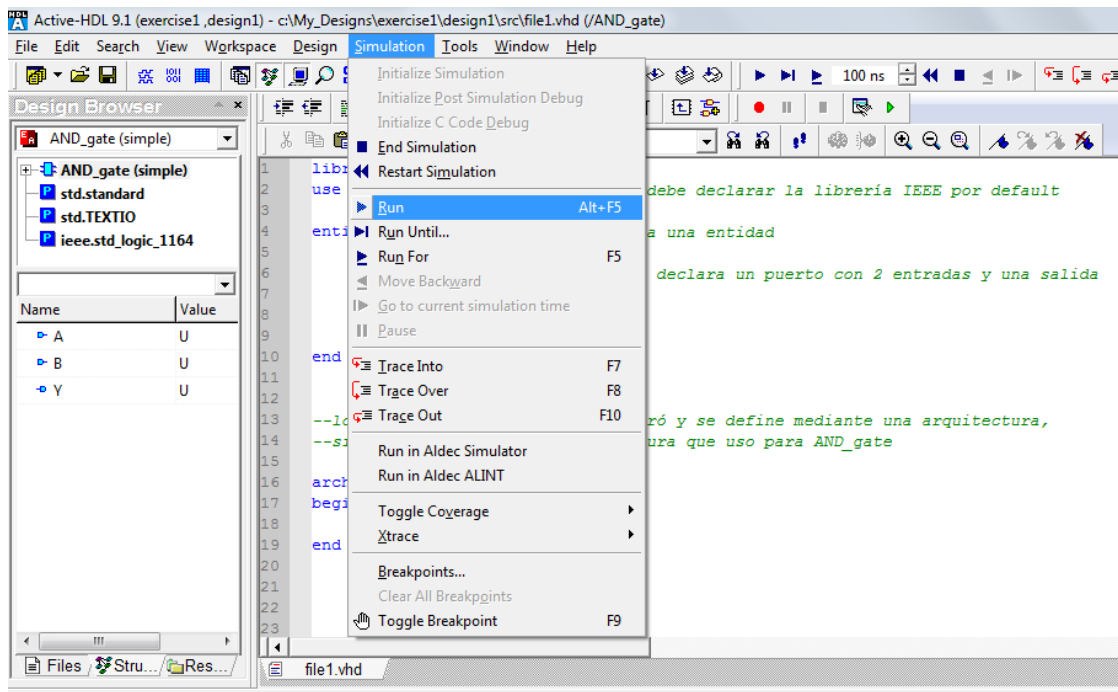


Figura 22. Ejecución de simulación sin previa activación de un ambiente de prueba

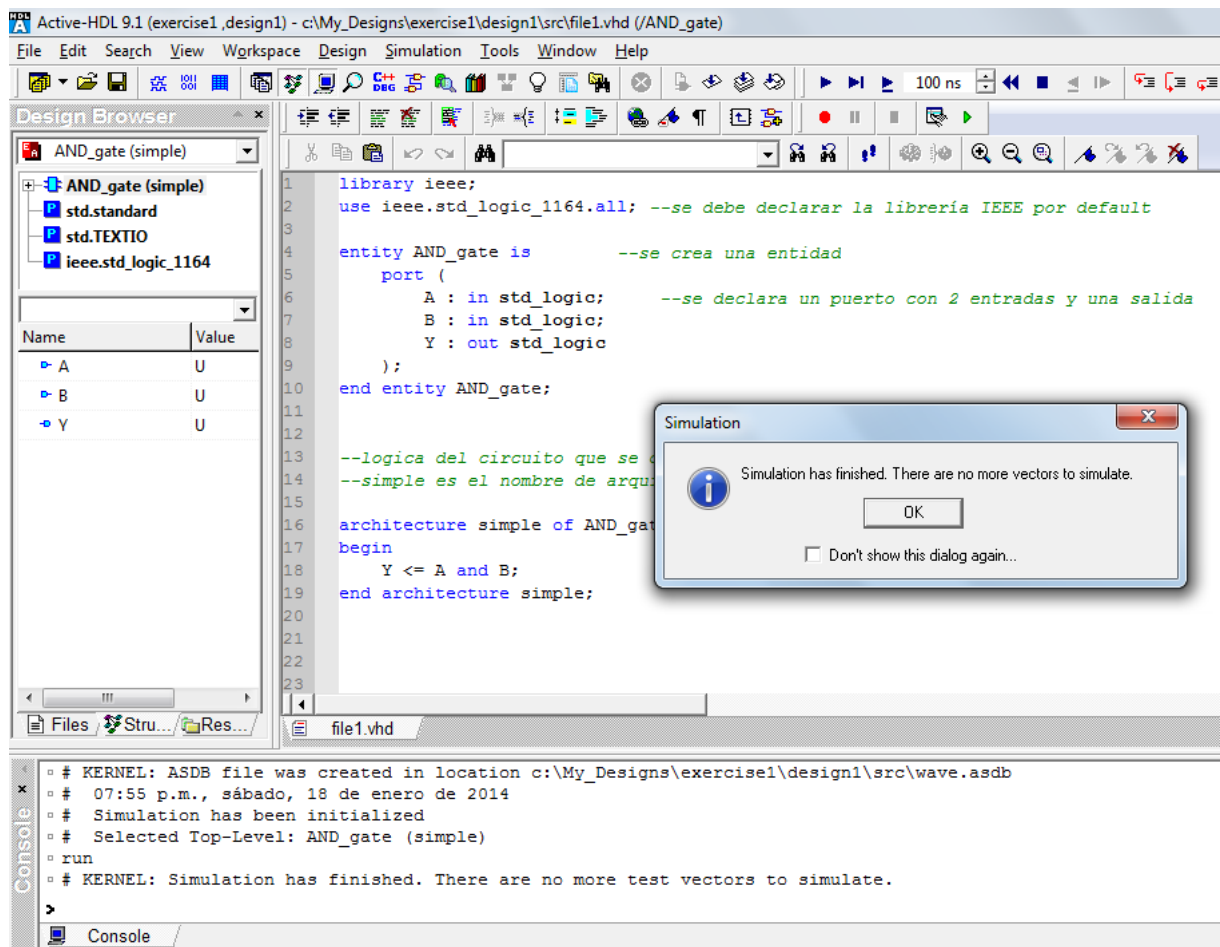


Figura 23. La Simulación fue satisfactoria aún a pesar de no tener activado un ambiente de prueba

Para efectos de mostrar evidencia en la simulación existe un mecanismo por medio del cual podemos observar el comportamiento de las señales que se van a insertar en nuestra entidad de prueba, y a través de la lógica implementada, también podrá observarse cuál es el comportamiento final dando lugar a una señal de salida, este mecanismo es la generación de una forma de onda para la representación de señales en una lógica, dentro de Active-HDL seleccionaremos las opciones *File, New, Waveform* como se muestra a continuación:

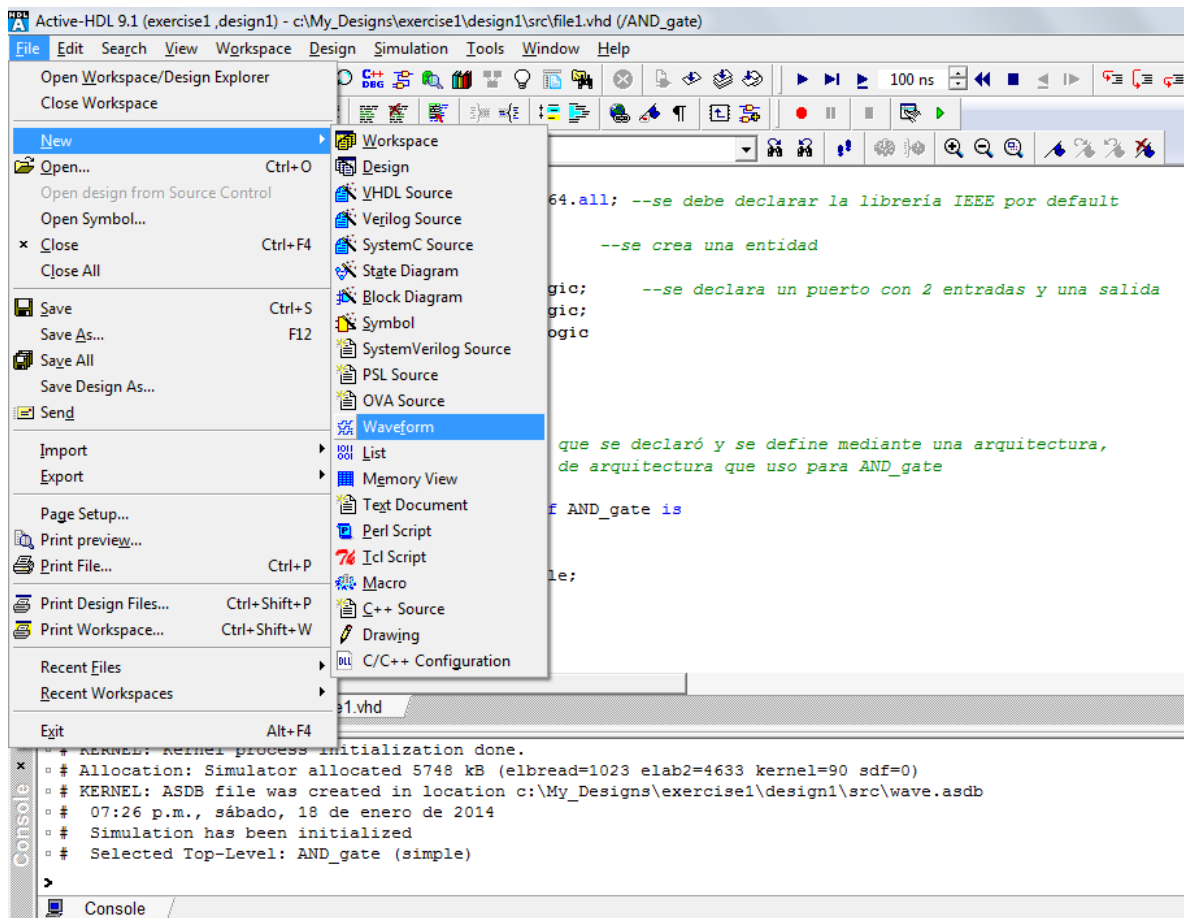


Figura 24. Generación de una Waveform para la representación de señales

A continuación se genera un área de trabajo en donde podremos inicializar nuestros parámetros de la entidad, definir los intervalos de tiempo en donde se moverán las señales de entrada y salida, y lo más importante, su representación gráfica que nos permite tener evidencia en todo momento de la ejecución de nuestra lógica dentro de nuestro componente:

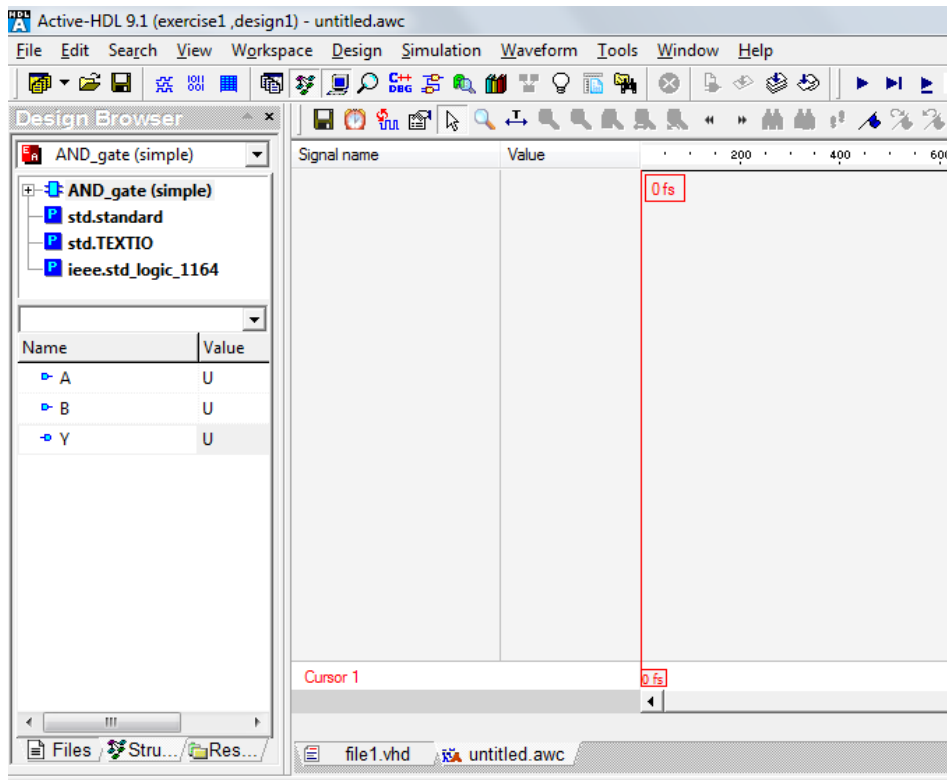


Figura 25. Vista de Waveform disponible para configurar el ambiente de prueba

El primer paso para configurar nuestro ambiente de prueba es tomar todas las señales de entrada y salida definidas en nuestra entidad, y ponerlas dentro de nuestra forma, esto se hace arrastrando cada señal, de la parte inferior izquierda a nuestro panel principal, en donde se leen los campos “Signal name”, “Value” como se muestra a continuación:

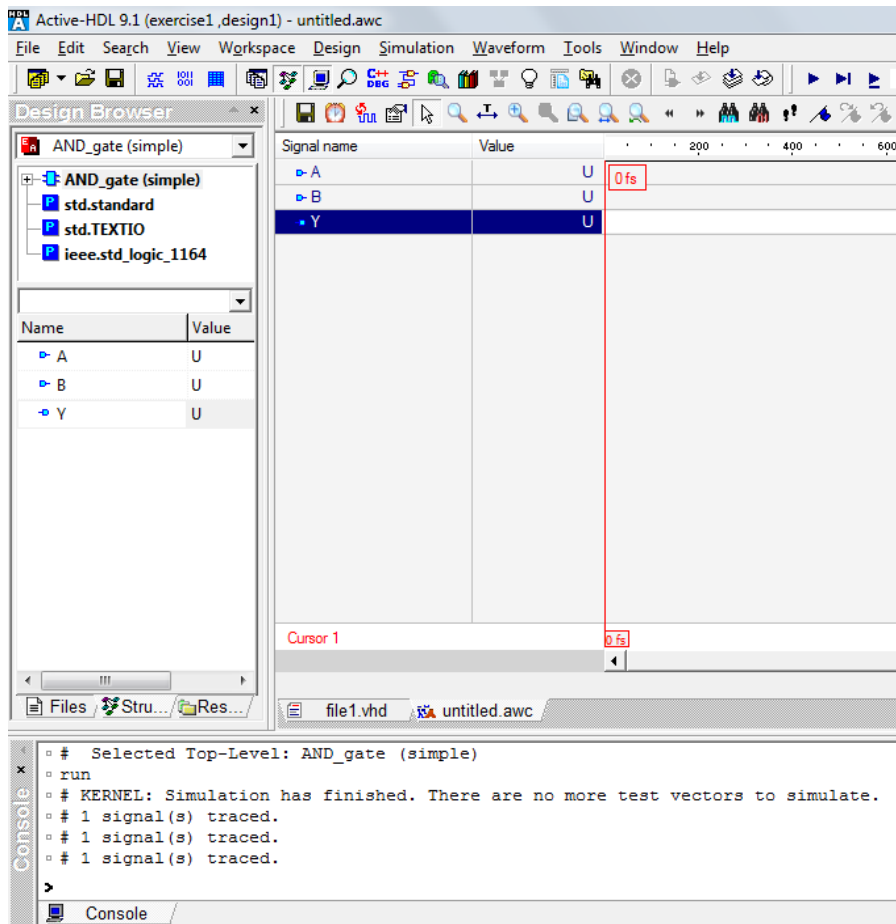


Figura 26. Las señales son agregadas al ambiente de prueba

Las señales han sido agregadas en el ambiente de prueba, cada señal involucrada tiene un valor “por Default” de “U” que representa valor no asignado “Unassigned”, en nuestro caso es preciso iniciar cada señal de entrada con un valor de tipo de dato válido para la lógica programada, particularmente la lógica implementada es una operación “booleana” y en ese contexto, los valores admitidos serán “0” y “1”.

El método para poder asignar los valores mencionados consiste en dar un “click” derecho sobre la señal que requiere ser inicializada (sobre el panel principal de Waveform) y después hay seleccionaremos la opción *Stimulators...*

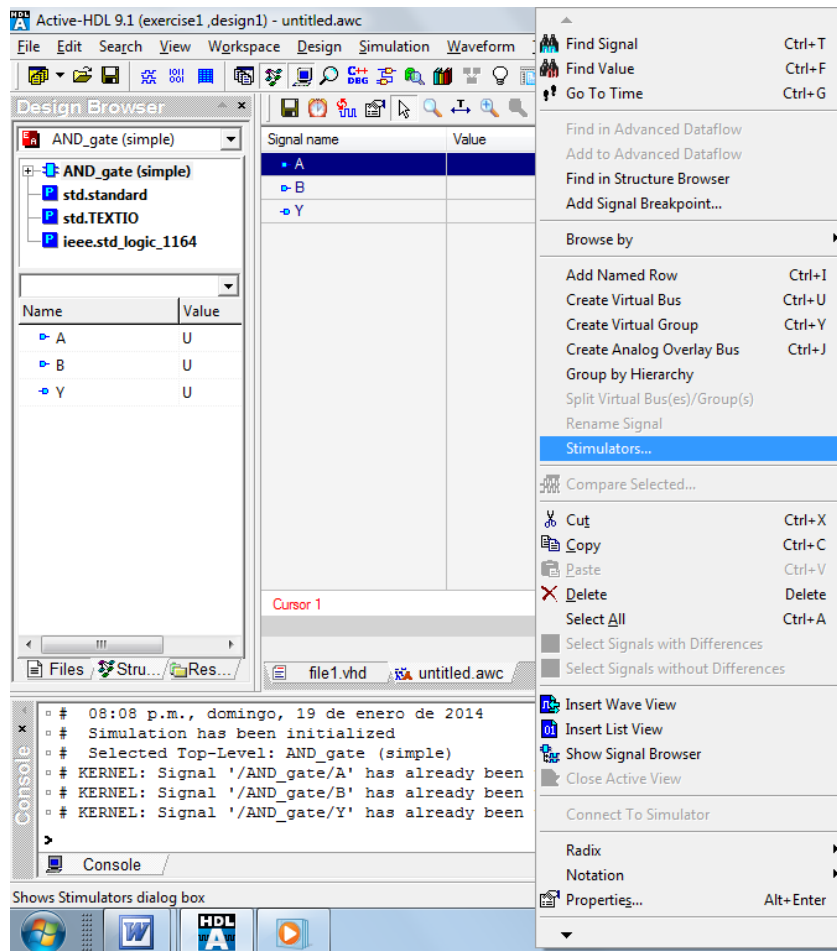


Figura 27. Inicialización de señales de entrada

Posterior a la selección de la opción *Stimulators...* aparecerá un cuadro de diálogo en donde podemos seleccionar los valores de entrada para nuestras señales, en este caso particular y de acuerdo a la lógica implementada, seleccionaremos del tipo de dato (Type) la opción “Value” para poder inicializar nuestras señales con “0” y “1”, una vez seleccionado el valor, este es confirmado mediante el botón de “Apply” y el cuadro de diálogo es cerrado mediante el botón “Close”:



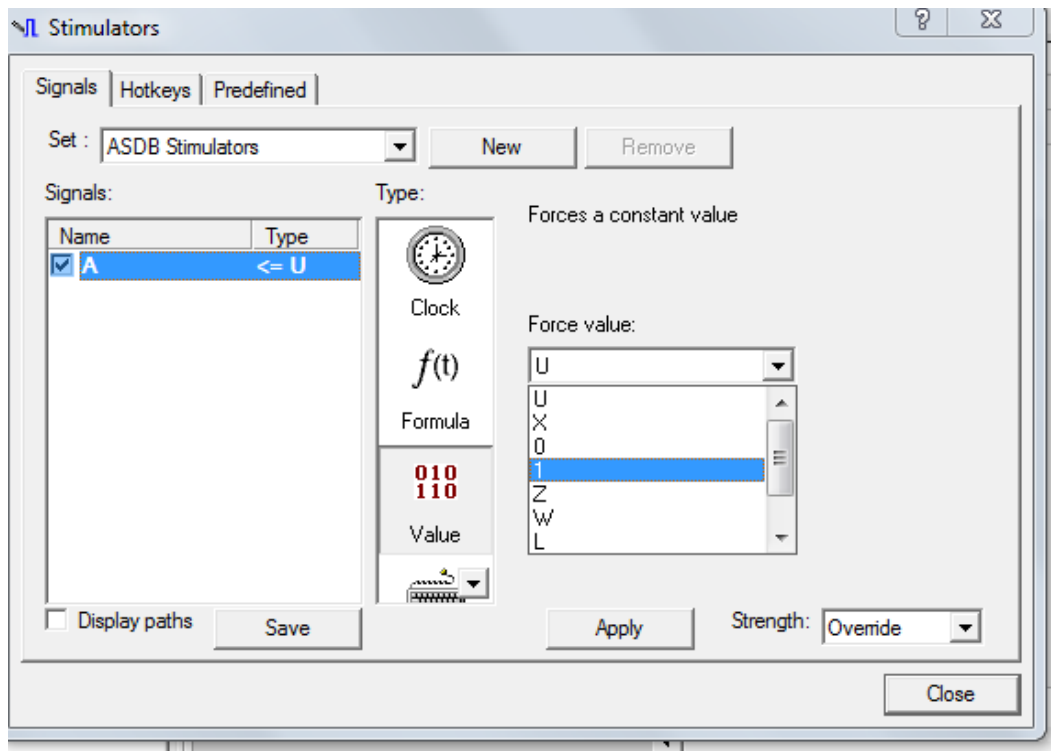


Figura 28. Inicialización de señales con valores lógicos "0" y "1"

El proceso de inicialización de valores se debe repetir para cada señal de entrada involucrada en nuestra lógica (A y B) con la finalidad de poder identificar si el comportamiento lógico implementado es el esperado para nuestro componente en términos de su señal de salida, adicional a la inicialización de valores lógicos es necesario establecer un intervalo de tiempo en donde se moverán nuestras señales de manera gráfica que nos proveerá una evidencia contundente del comportamiento que esperamos, o en su defecto una graficación incorrecta de la señal, que en términos de ingeniería representa una forma más sencilla de identificar un problema y resolverlo. El método para poder agregar un intervalo de tiempo dentro de nuestro ambiente se realiza mediante el cambio de valores en el campo de intervalo de tiempo de nuestra forma que se encuentra en la barra de Simulación de nuestro panel, como se representa a continuación:

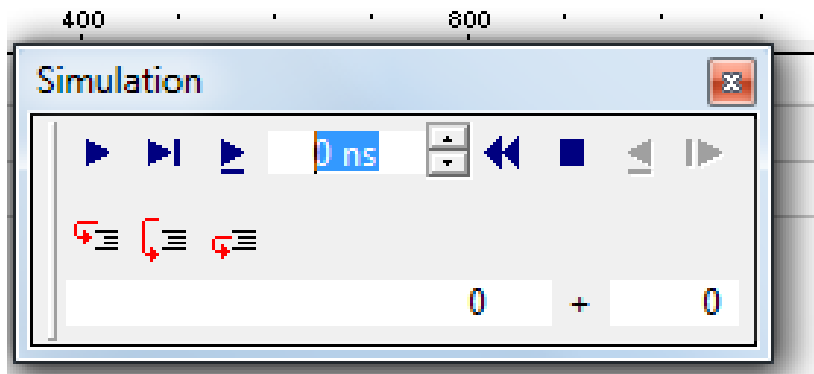


Figura 29. Barra de Simulación en una Waveform

El valor por defecto de nuestra barra de Simulación es “0 ns”, pero para poder observar un comportamiento lógico de nuestras señales, es necesario agregar un valor mayor a 0, (por ejemplo 1300 ns) y correr la simulación mediante *F5* (*Run*)

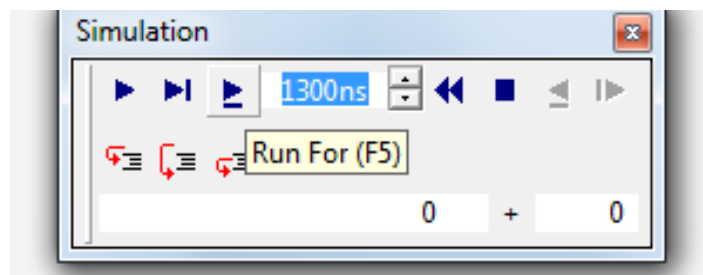


Figura 30. Ejecución de la simulación en 1300 ns

Después de la ejecución de la simulación podemos visualizar dentro del panel “Waveform” la representación gráfica de nuestras señales en un intervalo de tiempo de 1300 ns, como es esperado, el comportamiento que vemos en las dos señales de entrada (A y B), está graficado en el límite del reglón de la representación de cada señal, debido al valor de “1” previamente inicializado por medio de los *Stimulators...*, de igual manera, el comportamiento de la salida (Y) también es un “1” ejercitando la lógica implementada de una compuerta AND:

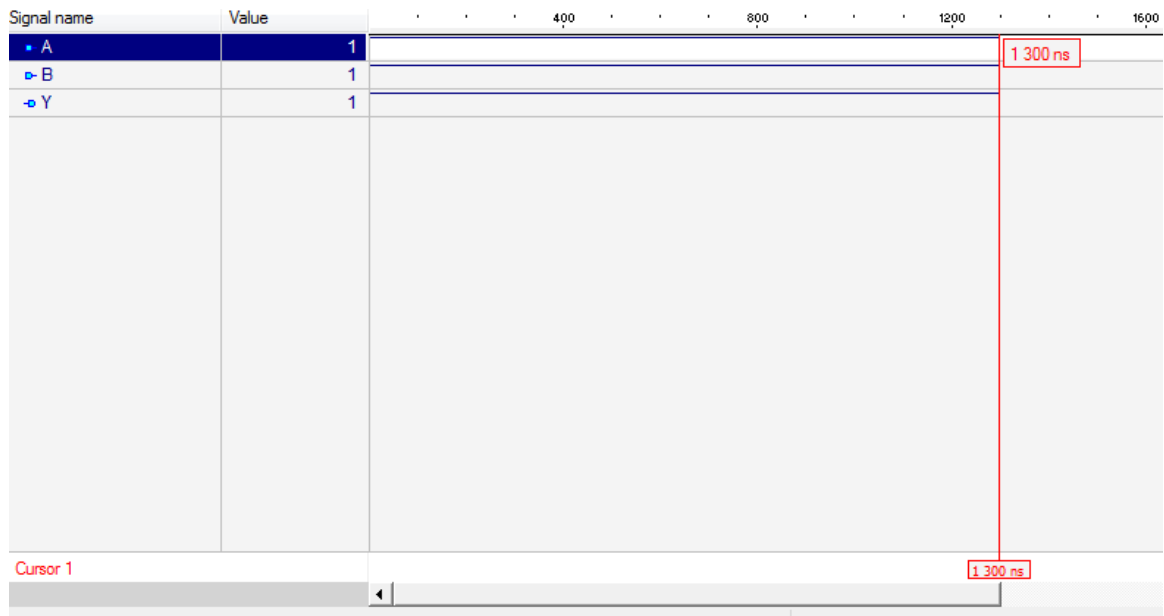


Figura 31. Representación gráfica de las señales en Waveform

Para efectos de comprobación del comportamiento de nuestra lógica, cambiaremos el valor de una de las señales de entrada con un “0” (A=“0”) y correremos nuevamente nuestra simulación agregando un nuevo intervalo de tiempo (1300 ns como se representa en la Figura 53), el resultado de la simulación será como a continuación se describe:

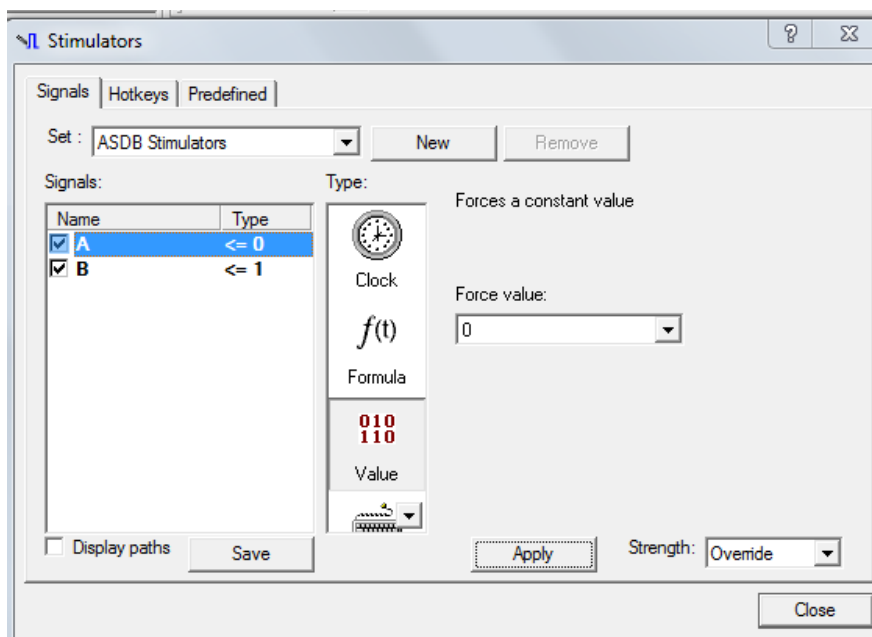


Figura 32. Cambio del valor de una de las señales de entrada para probar la lógica implementada

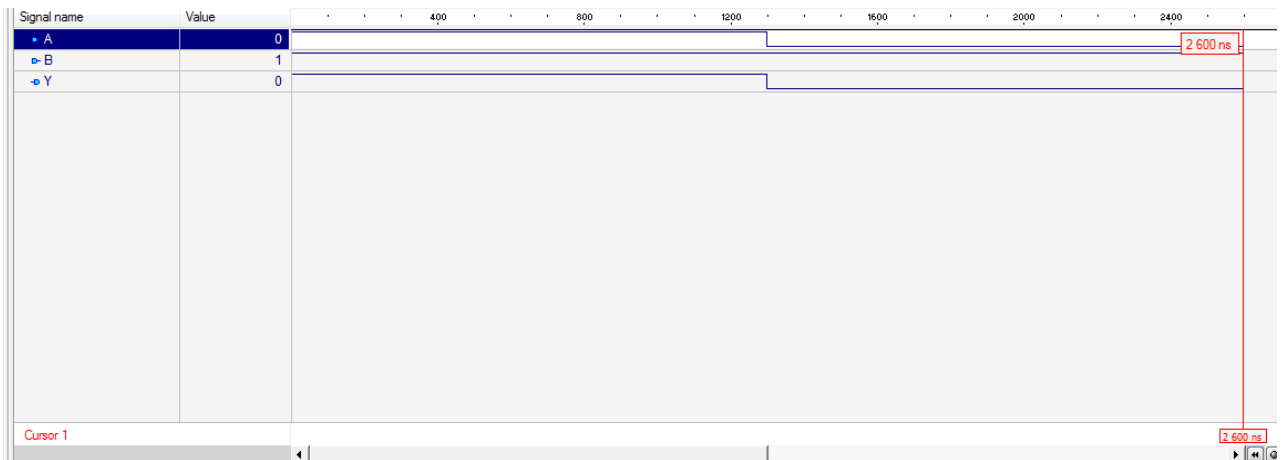


Figura 33. Gráfica del cambio de una de las señales de entrada que también afecta a la salida

Como producto del cambio de una de las señales de entrada ( $A=0$ ) podemos observar que el comportamiento en el reglón que representa dicha señal a partir de 1301 ns se ha graficado en la parte inferior, al mismo tiempo también se observa el mismo fenómeno pero en el reglón que representa la señal de salida (Y), prolongando el comportamiento hasta 2600 ns.

A través de este ejercicio pudimos comprobar el funcionamiento de nuestra lógica implementada a través de VHDL con la ayuda de un "Waveform", la gráfica de las señales nos indica los momentos en el tiempo en los que cada valor cambió durante la ejecución y cómo la señal de salida obedece a la lógica implementada, dándonos como resultado evidencia de una funcionalidad correcta en términos de simulación.

## Reconfiguración de dispositivos de hardware mediante VHDL

Hasta el momento hemos desarrollado lógica en VHDL que nos permite trabajar con señales que son configuradas de manera manual por el usuario, y donde también el intervalo de ejecución obedece a una manipulación directa, simulando un ambiente de prueba controlado. En términos de reconfiguración es preciso pensar en condiciones que no siempre resultan ser controladas y donde la mayoría de las veces, dichas condiciones pueden representar una

amenaza o un desequilibrio al sistema en ejecución. Tal podría ser el caso de una sistemas de hardware críticos que controlan el vuelo de cualquier avión comercial que todos los días transporta una cantidad importante de pasajeros alrededor del mundo, dichos sistemas son los encargados de tener control sobre los parámetros de vuelo en una aeronave, tales como temperatura, altitud, niveles de combustible, proximidad de aterrizaje y diagnostico de vuelo en términos generales. Todas estas condiciones son ejemplos de aspectos críticos que determinan si una aeronave puede iniciar o no, un proceso de viaje, si su desempeño en vuelo es correcto, si puede continuar con toda la ruta establecida al inicio de una travesía o si está en condiciones para poder lograr un aterrizaje.

Sabemos que para cualquier computadora y sus componentes de hardware, un sistema operativo es quién controla cada uno de los procesos lógicos que describen su funcionamiento mediante operaciones aritméticas y lógicas que lo hacen funcionar. Dentro de las operaciones aritméticas existen millones de cálculos que pueden ejecutarse al interior de la lógica de un sistema operativo y de manera consecutiva transformarse en señales lógicas o pulsos que estimulan a un dispositivo de hardware para que funcione de manera normal.

```
If ((Z1*CONST35)/v5) == v32 then --las variables Z1, v5, v32 y la constante  
{ --CONST35 provienen de una función  
  A=true; --previa en la lógica de control  
}  
else  
{  
  A=false; --A es la salida de la función por paso de  
} --parámetros
```

En el pseudocódigo descrito anteriormente se puede observar como tres variables y una constante previamente procesadas son comparadas mediante una rutina lógica para proveer un valor de salida a una variable booleana, este tipo de dato en una variable de salida es que comúnmente un dispositivo de hardware recibe para ejecución de alguna rutina física.

### **Consideraciones de diseño**

Cuando un fabricante decide iniciar la producción de un nuevo modelo de avión, junto a conocer la opinión de las compañías aéreas respecto a sus necesidades, analizar la evolución del precio del combustible, las restricciones ambientales impuestas por el ruido y otros problemas asociados con el medio ambiente -sin olvidar cualquier mejora que hubiera introducido en sus aviones la competencia-, se incorporan mejoras que hagan más seguro el avión.

Los aviones se diseñan y construyen para poder salir airosos de situaciones complicadas, tanto desde el punto de vista de maniobras propias del vuelo, como ante meteorología adversa. Hasta cierto límite, por supuesto. También hay que contar con la pericia de los pilotos.

Durante la fase de diseño, el primer condicionante con el que se trabaja es la seguridad. De ahí, que el compromiso sea garantizar que la probabilidad de que un solo fallo tenga efectos catastróficos para el avión sea de uno entre mil millones, es decir, extremadamente remota. De ese modo, prácticamente se garantiza que una situación de ese tipo no debería aparecer en toda la vida operativa de un modelo de avión, 25 ó 30 años, e incluso más. De todas formas, que ese sea el objetivo no quiere decir que en realidad se cumpla.

Los pilares sobre los que se asienta la fiabilidad de un avión son:

- La redundancia de sistemas críticos
  
- La robustez de la estructura, así como su resistencia frente a los efectos de la fatiga de los materiales y de tolerancia a los daños externos.

- La fiabilidad de funcionamiento de los sistemas.
- La efectividad de los sistemas de aviso y de detección de anomalías.
- El establecimiento de intervalos de mantenimiento programado, que garantice la detección a tiempo de cualquier problema.
- La mejora continua durante los años que dure la fabricación de cada modelo de avión.

Gracias a esta forma sistemática de trabajo los aviones actuales son muy fiables. De ahí, la evolución meteórica experimentada por la industria del transporte aéreo.

### **La cabina de los pilotos**

Una de las prioridades en el diseño de aviones se centra en la cabina de los pilotos y en la interacción de estos con los instrumentos y mandos de vuelo, lo que se conoce como ergonomía. Es la relación hombre-máquina.



*Figura 34. Cabina de un avión*

En este aspecto, el desarrollo de ordenadores, programas informáticos específicos y monitores de video, ha permitido sustituir los tradicionales instrumentos analógicos por pantallas multifunción y aumentar la fiabilidad de los sistemas, mejorando de ese modo la gestión de la información en cabina.

Siguiendo el principio de redundancia, cada avión se diseña en la actualidad de modo que, en caso de que alguno de sus equipos y sistemas falle, otro asuma sus funciones. Así, instrumentos de vuelo como los indicadores de velocidad y altitud, el horizonte artificial, los sistemas de comunicaciones y otros, se encuentran, incluso, por triplicado en la cabina de los pilotos. Además, entre otras mejoras llevadas a cabo se han sustituido numerosos avisos luminosos y acústicos por voces sintéticas -generalmente en inglés-, que llaman la atención de la tripulación sobre las incidencias que tienen lugar (Avión, 2013).

### **Sistemas en paralelo**

En una configuración en paralelo se precisa el funcionamiento de al menos una componente para que el sistema funcione. Se dice que las componentes son redundantes. La redundancia es uno de los métodos utilizados para mejorar la fiabilidad de un sistema (Wikipedia, 2013) y dentro de nuestra investigación presentamos las funciones estructura y de fiabilidad de avalan la continuidad de la funcionalidad provista por uno o varios dispositivos de hardware manipulados por la reconfiguración mediante el software. Para el caso de la función estructura, la ejecución secuencial de una funcionalidad deberá ser continua y dependiente de procedimientos previos que necesitan ser ejecutados en un orden lógico y congruente. En el mismo contexto, la función de fiabilidad permite una ejecución sin interrupciones (si es que la funcionalidad originalmente definida lo requiere), no importando el uso de elementos secundarios como pueden ser artefactos de respaldo o de emergencia que normalmente son re-configurados para ejecutar tareas de aplicaciones críticas.

La **función estructura** del sistema es:

$$X_S = 1 - \Phi(X_1, K, X_n) = \prod_{i=1}^n X_i = 1 - \prod_{i=1}^n (1 - X_i)$$



La **función de fiabilidad** de sistema es:

$$R_S = 1 - Q_S = 1 - P(X_1 = 0, K, X_n = 0) = 1 - ((1 - R_1) * \dots * (1 - R_n)) = \prod_{i=1}^n R_i = 1 - \prod_{i=1}^n (1 - R_i)$$

En un sistema en paralelo la componente más importante de cara a la fiabilidad es aquella que tiene la mayor fiabilidad de todas. La característica inherente al modelo paralelo se llama redundancia: Es decir existe más de un componente para desempeñar una función dada. La redundancia puede ser de dos clases:

Redundancia activa.- En este caso, todos los elementos redundantes están activos simultáneamente durante la misión.

Redundancia secuencial (llamada también stand-by o pasiva).-En esta ocasión, el elemento redundante sólo entra en juego cuando se le da la orden como consecuencia del fallo del elemento primario. Hasta que llega ese momento el elemento redundante ha permanecido inactivo, en reserva, pero ha podido estar:

- Totalmente inactivo (Ej.: La rueda de repuesto de un automóvil)
- Energizado total o parcialmente (Ej.: Un grupo electrógeno).

Los ingenieros de Boeing diseñan los aviones con dos cosas en mente: por un lado, diseñar para prevenir los fallos y por otro lado, incluir protecciones en el caso improbable de que se produzca un fallo. El objetivo es asegurar que ningún fallo reduce la seguridad o pone el avión en riesgo.

El proceso de validación contempla primero la división de los requisitos a nivel de avión en requisitos de los sistemas, subsistemas, y componentes; la validación de estos requisitos; su implementación y luego el análisis de los resultados. Esta aproximación rigurosa basada en “pilares fundamentales” asegura que los diseños cumplen con los requisitos y la función como estaba previsto.

Los sistemas de seguridad que forman parte del sistema eléctrico del 787 y que lo apoyan son diseñados para responder a varios escenarios de fallos e incluyen múltiples niveles de redundancia. Los controles de diseño buscan asegurar que ningún fallo por si solo pueda provocar un accidente; que los sistemas estén separados por función y por situación; que existen múltiples capas de redundancia, y que el avión tenga una red intensa de sistemas de reserva y de protección (Boeing, 2013).

Para efectos de nuestra investigación simularemos en Active-HDL la reconfiguración de dispositivos de hardware críticos para operar un avión en vuelo, tales como son: los motores de propulsión y el six pack de elementos de control de vuelo en la cabina del avión (Altímetro, Indicador de velocidad vertical o variómetro, Inclínómetro y coordinador de giro, Horizonte artificial, Indicador de Rumbos, ADF), donde para cada uno de estos elementos se trabaja con dispositivos iniciales que proyectan un comportamiento normal en un ambiente controlado, es decir, dos motores de avión activados a una propulsión estándar y 6 controles de navegación necesarios (4 de ellos críticos) para conocer los niveles de velocidad, temperatura, combustible y seguimiento de la ruta programada, para ello programaremos en código VHDL las entradas, salidas y procedimientos necesarios para poder habilitar la simulación:

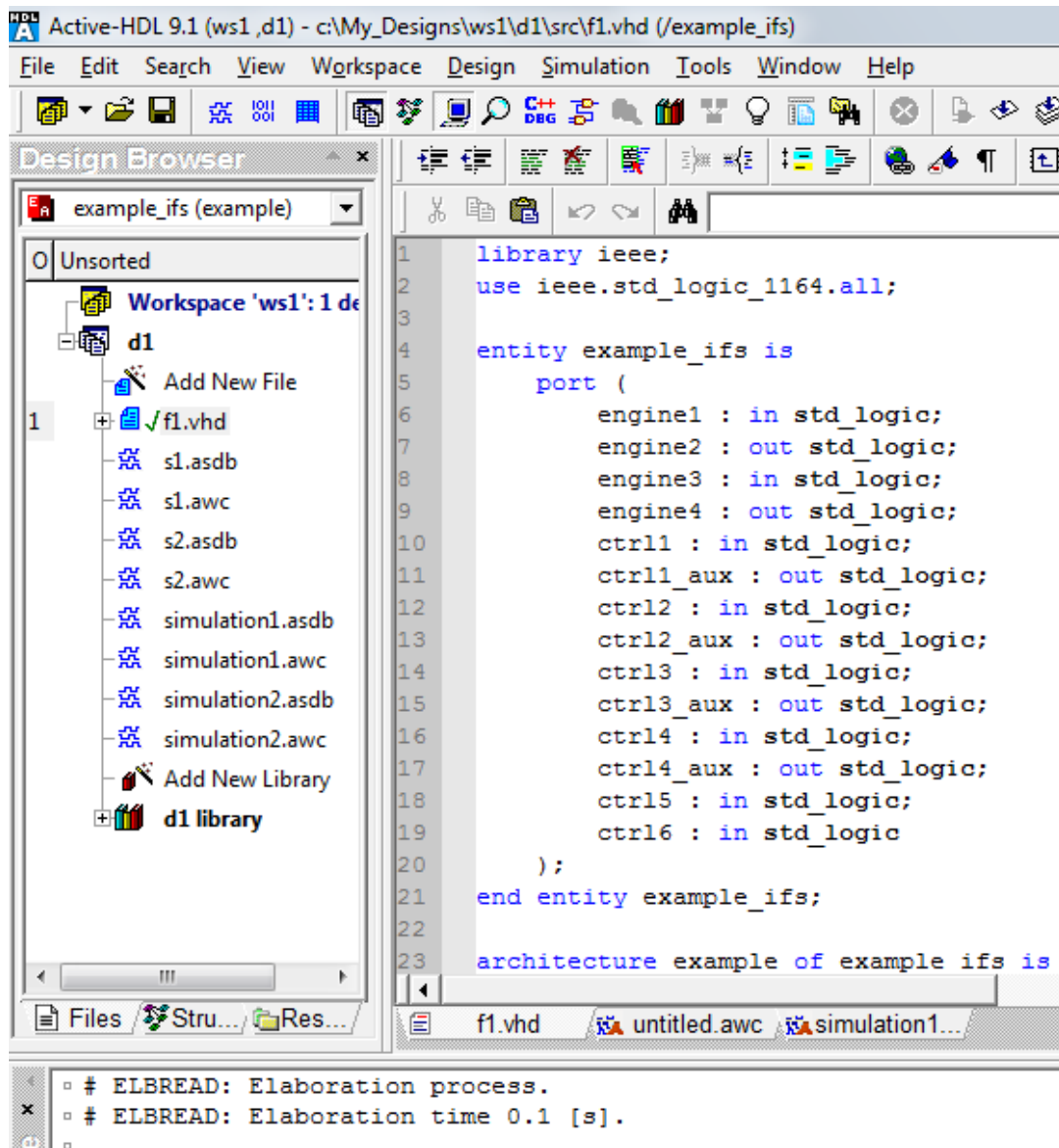


Figura 35. Declaración de entradas y salidas por cada elemento a simular

La primera parte de la codificación incorpora el uso de librerías por default y la definición de la entidad, teniendo como puertos de entrada a todos los elementos funcionando en un ambiente controlado (motores e indicadores de vuelo), y con definición de elementos de salida utilizaremos los dispositivos que deben ser activados en caso de una emergencia.

```

23 architecture example of example_ifs is
24 begin
25     process (engine1,engine3,ctrl11,ctrl12,ctrl13,ctrl14,ctrl15,ctrl16)
26 begin
27     if (engine1 = '0')then --activación de motor 2 y 4 en caso de emergencia
28         engine2 <= '1';
29     else
30         if (engine3 = '0')then
31             engine4 <= '1';
32         else
33             if ((engine1 = '0')and(engine3 = '0'))then
34                 engine2 <= '1';
35                 engine4 <= '1';
36             end if;
37         end if;
38     end if;
39     if (ctrl11 = '0') then --activación de controles (6 pack) en cabina del avión en caso de emergencia
40         ctrl11_aux <= '1';
41     else
42         if (ctrl12 = '0') then
43             ctrl12_aux <= '1';
44         else
45             if (ctrl13 = '0')then

```

```

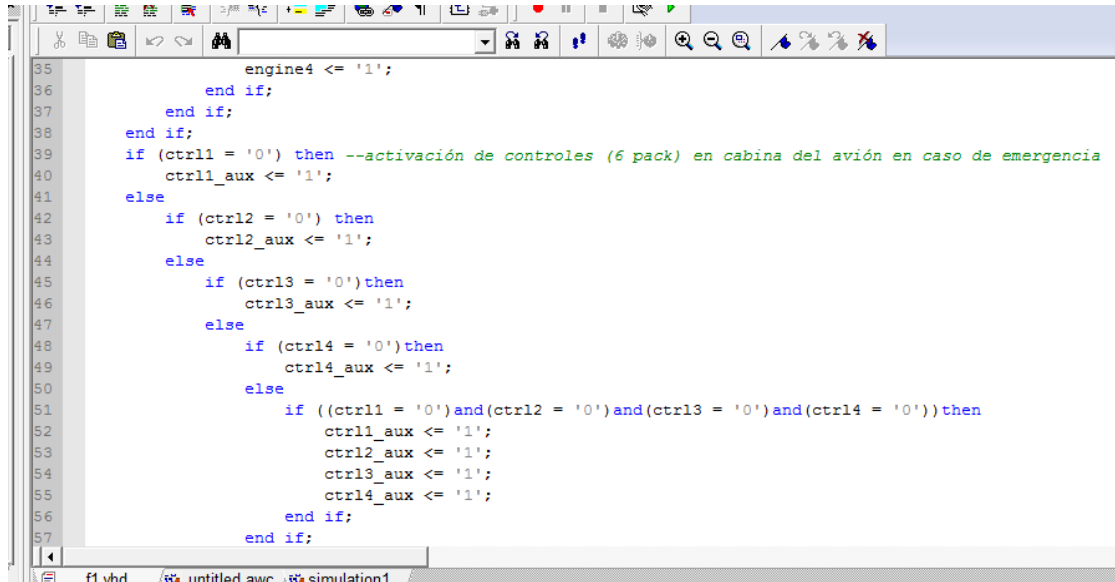
:ion process.
:ion time 0.1 [s].

```

Figura 36. Definición de la arquitectura a ser ejecutada

Cada elemento de entrada que corresponde a los dispositivos de funcionalidad crítica en vuelo, son ingresados mediante una definición de paso de parámetros a través de un proceso, con ese hecho, estamos en la facultad de poder crear un criterio de comparación para cada elemento de entrada y activar algún comportamiento de salida en caso de una emergencia.

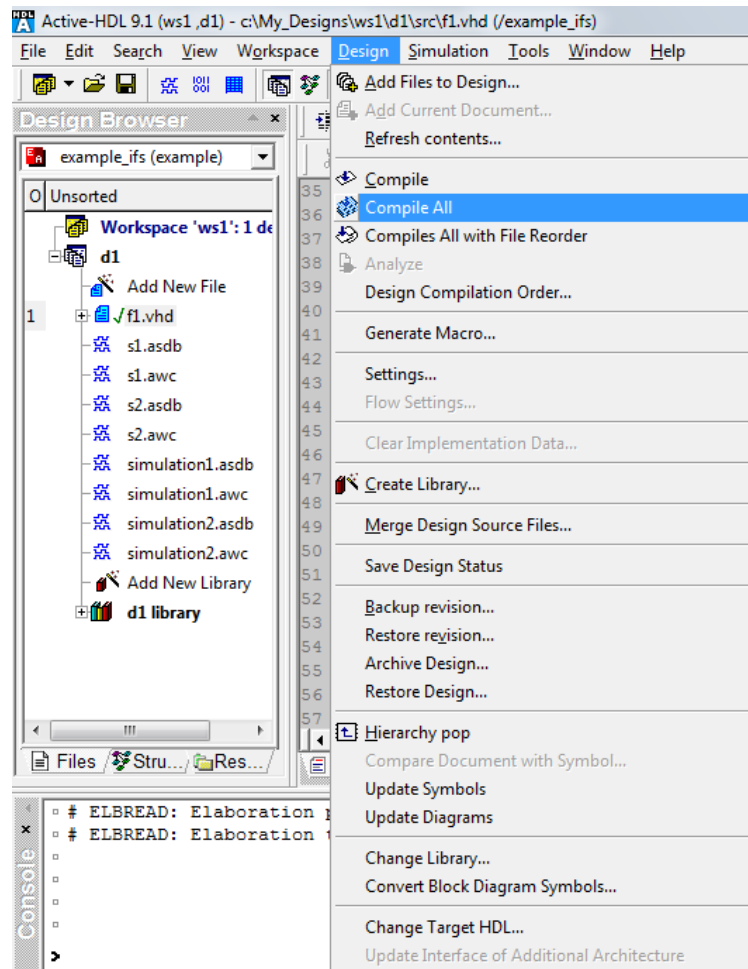
VHDL es un conjunto de instrucciones y métodos de configuración de dispositivos de hardware, pero visto desde el punto de vista de Ingeniería de Software, podemos decir que es un lenguaje de programación secuencial, en el cual, cada procedimiento y sub-procedimiento deberá ser anidado y perfectamente delimitado para poder hacer ejecuciones controladas y de acuerdo a los tiempos de corrida para poder reproducir la funcionalidad de algún evento, es por ello que dentro de nuestra simulación se crearon los métodos necesarios para poder representar la ejecución de los motores en una primera instancia, y una más para poder representar la ejecución de los controles de navegación:



```
35     engine4 <= '1';
36   end if;
37 end if;
38 end if;
39 if (ctrl1 = '0') then --activación de controles (6 pack) en cabina del avión en caso de emergencia
40   ctrl1_aux <= '1';
41 else
42   if (ctrl2 = '0') then
43     ctrl2_aux <= '1';
44   else
45     if (ctrl3 = '0') then
46       ctrl3_aux <= '1';
47     else
48       if (ctrl4 = '0') then
49         ctrl4_aux <= '1';
50       else
51         if ((ctrl1 = '0') and (ctrl2 = '0') and (ctrl3 = '0') and (ctrl4 = '0')) then
52           ctrl1_aux <= '1';
53           ctrl2_aux <= '1';
54           ctrl3_aux <= '1';
55           ctrl4_aux <= '1';
56         end if;
57       end if;
58     end if;
59   end if;
60 end if;
```

Figura 37. Programación secuencial utilizada en VHDL

Una vez que las señales y procedimientos lógicos de la simulación han sido implementados en VHDL podemos continuar con la compilación necesaria para verificar que nuestro código está libre de errores:



```

x  # Compile...
x  # Warning: DAGGEN_0523: The source is compiled without the -dbg switch. Line breakpoints, code coverage, and assertion debug will not be available.
x  # File: c:\My_Designs\ws1\di\src\f1.vhd
x  # Compile Entity "example_ifs"
x  # Compile Architecture "example" of Entity "example_ifs"
x  # Compile success 0 Errors 0 Warnings Analysis time : 0.5 [s]

```

Figura 38. Proceso de compilación de código VHDL

Después de haber obtenido una compilación libre de errores y “warnings”, el proceso de simulación puede ser iniciado, seleccionando las opciones *Simulation, Initialize Simulation* del menú principal:

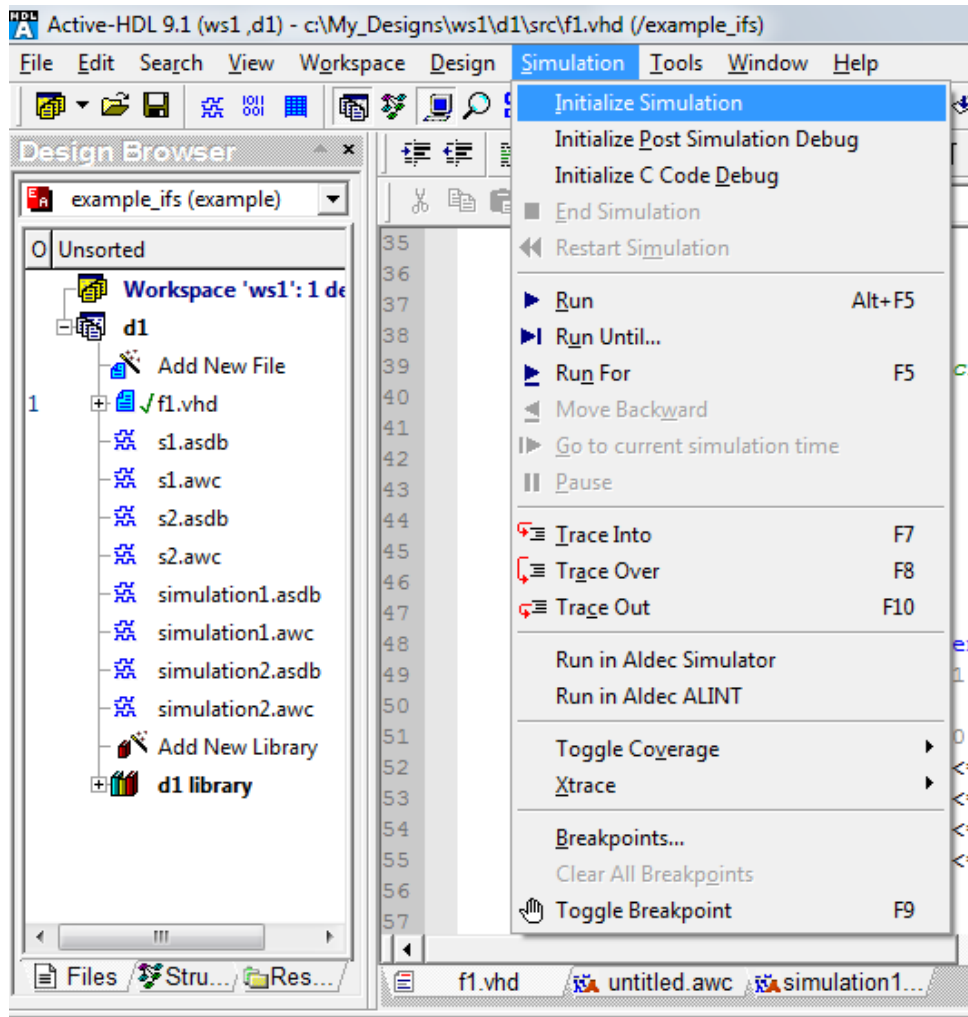


Figura 39. Inicialización de la Simulación

A continuación deberá iniciarse una vista de Waveform para poder dejar evidencia del proceso de simulación mediante señales:

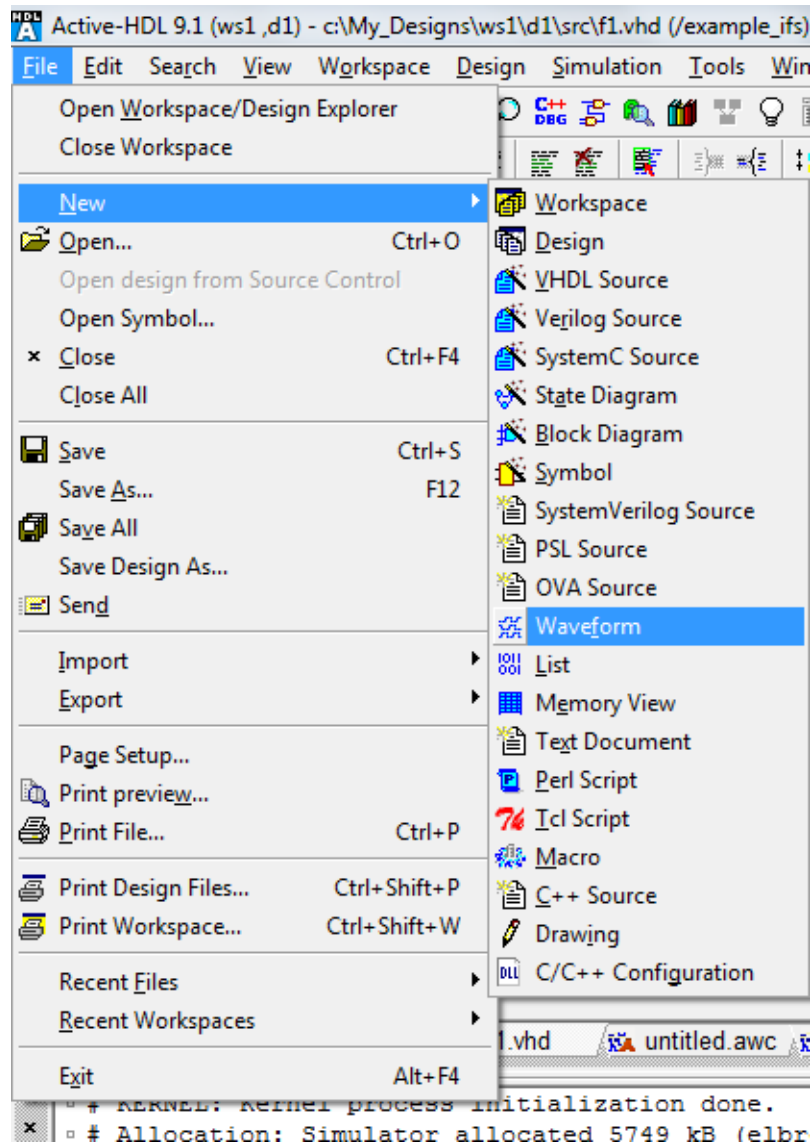


Figura 40. Generación de una vista de Waveform para poder mostrar evidencia del proceso de Simulación

Todos los elementos definidos en la entidad como entradas y salidas deberán ser llevados a la zona de definición de señales para la Simulación y posteriormente, cada señal de entrada, deberá ser inicializada mediante la opción de *Stimulators*:



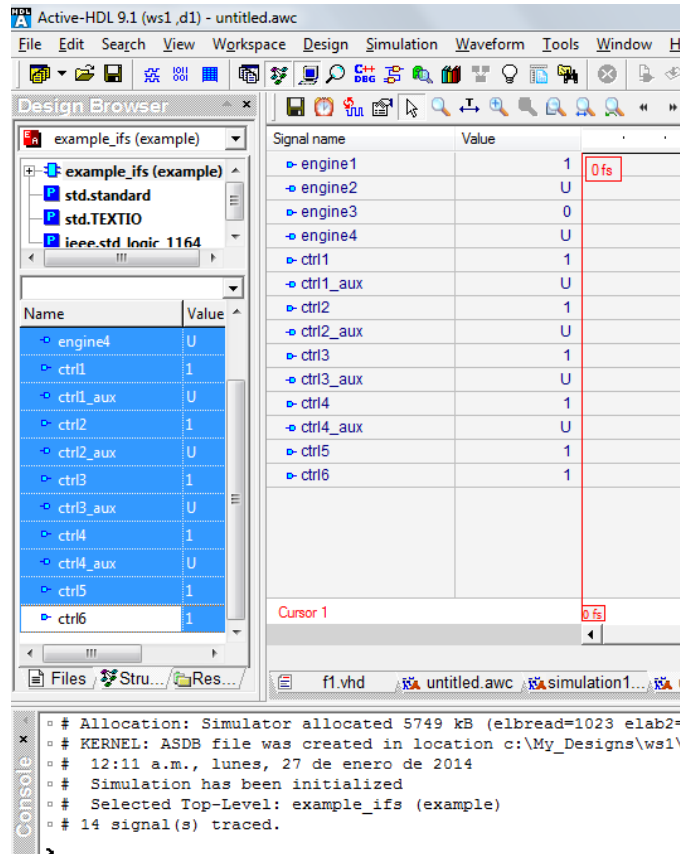


Figura 41. Señales de entrada y salida definidas en la entidad de la lógica

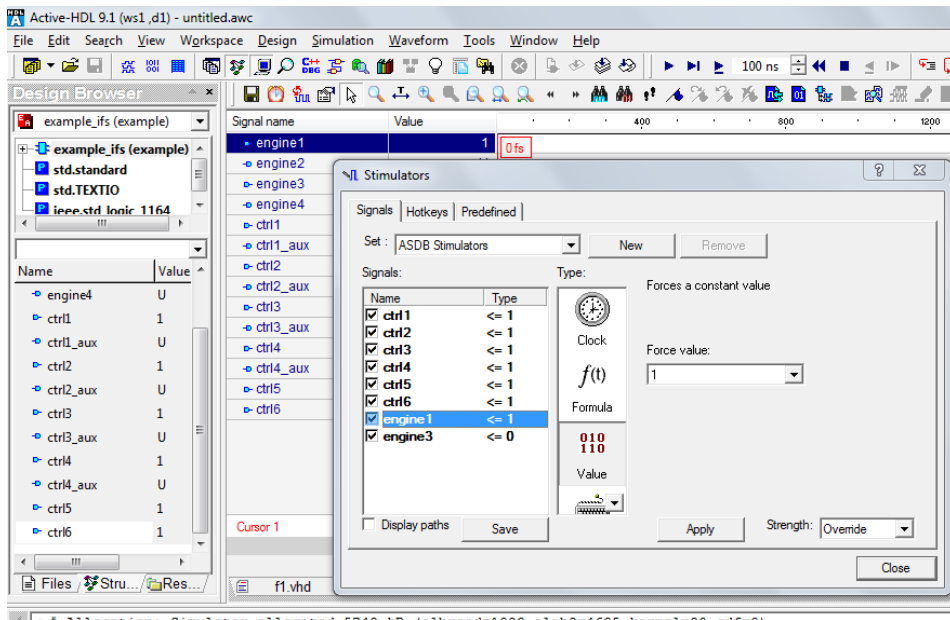


Figura 42. Inicialización de señales mediante la opción "Stimulators"

Las señales de entrada serán configuradas describiendo un comportamiento normal en vuelo, con el objetivo de saber si en condiciones normales, los elementos de emergencia están funcionando. Una vez concluido el proceso de configuración de señales en condiciones normales de simulación en vuelo, esta es ejecutada obteniendo los siguientes resultados en términos de señales lógicas:

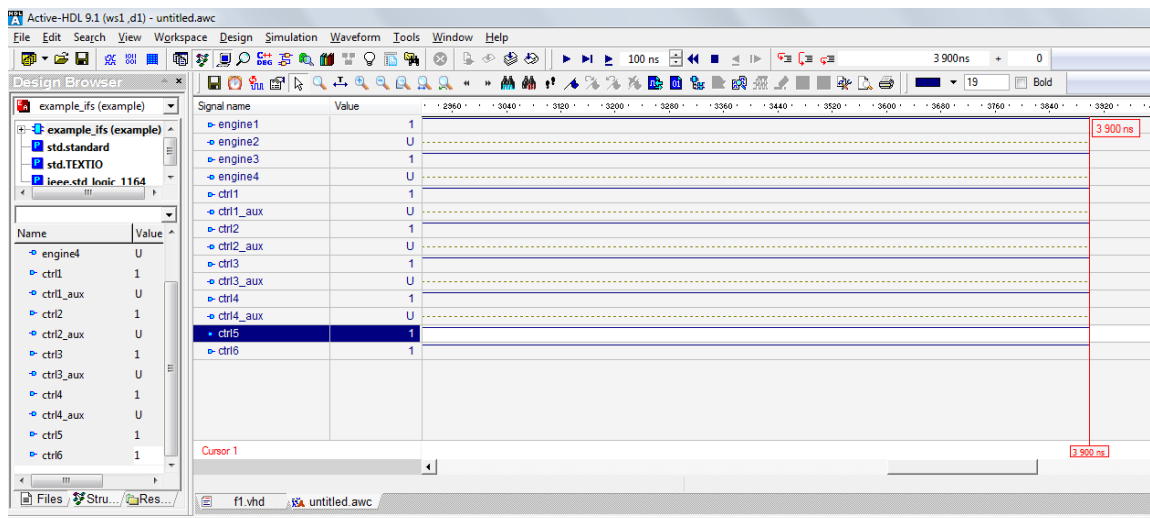


Figura 43. Ejecución de la funcionalidad de vuelo con una Simulación ejecutada a 3900 ns.

Las señales de entrada para motores “engine1” y “engine3” fueron configuradas para simular un comportamiento de vuelo normal a 3900 ns. ejecutando un funcionamiento esperado al no tener que activar ninguna señal que reconfigurara el funcionamiento para “engine2” y “engine4” (motores de emergencia), es por ello, que existen líneas punteadas durante todo el proceso de ejecución y los valores de salida en cada señal, no han sido modificados “U”. En el caso de las señales que corresponden a los controles del 6 pack para el vuelo: “ctrl1”, “ctrl2”, “ctrl3”, “ctrl4”, “ctrl5”, “ctrl6” también han sido configurados con valores iniciales que simulan un comportamiento normal. Como en el caso de los motores de emergencia que no fueron activados, también podemos observar el comportamiento de las señales que activarían los controles de emergencia: “ctrl1\_aux”, “ctrl2\_aux”, “ctrl3\_aux” y “ctrl4\_aux” con valores de “U” y líneas punteadas a lo largo de la ejecución, debido a que

ninguno de ellos ha sido activado debido al correcto funcionamiento de los elementos de control inicial. Cabe aclarar que dentro de la lógica de respaldo, sólo han sido considerados 4 de los 6 controles de navegación, y esto es porque sólo cuatro de ellos representan un aspecto crítico para la navegación: Altímetro, Indicador de velocidad vertical o variómetro, Inclínómetro y coordinador de giro y Horizonte artificial, mientras que el Indicador de Rumbos y el ADF pueden ser sustituidos en una situación crítica, para efectos de comprensión del documento se optó solamente por simular los primeros cuatro.

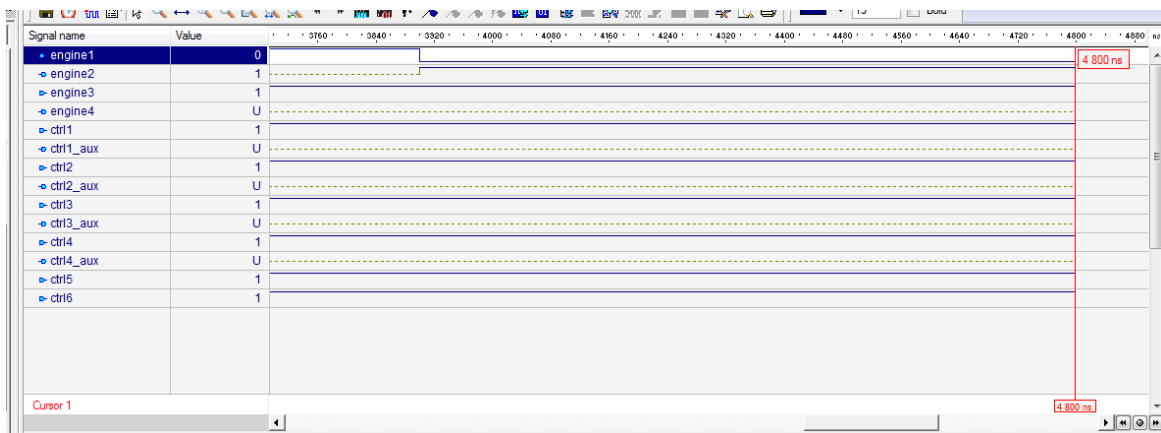


Figura 44. Inserción de una falla en el motor 1 y la activación en tiempo real del motor 2

Para poder validar el concepto de redundancia en la seguridad de nuestro modelo, se ha insertado una señal de falla en primer motor “engine1” y podemos confirmar el correcto funcionamiento del mecanismo de seguridad, observando como se ha activado el motor de emergencia “engine2” al recibir un fallo en vuelo (cambio en las señales de prueba en 3900 ns.)

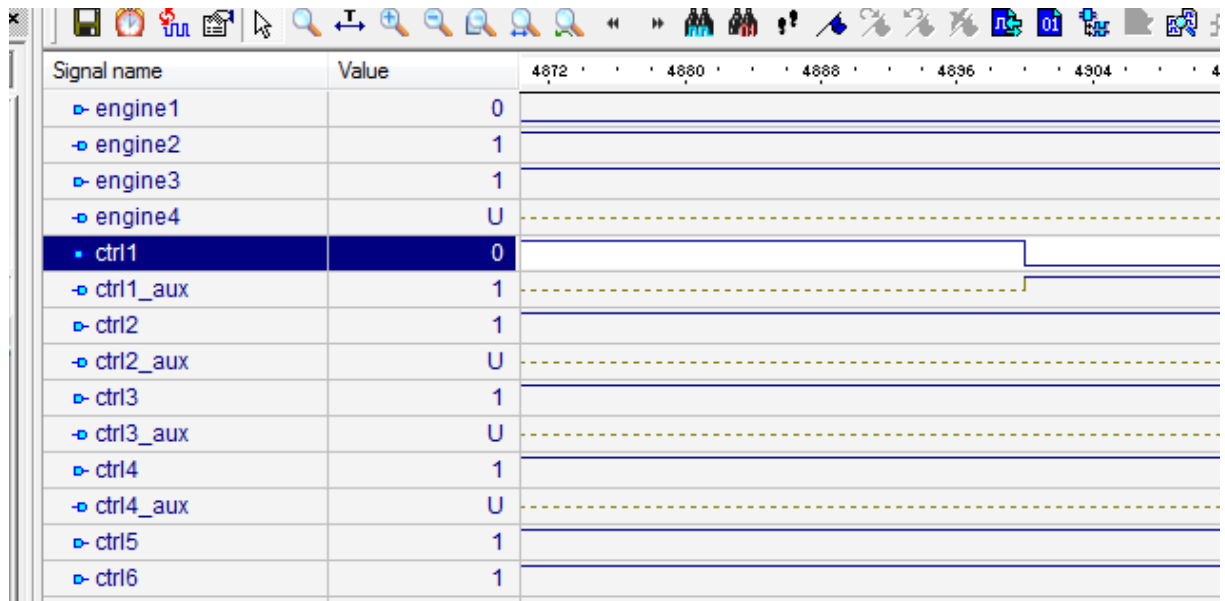


Figura 45. Inserción de una falla en uno de los controles principales

En el caso de los controles de navegación, también se ha insertado una falla y en el tiempo de simulación de 5000 ns podemos observar como la señal del Altímetro “ctrl1” recibe una alteración a su comportamiento normal, por lo cuál, la señal que activa el funcionamiento de un Altímetro auxiliar, es ejecutada, logrando con esto, la comprobación de que en VHDL podemos codificar el comportamiento de cualquier dispositivo mediante manipulación de lógica de cada señal que lo compone.

## IV. RESULTADOS Y DISCUSION

Como resultado de esta investigación, podemos concluir que dentro de nuestro mundo actual existen una gran cantidad de dispositivos de hardware que son configurados al momento de su fabricación por medio de un software embebido inherente al mismo dispositivo, que en la mayoría de los casos funciona bajo condiciones normales de trabajo y orientado a la tarea para la cual ha sido diseñado, pero ¿qué pasa? Cuando estas condiciones normales cambian y la tarea inicial tiene algún tipo de variación o alteración a su funcionamiento, en términos prácticos, el dispositivo puede ser desechado y uno nuevo vendrá a sustituirle, sin embargo en aplicaciones de funcionalidad crítica, difícilmente se puede proseguir de esta manera, es por ello que se ha estudiado la alternativa de reconfigurar ese hardware que anteriormente no podía ser re-utilizado o activado en tiempo real para propósitos de un nuevo uso, con ello, la reutilización de la implementación lógica de un artefacto es una opción tangible y a tan sólo unas líneas de código en VHDL, Lo más relevante de este trabajo es mostrar la relación estrecha que existe entre el estudio de la ingeniería de software y el estudio de hardware como campo de aplicación, que estimulado por la codificación y la configuración automática, permite la interacción de las máquinas y sus dispositivos casi de manera inteligente y en función de la conveniencia de procesos y tareas que les han sido asignadas o para las que fueron creadas, hoy en día cualquier dispositivo de hardware puede ser re-configurado, las implementaciones lógicas que antes parecían imposibles de re-codificar por su complejidad de arquitectura y de lógica, hoy en día toman un rumbo de certidumbre para nuevas aplicaciones y potencialización de recursos de hardware a través del software, VHDL sobre FPGAs es solamente una opción para poder hacerlo, pero este conocimiento permite proponer nuevas ideas y posibilidades en un mundo en donde los dispositivos y sus controles de software tienen cada vez más aplicación y propiedad de administración de recursos de cualquier tipo.

## LITERATURA CITADA

- [1] Christophe Bobda, "Introduction to Reconfigurable Computing" Architectures, algorithms and applications, 2007.
- [2] K. Compton, S. Hauck, "An introduction to Reconfigurable Computing" Northwestern University Evanston, IL USA & University of Washington Seattle, WA USA, 2007.
- [3] Heitor S. Lopes, Carlos R. Erig Lima and Norton J. Murata "A configware approach for high-speed parallel Analysis of genomic data", 2007
- [4] P. Banerjee, N. Shenoy, A. Choudhary, S. Hauck, C. Bachmann, M. Haldar, P. Joisha, A. Jones, A. Kanhare, A. Nayak, S. Periyacheri, M. Walkden, D. Zaretsky "A MATLAB Compiler For Distributed, Heterogeneous, Reconfigurable Computing Systems" 2000 IEEE Symposium on Field-Programmable Custom Computing Machines
- [5] P. Banerjee et al, "MATCH: A MATLAB Compiler for Configurable Computing Systems." Technical Report, Center for Parallel and Distributed Computing, Northwestern University, Aug. 1999, CPDC-TR-9908-013
- [6] Eduardo Ros, Francisco José Pelayo, Alberto Prieto y Begoña del Pino, "Ingeniería Neuromórfica: El papel del hardware Reconfigurable" II Jornadas sobre Computación Reconfigurable y Aplicaciones, JCRA 2002.
- [7] S. A. Ito and L. Carro, "A comparison of microcontrollers targeted to FPGA-based embedded applications", Proc. IEEE 13th Symp. Integrated Circuits and Systems Design, 2000.
- [8] J. Becker and R. Hartenstein, Configware and morphware going mainstream, *J. Syst.* 2000

- [9] K. Compton, S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software", ACM Computing Surveys, Vol. 10, No. 3, June 2002.
- [10] D. Buell, J.M. Arnold, W.J. Kleinfelder, Splash 2: FPGAs in a Custom Computing Machine. IEEE Computer Society Press, 1996.
- [11] P. Bertin, Memoires Actives Programmables: Conception, Realisation et Programmation, PhD thesis, Universite Paris 7, 1993.
- [12] C. Ebeling, D.C. Cronquist, P. Franklin, "RaPiD—Reconfigurable Pipelined Datapath", Field Programmable Logic Conference, 1996.
- [13] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, R. Taylor, "PipeRench: A Reconfigurable Architecture and Compiler", 2000.
- [14] J.R. Hauser, J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor", Int'l Symp. FCCM, April 1997.
- [15] T. Miyamori, K. Olukotun, "REMARC: Reconfigurable Multimedia Array Coprocessor", Proc. Int'l Symp. FPGA, February 1998.
- [16] Z.A. Ye, A. Moshovos, S. Hauck, P. Banerjee, "CHIMAERA: A High-Performance Architecture with a Tightly-Coupled Reconfigurable Functional Unit", Proc. Int'l Symp. Computer Architecture, June 2000.
- [17] R.D. Wittig, P. Chow, "OneChip: An FPGA Processor With Reconfigurable Logic", Int'l Symp. FCCM, April 1996.
- [18] R. Allen, D. Gajski, "The Case for C/C++ Hardware Design", EEDesign, June 9, 2000.
- [19] T. Callahan, J.R. Hauser, J. Wawrzynek, "The Garp Architecture and C Compiler", IEEE Computer, April 2000.

[20] E. Darnell , Y. Li, T. Callahan, , R. Harr, U. Kurkure, J. Stockwood, "Hardware-Software Co-Design of Embedded Reconfigurable Architectures", Design Automation Conference, 2000.

[21] G. Brebner, "A Virtual Hardware Operating System for the Xilinx XC6200", FPL 1996.

[22] <http://es.scribd.com/doc/106578918/Active-Hdl>, 2014

[23] <http://www.hispaviacion.es/articulos/jorgeavion.html>, 2014

[24] [http://es.wikipedia.org/wiki/Fiabilidad\\_de\\_sistemas](http://es.wikipedia.org/wiki/Fiabilidad_de_sistemas), 2014

[25] <http://www.boeing.es/Sala-de-Prensa/EI-787-Dreamliner/EI-sistema-Elctrico-del-Dreamliner-787>, 2014