****

**UNIVERSIDAD TECNOLÓGICA CENTROAMERICANA.**

**FACULTAD DE INGENIERÍA.**

**PROYECTO DE GRADUACIÓN.**

**ARCHSIM: UNA HERRAMIENTA PARA EL DISEÑO Y LA SIMULACIÓN DE HARDWARE.**

**PREVIO A LA OBTENCIÓN DEL TÍTULO**

**INGENIERO EN SISTEMAS COMPUTACIONALES.**

**PRESENTADO POR:**

**21011286 NÉSTOR ALEJANDRO BERMÚDEZ SARMIENTO**

**ASESOR: ING. CÉSAR ORELLANA.**

**CAMPUS SAN PEDRO SULA**

**ABRIL, 2014**

**Autorización.**

*AUTORIZACIÓN DEL AUTOR(ES) PARA LA CONSULTA, LA REPRODUCCIÓN PARCIAL O TOTAL, Y PUBLICACIÓN ELECTRÓNICA DEL TEXTO COMPLETO DE TESIS DE GRADO.*

Señores

CENTRO DE RECURSOS PARA EL APRENDIZAJE Y LA INVESTIGACIÓN (CRAI)   
San Pedro Sula

Estimados Señores:

La presentación del documento de tesis forma parte de los requerimientos y procesos establecidos de graduación para alumnos de pregrado de UNITEC.

Yo, Néstor Alejandro Bermúdez Sarmiento, de San Pedro Sula autor del trabajo de grado titulado: ArchSim: una herramienta para la definición y simulación de hardware., presentado y aprobado en el año 2014, como requisito para optar al título de Profesional de Ingeniero En Sistemas Computacionales, autorizo a:

Las Bibliotecas de los Centros de Recursos para el Aprendizaje y la Investigación (CRAI) de la Universidad Tecnológica Centroamericana (UNITEC), para que, con fines académicos, pueda libremente registrar, copiar y usar la información contenida en él, con fines educativos, investigativos o sociales de la siguiente manera:

Los usuarios puedan consultar el contenido de este trabajo de grado en la salas de estudio de la biblioteca y la página Web de la universidad.

Permita la consulta y la reproducción, a los usuarios interesados en el contenido de este trabajo, para todos los usos que tengan finalidad académica, ya sea en formato CD o digital desde Internet, Intranet, etc., y en general para cualquier formato conocido o por conocer.

De conformidad con lo establecido en el artículo 19 de la Ley de Derechos de Autor y de los Derechos Conexos; los cuales son irrenunciables, imprescriptibles, inembargables e inalienables.

Es entendido que cualquier copia o reproducción del presente documento con fines de lucro no está permitida sin previa autorización por escrito de parte de los principales autores.

En fe de lo cual, se suscribe la presente acta en la ciudad de San Pedro Sula a los cuatro días del mes de Marzo de dos mil catorce

[Fecha de la defensa]

Néstor Alejandro Bermúdez Sarmiento

21011286

**Hoja De Firmas.**

Los abajo firmantes damos fe, en nuestra posición de miembro de Terna, Asesor y/o Jefe Académico y en el marco de nuestras responsabilidades adquiridas, que el presente documento cumple con los lineamientos exigidos por la Facultad de Ingeniería y los requerimientos académicos que la Universidad dispone dentro de los procesos de graduación.

Ing. César Orellana.

Asesor UNITEC.

|  |  |  |
| --- | --- | --- |
| Ing. X o Arq. X  Miembro de Terna | Ing. X o Arq. X  Miembro de Terna | Ing. X o Arq. X  Miembro de Terna |

|  |  |
| --- | --- |
| Ing. X o Arq. X  Jefe Académico de X | UNITEC | Ing. X o Arq. X  Decano de Facultad de Ingeniería UNITEC |

**Dedicatoria Y Agradecimiento.**

A Dios.

Por haberme permitido llegar hasta donde he llegado y ayudarme a mantener la constancia y motivación necesaria para culminar este viaje.

A mis padres.

Por el apoyo incondicional, por los sacrificios que fueron necesarios para continuar mis estudios y sobre todo por el amor e interés brindados.

A Banco Central de Honduras.

Porque sin su apoyo económico no hubiese comenzado lo que hoy estoy terminando.

A mis maestros.

Por todo el conocimiento brindado, en particular al Ing. Rogger Vásquez e Iván Deras por su confianza, conocimiento y experiencias compartidas y la Licda. Mirtha Coto por tolerarme y ayudarme siempre que lo necesité.

A mis muchos asesores.

Ing. César Orellana, Ing. Carlos Castro, Ing. Jared Ocampo e Ing. Iván Deras. Cada uno me asesoró en cierto aspecto y momento, y fue gracias a ello que este trabajo fue terminado exitosamente.

**Resumen Ejecutivo**

El presente documento se basa en la investigación realizada para llevar a cabo una aplicación con el objetivo de facilitar el diseño y la simulación del comportamiento de componentes de hardware, que, a su vez, se espera mejore la comprensión de conceptos de y funcionamiento de estructuras más complejas como los procesadores.

Actualmente, no hay intérpretes de código libre para el lenguaje de descripción de hardware a utilizar, Verilog, por lo que el desarrollo del mismo es un elemento fundamental de este proyecto y de la investigación como tal.

Por otro lado, la aplicación permitirá la interconexión entre componentes definidos por el usuario, su ejecución y depuración a través de la ventana de simulación.

Para concluir, se encontrará un estudio realizado con alumnos y exalumnos de Ingeniería en Sistemas Computacionales para evaluar varios aspectos de aplicación desarrollada.

[I. Introducción 1](#_Toc384821707)

[II. Objetivos. 2](#_Toc384821708)

[**2.1** **Objetivo General** 2](#_Toc384821709)

[**2.2** **Objetivos Especificos.** 2](#_Toc384821710)

[III. Descripción Del Proyecto. 3](#_Toc384821711)

[**3.1** **Descripción General.** 3](#_Toc384821712)

[**3.2** **Alcance.** 3](#_Toc384821713)

[**3.3** **Actividades A Desarrollar.** 4](#_Toc384821714)

[**3.4** **Recursos A Utilizar.** 5](#_Toc384821715)

[IV. Marco Teórico. 6](#_Toc384821716)

[**4.1** **Sobre La Simulación.** 6](#_Toc384821717)

[4.1.1 Conceptos. 6](#_Toc384821718)

[4.1.2 SWING FRAMEWORK. 6](#_Toc384821719)

[**4.2** **DISEÑO DE HARDWARE.** 8](#_Toc384821720)

[4.2.1 LENGUAJES DE DESCRIPCIÓN DE HARDWARE. 9](#_Toc384821721)

[**4.3** **SOBRE ELECTRÓNICA.** 12](#_Toc384821722)

[4.3.1 CONCEPTOS. 12](#_Toc384821723)

[4.3.2 TABLAS DE VERDAD. 14](#_Toc384821724)

[4.3.3 SIMBOLOGÍA. 16](#_Toc384821725)

[**4.4** **SOBRE EL DESARROLLO.** 18](#_Toc384821726)

[4.4.1 METODOLOGÍA. 18](#_Toc384821727)

[4.4.1.1 METODOLOGÍAS AGILES. 18](#_Toc384821728)

[4.4.1.2 Scrum. 19](#_Toc384821729)

[4.4.2 SOBRE EL COMPILADOR. 22](#_Toc384821730)

[4.4.2.1 JFlex 24](#_Toc384821731)

[4.4.2.2 Cup 25](#_Toc384821732)

[4.4.3 OTRAS HERRAMIENTAS Y TECNOLOGÍAS. 26](#_Toc384821733)

[4.4.3.1 RSyntaxTextArea 26](#_Toc384821734)

[4.4.3.2 XML. 27](#_Toc384821735)

[**4.5** **ANTECEDENTES.** 29](#_Toc384821736)

[4.5.1 XILINX. 29](#_Toc384821737)

[4.5.2 ALTERA. 30](#_Toc384821738)

[V. Metodología. 31](#_Toc384821739)

[**5.1** **CRONOLOGÍA DE TRABAJO.** 31](#_Toc384821740)

[**5.2** **METODO ESTADÍSTICOS PARA EVALUACIÓN DE OBJETIVOS.** 40](#_Toc384821741)

[VI. Resultados Y Análisis. 41](#_Toc384821742)

[VII. Conclusiones. 42](#_Toc384821743)

[VIII. Recomendaciones. 43](#_Toc384821744)

[IX. Bibliografía 45](#_Toc384821745)

[X. Anexos. 46](#_Toc384821746)

[**10.1** **ANEXO A: CARACTERÍSTICAS DE VERILOG ACEPTADAS.** 46](#_Toc384821747)

[**10.2** **ANEXO B: TABLAS DE VERDAD DE COMPUERTAS LÓGICAS.** 67](#_Toc384821748)

[**10.3** **ANEXO C: ESTRUCTURAS XML.** 70](#_Toc384821749)

[10.3.1 ARCHIVO DE CONFIGURACIONES. 70](#_Toc384821750)

[10.3.2 ARCHIVO DE DEFINICIÓN DE MÓDULO. 71](#_Toc384821751)

[10.3.3 ARCHIVO DE METADATA. 72](#_Toc384821752)

[**10.4** **ANEXO D: ENCUESTA.** 73](#_Toc384821753)

[**10.5** **ANEXO E: MANUAL DE USUARIO.** 75](#_Toc384821754)

[10.5.1 PANTALLA PRINCIPAL. 75](#_Toc384821755)

[10.5.2 VENTANA DE DISEÑO. 76](#_Toc384821756)

[10.5.3 VENTANA DE SIMULACIÓN. 81](#_Toc384821757)

[10.5.4 MONITOR DE VARIABLES. 88](#_Toc384821758)

[10.5.5 EDITOR DE CONFIGURACIONES. 90](#_Toc384821759)

**Índice De Imágenes.**

[**Ilustración 1. Diagrama de clases JComponent** 7](#_Toc384819767)

[**Ilustración 2. Jerarquía de clases de JFrame** 8](#_Toc384819768)

[**Ilustración 3. Niveles de abstracción en el diseño de descripción de hardware.** 11](#_Toc384819769)

[**Ilustración 4. Imagen comparativa entre sintaxis de Verilog y VHDL** 12](#_Toc384819770)

[**Ilustración 5. Circuito Integrado.** 13](#_Toc384819771)

[**Ilustración 6. Símbolo IEC para una compuerta Y** 17](#_Toc384819772)

[**Ilustración 7. Símbolo ANSI para una compuerta Y** 17](#_Toc384819773)

[**Ilustración 8. Símbolo IEC para una compuerta O** 17](#_Toc384819774)

[**Ilustración 9. Símbolo ANSI para una compuerta O** 17](#_Toc384819775)

[**Ilustración 10. Símbolo IEC para una compuerta Nor** 17](#_Toc384819776)

[**Ilustración 11. Símbolo ANSI para una compuerta Nor** 17](#_Toc384819777)

[**Ilustración 12. Símbolo IEC para una compuerta Not** 17](#_Toc384819778)

[**Ilustración 13. Símbolo ANSI para una compuerta Not** 17](#_Toc384819779)

[**Ilustración 14. Símbolo IEC para una compuerta Nand** 18](#_Toc384819780)

[**Ilustración 15. Símbolo ANSI para una compuerta Nand** 18](#_Toc384819781)

[**Ilustración 16. Símbolo IEC para una compuerta Xnor** 18](#_Toc384819782)

[**Ilustración 17. Símbolo ANSI para una compuerta Xnor** 18](#_Toc384819783)

[**Ilustración 18. Símbolo IEC para una compuerta Xor** 18](#_Toc384819784)

[**Ilustración 19. Símbolo ANSI para una compuerta Xor** 18](#_Toc384819785)

[**Ilustración 20. Ejemplo de una pizarra de Scrum** 21](#_Toc384819786)

[**Ilustración 21. Captura de Pantalla de la pizarra de este proyecto** 21](#_Toc384819787)

[**Ilustración 22. Etapas del proceso de compilación.** 23](#_Toc384819788)

[**Ilustración 23. Representación de árbol de un archivo XML.** 28](#_Toc384819789)

[**Ilustración 24. Interfaz gráfica de ISim** 30](#_Toc384819790)

[**Ilustración 25. Interfaz de simulación de Quartus II** 30](#_Toc384819791)

[**Ilustración 26. Simbología de colores para agrupar tareas.** 31](#_Toc384819792)

[**Ilustración 27. Backlog de actividades a realizar en la fase I.** 32](#_Toc384819793)

[**Ilustración 28. Board de Trello al finalizar Fase I** 35](#_Toc384819794)

[**Ilustración 29. Backlog de Tareas a realizar en Fase 2** 36](#_Toc384819795)

[**Ilustración 30. Más tareas a realizar durante Fase 2** 37](file:///C:\Users\Nestor%20Bermudez\Documents\GitHub\GradProjectJava\VerdaderoMarcoTeorico.docx#_Toc384819796)

[**Ilustración 31. Opciones en el menú de Ayuda** 76](#_Toc384819797)

[**Ilustración 32. Ventana de diseño de componentes** 76](#_Toc384819798)

[**Ilustración 33. Diseño de componente: Opciones en File** 77](#_Toc384819799)

[**Ilustración 34. Diseño de componente: Opciones de Edit** 77](#_Toc384819800)

[**Ilustración 35. Diseño de componente: Opciones de Build** 77](#_Toc384819801)

[**Ilustración 36. Diseño de componente: Vista Previa** 78](#_Toc384819802)

[**Ilustración 37. Diseño de componente: definición de comportamiento usando Verilog** 79](#_Toc384819803)

[**Ilustración 38. Diseño de componente: Log de errores** 80](#_Toc384819804)

[**Ilustración 39. Ventana de simulación** 81](#_Toc384819805)

[**Ilustración 40. Ventana de simulación: Opciones de File** 81](#_Toc384819806)

[**Ilustración 41. Ventana de simulación: Opciones de Design** 82](#_Toc384819807)

[**Ilustración 42. Ventana de simulación: Opciones de Execution** 82](#_Toc384819808)

[**Ilustración 43. Ventana de simulación: acciones sobre componentes** 83](#_Toc384819809)

[**Ilustración 44. Ventana de simulación: selección en área** 84](#_Toc384819810)

[**Ilustración 45. Ventana de simulación: módulos definidos por el usuario** 85](#_Toc384819811)

[**Ilustración 46. Ventana de simulación: otras características** 86](#_Toc384819812)

[**Ilustración 47. Ventana de simulación: Monitor de Variables** 88](#_Toc384819813)

[**Ilustración 48. Selector de Variables** 88](file:///C:\Users\Nestor%20Bermudez\Documents\GitHub\GradProjectJava\VerdaderoMarcoTeorico.docx#_Toc384819814)

[**Ilustración 49. Monitor de Variables** 89](#_Toc384819815)

[**Ilustración 50. Editor de Configuraciones** 90](#_Toc384819816)

1. **Introducción**

Los lenguajes de descripción de hardware han existido desde 1960 y, en la actualidad, son fundamentales para empresas que se dedican al diseño de hardware o electrónica en general debido a que es mucho más barato y rápido hacer pruebas simuladas que manufacturar los componentes para probarlos.

El propósito de este trabajo es brindar un simulador de componentes de hardware con un objetivo diferente: mejorar la experiencia de diseño y simulación de los componentes y la comprensión de los mismos.

Para ello, el proceso se dividió en dos grandes partes: la interpretación del lenguaje de descripción de hardware elegido (Verilog) y la interfaz gráfica, cuya parte primordial es la ventana de simulación.

Originalmente se pensó en reemplazar la primera parte por un intérprete de Verilog de código libre, pero no se pudo encontrar ninguno, al menos no completamente funcional o compatible con las tecnologías que se utilizarían. Por esto se decidió desarrollar uno desde cero, que si bien no soporta toda la definición oficial de Verilog, soporta lo necesario para la simulación de componentes tan complejos como un procesador.

En este documento se pondrá en evidencia el proceso de desarrollo de la aplicación así como los fundamentos teóricos en los que se basa.

En las últimas páginas también se encontrará un análisis estadístico realizado con el propósito de evaluar si se cumplieron los objetivos planteados inicialmente.

Cabe mencionar que la aplicación desarrollada no es un reemplazo a herramientas existentes similares como ISE de Xilinx o Quartus II de Altera, debido a que estas cuentan con programas especializadas para la carga del componente en chips programables conocidos como FPGA; esta aplicación es un complemento y trata de brindar bondades que las anteriores no tienen como ser el Monitor de Variables y la posibilidad de simular la interacción entre componentes.

1. **Objetivos.**
   1. **Objetivo General**

Proveer una herramienta para simulación de diseño de hardware fácil de usar, que permita la definición y simulación de componentes de hardware y que permita la detección de errores en tiempo de simulación con la ayuda de un Monitor de Variables.

* 1. **Objetivos Especificos.**

1. Crear una aplicación integral que cuya alcance vaya desde el diseño de un nuevo componente de hardware hasta la simulación del mismo.
2. Crear un ambiente de simulación con dos modos:
   1. Un modo de tiempo real, donde la simulación inicia, se ejecuta y termina para luego analizar el estado final del sistema.
   2. Un modo paso a paso, donde la simulación se pausa luego de cada iteración para permitir el análisis en cualquier momento.
3. Implementar un Monitor de Variables para facilitar el análisis del comportamiento de un componente de hardware y mejorar la detección de errores en tiempo de simulación.
4. Analizar la retroalimentación del usuario final para determinar las fortalezas y deficiencias de la aplicación desarrollada.
5. **Descripción Del Proyecto.**
   1. **Descripción General.**

Ciencias de la Computación es un área de estudio desafiante debido a la complejidad y abstracción de muchos de sus tópicos.

Uno de los más difíciles es el estudio de la micro-arquitectura de una computadora. Como parte de la clase de Organización de Computadoras se busca entender el funcionamiento de un procesador, en donde se analizan sus partes y se modela uno haciendo uso de un lenguaje de descripción de hardware llamado Verilog mediante una IDE de Xilinx.

Esta IDE cuenta con un simulador de señales digitales llamado iSim que, hasta el momento, es la única forma de detección de errores con la que cuentan los alumnos cuando trabajan en su procesador.

Resumiendo el problema:

No se cuenta con una herramienta en la que se pueda implementar componentes para el diseño de hardware con ayuda gráfica y con métodos de detección de errores prácticos.

* 1. **Alcance.**

La aplicación contará con una interfaz gráfica para definir el diseño y comportamiento de las diferentes partes de un procesador (Instruction Memory, Register File, ALU, Data Memory) además de otros componentes electrónicos necesarios en el diseño de un procesador entre ellos, multiplexores y sumadores. Para hacer la modelación del comportamiento de cada componente será necesario hacer un intérprete de Verilog.

Estos componentes luego pasarán a formar parte de un toolbox. Los elementos del toolbox se podrán conectar entre ellos en una ventana de diseño para formar estructuras más complejas (un procesador, por ejemplo).

Finalmente se podrán ejecutar de dos posibles formas: Tiempo real (es una ejecución rápida donde lo que interesa es el resultado final) y Simulación (permitirá ir paso a paso por los elementos del diseño, ideal para la depuración de errores).

* 1. **Actividades A Desarrollar.**

1. Crear pantalla principal de la aplicación:
   1. Opción para crear nuevos componentes (ver inciso 2)
   2. Opción para unir componentes y ejecutarlos (ver inciso 3)
   3. Guardar/Cargar los componentes definidos.
2. . Crear interfaz para definir componentes (diseño y comportamiento).
3. Crear interfaz de diseño con formas básicas para utilizar.
4. Crear editor de texto para definir el comportamiento usando Verilog (una versión minimalista del mismo) con características básicas de autocompletación y resaltado de palabras clave, además de Copy &Paste y Undo/Redo.
5. Los elementos se podrán mover libremente y ser eliminados de la pantalla de diseño.
6. Crear interfaz para unir componentes y ejecutarlos.
   1. Ejecución en modo “Tiempo Real”
   2. Ejecución en modo “Simulación”
7. Creación del toolbox donde estarán los componentes para su reutilización.
8. Conversión de Verilog a una representación intermedia para utilizarse en la ejecución.
   1. **Recursos A Utilizar.**

Todos los recursos utilizados son herramientas de software, no hubo hardware involucrado, a excepción del equipo en el que se desarrolló.

1. Java: lenguaje de programación orientado a objetos elegido para el desarrollo de la aplicación.
2. NetBeans 7.1: IDE (Integrated Development Environment) o Ambiente de Desarrollo Integrado que permite y facilita la programación en Java.
3. Trello: aplicación web que permite el manejo de una planeación ágil.
4. **Marco Teórico.**
   1. **Sobre La Simulación.**
      1. Conceptos.

Según (Banks, 1998): una simulación es una imitación de la operación de un proceso o sistema de la vida real. Una simulación es usada para describir y analizar el comportamiento de un sistema, para preguntarse cosas sobre el mundo real y para mejorar el diseño de sistemas reales.

En general, una simulación puede pertenecer a una de dos categorías:

1. Simulación de Estado Continuo: se aplica a sistemas donde la noción de estado es continua y, típicamente, involucra resolver de ecuaciones diferenciales. Simuladores a nivel de circuito son un ejemplo de ellos.
2. Simulación de Estado Discreto: es aplicable a sistemas en los que sus cambios se dan cada cierta cantidad finita de tiempo. De esta categoría hay dos tipos: avance de tiempo fijo o variable.

Si bien en la aplicación se pueden encontrar componentes que tiene lógica combinacional (Ver Electrónica para más detalle) es un Simulador de Estado Discreto porque la evaluación de las conexiones entre componentes se hace cada cierto tiempo predefinido que ando en el rango de los 100 milisegundos.

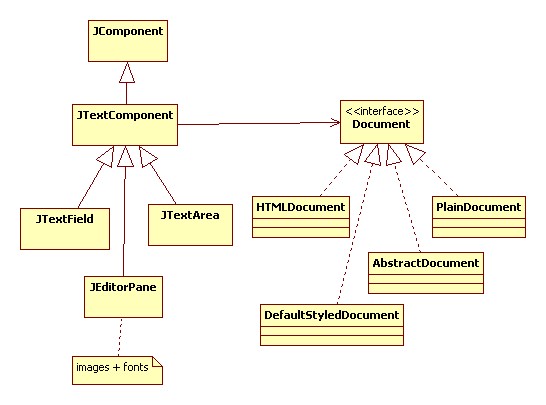
* + 1. Swing Framework.

Según la definición de Sun, Swing es un conjunto de componentes gráficos personalizables los cuales pueden cambiar su apariencia en tiempo de ejecución.

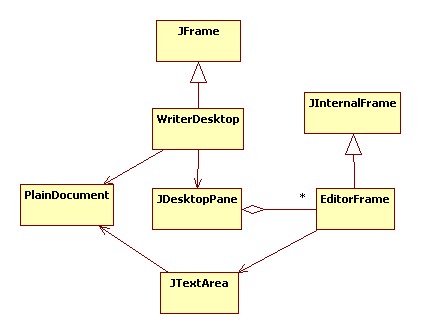
Sin embargo, Swing es mucho más que eso. Fue creado para el desarrollo empresarial de aplicaciones utilizando Java. Por esto se refiere a aplicaciones a gran escala con un amplio set de poderosos componentes. (Cole, Eckstein, Elliot, Loy, & Wood, 2002)

Además, todos ellos pueden extenderse o modificarse a conveniencia.

La jerarquía de clases de algunos de los componentes utilizados se presenta a continuación:



**Ilustración 1. Diagrama de clases JComponent**



**Ilustración 2. Jerarquía de clases de JFrame**

Las jerarquías anteriores son las principales del proyecto debido a que JComponent es el padre de casi todos los otros componentes de Swing. Y fue la clase que se extendió para definir las formas que se dibujan en la aplicación que este informe apoya.

Swing permite crear aplicaciones que simulan el escritorio de un sistema operativo con lo cual se pueden tener múltiples ventanas en un solo contenedor. Dichas aplicaciones son llamadas MDI (Multiple Document Interface). Una aplicación MDI se puede crear a través de las clases JDesktopPane y JInternalFrame que muestra la Ilustración 2

Drag and Drop.

Si bien los componentes de Swing traen funcionalidad de “Drag and Drop”, es bastante básica y rigida. Para el desarrollo de este proyecto fue necesario implementar dicha funcionalidad utilizando las interfaces MouseMotionListener y MouseListener. El trabajo es mayor pero permite gran libertad de diseño.

JTable es un componente más de Swing, este permite mostrar información en forma de tabla. El comportamiento estándar de un JTable no permite, entre muchas cosas, tener tablas jerárquicas, la cual fue necesaria para mostrar el contenido de las variables en el Monitor de Variables con el que cuenta la simulación.

Además el modelo de datos que recibe con la información a mostrar no se apegaba a lo requerido.

Se siguieron pasos de (Robinson & Vorobiev) para sobrecargar un DefaultTableModel para conseguir la funcionalidad requerida.

* 1. **Diseño De Hardware.**

El diseño de hardware o diseño electrónico es la acción de desarrollar un circuito interconectando dispositivos electrónicos bien conocidos de tal forma que cumpla con las especificaciones dadas.

Donde una especificación es una sentencia exacta que determina las necesidades que deben ser satisfechas, además deben ser medibles para poder determinar si dicha especificación se cumple o no.

“El diseño electrónico es el proceso de convertir la descripción de comportamiento (lo que pasa cuando…) a una descripción estructural (qué está conectado con qué y cómo)” (Sharma, 2006)

* + 1. Lenguajes De Descripción De Hardware.

Un lenguaje de descripción de hardware no es más que un lenguaje de programación utilizado para definir el comportamiento de componentes electrónicos. “Un lenguaje de descripción de hardware o HDL por sus siglas en inglés, pertenece a alguna de las siguientes categorías: lenguaje de computadora, lenguaje de especificación o lenguaje de modelado y comúnmente modelan lógica digital para la simulación.” (Princeton University, 2013)

La gran importancia de un lenguaje de descripción de hardware reside en que, en conjunto con los sistemas de simulación, permiten probar el comportamiento de componentes electrónicos antes de comenzar su producción física.

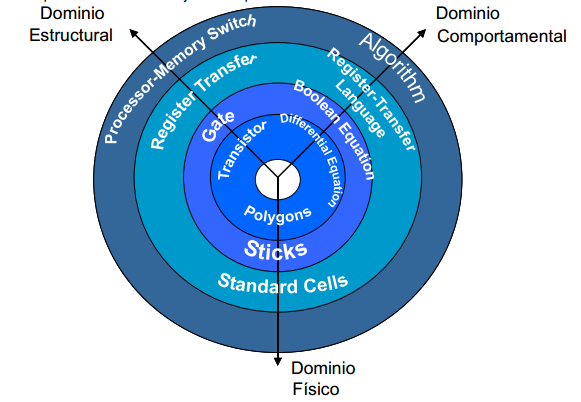
La mayor diferencia entre un lenguaje de descripción de hardware y un lenguaje de programación como C/C++ es que los últimos no disponen de mecanismos de reloj que son fundamentales en el diseño de hardware. Además los lenguajes de descripción de hardware tienen habilidades para el manejo de concurrencia.

(Vega-Castillo, 2007) menciona:

Los sistemas digitales pueden describirse con HDLs utilizando un enfoque comportamental, de flujo de datos o estructural. Los primeros dos enfoques y/o una combinación de los dos se conocen como código de nivel de transferencia de registros (RTL, register transfer level). En algunos textos este nivel se conoce también como nivel comportamental (behavioral) o código arquitectónico (ARTL, architectural RTL). Por otra parte, el enfoque estructural se conoce como código estructural, puesto que es una lista textual de compuertas y sus interconexiones. Puede pensarse que el código estructural es un archivo de texto describiendo un esquemático. (p. 1)

Para implementar los diferentes enfoques de descripción de hardware, el diseñador puede utilizar diferentes niveles de abstracción dependiendo de los requerimientos específicos:

1. Nivel de conmutadores (switch level), utilizado para describir el circuito en términos de transistores y cables
2. Nivel de compuertas (gate level), para describir el circuito en términos de compuertas lógicas y elementos de almacenamiento como flip-flops. Esta es una descripción textual del esquemático.
3. Nivel de flujo de datos, que describe el circuito en términos de flujo de datos entre registros
4. Nivel algorítmico o comportamental similar a un programa en un lenguaje de alto nivel como C. Incluye instrucciones de alto nivel tales como lazos, comandos de decisión y otros.



**Ilustración 3. Niveles de abstracción en el diseño de descripción de hardware.**

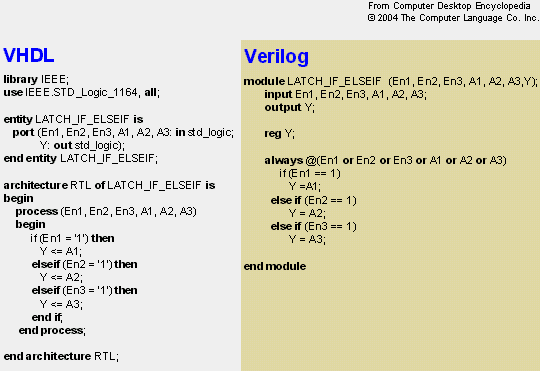
Por los últimos años, los dos lenguajes de descripción de hardware más utilizados han sido VHDL y Verilog.

La sintaxis de Verilog es más sencilla que la de VHDL y se asemeja más a la sintaxis de los lenguajes de programación que se suelen tratar en clases de Ciencias de la Computación.

Además, al momento de ser escrito este informe, es el que se utiliza en UNITEC en clases relacionadas a Micro controladores y Organización y Arquitectura de Computadoras.

Por estas razones se decidió utilizar Verilog para definir los componentes. Cabe mencionar que, por razones prácticas, la aplicación no soporta todas las características de Verilog.

Ver Anexo A para ver la definición léxica y sintáctica de la porción de Verilog aceptada por esta aplicación. La definición oficial del lenguaje fue tomada de (Sutherland, 2001).



**Ilustración 4. Imagen comparativa entre sintaxis de Verilog y VHDL**

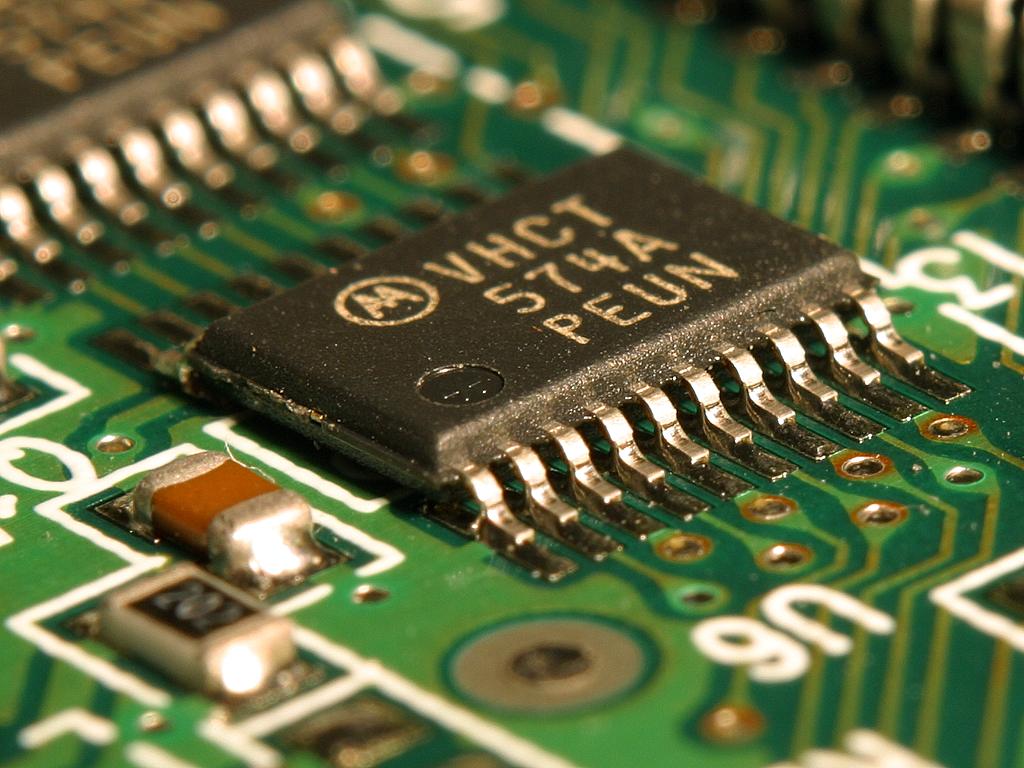
* 1. **Sobre Electrónica.**
     1. Conceptos.

1. Circuito Integrado: es un componente electrónico formado generalmente por transistores, resistores y otra variedad de componentes, puestos junto para llevar a cabo una tarea particular. (Ross, Shamieh, & McComb, 2010)

Un circuito integrado puede ser de dos tipos: lineales o digitales.

Lineales: son circuitos integrados que procesan señales analógicas, que consisten de voltajes y corrientes continuamente cambiantes.

Digitales: estos circuitos integrados procesan señales digitales, las cuales consisten de solo dos niveles de voltaje o corriente que representan datos binarios. Cabe destacar que en ciertos casos se encuentran circuitos integrados que tienen características de ambos.



**Ilustración 5. Circuito Integrado.**

1. Compuerta lógica: también llamadas simplemente compuertas, son diminutos circuitos digitales que aceptan una o más entradas binarias o genera una salida de un solo bit en base a una regla específica.

Hay siete compuertas lógicas, aunque algunas pueden ser construidas a partir de otras o una combinación de ellas. Estas compuertas son:

1. Compuerta Y (And Gate): su salida es un 1 (ALTO/HIGH) si y solo si todas sus entradas son 1.
2. Compuerta O (Or Gate): su salida es 1 si y solo si, al menos una de sus entradas es 1.
3. Compuerta NOT (Not Gate): produce un 1 si su entrada es un cero y viceversa.
4. Compuerta XOR (Xor Gate): produce un 1 como salida si exactamente una de sus entradas es 1, de lo contrario es 0. A diferencia de todas las otras compuertas, las compuertas Xor solo poseen 2 entradas, pero pueden ponerse en cascada para mayor complejidad.
5. Compuerta NAND (Nand Gate)
6. Compuerta NOR (Nor Gate)
7. Compuerta XNOR (Xnor Gate)

Las últimas tres compuertas son la negación de las compuertas Y, O y Xor respectivamente.

La forma más común de representar la operación que realiza cada una de las compuertas es una tabla de verdad. En la siguiente sección se explica a detalle este tema.

* + 1. Tablas De Verdad.

El matemático George Boole, escribió una tesis llamada “The Mathematical Analysis of Logic” en 1847. Su trabajo consistía, básicamente, en encontrar fundamentos básicos en el proceso del pensamiento lógico. Por pensamiento lógico se entiende el decidir si una proposición dada es verdadera o no de manera objetiva y basada en principios invariables, en este caso expresados en términos matemáticos.

Para comenzar, entiéndase por proposición una sentencia que puede ser ratificada o negada. (Boole, 1847)

Además del término, Boole definió cinco operaciones básicas que se pueden aplicar a toda proposición.

Dichas operaciones son:

1. Conjunción, también llamada Unión, y usualmente pronunciada en proposiciones como “y”.
2. Disyunción, también llamada Intersección, y pronunciada “o”.
3. Negación, pronunciada “no”
4. Implicación, la cual consiste de dos proposiciones, si el resultado de la primera proposición es verdadero entonces se ratifica la segunda.
5. Implicación bidireccional, pronunciada “si y solo si”.

Se puede determinar el valor de cualquier proposición descomponiéndola en varias proposiciones separadas por alguno de los operadores mencionados antes.

Con el objetivo de saber el valor booleano de una proposición compleja dado el valor de las proposiciones simples que la componen se crearon las tablas de verdad.

Por ejemplo: la proposición “Esta es una pera y está madura.” puede ser descompuesta en dos proposiciones separadas por una operación de conjunción (“y”):

1. Esta es una pera
2. (La pera) está madura.

¿Cómo se crea la tabla de verdad?

1. Por cada una de las proposiciones simples se agrega una columna a la tabla.
2. Se agrega una columna al final para el valor de la proposición original.
3. Por cada proposición simple se agregan dos filas, esto debido a que una proposición solo puede tener un valor booleano falso o verdadero.
4. Por cada operación aplicada agregar una columna (excepto la original, que ya fue agregada).

Una vez creada la tabla, la misma debe llenarse, para ello en cada fila se escribe una combinación de “verdadero” o “falso” hasta cubrir todas las posibilidades. Note que los valores “verdadero” o “falso” se escriben solo en las celdas que corresponden a proposiciones primitivas. El resto debe calcularse dependiendo de la operación que se aplica a las proposiciones involucradas.

Siguiendo con el ejemplo de la pera, la tabla de verdad generada es la siguiente:

|  |  |  |
| --- | --- | --- |
| **Esta es una pera** | **(La pera) está demasiado madura** | **Esta es una pera y está demasiado madura.** |
| **verdadero** | **verdadero** | **verdadero** |
| **verdadero** | **falso** | **falso** |
| **falso** | **verdadero** | **falso** |
| **falso** | **falso** | **falso** |

En el Anexo B se pueden encontrar las tablas de verdad de cada una de las compuertas mencionadas en la sección anterior.

* + 1. Simbología.

Una simbología es un conjunto de elementos gráficos que permiten expresar y entender diagramas o lenguajes. En el caso de la electrónica esta simbología permite reconocer componentes electrónicos a partir de sus símbolos.

En general hay muchos símbolos para representar una misma cosa, el símbolo usualmente varía según el continente o periodo de tiempo.

En electrónica se pueden encontrar tres simbologías principales, la simbología MIL (MILitary) la cual es aceptada por la ANSI (American National Standards Institute) que se encarga de que las características de un producto sea consistente a nivel mundial; también existe la simbología IEC que es usada ampliamente en Europa y es avalada por la IEC (International Electrotechnical Commission); finalmente está la simbología DIN que ya no se utiliza pero aún se puede encontrar en equipos viejos.

En América, el estándar más usado es el ANSI, por lo que se decidió utilizar la simbología MIL para los componentes electrónicos diagramados.

A continuación se mostrarán los símbolos utilizados en la aplicación así como los equivalentes en IEC para referencia.

Compuerta Y:

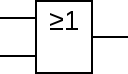
AND IEC.svg

**Ilustración 6. Símbolo IEC para una compuerta Y**

AND ANSI.svg

**Ilustración 7. Símbolo ANSI para una compuerta Y**

Compuerta O:



**Ilustración 8. Símbolo IEC para una compuerta O**

OR ANSI Labelled.svg

**Ilustración 9. Símbolo ANSI para una compuerta O**

Compuerta NOR:

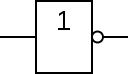
NOR IEC.svg

**Ilustración 10. Símbolo IEC para una compuerta Nor**

NOR ANSI Labelled.svg

**Ilustración 11. Símbolo ANSI para una compuerta Nor**

Compuerta NOT:



**Ilustración 12. Símbolo IEC para una compuerta Not**

http://upload.wikimedia.org/wikipedia/commons/thumb/9/9f/Not-gate-en.svg/128px-Not-gate-en.svg.png

**Ilustración 13. Símbolo ANSI para una compuerta Not**

Compuerta NAND:

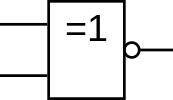
NAND IEC.svg

**Ilustración 14. Símbolo IEC para una compuerta Nand**

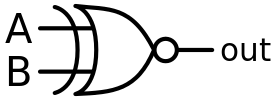
NAND ANSI Labelled.svg

**Ilustración 15. Símbolo ANSI para una compuerta Nand**

Compuerta XNOR:



**Ilustración 16. Símbolo IEC para una compuerta Xnor**



**Ilustración 17. Símbolo ANSI para una compuerta Xnor**

Compuerta XOR:

http://upload.wikimedia.org/wikipedia/commons/thumb/4/4e/XOR_IEC.svg/100px-XOR_IEC.svg.png

**Ilustración 18. Símbolo IEC para una compuerta Xor**

http://upload.wikimedia.org/wikipedia/commons/thumb/0/01/XOR_ANSI.svg/100px-XOR_ANSI.svg.png

**Ilustración 19. Símbolo ANSI para una compuerta Xor**

* 1. **Sobre El Desarrollo.**
     1. Metodología.

En todo proyecto es necesario la planeación del mismo, no solo por regla o documentación. Es la mejor manera de medir si se está cumpliendo con las metas propuestas.

Si bien en el desarrollo de software han existido muchos paradigmas con respecto a la planeación el que actualmente predomina es el Método Ágil.

* + 1. Metodologías Agiles.

Según (Rasmusson, 2010), los principios fundamentales de todas las metodologías agiles son:

1. Separa problemas grandes en varios más pequeños.
2. Concéntrate en lo realmente importante y olvídate del resto.
3. Asegúrate de lo que entregas funciona.
4. Ve a buscar retroalimentación.
5. Cambia el curso de ser necesario.
6. El progreso debe ser medible.

La metodología Ágil se basa en iteraciones, siendo una iteración una secuencia de etapas que comúnmente incluyen: Análisis, Desarrollo, Pruebas Técnicas y Pruebas o Demostraciones con el usuario.

Por regla toda iteración debe ser corta y debe tener la duración adecuada para entregar algo que el usuario considera un porción del sistema.

El objetivo de tener iteraciones cortas es detectar errores o cambio de planes lo antes posible pero, primordialmente, obtener retroalimentación del cliente o usuario.

Para hacer una planeación ágil debe existir un Macro Plan o “Master Story List”. De las tareas que existen en el Master Story List se toman algunas que conformarán el trabajo de una iteración.

Una de las tantas metodologías agiles que existe es Scrum, la cual fue utilizada en el desarrollo de este proyecto.

* + - 1. Scrum.

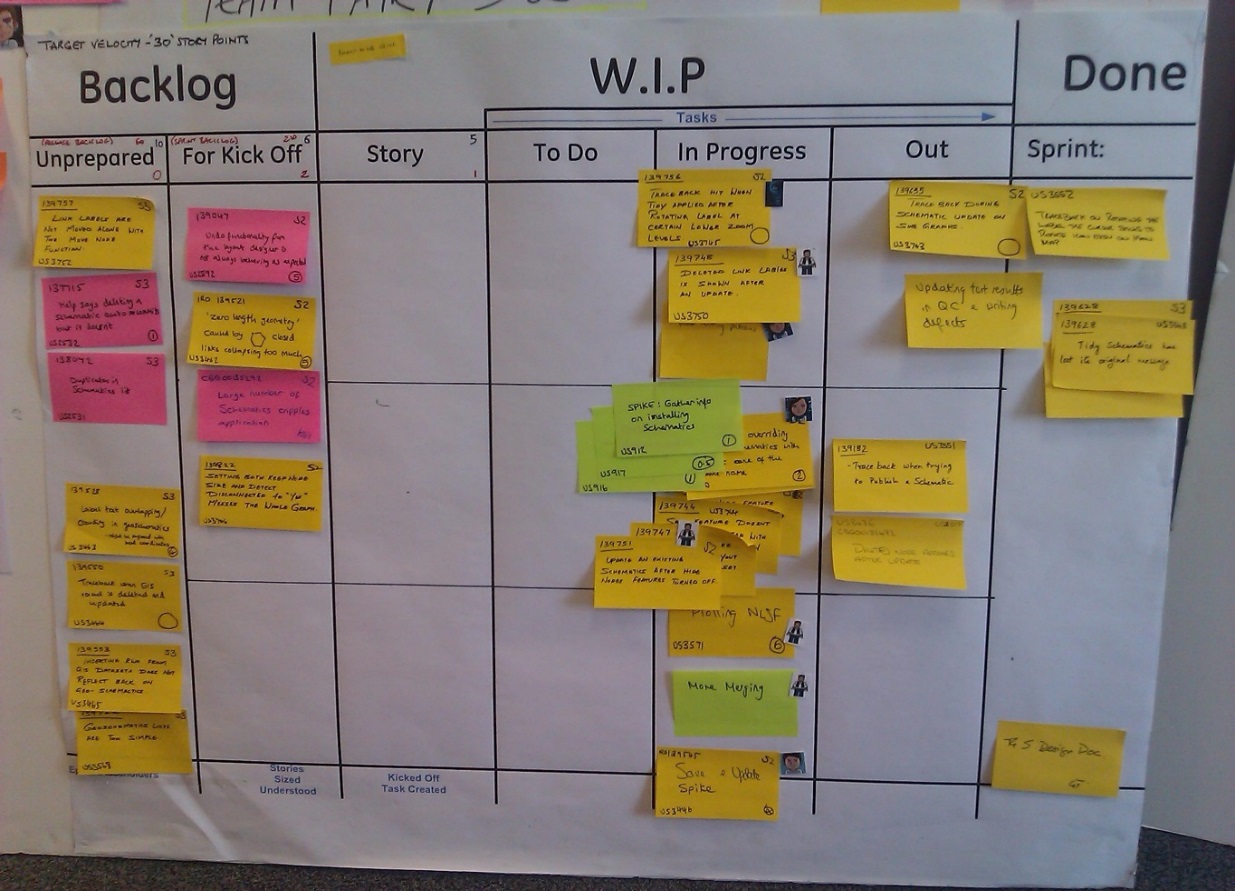
Fuera del contexto de la administración de proyectos Scrum es un movimiento de rugby en el cual todos los miembros se mueven juntos para llevar la bola de un extremo del campo al opuesto.

Justamente ese es el objetivo de Scrum en la administración de proyectos, todos los miembros del equipo trabajando para llevar una asignación o “Story” (según los términos de Scrum) del análisis a su finalización.

El proceso de Scrum inicia revisando el Product Backlog (Master Story List) con el cliente. Se identifican las tareas de mayor prioridad y se estima cuantas de estas tareas pueden entrar en sprint. Estas tareas forman el Sprint Backlog. Un sprint es un periodo de tiempo, usualmente entre 2 y 4 semanas, durante las cuales el equipo analiza, diseña, construye, prueba y documenta las tareas seleccionadas. (Smith & Sidky, 2009)

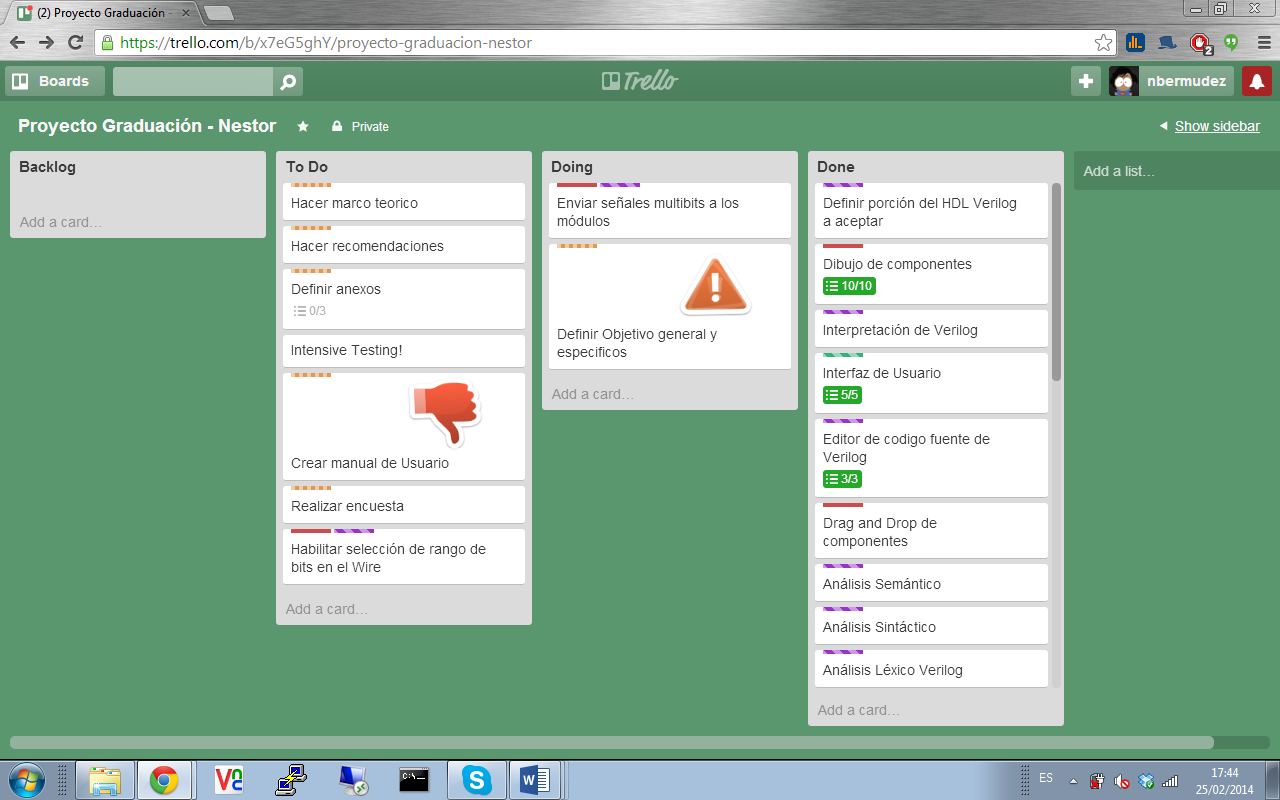
El método de control de progreso de un proyecto para Scrum es una pizarra dividida en tres secciones principales:

1. Backlog: contiene las todas las asignaciones, llamadas historias de usuario, que no han sido comenzadas o completadas.
2. Work In Progress: contiene las historias de usuario que se están trabajando, en condiciones ideales deberían haber historias sólo del sprint actual. Esta sección suele subdividirse en varias columnas dependiendo del proceso por el que se quiere pase una historia. Las más comunes son: To Do (donde están las historias que no se han comenzado), In Progress (las tareas que se están trabajando actualmente), Ready to Test (tareas terminadas pero sin aprobación de calidad).
3. Done: contiene las tareas que han sido completadas. Note que una asignación nunca debe retroceder en la pizarra. En caso de que una historia que estaba en Done necesita cambios no se muevo la historia a ningún lado, se debe crear una nueva historia para los cambios requeridos.



**Ilustración 20. Ejemplo de una pizarra de Scrum**

Hay herramientas en línea que permiten manejar la pizarra de un proyecto sin necesidad de una pizarra física. La planeación del proyecto se realizó utilizando una de ellas llamada Trello (<https://trello.com>)



**Ilustración 21. Captura de Pantalla de la pizarra de este proyecto**

* + 1. Sobre El Compilador.

Esta sección da por supuesto un conocimiento básico sobre el proceso de compilación de un programa y cierta terminología relacionada a compiladores.

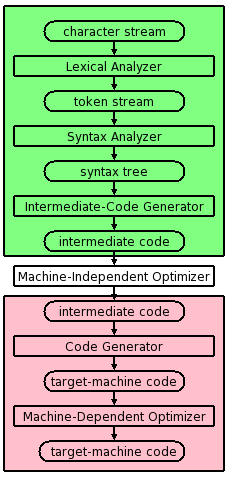
Un compilador es un sistema de software que realiza la conversión de un lenguaje de programación a una representación que una computadora pueda ejecutar llamado lenguaje objeto. (Aho & Ullman, 2006, pág. 1)

El proceso de compilación consiste de tres grandes fases:

1. Análisis lineal: también llamado análisis léxico, consiste en agrupar secuencias de caracteres en componentes léxicos que serán la entrada de la siguiente fase. Comúnmente dichos componentes son llamados tokens.
2. Análisis jerárquico: también conocido como análisis sintáctico, es el equivalente a las reglas sintácticas que rigen los lenguajes naturales como el español. Consiste en agrupar elementos mínimos, los tokens, de tal forma que tengan sentido, hablando de orden de los diferentes tokens.

Existen muchos tipos de analizadores sintácticos, LL, LALR y LR. Dependiendo de si son descendentes o ascendentes. Siendo el más poderoso el LR. El cual se llama así porque la entrada se analiza de izquierda a derecha (Left-to-Right) y porque utiliza la producción de la gramática más a la derecha (Rightmost Derivation). (Aho & Ullman, 2006)

1. Análisis semántico: (Maurer, 1946) trata de comprender el significado de una sentencia o expresión sintácticamente correcta.
2. Optimizaciones de interpretación intermedia: cuando se escribe código frecuentemente se hace de forma ineficiente. Aun cuando para el desarrollador sea eficiente no lo será para el procesador, es por esto que se crea un interpretación intermedia del programa para manipularse en memoria y tratar de hacerlo óptimo para la máquina.



**Ilustración 22. Etapas del proceso de compilación.**

La imagen anterior muestra un diagrama de la secuencia de pasos por las que pasa un programa para ser compilado. Para este proyecto, los pasos comprenden la sección de color verde debido a que el objetivo es interpretar un programa de Verilog y no generar código de máquina para el mismo. Lo anterior también se conoce como Interprete dado que solo interpreta el código original sin generar un archivo compilado como salida.

Las primeras fases pueden automatizarse mediante el uso de algunas aplicaciones generadoras de análisis léxicos y sintácticos. Para el desarrollo de este proyecto se eligieron, JFlex y Cup para el análisis léxico y sintáctico respectivamente.

* + - 1. JFlex

JFlex es un generador de analizadores léxicos para Java escrito en Java. Fue inspirado en JLex, que fue escrito por Elliot Berk en la Universidad de Princeton.

JFlex toma como entrada un archivo con una sintaxis particular y genera una clase de Java la cual es el analizador léxico descrito en el archivo de entrada.

Las instrucciones de instalación y uso pueden ser encontradas a detalle en la página oficial de la aplicación: <http://jflex.de/manual.html>, pero a grandes rasgos las instrucciones son:

Pasos para instalar JFlex:

1. Descargar el archivo comprimido encontrado en: <http://jflex.de/download.html>
2. Descomprimirlo en el directorio de elección.
3. Abrir el archivo jflex.bat que se encuentra dentro de la carpeta bin y cambiar los valores de las variables JAVA\_HOME y JFLEX\_HOME a los directorios correspondientes.
4. Agregar la carpeta bin a las variables de entorno de Windows.

Otros autores (Gálvez & Mora, 2005) denomina a herramientas como JFlex o Cup como metacompiladores porque son, en resumen, compiladores para escribir compiladores.

El propósito de un analizador léxico es generar tokens a partir de un stream de entrada. Para utilizar JFlex es necesario definir una expresión regular que represente cada uno de los posibles tokens que retornará el analizador.

Una expresión regular según (Goyvaerts, 2009) “es un tipo específico de patrón de texto que puede ser usado para verificar que cierta entrada concuerde con un patrón particular.”

Por ejemplo, en Java, C/C++ un identificador debe iniciar con una letra y luego contener una combinación de letras o dígitos. No se entrará en detalles sobre cómo escribir expresiones regulares ni sobre JFlex pero la expresión regular para un identificador es:   
[a-zA-Z]([a-zA-Z0-9])\*. Dicho lo anterior, una entrada en JFlex para ese token puede ser así:

%%

[a-zA-Z]([a-zA-Z0-9])\* { return IDENTIFIER; }

%%

Una vez terminada la definición del analizador léxico utilizando la sintaxis de JFlex es necesario que se compile utilizando la línea de comandos. Para ello, navegar hasta la carpeta que contiene el archivo .lex, en caso de querer regenerar el archivo de esta aplicación se puede encontrar en la carpeta VerilogCompiler dentro de la raíz de la aplicación.

El comando que debe executarse para generar el analizador léxico es:

jflex VerilogLexer.flex

Una vez generado el archivo, para utilizar el analizador con cualquier clase que extienda de la interfaz Reader de Java. Los dos más comunes son InputStreamReader que permite leer el contenido de un archivo y con él alimentar el analizador léxico y el StringReader con el cual se puede hacer pruebas con cadenas en duro.

* + - 1. Cup

Cup es un analizador sintáctico LALR desarrollado en el Instituto de Tecnología de Georgia que genera codigo de Java y que permite introducir acciones semánticas en dicho lenguaje. (Gálvez & Mora, 2005)

El manual de Cup puede encontrarse en: <https://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html>. Entre los temas de interés del mismo se encuentran:

1. ¿Cómo definir la precedencia de operadores?
2. ¿Cómo remover ambigüedad?
3. Opciones de ejecución.
4. Recuperación de errores.
   * 1. Otras Herramientas Y Tecnologías.

Suele suceder que un proyecto pierde enfoque cuando se debe trabajar en características que no son el centro de la aplicación.

Por esta razón se buscaron herramientas “Open Source” que pudieran ser utilizadas en partes del proceso de desarrollo.

Una de estas herramientas es RSyntaxTextArea.

* + - 1. RSyntaxTextArea

RSyntaxTextArea es un componente que permite resaltado de texto basado en sintaxis. Extiende la clase JTextComponent así que se integra perfectamente con el resto de componentes de Swing. Puede ser usado para ver o editar código fuente. (Fifesoft, 2014)

Si bien RSyntaxTextArea trae predefinido resaltados para muchos lenguajes como Java, C/C++, HTML, etc, no soportaba resaltado para Verilog por lo que fue necesario extender el componente.

Las mismas personas que crearon RSyntaxTextArea crearon una extensión para el mismo, es una herramienta de auto-completación llamada AutoComplete.

AutoComplete no necesitó modificaciones pero se necesitó definir el set de valores que pueden ser autocompletados.

Este set incluye:

1. Palabras Clave
2. Atajos de teclado para generar prototipos de código. Referirse al Manual de Usuario para más detalle.
   * + 1. XML.

XML significa eXtensible Markup Language, por sus siglas en inglés. Y es una tecnología, no un lenguaje de programación, que fue diseñada para el almacenamiento y recuperación de datos.

XML es en realidad un subconjunto de SGML, el cual es mucho más complejo y poderoso que XML. SGML tiene la capacidad de describirse a sí mismo a través de más meta data.

Otro subconjunto de SGML es el bien conocido HTML, que, a diferencia de XML, fue diseñado para mostrar información y no contenerla. Además, a diferencia de XML, HTML sí es un lenguaje dado que la cantidad de palabras de las que dispone es finita.

Según (Hunter, y otros, 2007):

Es importante hacer énfasis en que XML no es un lenguaje en lo absoluto, es un estándar para crear lenguajes que cumplen las características de XML. En otras palabras, XML describe una sintaxis que puede ser usada para crear los propios lenguajes.

Un archivo XML, a decir verdad, no hace nada, simplemente es un contenedor de información mediante segmentos llamados tags.

Un archivo XML tiene una estructura de árbol, que comienza con una “raíz” y se separa en “ramas” y estas a su vez en más ramas u “hojas”.

Ejemplo:

<bookstore>  
  <book category="COOKING">  
    <title lang="en">Everyday Italian</title>  
    <author>Giada De Laurentiis</author>  
    <year>2005</year>  
    <price>30.00</price>  
  </book>  
  <book category="CHILDREN">  
    <title lang="en">Harry Potter</title>  
    <author>J K. Rowling</author>  
    <year>2005</year>  
    <price>29.99</price>  
  </book>  
  <book category="WEB">  
    <title lang="en">Learning XML</title>  
    <author>Erik T. Ray</author>  
    <year>2003</year>  
    <price>39.95</price>  
  </book>  
</bookstore>

En el ejemplo anterior, la primera línea es usada para definir la versión del documento.

La siguiente línea es la raíz del documento, el cual cuenta con cuatro horas.

La estructura anterior puede representarse en una vista de árbol de la siguiente forma:



**Ilustración 23. Representación de árbol de un archivo XML.**

Otro tema relacionado a XML que no se ha mencionado es DOM.

DOM significa Document Object Model, cuya traducción es Modelo de Objetos del Documento. DOM es una interfaz de programación de aplicaciones (API) para documentos XML válidos. Define la estructura lógica de los documentos y el modo en que se accede y manipula.

Con DOM, los programadores pueden construir documentos, navegar por su estructura y añadir, modificar y eliminar elementos y contenido.

Si bien, en actualidad hay alternativas dependientes de la plataforma de desarrollo, DOM y la versión uno de XML están estandarizadas por lo que utilizarlas garantiza que, al menos en el futuro cercano, sigan vigente.

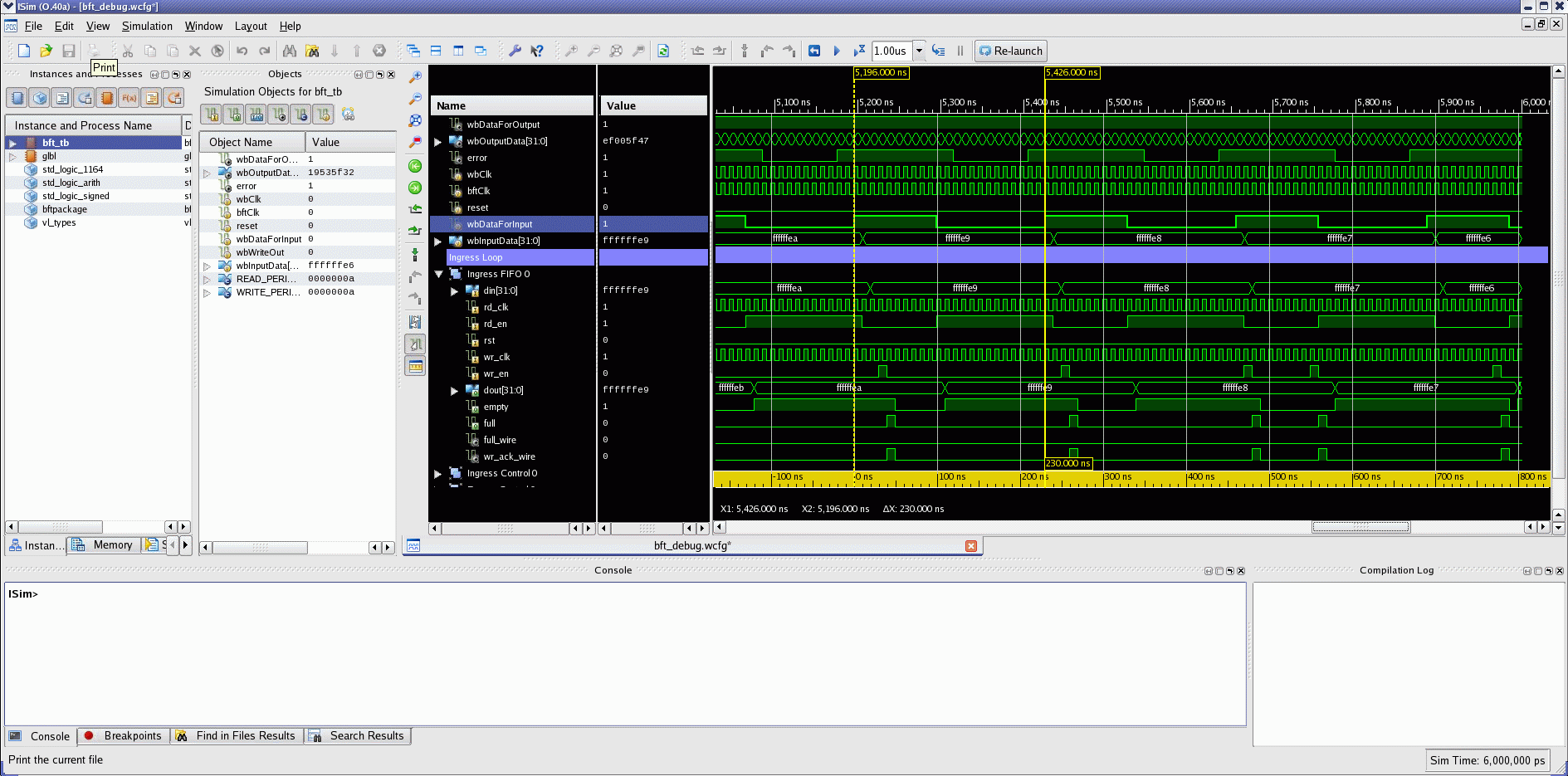
* 1. **Antecedentes.**
     1. Xilinx.

Xilinx es uno de los mayores proveedores de dispositivos programables en el mundo. Si bien originalmente solo se dedicaban al hardware, en la actualidad también se dedican a software relacionado al diseño, simulación y síntesis de componentes electrónicos.

Una de las tantas aplicaciones que tienen en su catálogo es ISE Design Suite.

ISE Design Suite es un conjunto de aplicaciones que comprenden todo el proceso de síntesis de un nuevo componente electrónico o de hardware; este proceso va desde su definición utilizando Verilog o VHDL, pasando por la simulación modular del componente, el mapeo de señales lógicas a los pines de un chip programable, hasta llegar a la síntesis de la definición para ser cargada en el chip programable.

Cada uno de las tareas anteriores se lleva a cabo en una aplicación independiente que es parte de la Suite. La que se encarga de la simulación se llama ISim, a continuación se muestra una imagen de la interfaz gráfica de la misma.

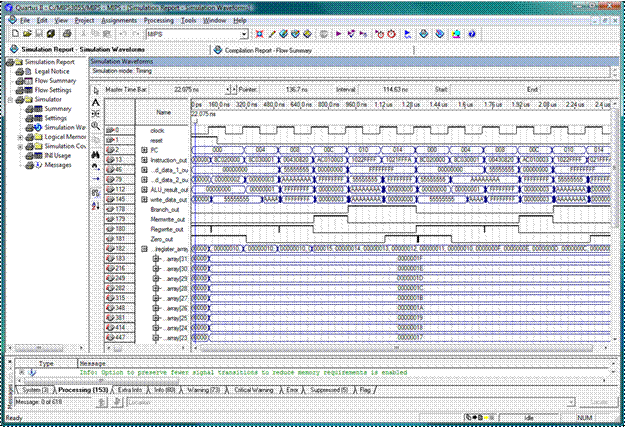
****

**Ilustración 24. Interfaz gráfica de ISim**

* + 1. Altera.

Altera es el otro gigante de los chips programables que, al igual que Xilinx, también tiene software para el diseño y síntesis.

La aplicación de Altera se llama Quartus II. A continuación la interfaz de simulación de ella:

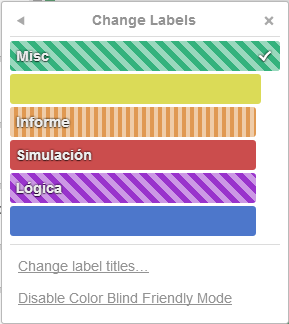
****

**Ilustración 25. Interfaz de simulación de Quartus II**

1. **Metodología.**
   1. **Cronología De Trabajo.**

Iniciando la primera fase del proyecto de graduación se definió una lista de tareas a realizar a lo largo del período. Como se mencionó antes, se utilizaría Trello para llevar la planificación de trabajo.

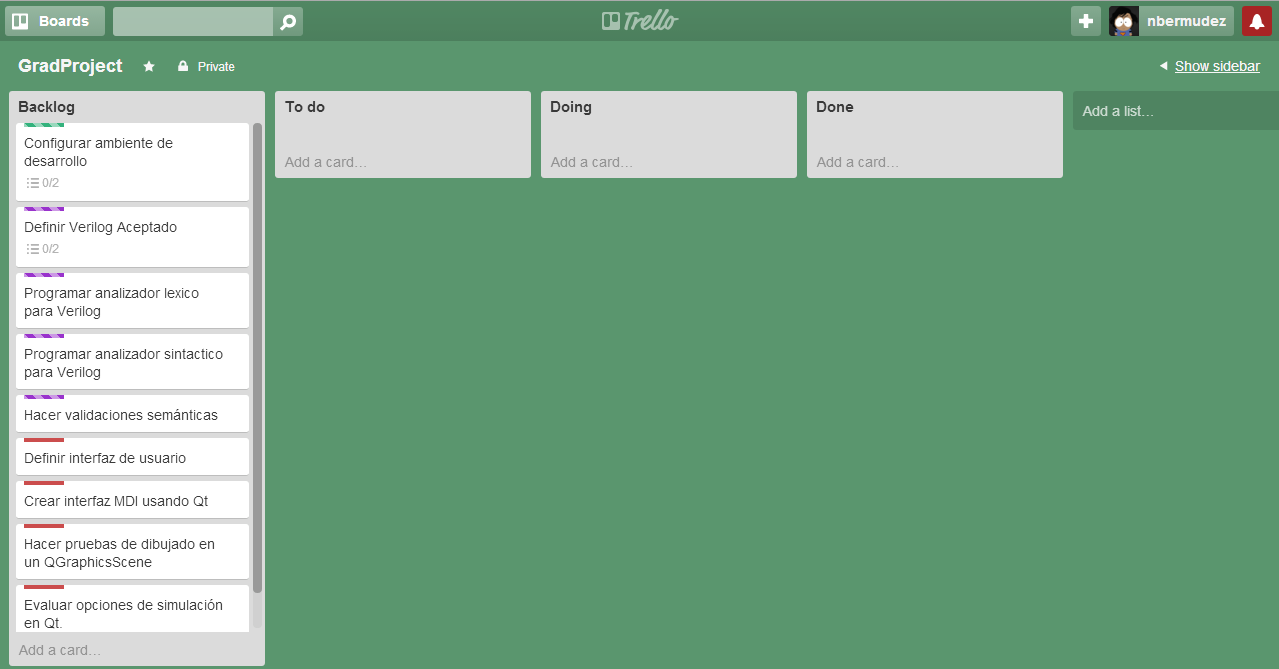
Antes de mostrar la planeación es necesario entender el significado del color de cada etiqueta en una tarjeta.



**Ilustración 26. Simbología de colores para agrupar tareas.**

Estos colores aparecerán en una pequeña barra en la parte superior de una tarjeta para representar que es una tarea relacionada al Informe, Simulación o Lógica; cuando una tarea no logra clasificarse en ninguna de las anteriores se utilizará Misc.

A continuación se encuentra una imagen general que muestra las tareas planificadas aun sin iniciar en Trello.



**Ilustración 27. Backlog de actividades a realizar en la fase I.**

En la primera fase no se llevó un registro tan detallado del trabajo realizado semana a semana.

**Semana 1.**

Se trabajó en la configuración del ambiente de desarrollo. Y se inició el aprendizaje de una de las herramientas a utilizar: Flex, que permite la definición del análisis léxico de un lenguaje.

**Semana 2.**

Se inició la definición de la interfaz de usuario, además se siguió con el proceso de aprendizaje de Flex y se inició el de Bison.

**Semana 3.**

Se comenzó a trabajar en la implementación de la interfaz de usuario definida en la semana anterior. También se inició el proceso de decidir qué características de Verilog serían soportadas por la aplicación dado que manejar todo el lenguaje se salía del margen de tiempo y utilidad del mismo.

**Semana 4.**

Se terminó la interfaz de usuario usando una aplicación MDI para permitir varias ventanas de diseño y simulación abiertas al mismo tiempo.

Se utilizó Flex para escribir la definición formal del analizador léxico de Verilog.

**Semana 5.**

Se comenzó con la escritura del analizador sintáctico de Verilog usando Bison. Paralelamente se comenzó a hacer pruebas de dibujado básico usando QGraphicsItems y QGraphicsScene de Qt.

**Semana 6.**

Se finalizó el analizador sintáctico de Verilog y se comenzó a trabajar en la creación de las clases del Árbol Sintáctico. Con respecto a la simulación, se hicieron pruebas para detectar colisiones entre componentes simples.

**Semana 7.**

Se implementó la funcionalidad de “Drag and drop” y “Resize” de elementos gráficos.

Simultáneamente se estudió y utilizó TreeCC para generar las clases del Árbol Sintáctico.

**Semana 8.**

Se comienzan a hacer pruebas integrales del proceso de compilación, que incluían desde el análisis léxico hasta la generación del árbol sintáctico.

Se nota que la interacción entre las partes comienza a dar problemas y que se pierde demasiado tiempo tratando de resolver problemas que deberían ser simples.

Inicialmente se decidió utilizar Qt por la familiaridad con el mismo. Relacionado a Qt no hubo inconvenientes, pero los problemas con el resto de las aplicaciones llevaron a la decisión de cambiar la plataforma de trabajo. A sugerencia del Asesor de este proyecto se decidió utilizar Java y la contraparte de Bison y Flex en Java: Cup y JFlex.

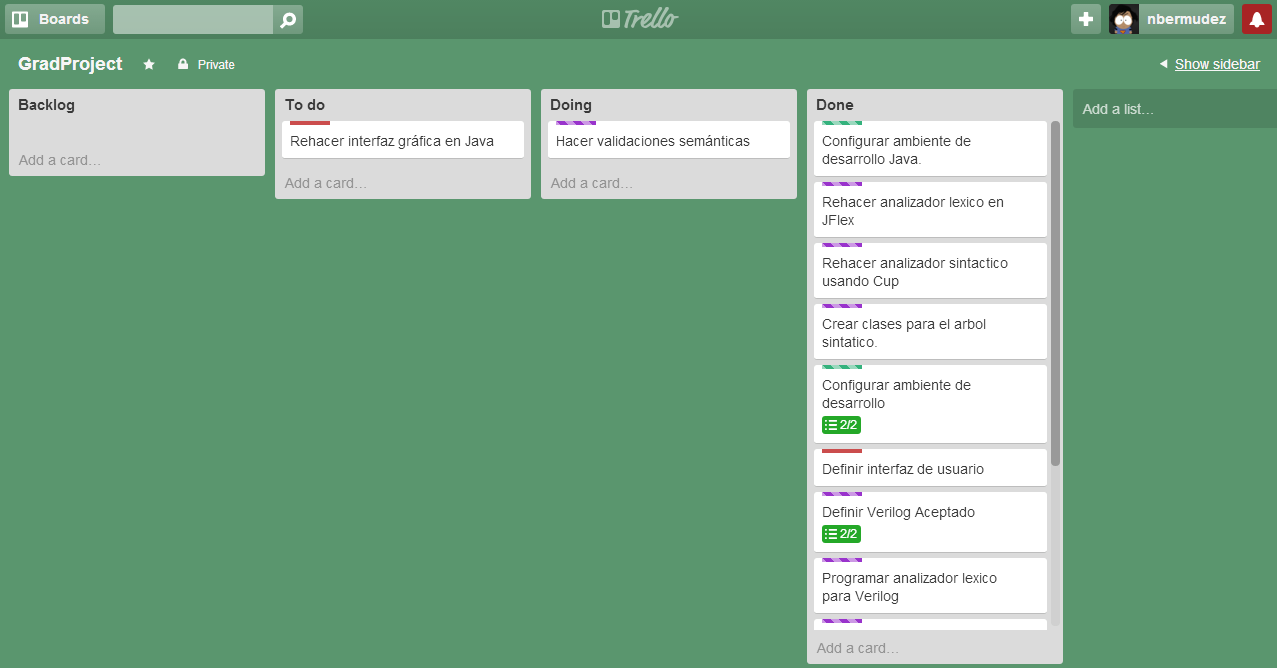
**Semana 9.**

Se acordó con el Asesor que durante las siguientes dos semanas se re-trabajara lo hecho hasta el momento en la nueva plataforma. Esta primera semana se utilizó para configurar el nuevo ambiente de trabajo y comenzar con la traducción de lo hecho en Qt a Java.

**Semana 10.**

Se terminó el analizador léxico y sintáctico utilizando Cup y JFlex. Se realizaron evaluaciones de progreso con el Asesor.

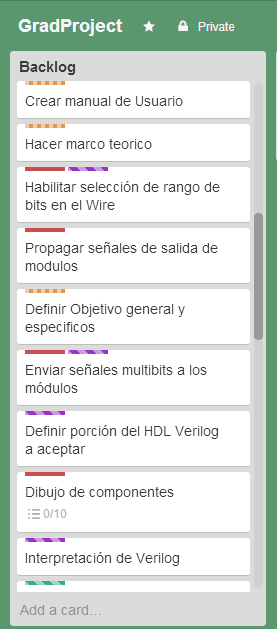
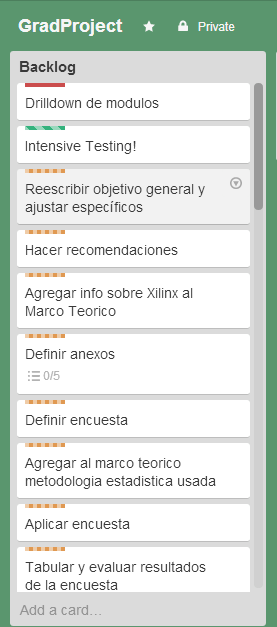
La siguiente imagen muestra el resultado final del board de Trello al finalizar la primera fase.



**Ilustración 28. Board de Trello al finalizar Fase I**

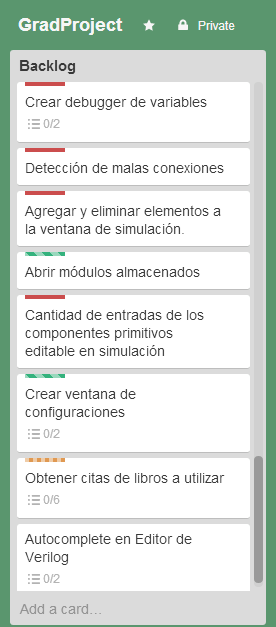
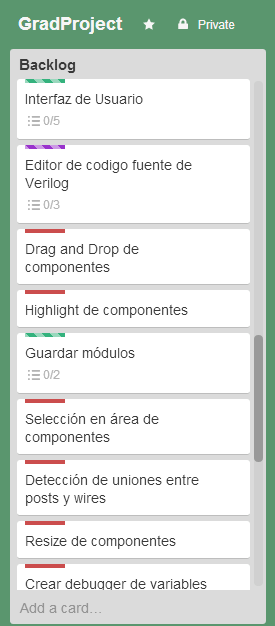
Como se puede ver la imagen, quedaron un par de cosas por hacer en la primera fase. No se pudieron terminar debido a las dos semanas de re-trabajo debidas al cambio de plataforma.

Si bien se trabajó en el proyecto en lapso entre la primera fase y la segunda, las tareas desarrolladas se programaron en las semanas dentro de la fase dos. Solo se mostrarán imágenes del Backlog del board dado que el resto no cambió con respecto a la imagen anterior.



**Ilustración 29. Backlog de Tareas a realizar en Fase 2**

**Ilustración 30. Más tareas a realizar durante Fase 2**



Al igual que en la primera fase, el código de colores se mantiene. Cabe destacar que en esta segunda fase la cantidad de tarjetas que involucran el informe aumentaron y el de número de las que involucraban desarrollo se disminuyeron.

Algo que sí cambió es que los entregables de ambas cosas, el informe y la aplicación, pasaron de ser semanales a cada dos semanas, debido a que se consideró que se era el lapso mínimo para tener algo presentable.

**Semana 11 y 12.**

Se crearon los componentes gráficos básicos: compuertas lógicas, cables, entradas y salidas lógicas. Se terminó el análisis semántico de Verilog y se inició el intérprete del mismo. Se rehicieron las características de “Drag and drop”, “Resize” y creación de los componentes definidos.

Se diseñaron y crearon estructuras XML para almacenar la definición de los componentes. También se trabajó en la lógica para cargar los componentes guardados.

Se trabajó en la conexión entre los componentes y la detección de malas conexiones.

Se analizó el requerimiento original de un modo de simulación paso a paso y se concluyó que no tenía sentido, para equilibrar el trabajo reducido se decidió agregar un Monitor de Variables.

Se comenzó a trabajar en el informe: el primer requerimiento fue la obtención de fuentes bibliográficas y la selección del texto a utilizar. Dijo requerimiento debería entregarse en un plazo de dos semanas para apegarse a la programación ya iniciada.

**Semana 13 y 14.**

El intérprete de Verilog fue terminado. Se implementó la propagación de señales de un solo bit entre todos los componentes. Se diseñó la representación gráfica de un módulo, un chip. Se creó una versión editable del chip para que pudiese reubicarse los puertos en el momento de la definición del componente.

Se definió el siguiente avance del informe: definición formal de los objetivos (general y específicos) y el marco teórico.

**Semana 15 y 16.**

Se investigó y escribió la bibliografía a utilizar para este documento. Se creó un componente para entradas de múltiples bits, necesario para alimentar los módulos.

Se corrigieron errores varios encontrados en la simulación y la propagación de señales.

Se crearon estructuras de datos para que sirvieran de repositorios de módulos, lógica de módulos y representación gráfica de ellos.

Se definió el siguiente entregable del informe: todo lo que faltaba excepto las conclusiones y análisis de resultados dado que la encuesta no sería aplicada hasta la siguiente semana.

**Semana 17 y 18.**

Se implementó el Monitor de Variables y se siguieron arreglando errores encontrados en el proceso. Se implementó la propagación de señales de varios bits entre módulos y otros componentes.

Se comenzó a trabajar en una nueva característica de la aplicación, un “Drilldown” de componentes complejos en momento de simulación.

Se escribió el Manual de Usuario y los anexos del informe.

El ultimo entregable del informe: el informe terminado, sujeto a revisión. Para ello habría que aplicar la encuesta, analizar los resultados y escribir las conclusiones en base a eso.

**Semana 19 y 20.**

Se hicieron varias pruebas con los módulos que compondrían un procesador. Se hicieron pruebas individuales y de interacción entre módulos. Debido a esas pruebas se encontraron errores que debieron ser corregidos para continuar con las pruebas.

Además se cambió la presentación del informe para que se apegara a las normativas de la universidad.

Se hicieron presentaciones de demostración a varios grupos de estudiantes para obtener su retroalimentación.

Finalmente se analizó dicha retroalimentación para concluir en base a los objetivos.

* 1. **Método Estadísticos Para Evaluación De Objetivos.**

Según (Ocampo, 2012), el tamaño de la muestra a utilizar está dado por la siguiente fórmula:

**Ecuación 1.**

Donde:

es el tamaño de la muestra.

es el tamaño de la población.

es una constante que depende del nivel de confianza que se quiera obtener. El valor se obtiene de la siguiente tabla relación entre y el nivel de confianza.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Valor de | 1.15 | 1.28 | 1.44 | 1.65 | 1.96 | 2.24 | 2.58 |
| Nivel de confianza | 75% | 80% | 85% | 90% | 95% | 97.5% | 99% |

y son el porcentaje de la población que tiene las características del caso de estudio, usualmente es desconocido y se utilizan valores de 0.5 para cada uno.

es el porcentaje de error que se espera.

Según cálculos aproximados, la cantidad de personas que cursan clases relacionadas al diseño de hardware en UNITEC en un año es de 24 personas.

Escogiendo un nivel de confianza de 95% y un 10% de error, el tamaño de la muestra es:

1. **Resultados Y Análisis.**
2. **Conclusiones.**
3. No se implementó la simulación paso a paso dado que, luego de un análisis del mismo, se concluyó que no tenía sentido porque el comportamiento del hardware, en su gran mayoría, es combinatorio y no discreto, lo que dificultó la definición de un “paso” de la simulación. Para compensar el tiempo de desarrollo se acordó la implementación de un Monitor de Variables el cual no estaba en el alcance original de este proyecto.
4. Se desarrolló una aplicación tipo escritorio que permite múltiples ventanas para evitar el tener que abrir aplicaciones independientes para el diseño de un componente de hardware y su simulación.
5. El Monitor de Variables logró implementarse de tal forma que se actualiza automáticamente según los cambios internos generado por los componentes, su utilidad se evaluó mediante la pregunta X de la encuesta. El análisis de la misma permite concluir que dicha herramienta es (o no es) útil para el usuario final.
6. Se logró conseguir un grupo de X personas para que utilizaran la aplicación y, posteriormente, respondieran la encuesta diseñada. Esta tarea permite concluir que, en general, la aplicación cumple (o no cumple) con las características necesarias para ser usado para el diseño y simulación de componentes de hardware.
7. **Recomendaciones.**

**A los alumnos.**

Para obtener un mejor rendimiento y trabajar sin otras preocupaciones o distracciones, se recomienda no cursar ninguna clase junto el Proyecto de Graduación. Dado que la forma de calificar es más rigurosa para una clase que para la primera fase del proyecto, se tiende a invertir más tiempo en dicha clase que en el proyecto.

Utilizar un método de planeación es muy importante y de gran ayuda, principalmente si se tienen otros compromisos como clases o trabajos. No se debe pensar en el tiempo de planeación como un tiempo desperdiciado, es la forma perfecta para saber el punto de progreso en el que se encuentra y qué está por delante. En este proyecto se utilizó Scrum en conjunto con Trello, pero hay infinidad de metodologías y herramientas, escojan la que mejor se adapte a su situación.

Avanzar en el proyecto durante vacaciones no es una pérdida de tiempo, si el proyecto marcha el trabajo fuera del periodo académico les dará un margen de tranquilidad; si el proyecto está atrasado servirá para ponerse al día. Se recomienda avanzar lo más que se pueda antes de comenzar la fase dos del proyecto dado que en esa fase tendrán que trabajar en el informe además del proyecto per se.

Pedir retroalimentación temprana, ya sea del asesor asignado al proyecto como de la persona con la que se definió el proyecto. Eso permitirá hacer ajustes al desarrollo y al plan y así evitar re trabajar partes del proyecto que se creían de una manera pero eran de otra.

**A quienes quieran continuar el proyecto.**

Escribir código fácil de entender siempre que sea posible, nombrar las variables con el amor con que nombrarían un hijo y nunca pensar en que nadie más verá el código escrito.

Incorporar elementos análogos y fuentes de voltaje de corriente directa y alterna ampliaría las áreas de uso de la aplicación mucho más allá de las actuales. En este momento la aplicación mantiene matrices y vectores de voltajes y fuentes de voltaje, así como métodos para la simplificación de ellas, pero no hay elementos que las alimenten.

El intérprete de Verilog es independiente al proceso de simulación por lo que es posible desarrollar intérpretes para otros lenguajes de descripción de hardware como VHDL sin alterar la simulación.

1. **Bibliografía**

Aho, A., & Ullman, J. (2006). *Compiladores. Principios, Técnicas y Herramientas.* Harlow: Pearson Education Limited.

Banks, J. (1998). *Handbook of Simulation. Principles, Methodology, Advances, Applications and Practice.* New York: Wiley-Interscience Publication.

Boole, G. (1847). *The Mathematical Analysis of Logic.* Cambridge.

Cole, B., Eckstein, R., Elliot, R., Loy, M., & Wood, D. (2002). *Java Swing.* O'Reilly Media.

Fifesoft. (10 de Febrero de 2014). *RSyntaxTextArea*. Obtenido de http://fifesoft.com/rsyntaxtextarea/

Gálvez, S., & Mora, M. (2005). *Java a tope: Compiladores. Traductores y Compiladores con LEX/YACC, JFLEX/CUP y JAVACC.*

Goyvaerts, J. (2009). *Regular Expressions Cookbook.* O'Reilly Media.

Hunter, D., Rafter, J., Ayers, D., Ducket, J., Watt, A., & McKinnon, L. (2007). *Beginning XML.* Indianapolis: Wiley Publishing, Inc.

Maurer, D. (1946). *Compiler Design.* Addison-Wesley Publishing Co.

Princeton University. (10 de Noviembre de 2013). *Hardware Description Language.* Obtenido de https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Hardware\_description\_language.html

Rasmusson, J. (2010). *The Agile Samurai. How Agile Masters Deliver Great Software.* The Pragmatic Bookshelf.

Robinson, M., & Vorobiev, P. (s.f.). *Swing.*

Ross, D., Shamieh, C., & McComb, G. (2010). *Electronics for Dummies.* Chichester: Jonh Wiley & Sons, Ltd.

Sharma, D. (Mayo de 2006). *Hardware Description Languages. Basic Concepts*. Obtenido de http://www.ee.iitb.ac.in/~smdp/DKStutorials/hdl.pdf

Smith, G., & Sidky, A. (2009). *Becoming Agile in an imperfect world.* Greenwich: Manning Publications Co.

Sutherland, S. (2001). *Verilog HDL. Quick Reference Guide.* Tualatin: Sutherland HDL, Inc.

Vega-Castillo, P. (Junio de 2007). *Escuela de Ingeniería Electrónica*. Obtenido de http://www.ie.itcr.ac.cr/pvega/Lab. Diseño Sist. Digitales/Lenguajes de descripción de hardware.pdf

1. **Anexos.**
   1. **Anexo A: Características De Verilog Aceptadas.**

**Palabras Reservadas.**

**always**

**and**

**assign**

**begin**

**case**

**deassign**

**default**

**else**

**end**

**endcase**

**endmodule**

**for**

**forever**

**if**

**initial**

**inout**

**input**

**integer**

**module**

**nand**

**negedge**

**nor**

**or**

**output**

**parameter**

**posedge**

**reg**

**repeat**

**supply0**

**supply1**

**wand**

**while**

**wire**

**wor**

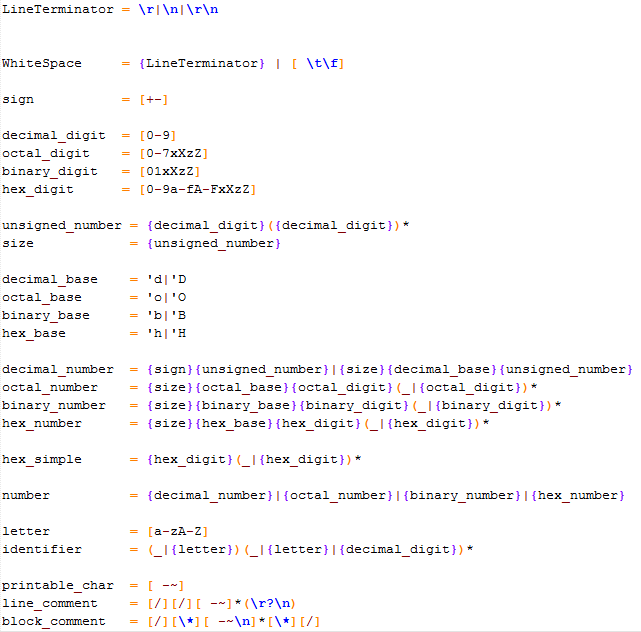
**xnor**

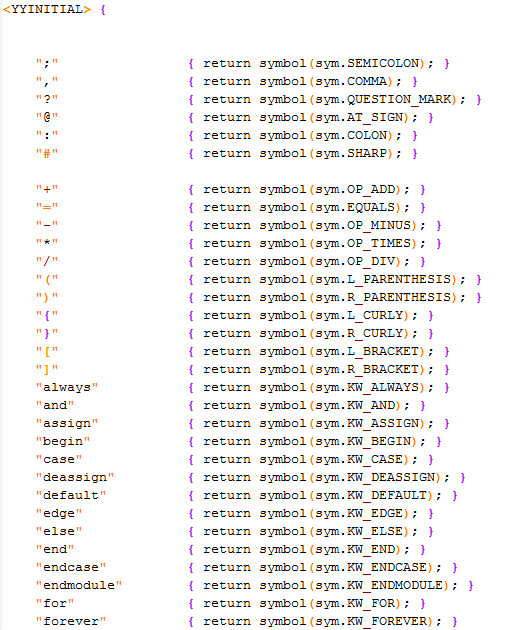
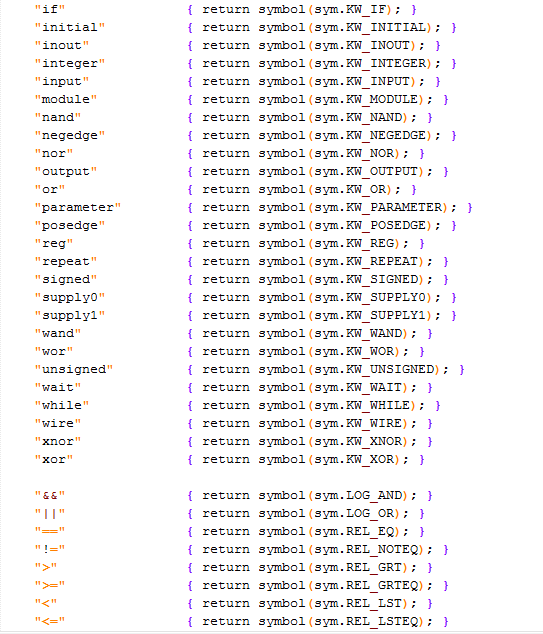
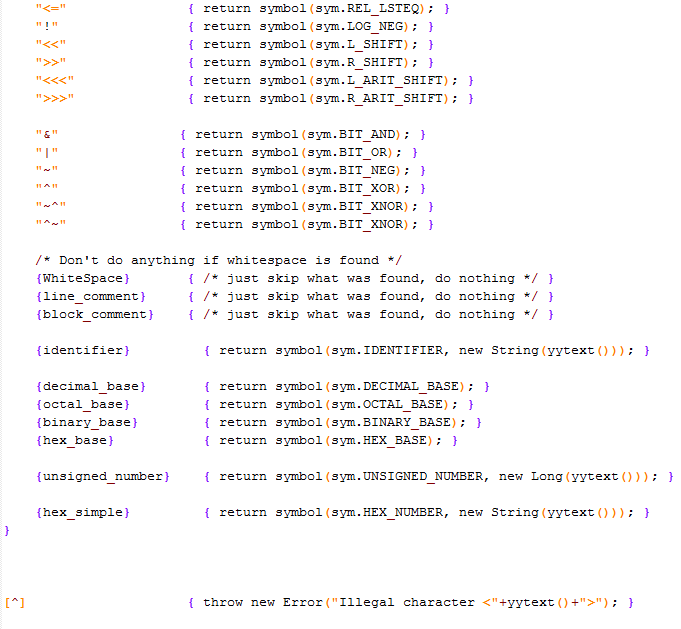
**xor**

**Convenciones Léxicas.**

1. Caracteres de espacio en blanco: espacios, tabs, saltos de línea.
2. Comentarios. Al estilo C, el de línea inicia con // y el de bloque /\*
3. Identificadores. Deben comenzar con un carácter alfabético o un guion bajo. Luego puede también contener dígitos.
4. Literales numéricas. Pueden estar formados de 3 componentes, el número de bits que utiliza, una base(‘o, ‘b, ‘h, ‘d).

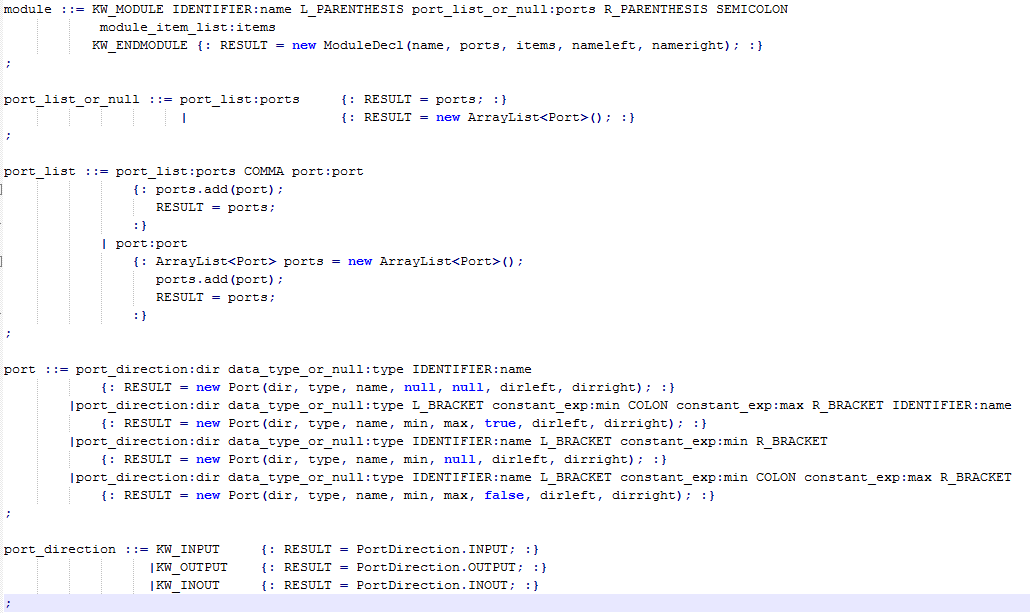
A continuación el archivo de JFlex que lo define.

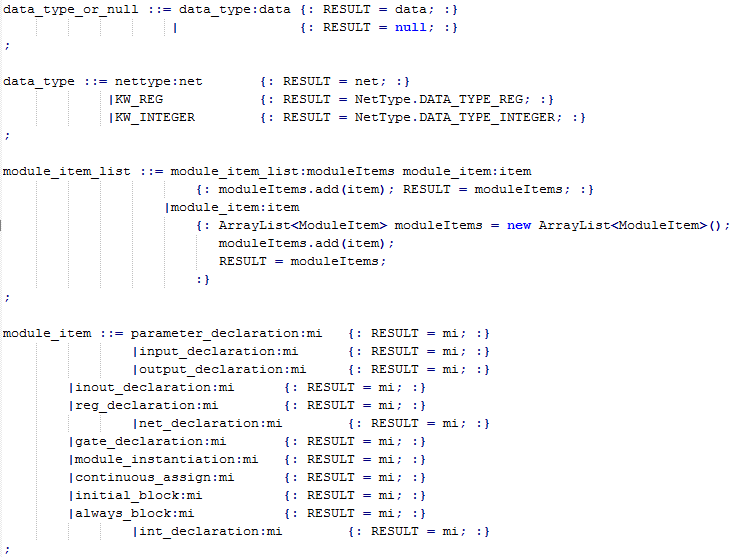


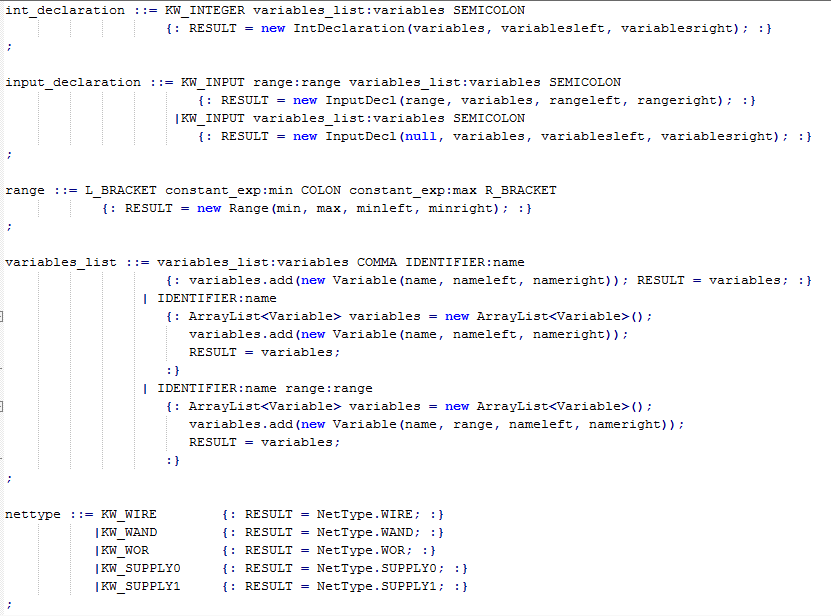
  

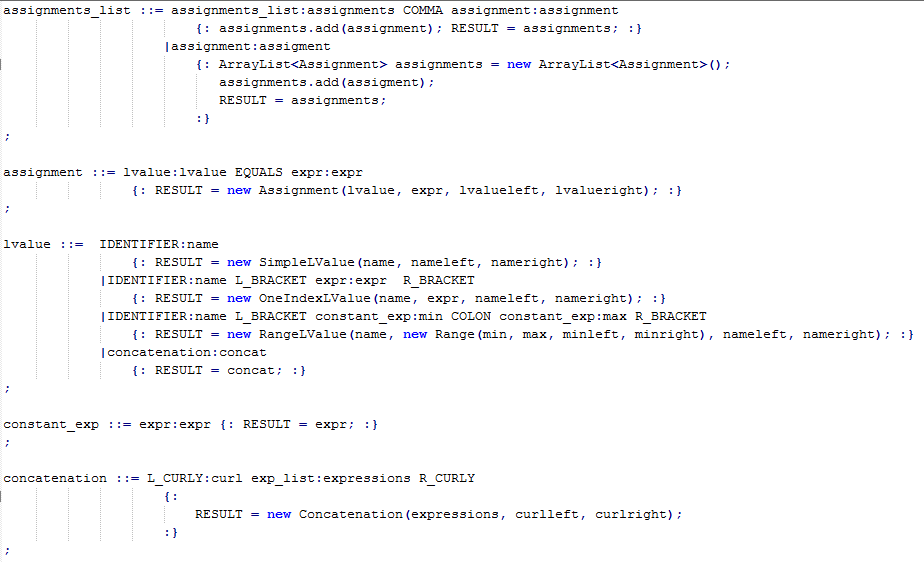
**Gramática del lenguaje.**

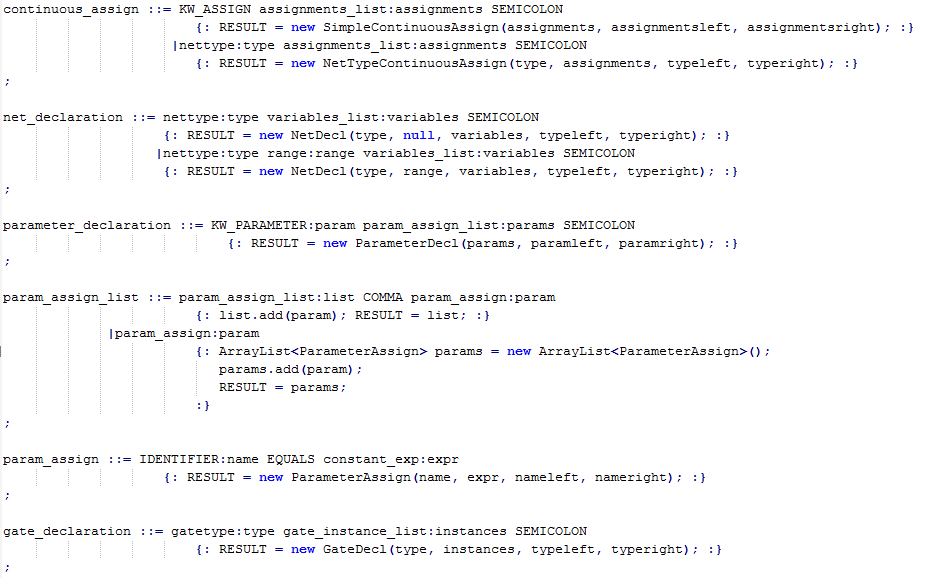
Esta se especificará con la sintaxis de Cup, la cual es parecida a la de BISON. Se asume cierto conocimiento del mismo.

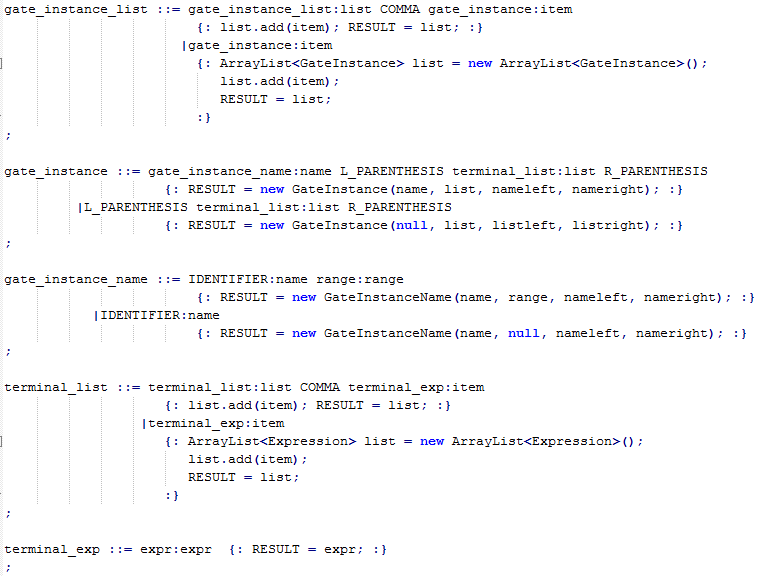


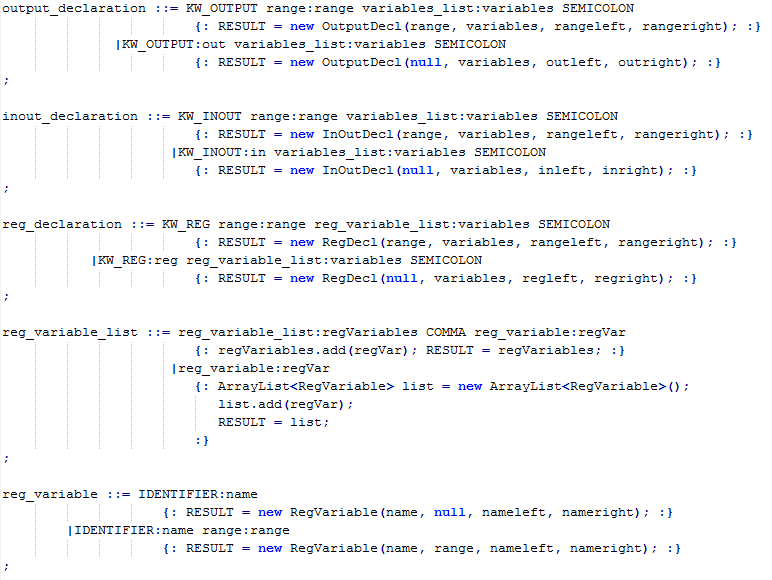


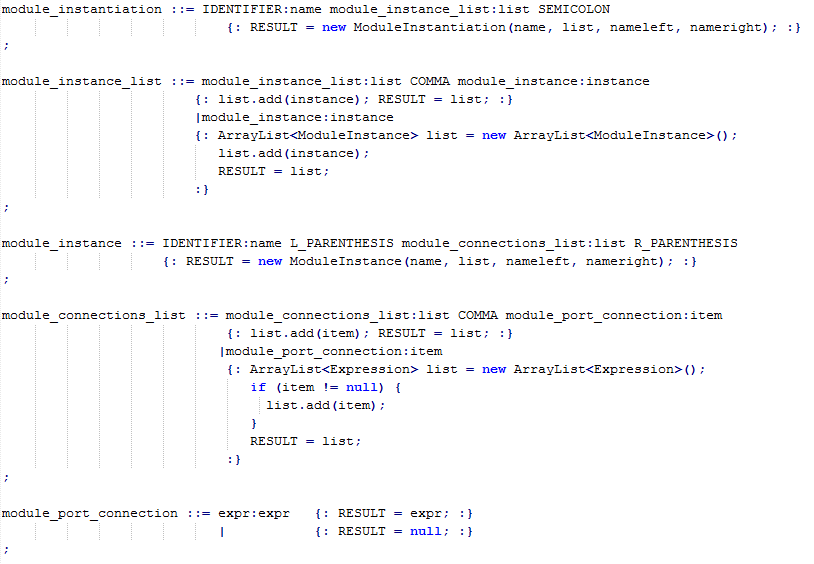


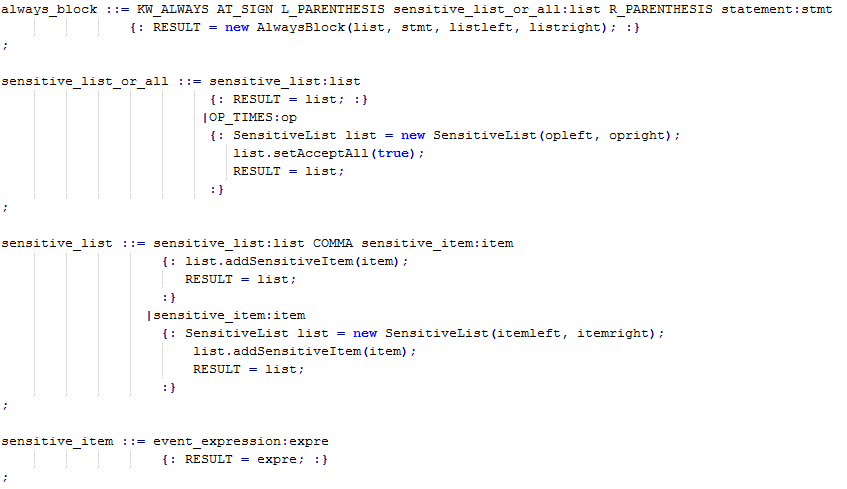


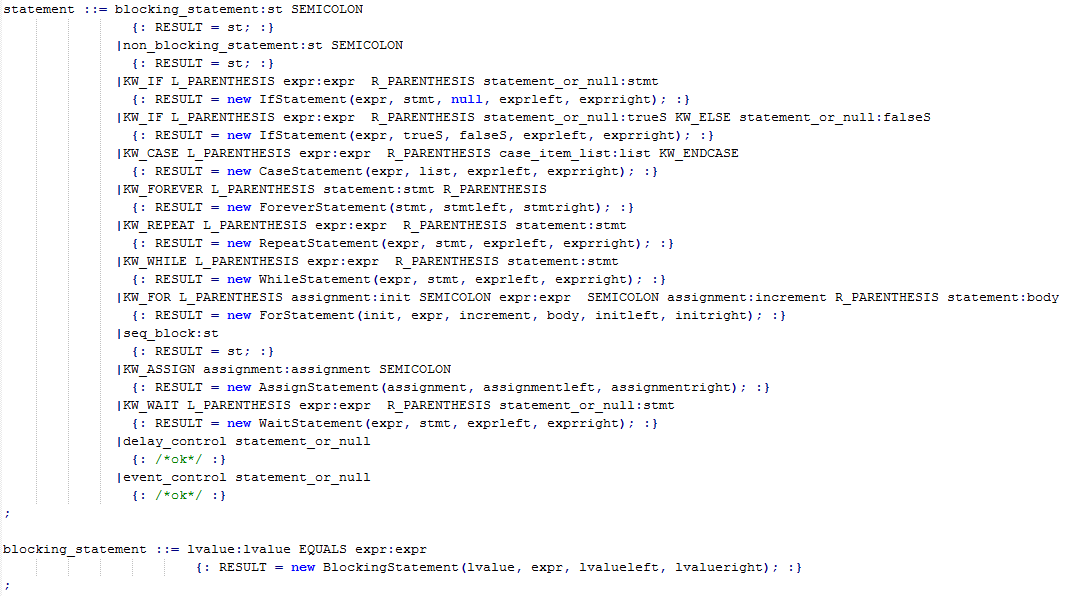


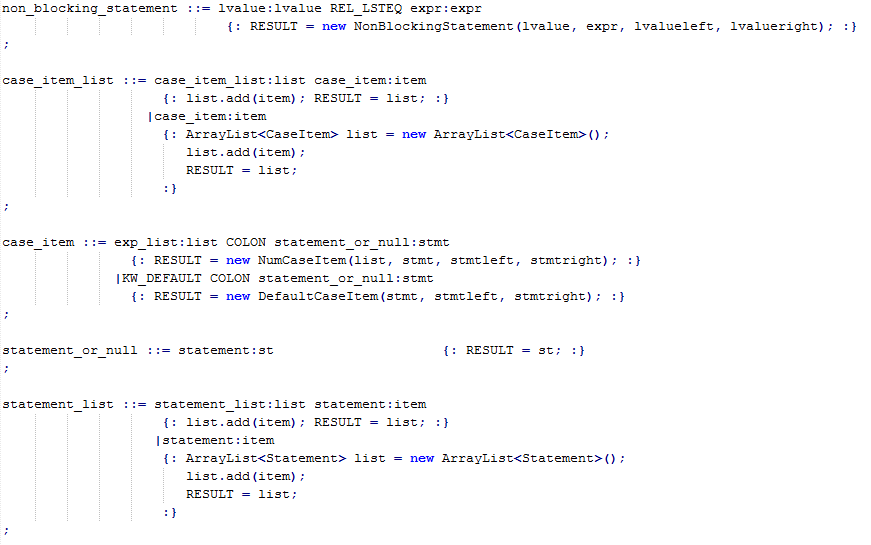


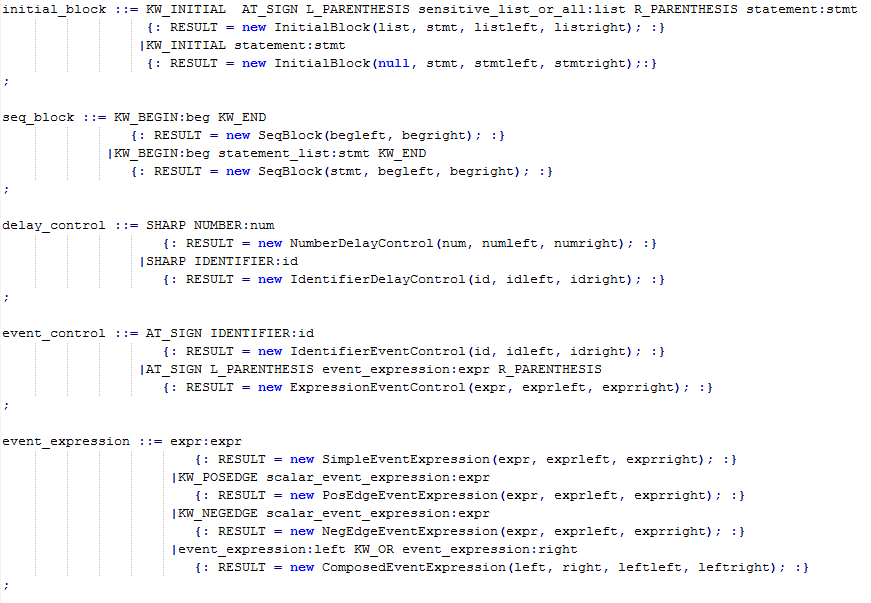


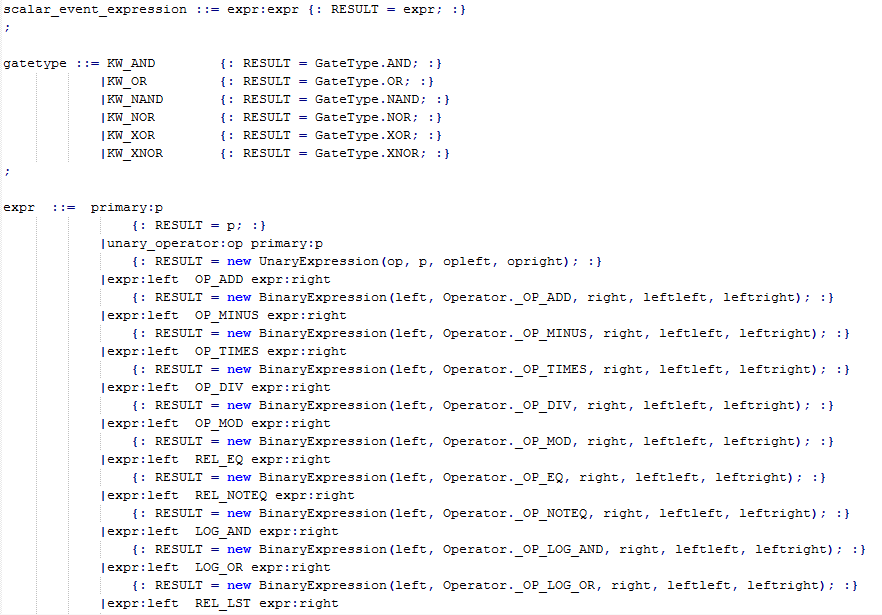


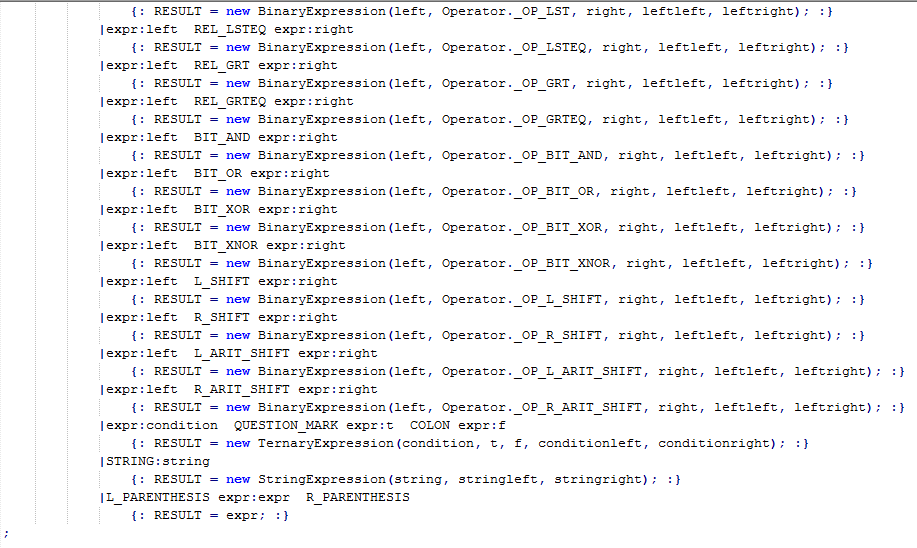


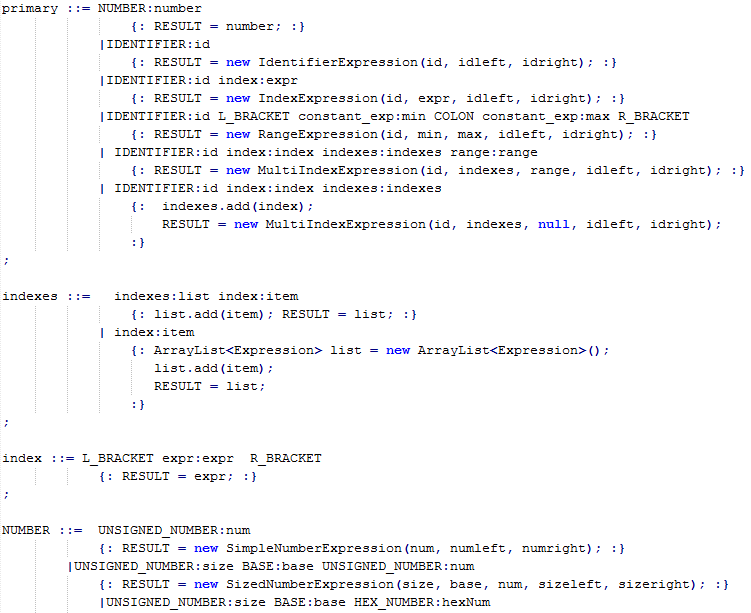


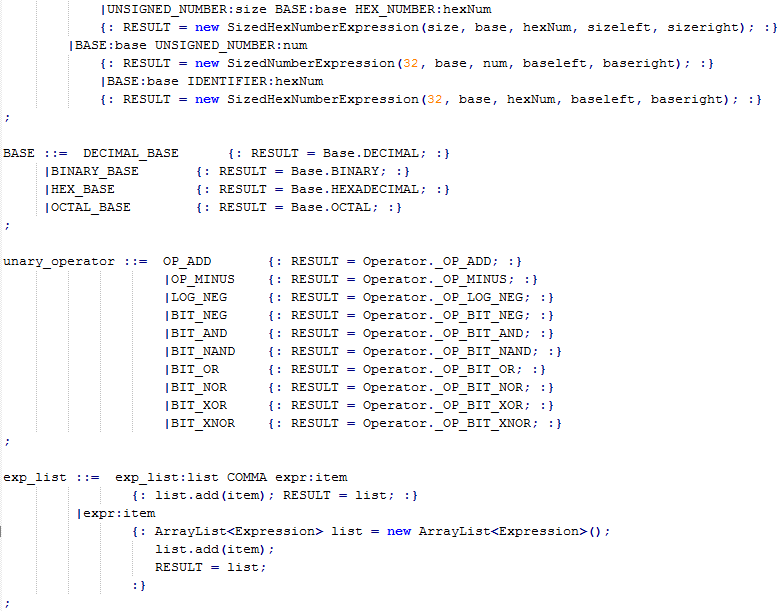












* 1. **Anexo B: Tablas De Verdad De Compuertas Lógicas.**

Hay un total de 7 compuertas lógicas. La salida de cada una de ellas depende de una función específica que depende de la combinación de sus entradas. En seguida se muestran las tablas de verdad de cada una de esas compuertas.

**Compuerta AND:**

|  |  |  |
| --- | --- | --- |
| Entrada A. | Entrada B | Salida |
| Verdadero | Verdadero | Verdadero |
| Verdadero | Falso | Falso |
| Falso | Verdadero | Falso |
| Falso | Falso | Falso |

**Compuerta OR:**

|  |  |  |
| --- | --- | --- |
| Entrada A. | Entrada B | Salida |
| Verdadero | Verdadero | Verdadero |
| Verdadero | Falso | Verdadero |
| Falso | Verdadero | Verdadero |
| Falso | Falso | Falso |

**Compuerta NOT:**

|  |  |
| --- | --- |
| Entrada. | Salida |
| Verdadero | Falso |
| Falso | Verdadero |

**Compuerta XOR:**

|  |  |  |
| --- | --- | --- |
| Entrada A. | Entrada B | Salida |
| Verdadero | Verdadero | Falso |
| Verdadero | Falso | Verdadero |
| Falso | Verdadero | Verdadero |
| Falso | Falso | Falso |

**Compuerta NAND:**

|  |  |  |
| --- | --- | --- |
| Entrada A. | Entrada B | Salida |
| Verdadero | Verdadero | Falso |
| Verdadero | Falso | Verdadero |
| Falso | Verdadero | Verdadero |
| Falso | Falso | Verdadero |

**Compuerta NOR:**

|  |  |  |
| --- | --- | --- |
| Entrada A. | Entrada B | Salida |
| Verdadero | Verdadero | Falso |
| Verdadero | Falso | Falso |
| Falso | Verdadero | Falso |
| Falso | Falso | Verdadero |

**Compuerta XNOR:**

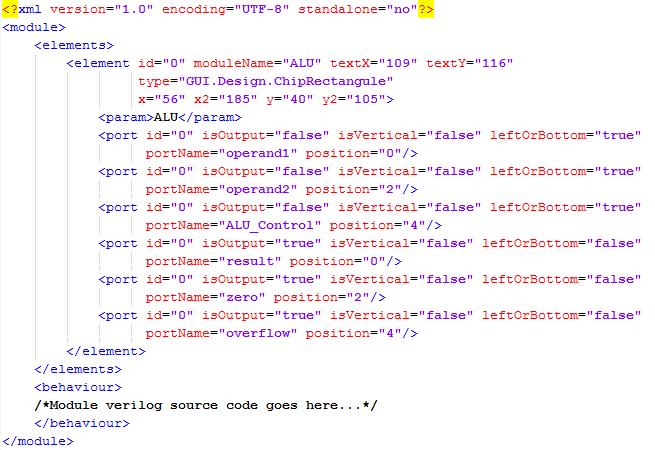
|  |  |  |
| --- | --- | --- |
| Entrada A. | Entrada B | Salida |
| Verdadero | Verdadero | Verdadero |
| Verdadero | Falso | Falso |
| Falso | Verdadero | Falso |
| Falso | Falso | Verdadero |

* 1. **Anexo C: Estructuras Xml.**
     1. Archivo De Configuraciones.



Cada una de las configuraciones consiste de tres campos, el nombre de la propiedad, el tipo de datos (usado para hacer conversión al valor en memoria) y el valor actual de la propiedad.

* + 1. Archivo De Definición De Módulo.

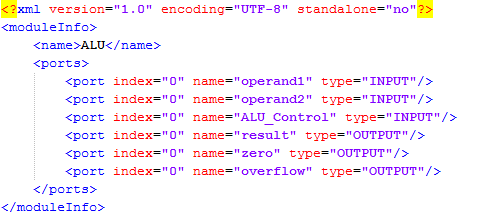


La información de cada elemento es usada para crear, via Reflection, una representación gráfica del mismo. Para hacer la construcción lo más genérica posible, el constructor utiliza la pareja de puntos “x”, “y”, “x2” y “y2” como argumentos de construcción comunes a todos los elementos y la lista de atributos “param” que se encuentran en el elemento para pasar parámetros extra.

El atributo “type” identifica completamente la clase que se instanciará para representar el elemento gráficamente.

El archivo XML de para la definición de un módulo y para una simulación sin básicamente el mismo. Las diferencias son las siguientes:

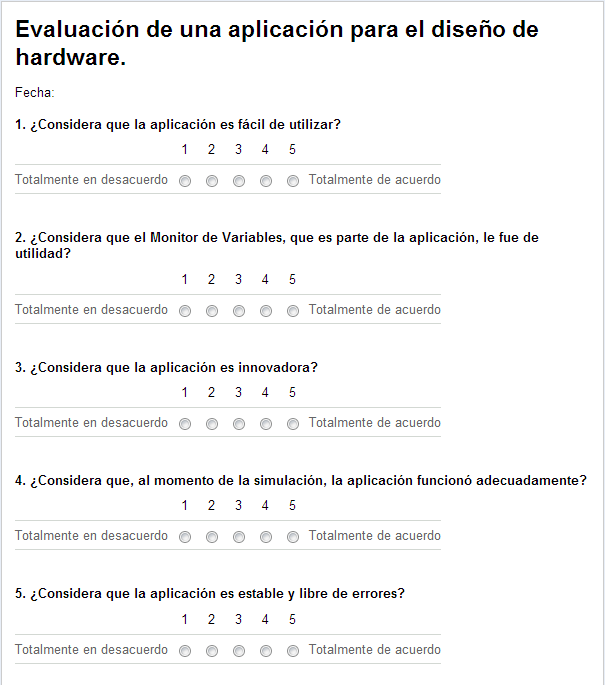
1. En Simulación no se utiliza el tag “behaviour”, dado que ya está en la definición del módulo.
2. En Definición solo hay un elemento en la lista de elementos, esto es lógico dado que en el momento de la definición el único elemento que existe es el componente siendo definido.
3. En Simulación, en vez de utilizar la clase ChipRectangule, para representar un componente definido por el usuario usando Verilog se utiliza ModuleChip. Esta decisión radica en que, durante la definición de un componente, los puertos del chip pueden moverse libremente mientras que en modo de Simulación son estáticos y tienen información extra generada en el momento de definición.
   * 1. Archivo De Metadata.

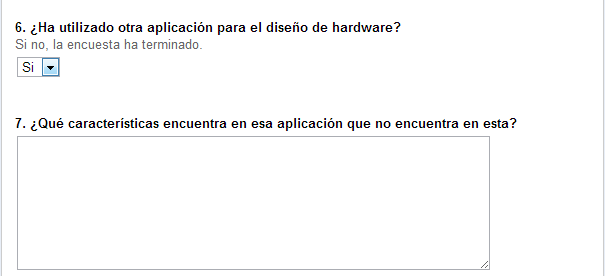


El propósito de este archivo es que no sea necesario cargar un archivo más extenso (el de definición del módulo) y pasarlo por el proceso de compilación simplemente para saber si los puertos del componente son de entrada o salida. El campo “index” es un vestigio de versiones anteriores, actualmente no se utiliza dado que la relación entre puerto y módulo se hace vía “name”.

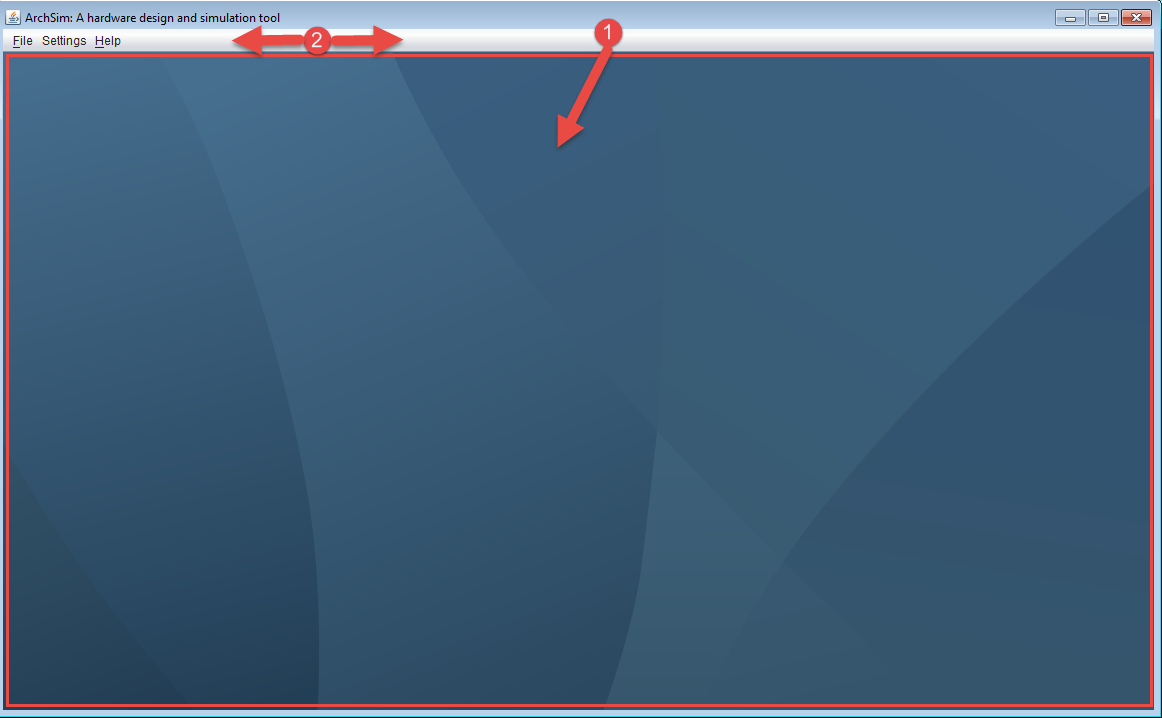
* 1. **Anexo D: Encuesta.**

Para poder evaluar el cumplimiento de los objetivos debió diseñarse y aplicarse una encuesta a alumnos y exalumnos de UNITEC. A continuación se presenta la encuesta utilizada.





* 1. **Anexo E: Manual De Usuario.**
     1. Pantalla Principal.



1. Escritorio. En esta área se mostrarán las ventanas abiertas ya sean de diseño o simulación. Estas ventanas se pueden cerrar, minimizar, maximizar y moverse por todo el escritorio.
2. Barra de Menú: cuenta con tres menús principales:
   1. File: tiene acciones para crear una nueva ventana de diseño o simulación, abrir un archivo guardado previamente y cerrar la aplicación.



* 1. Settings: cuenta con una sola opción que muestra una ventana para editar algunas configuraciones generales y de simulación.



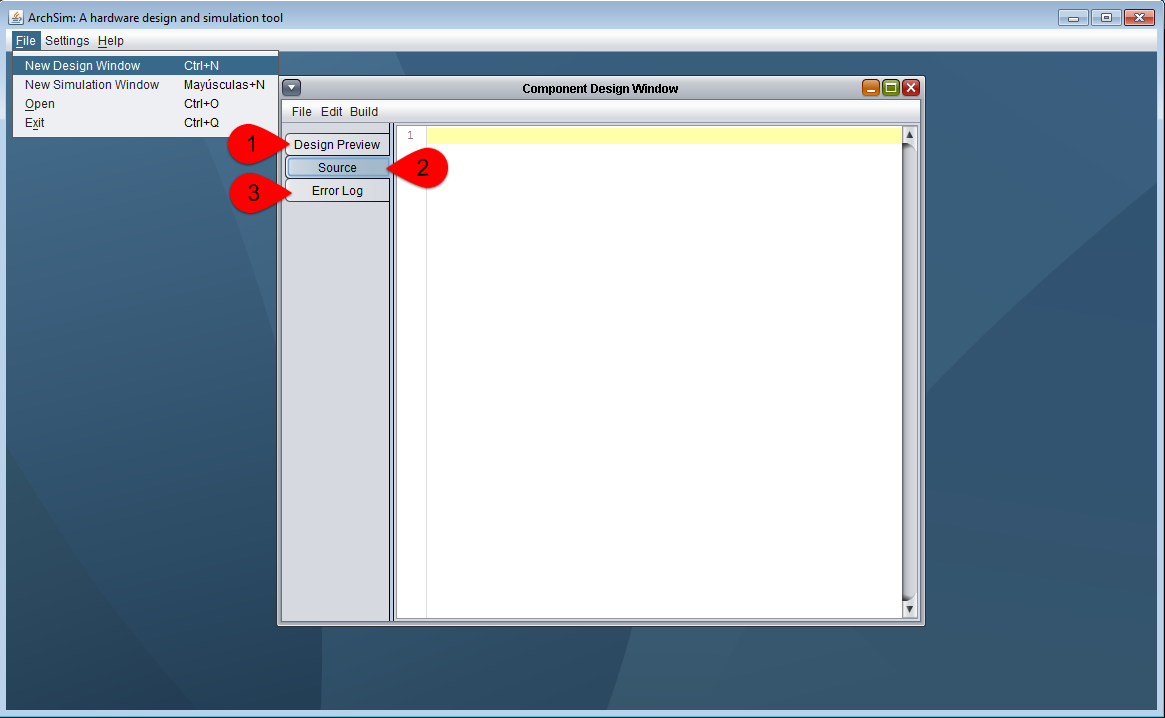
* 1. Help: por el momento solo cuenta con una opción, About que muestra información general y de contacto del desarrollador, se espera que, a futuro, haya una página de ayuda sobre la aplicación.



**Ilustración 31. Opciones en el menú de Ayuda**

* + 1. Ventana De Diseño.

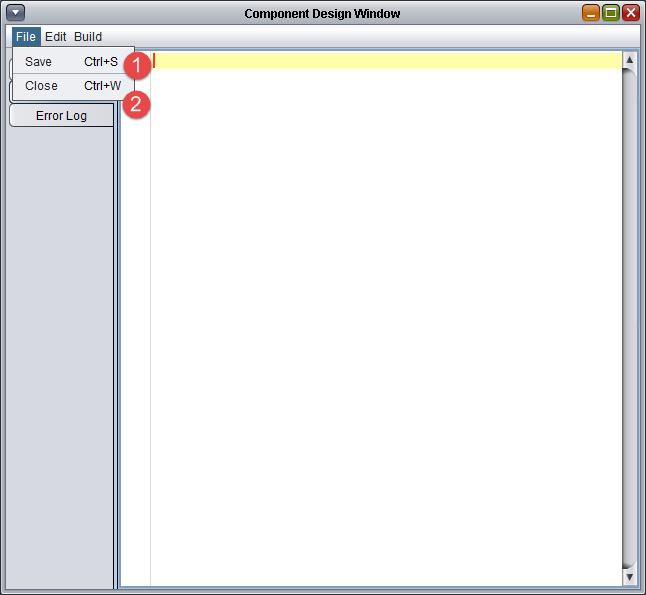
En esta ventana se diseña un nuevo componente usando Verilog para definir su comportamiento.



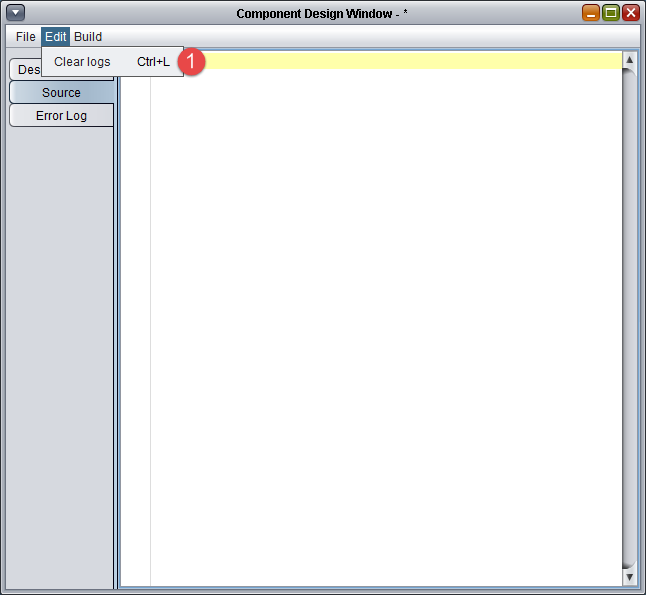
**Ilustración 32. Ventana de diseño de componentes**

1. Design Preview: cuando se compila exitosamente el código de Verilog escrito en “Source” se genera una vista previa del componente. En esta pantalla se puede cambiar la posición de los puertos del módulo y la posición del nombre del módulo.
2. Source: esta sección es, básicamente, un editor de texto donde se escribe el código de Verilog.
3. Error Log: cuando el proceso de compilación del código de Verilog falla, no se genera una vista previa, en su lugar se muestra una lista de los errores ocurridos. Esta lista no se borra por cada intento de compilación, puede borrarse con la opción “Clear Logs” en el menú Edit.

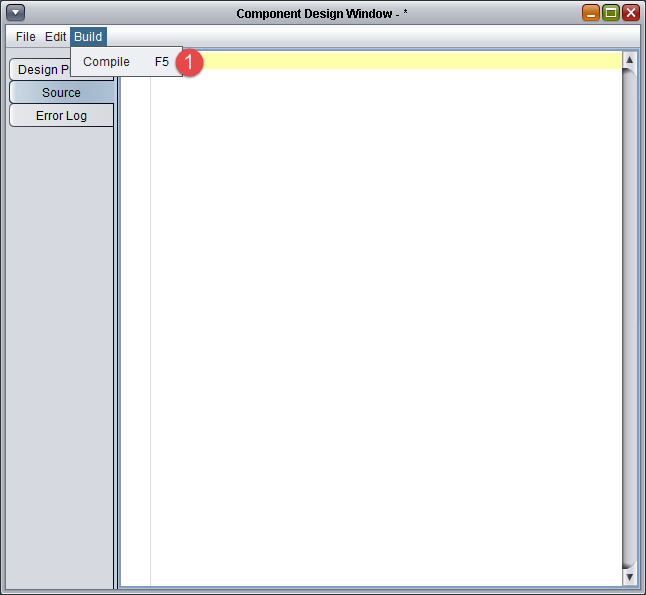
Las opciones en los menús son:



**Ilustración 33. Diseño de componente: Opciones en File**

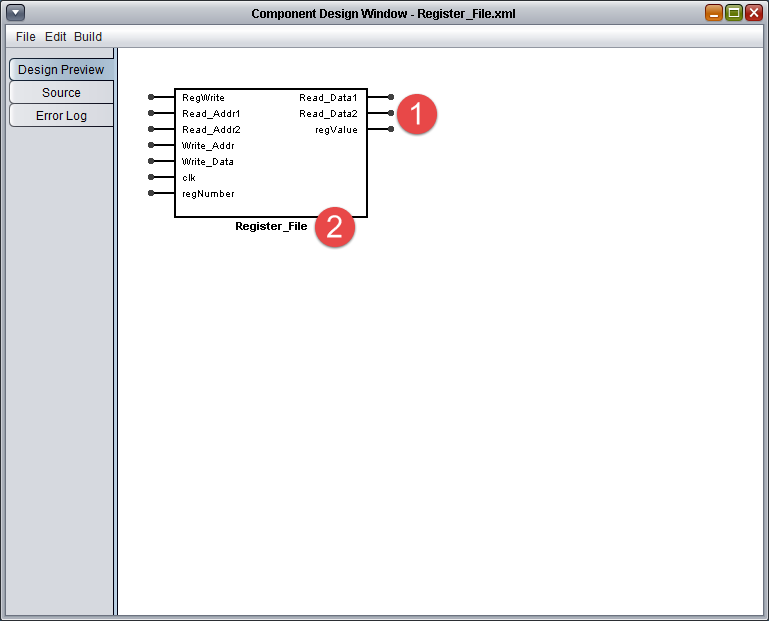


**Ilustración 34. Diseño de componente: Opciones de Edit**



**Ilustración 35. Diseño de componente: Opciones de Build**

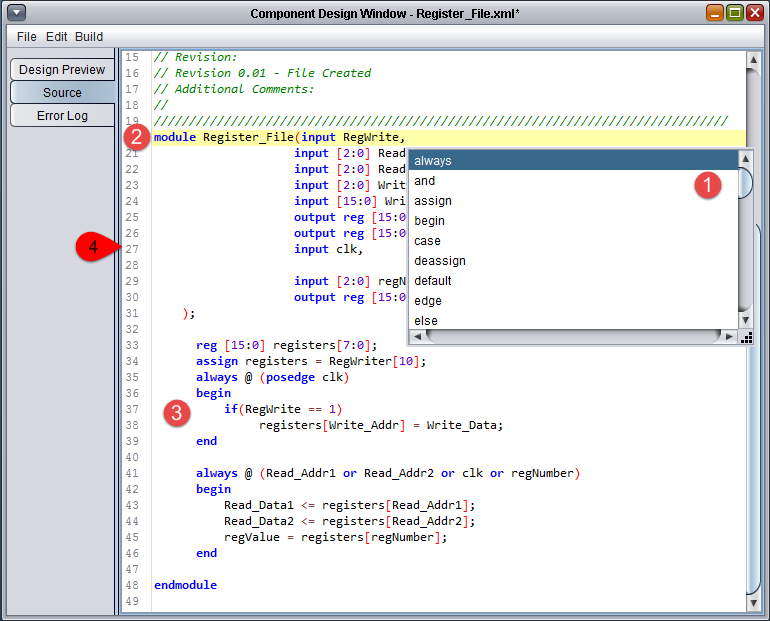
La sección de vista previa usualmente luce similar a la siguiente imagen:



**Ilustración 36. Diseño de componente: Vista Previa**

1. Por cada entrada y salida se genera un puerto en el dibujo del chip. Las entradas van a la izquierda y las salidas a la derecha. Estos puertos pueden moverse libremente a cualquiera de los cuatro lados del chip.
2. El nombre del módulo también puede moverse, pero el texto no puede invertirse ni rotar. Tampoco puede alejarse mucho del rectángulo del chip, puede colocarse un poco abajo del borde inferior o un poco arriba del superior.

Recompilar el código fuente vuelve este diagrama a su configuración original por lo que editar su posición debería ser el último paso en el proceso.

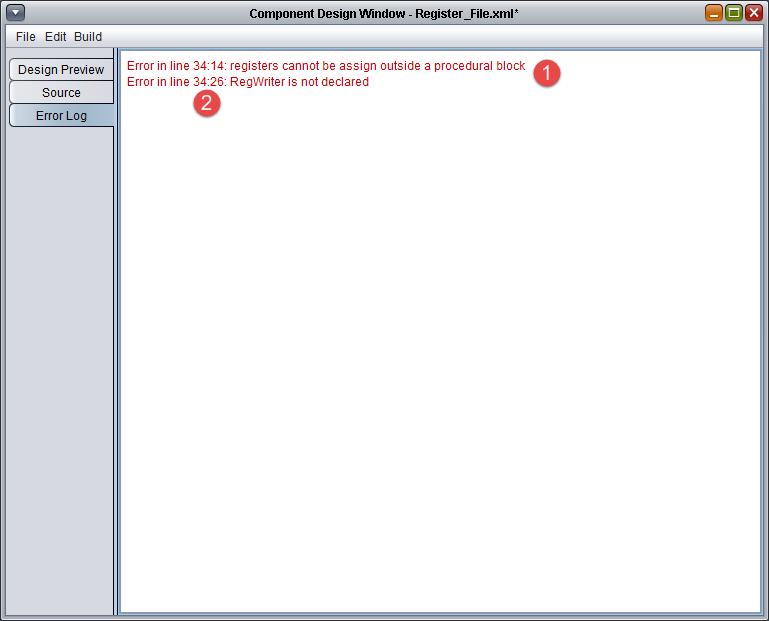


**Ilustración 37. Diseño de componente: definición de comportamiento usando Verilog**

1. Al momento de escribir código de Verilog se puede presionar la combinación de teclas Ctrl+Espacio para abrir una lista de autocompletado según se vaya escribiendo. Por el momento la autocompletación es solamente de palabras reservadas.
2. Para crear un esqueleto vacío de un nuevo módulo puede escribir “nmod” y presionar la combinación de teclas Ctrl+Shift+Espacio.
3. El editor cuenta con sobresaltado de palabras claves, de símbolos de agrupación como paréntesis, llaves y corchetes así como emparejamiento y sobresaltado especial de los mismos. Además se muestra marcada la línea actual.

El esquema de colores puede cambiarse a través de la ventana de Settings.

1. En el lado izquierdo del editor se encuentra una columna que muestra el número de línea para facilitar la ubicación de errores de compilación.



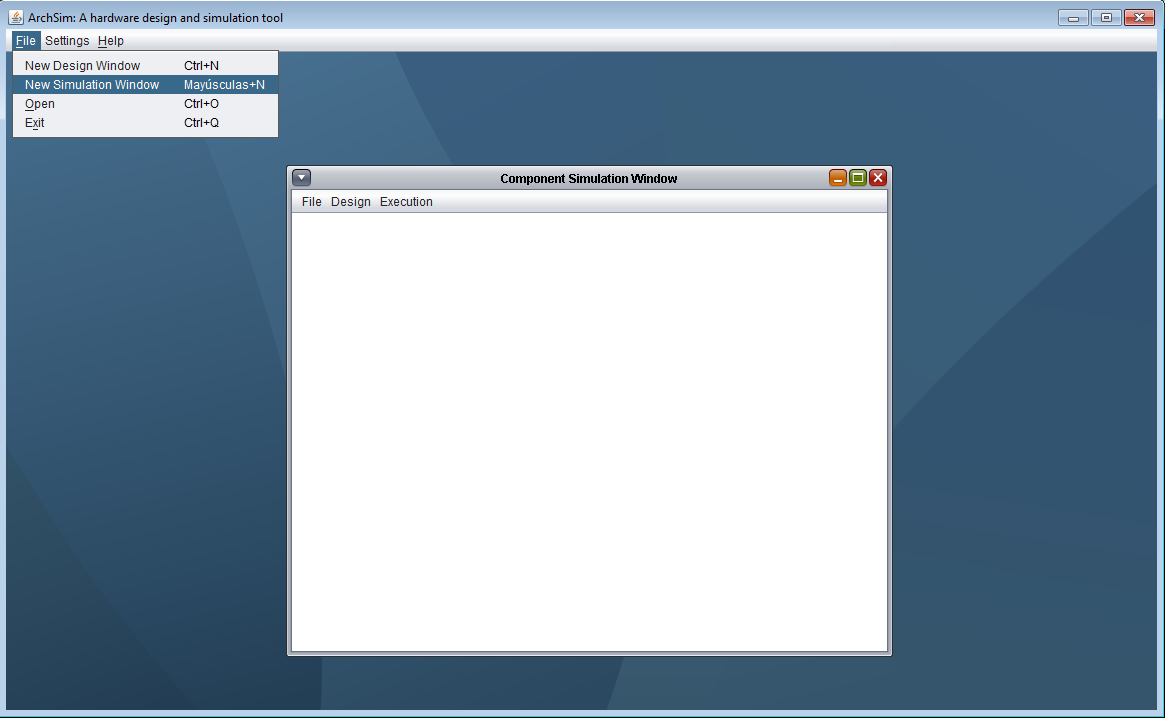
**Ilustración 38. Diseño de componente: Log de errores**

1. En esta área se muestran todos los errores que ocurrieron durante el proceso de compilación, sin borrar los anteriores.
2. Los errores muestran el número de línea y columna en la que sucedió dicho error.

El color rojo del texto no puede cambiarse a través de las configuraciones, el código fuente está disponible si se desea cambiar o implementar una forma de cambiarlo.

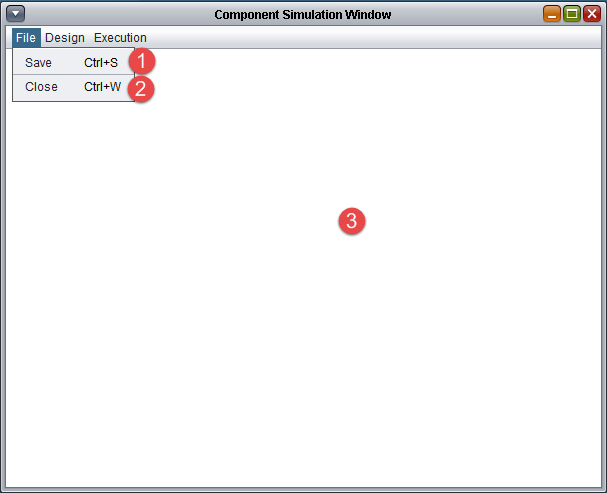
* + 1. Ventana De Simulación.

En esta ventana se combinan elementos electrónicos básicos como compuertas lógicas, multiplexores, entradas lógicas de uno o más bits y módulos definidos por el usuario usando la ventana de diseño.



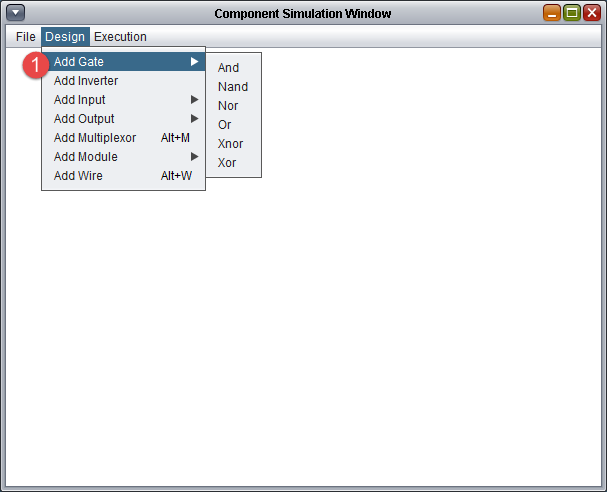
**Ilustración 39. Ventana de simulación**

La ventana cuenta con una barra de menús independientes a la ventana principal con las siguientes opciones.



**Ilustración 40. Ventana de simulación: Opciones de File**

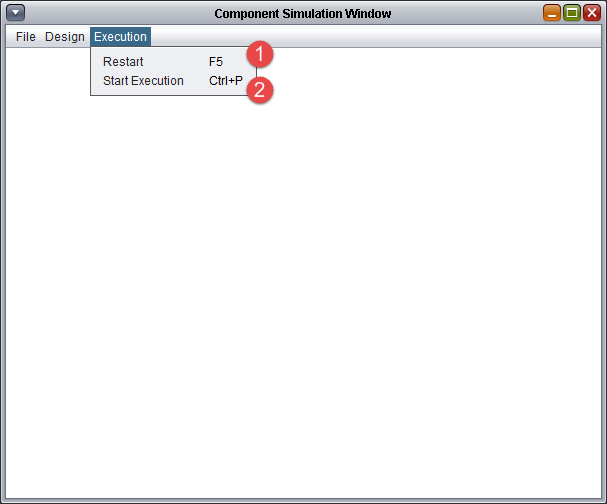
1. Permite guardar los elementos que se encuentran actualmente en la ventana de simulación en un formato que puede abrir posteriormente por esta misma aplicación.
2. Cierra la ventana, si hay elementos pregunta si se desea guardar antes de salir.



**Ilustración 41. Ventana de simulación: Opciones de Design**

1. En Design se encuentra la lista de elementos que pueden ser agregados a la simulación. Hay 7 diferentes compuertas (las 6 que se muestran y el Inverter, que es una compuerta Not); 2 posibles entradas, lógica de un bit y de múltiples bits; lo mismo para las salidas; multiplexores, cables y los módulos definidos por el usuario.

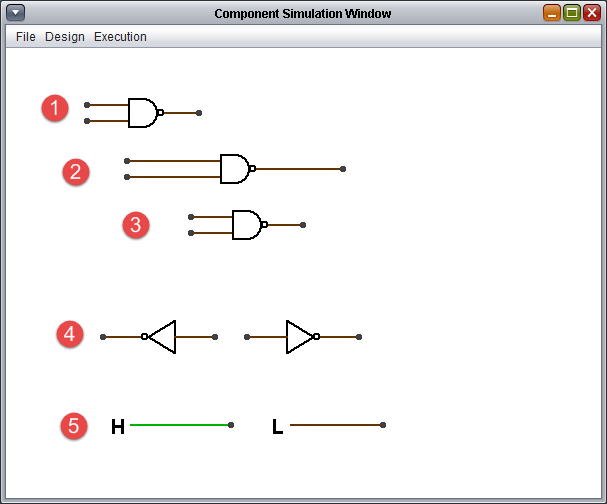
La lista de módulos se actualiza si se crea un nuevo módulo a través de una ventana de definición así que no es necesario reiniciar la aplicación para utilizar un módulo recién definido.



**Ilustración 42. Ventana de simulación: Opciones de Execution**

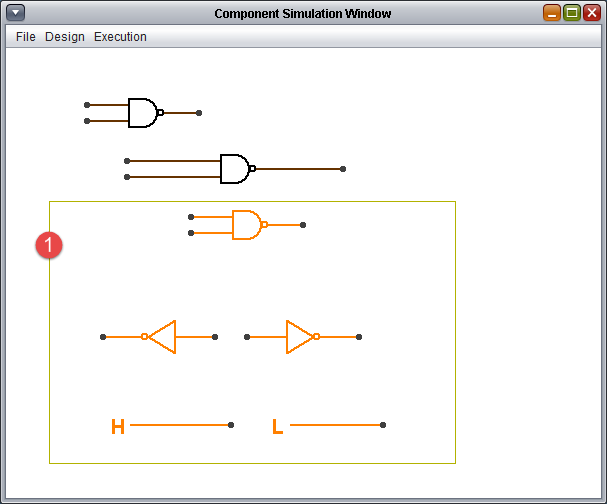
1. Permite hacer que la ejecución comience de nuevo, esto es de gran utilidad debido a que los módulos pueden contar con un bloque de inicialización (initial block) que solo se ejecuta una vez al iniciar la simulación.
2. La segunda opción es en realidad tres opciones en una, cuando la ejecución no ha iniciado aparece con el texto “Start Execution” y permite iniciar la simulación; una vez iniciado aparece con el texto “Pause” y sirve para pausar la simulación; finalmente, cuando está pausada el texto cambia a “Continue” y permite reanudar la simulación donde se había quedado.

Sobre los componentes de la ventana de simulación:



**Ilustración 43. Ventana de simulación: acciones sobre componentes**

1. Las entradas y salidas de los componentes pueden tener dos colores:
   1. Verde: significa que el valor en dicha conexión es un uno lógico, el valor del voltaje que representa puede modificarse en la ventana de Settings.
   2. Café: el valor es un cero lógico o es un valor de más de un bit.
2. El usuario puede cambiar el tamaño de los componentes, para ello seleccione un elemento posicionándose sobre él, mantenga presionado la tecla Ctrl y arrastre el cursor libremente. La aplicación identificará automáticamente el extremo más cercano al ratón y el extremo opuesto se mantiene inmóvil durante el cambio de tamaño.
3. Los componentes pueden moverse por toda el área de simulación. Esto se logra arrastrando el elemento con el botón izquierdo del ratón siendo presionado.
4. Los componentes pueden invertirse de dirección, simplemente debe mover un extremo, como cuando se cambia de tamaño un elemento, hacia la dirección opuesta.
5. Hay elementos que cambian su estado con solo ser presionados, uno de ellos es la entrada lógica de un bit. La entrada tiene un valor por defecto de uno o “high”, al dar click sobre él cambia a cero y viceversa.



**Ilustración 44. Ventana de simulación: selección en área**

Además de lo anterior, también puede seleccionarse varios elementos a la vez. Esto se logra arrastrando el puntero desde una sección del área de simulación que esté vacía y, manteniendo presionado mueva el puntero, el área seleccionada se marca por un rectángulo de color amarillo-verdoso.

Los elementos seleccionados, ya sea por selección de área o por posicionarse sobre él, se muestran de un color naranja.

Los componentes que están seleccionados por área se pueden mover de igual forma que se mueve un solo elemento.

Adicionalmente, hay una opción que permite mover todos los elementos a la vez, sin necesidad de seleccionarlos. Esto fue pensado para la reubicación de los elementos porque no todos pueden estar dentro del rango de visibilidad.

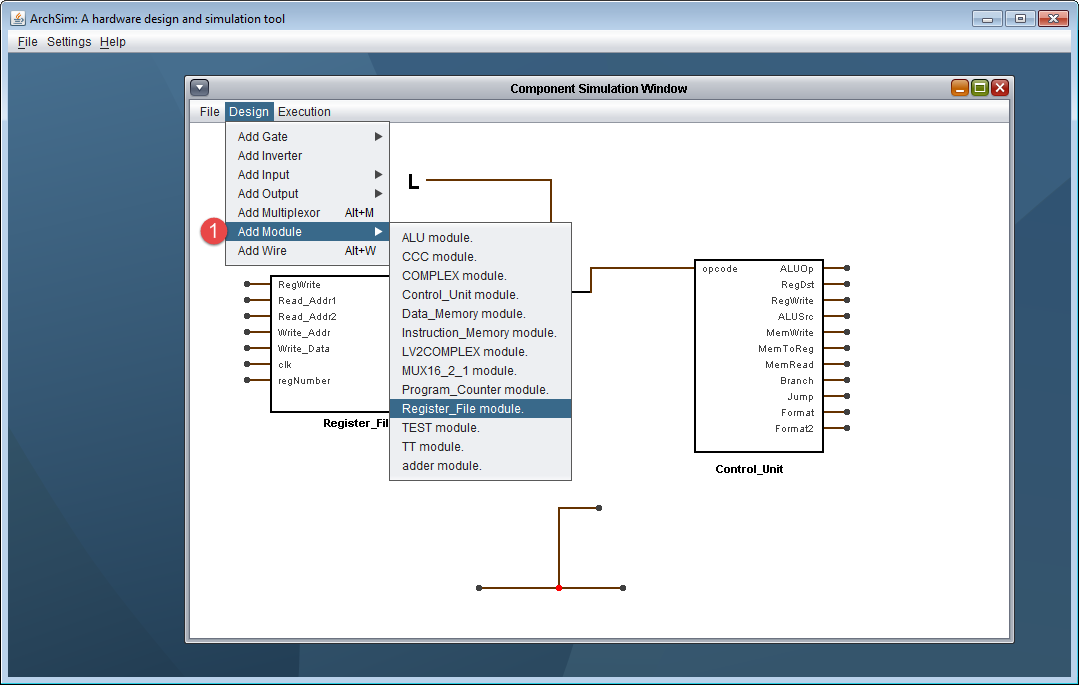
Se pueden arrastrar de esta manera manteniendo presionado la tecla Shift y arrastrando el puntero a voluntad.

Los elementos seleccionados en un área pueden borrarse con solo presionar la tecla “Delete” o “Supr”.

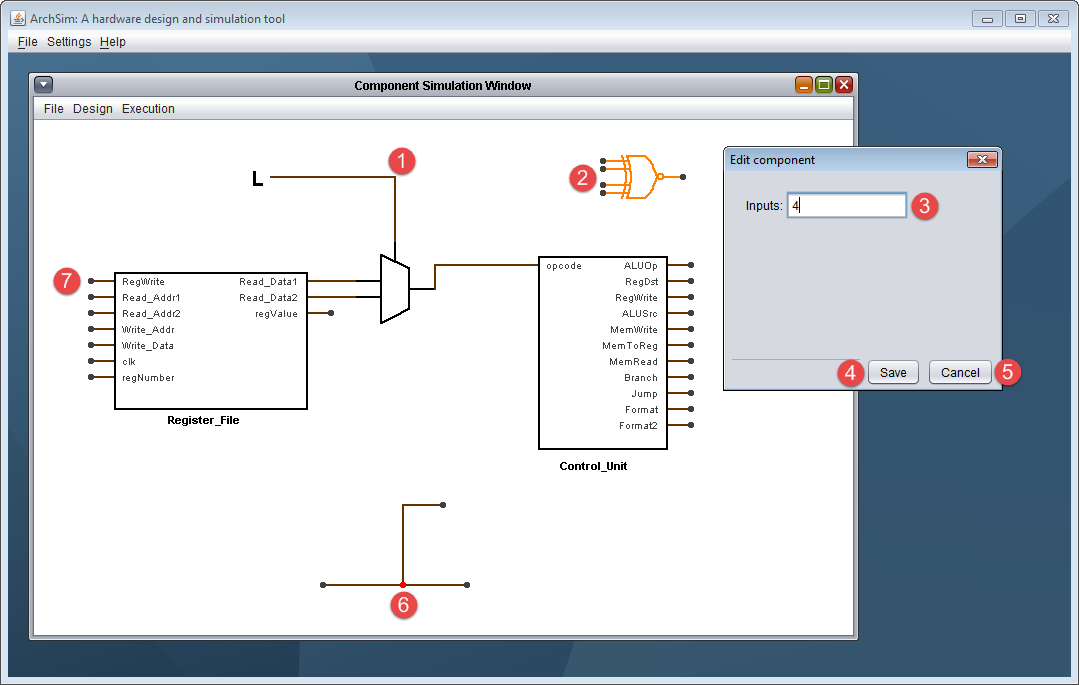
Si el puntero está sobre un elemento o un área vacía cuando se presiona la tecla Delete, el icono del puntero cambia a una mano, en este modo cualquier elemento que se presione será eliminado. Para salir de este modo es necesario presionar la tecla Esc.

Otra cosa que cabe mencionar es que los módulos no pueden cambiar de tamaño. El tamaño que adquieren depende de la longitud del nombre de sus puertos y no varía.

La lista de módulos se genera a partir de los elementos creados por el usuario.



**Ilustración 45. Ventana de simulación: módulos definidos por el usuario**

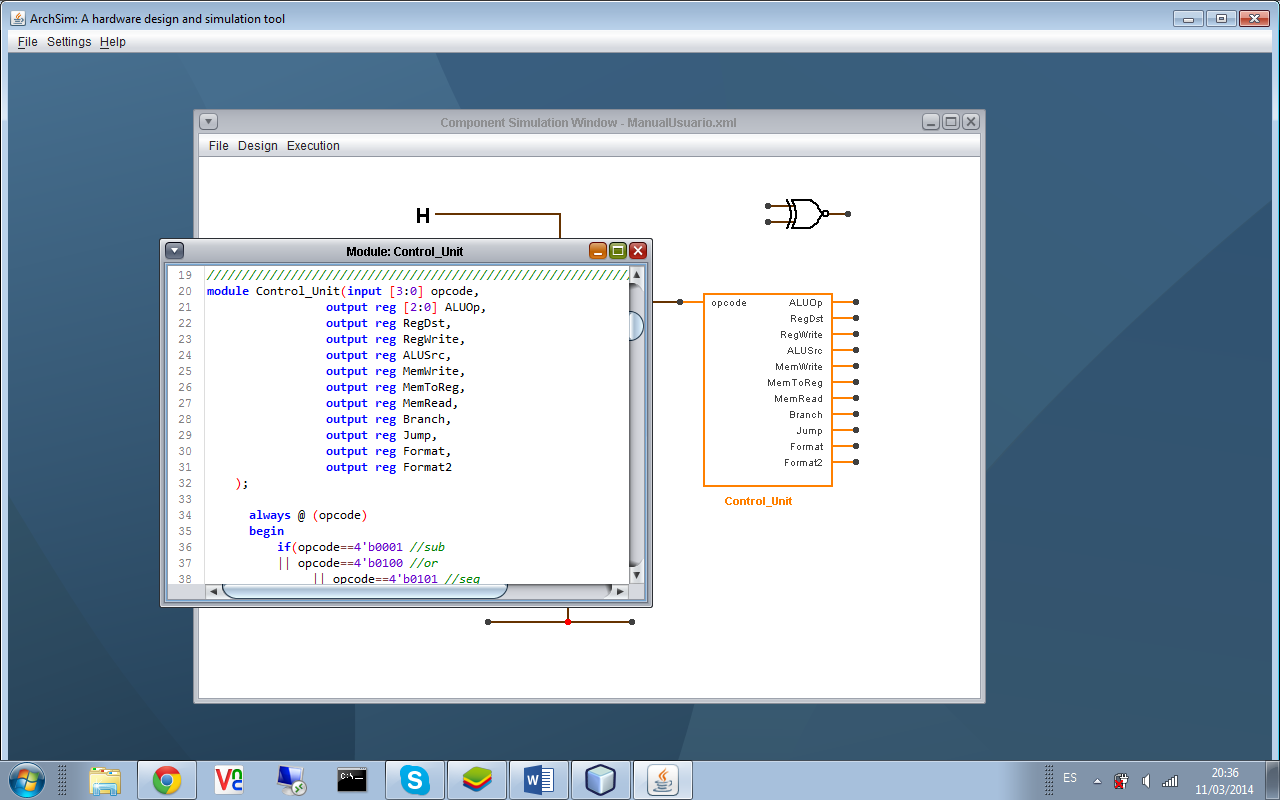


**Ilustración 46. Ventana de simulación: otras características**

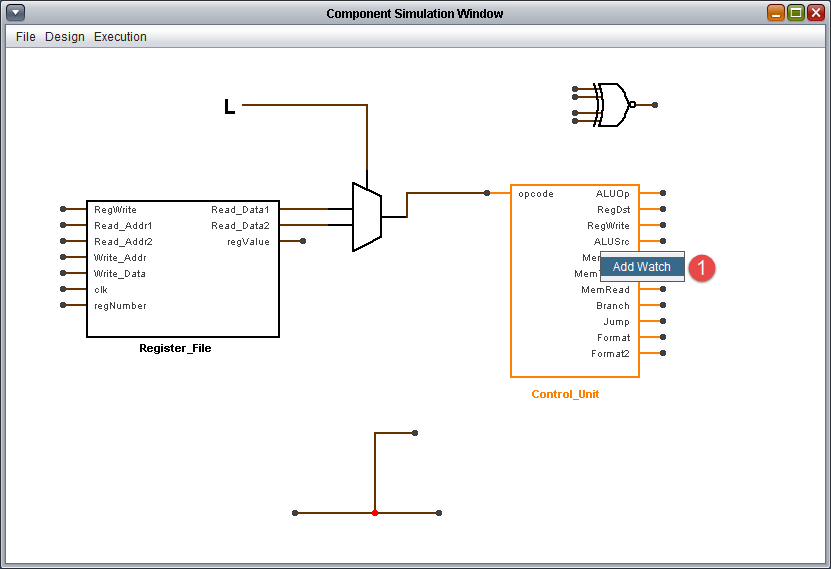
1. Un puerto o extremo no conectado siempre se muestra con un pequeño círculo negro en el extremo como se muestra cerca del marcador con el número 7. Cuando el extremo está correctamente conectado a otro el círculo desaparece para dar la impresión de continuidad. Además cuando un puerto está en contacto con otro componente pero no correctamente, el extremo se muestra en color rojo como en el marcador número 6 en la imagen.
2. Algunos elementos, como las compuertas (a excepción del Inversor) pueden tener más de dos entradas. Además hay otros componentes que pueden editarse, si bien la cantidad de entradas no es una de ellas, por ejemplo a los cables se les puede asociar qué porción de una señal de varios bits transmitirá; a una entrada de múltiples bits también se le puede modificar el valor que transmitirá.
   * 1. Para abrir la ventana de edición basta con dar doble click sobre el componente a editar. En los módulos, el comportamiento del doble click es otro, se explicará a detalle más adelante.
3. La ventana de edición cuenta con una serie de campos a editar, en la imagen se muestra la cantidad de entradas de una compuerta XNor. Una vez editado los valores puede usarse los botones Save o Cancel para aplicar o cancelar los cambios respectivamente.

Al dar doble click sobre un módulo puede suceder una de dos cosas:

1. Si el módulo no tiene instancias de módulos dentro de él lo que se muestra es el código de Verilog que define su comportamiento.
2. Si el módulo si tiene otros módulos instanciados lo que se mostrará es la conexión interna de esos componentes, esto es un trabajo en proceso y posiblemente no sea parte de la primera liberación.



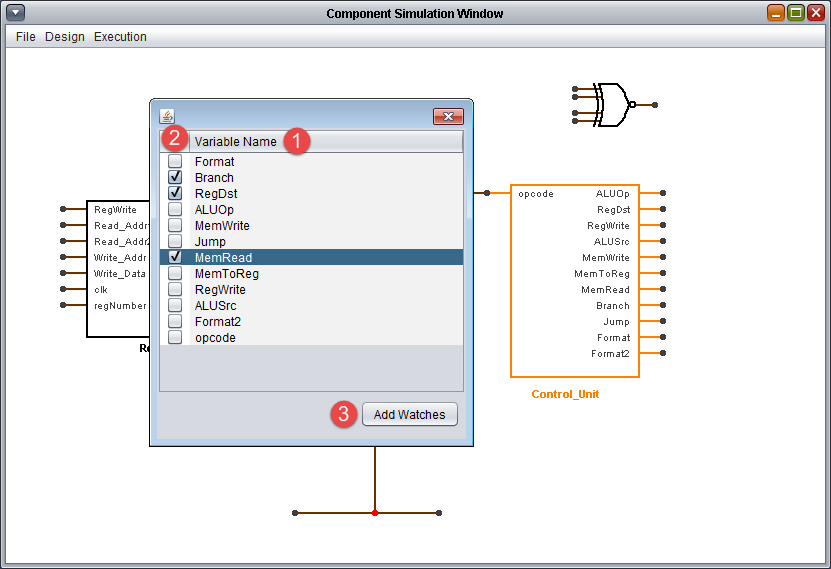
* + 1. Monitor De Variables.



**Ilustración 47. Ventana de simulación: Monitor de Variables**

Para utilizar el Monitor de Variables, debe haber al menos un módulo en la simulación.

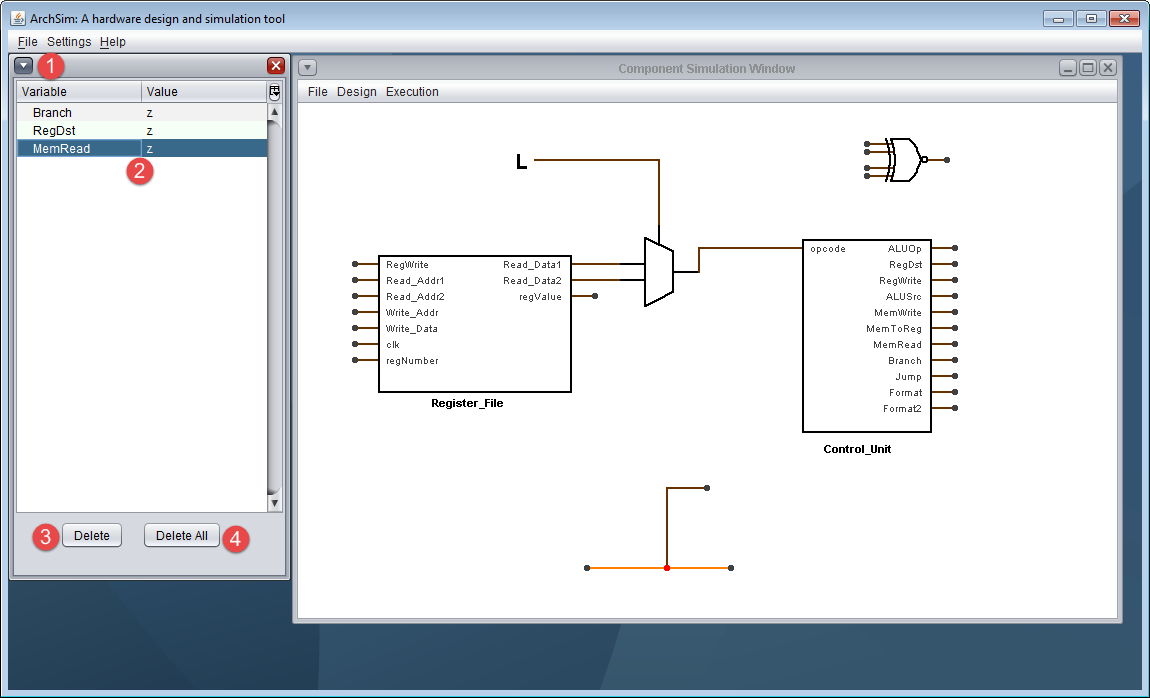
Al dar click derecho sobre alguno de los módulos aparece un menú contextual con una sola opción, por ahora, “Add Watch”. Al dar click en esa opción aparece la siguiente ventana.

La ventana muestra una lista de todos los puertos de ese módulo, y al lado de cada variable aparece una cajita para marcar si se quiere monitorear o no dicha variable.

**Ilustración 48. Selector de Variables**

Una vez seleccionada la lista de puertos a monitorear puede dar click en el botón “Add Watches”.

La primera vez, esta acción creará una nueva ventana como la que se muestra a continuación. Sino, las nuevas variables solo se agregan a la ventana ya existente.



**Ilustración 49. Monitor de Variables**

La ventana de monitoreo no es más que una tabla de dos columnas, el nombre de la variable y el valor de la misma.

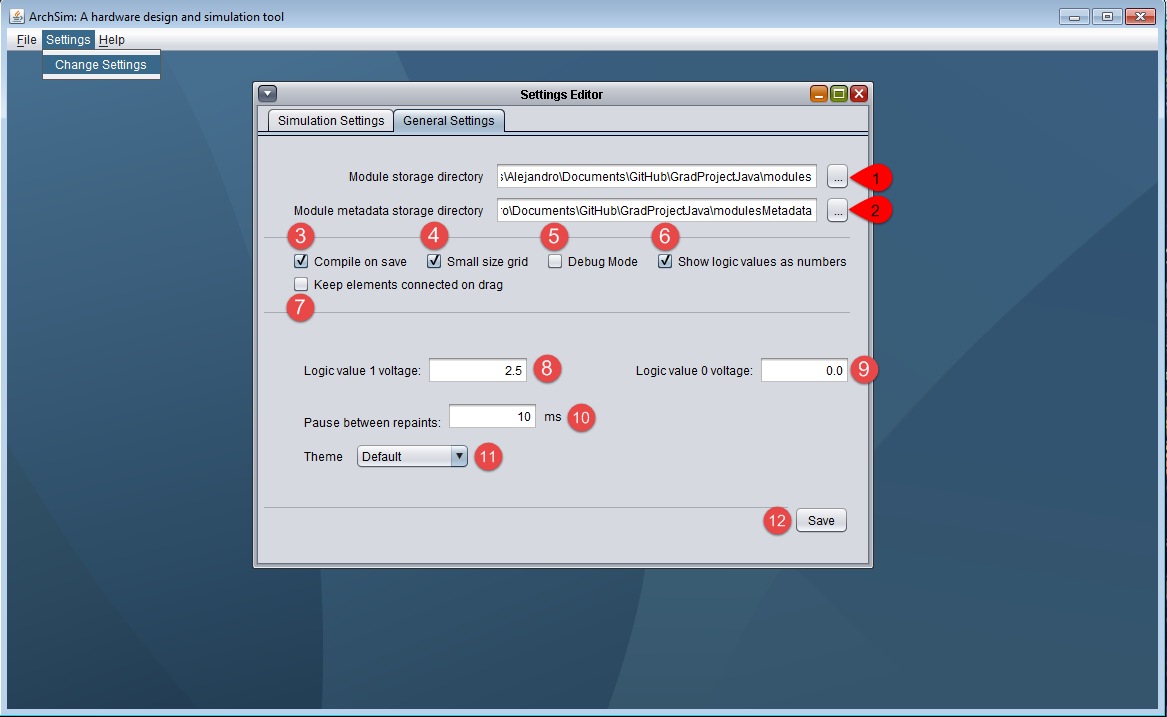
Pueden agregarse nuevas variables al Monitor de la misma forma que se hizo antes. Para dejar de monitorear alguna variable basta con seleccionarla y dar click en el botón Delete.

El botón Delete All borra deja de hace que se deje de monitorear todas las variables.

Es importante mencionar que los valores de las variables se actualizan automáticamente según se ejecuta la simulación, no hay necesidad de refrescar o hacer foco a la ventana.

* + 1. Editor De Configuraciones.

En esta ventana se pueden cambiar algunas configuraciones generales de la aplicación así como algunas relacionadas a la simulación y, posiblemente a futuro, configuraciones especializadas en la simulación.



**Ilustración 50. Editor de Configuraciones**

1. Este botón abre un dialogo para seleccionar una carpeta, en dicha carpeta se almacenará los archivos que definen un módulo que hayan sido definidos por el usuario. Si bien el archivo puede guardarse en cualquier ubicación, la carga de ellos para poder utilizarlos en la simulación solo se hace de los archivos que estén en la carpeta especificada aquí.
2. A decir verdad, la definición de un nuevo módulo consta de dos archivos, el primero tiene la definición y representación gráfica del componente y el segundo información de la cantidad de puertos, si son de entrada o salida y otra información similar que se conoce como metada. Note que este segundo archivo se crea con el mismo nombre del primero por lo que se aconseja que el directorio escogido no sea el mismo del inciso 1, sino puede darse comportamientos inesperados.
3. Si esta opción está marcada hará que cada vez que se guarde un módulo se compile el código de Verilog y se genere el diseño del componente. Compilar un componente hace que este esté disponible para la ventana de simulación.
4. La idea de esta opción era que hubiesen dos tamaños de elementos gráficos, aún no ha sido implementado así que esta opción, temporalmente, no hace nada.
5. Debug Mode hace que durante la ejecución se registre mucha de la información y mensajes de desarrollador en la salida estándar, si la aplicación fue abierta desde una terminal los mensajes se imprimirán ahí.
6. Por defecto los valores lógicos se muestran como una “H” para representar un “high” o alto (uno lógico) o “L” de “low” o bajo (cero lógico). Esta opción le dice a la ventana de simulación que en vez de mostrar las letras muestre los números.
7. Esta opción está en una fase beta, tenerla marcada causa que, cuando un componente conectado a otros se mueve por el área de simulación, las conexiones se ajustan para evitar perder el enlace entre los componentes.
8. Los valores especificados en esta opción así como en la siguiente actualmente no se utilizan, pero fueron pensadas para hacer posible la comunicación entre elementos lógicos y analógicos mediante una equivalencia entre valores digitales y reales. Actualmente no es de interés dado que todos los componentes que existen son lógicos.
9. Igual que el inciso anterior.
10. Determina el tiempo de espera entre un pintado y el siguiente en algunas circunstancias que la simulación lo requiere. Asignar valores muy grandes podría causar que la aplicación deje de responder.
11. El control lista una serie de temas (esquema de colores) predefinidos que pueden aplicarse a la ventana de diseño de componentes, específicamente al área de texto. Note que el esquema se aplica a las ventanas abiertas y no a las que se van creando. Aún se está considerando hacer el cambio en este comportamiento.