tabla de contenido.

[I. RESUMEN EJECUTIVO 2](#_Toc381382866)

[II. INTRODUCCION 3](#_Toc381382867)

[III. OBJETIVOS. 4](#_Toc381382868)

[**3.1** **OBJETIVO GENERAL** 4](#_Toc381382869)

[**3.2** **OBJETIVOS ESPECIFICOS.** 4](#_Toc381382870)

[IV. MARCO TEORICO. 5](#_Toc381382871)

[**4.1** **SOBRE LA SIMULACIÓN.** 5](#_Toc381382872)

[4.1.1 CONCEPTOS. 5](#_Toc381382873)

[4.1.2 SWING FRAMEWORK. 5](#_Toc381382874)

[**4.2** **DISEÑO DE HARDWARE.** 7](#_Toc381382875)

[4.2.1 LENGUAJES DE DESCRIPCIÓN DE HARDWARE. 8](#_Toc381382876)

[**4.3** **SOBRE ELECTRÓNICA.** 11](#_Toc381382877)

[4.3.1 CONCEPTOS. 11](#_Toc381382878)

[4.3.2 TABLAS DE VERDAD. 13](#_Toc381382879)

[4.3.3 SIMBOLOGÍA. 15](#_Toc381382880)

[**4.4** **SOBRE EL DESARROLLO.** 17](#_Toc381382881)

[4.4.1 METODOLOGÍA. 17](#_Toc381382882)

[4.4.1.1 METODOLOGÍAS AGILES. 17](#_Toc381382883)

[4.4.1.2 Scrum. 18](#_Toc381382884)

[4.4.2 SOBRE EL COMPILADOR. 20](#_Toc381382885)

[4.4.2.1 JFlex 23](#_Toc381382886)

[4.4.2.2 Cup 24](#_Toc381382887)

[4.4.3 OTRAS HERRAMIENTAS Y TECNOLOGÍAS. 25](#_Toc381382888)

[4.4.3.1 RSyntaxTextArea 25](#_Toc381382889)

[4.4.3.2 XML. 26](#_Toc381382890)

[**4.5** **ANTECEDENTES.** 28](#_Toc381382891)

[4.5.1 XILINX. 28](#_Toc381382892)

[4.5.2 ALTERA. 29](#_Toc381382893)

[V. CONCLUSIONES. 30](#_Toc381382894)

[VI. RECOMENDACIONES. 31](#_Toc381382895)

[VII. BIBLIOGRAFÍA 32](#_Toc381382896)

[VIII. ANEXOS. 32](#_Toc381382897)

1. **RESUMEN EJECUTIVO**
2. **INTRODUCCION**
3. **OBJETIVOS.**
   1. **OBJETIVO GENERAL**

Proveer una herramienta para simulación de diseño de hardware más 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 en la que se pueda definir componentes de hardware y simular su comportamiento, sin necesidad de una segunda aplicación.
2. 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.
3. Encuestar a alumnos o exalumnos que han tenido la oportunidad de utilizar las herramientas actuales de diseño de hardware para que las comparen con la aplicación desarrollada.
4. **MARCO TEORICO.**
   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:

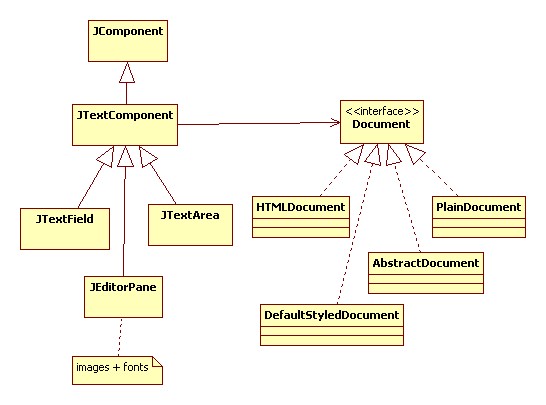


Ilustración 1. Diagrama de clases JComponent

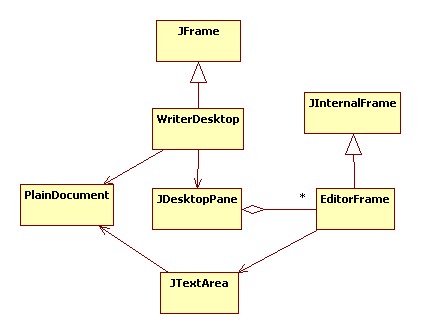


Ilustración 2. Jerarquía de clases de JFrame

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.

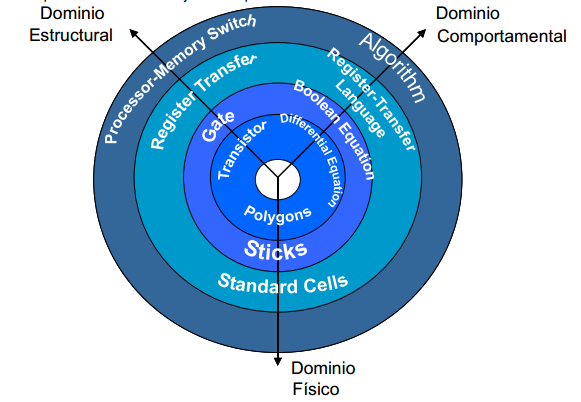


Ilustración 3. Niveles de abstracción en el diseño de descripción de hardware.

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).

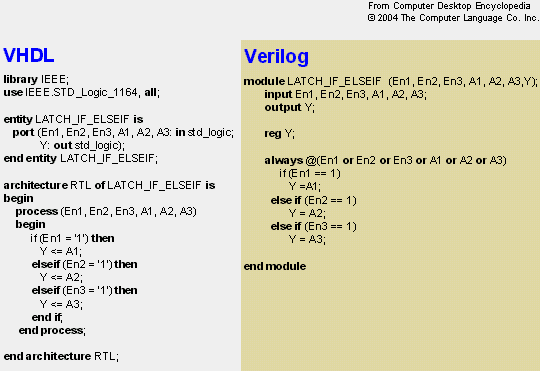


Ilustración 4. Imagen comparativa entre sintaxis de Verilog y VHDL

* 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.

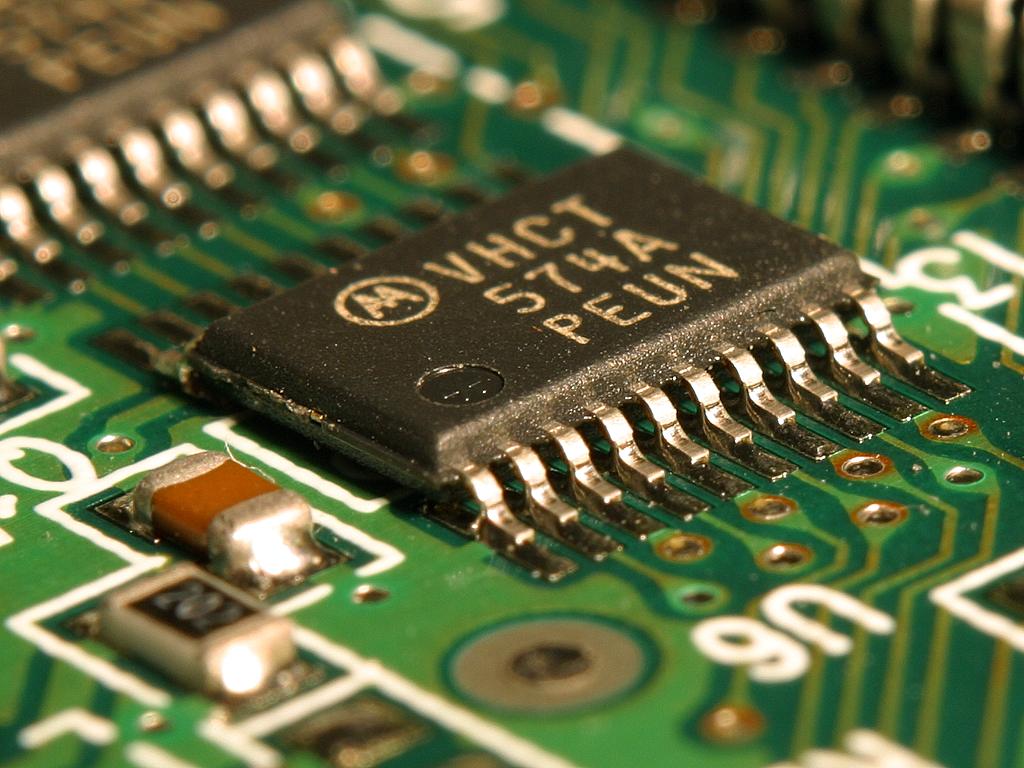


Ilustración 5. Circuito Integrado.

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:

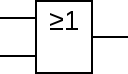


Ilustración 8. Símbolo IEC para una compuerta O

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:

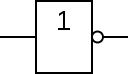


Ilustración 12. Símbolo IEC para una compuerta Not

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:

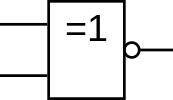


Ilustración 16. Símbolo IEC para una compuerta Xnor

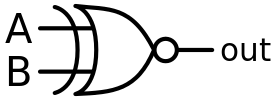


Ilustración 17. Símbolo ANSI para una compuerta Xnor

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.

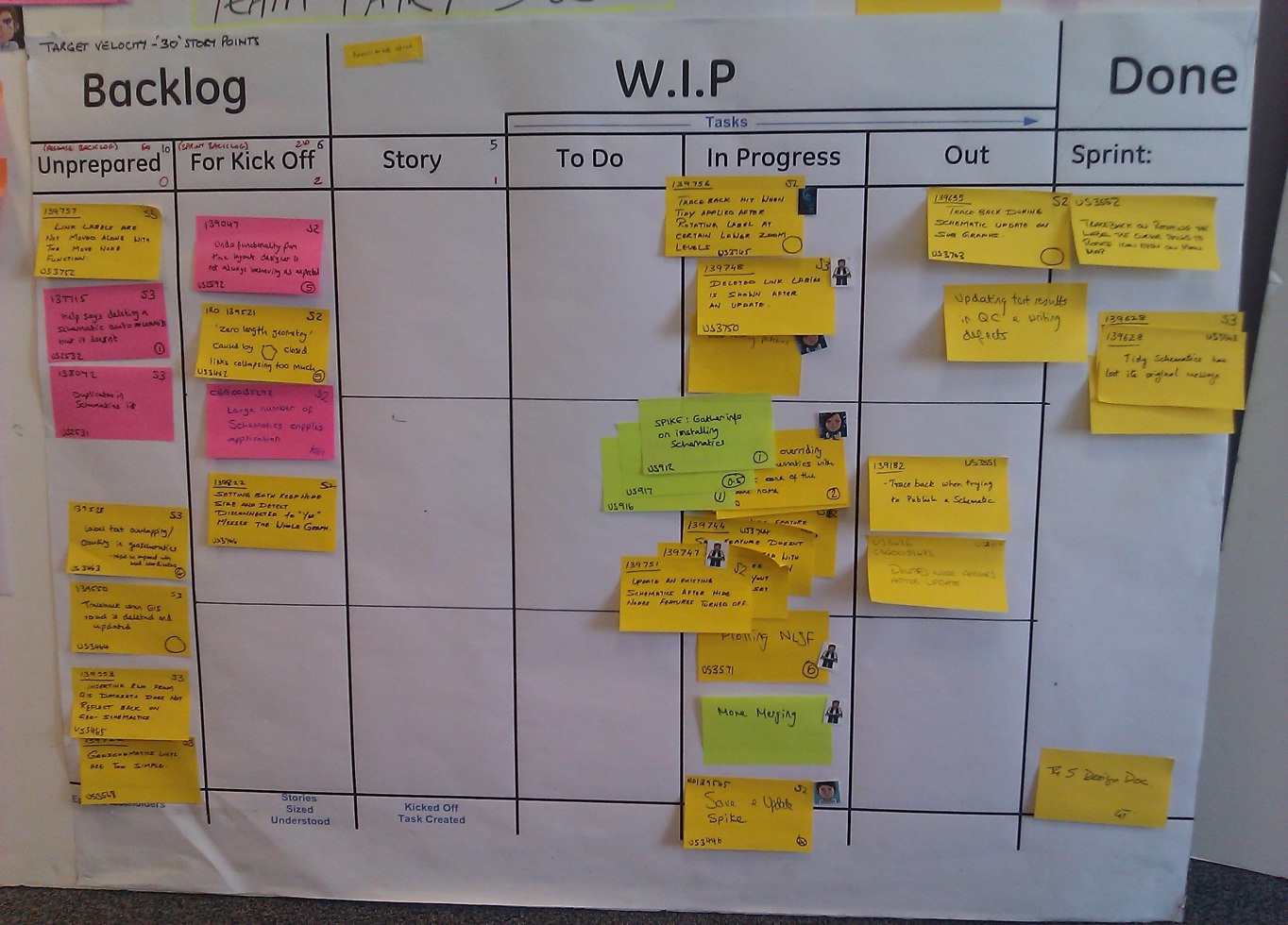


Ilustración 20. Ejemplo de una pizarra de Scrum

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>)

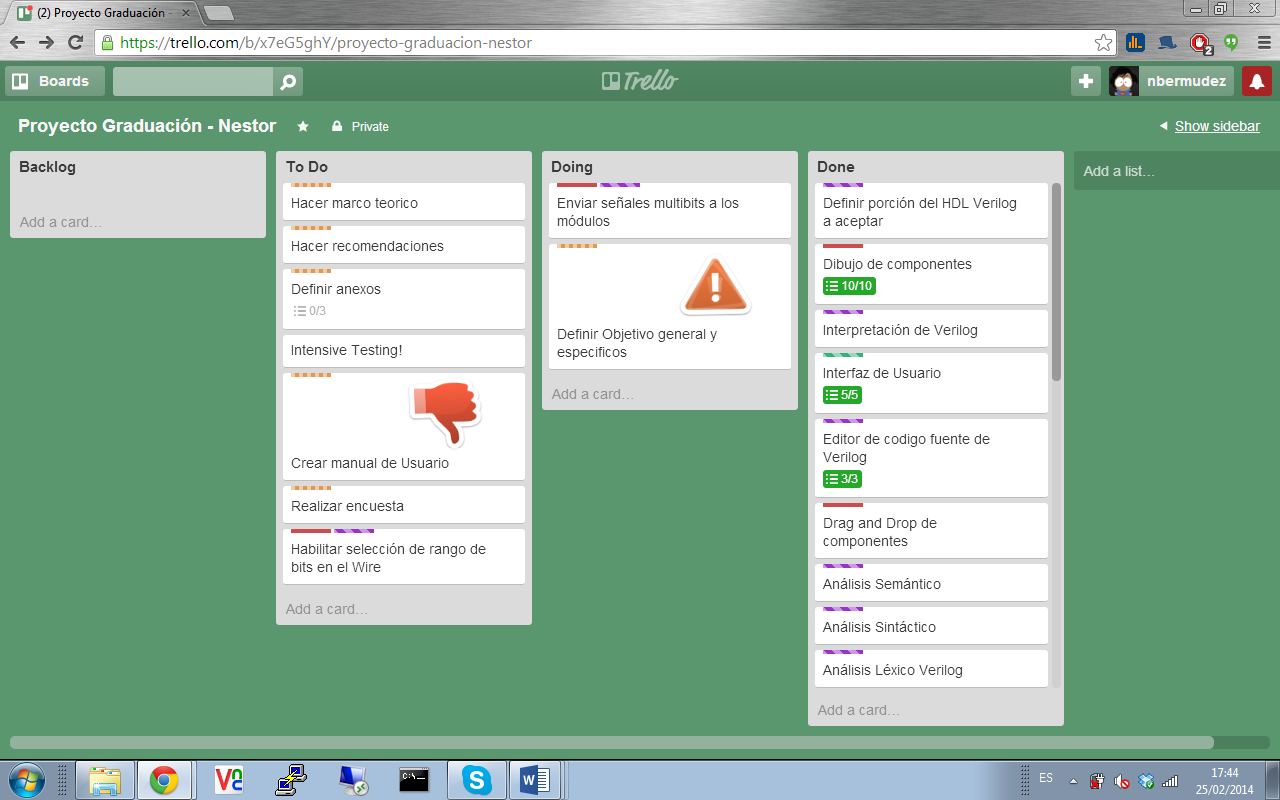


Ilustración 21. Captura de Pantalla de la pizarra de este proyecto

* + 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.

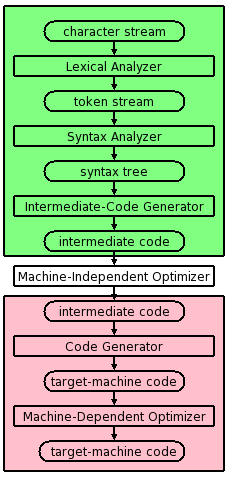


Ilustración 22. Etapas del proceso de compilación.

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.

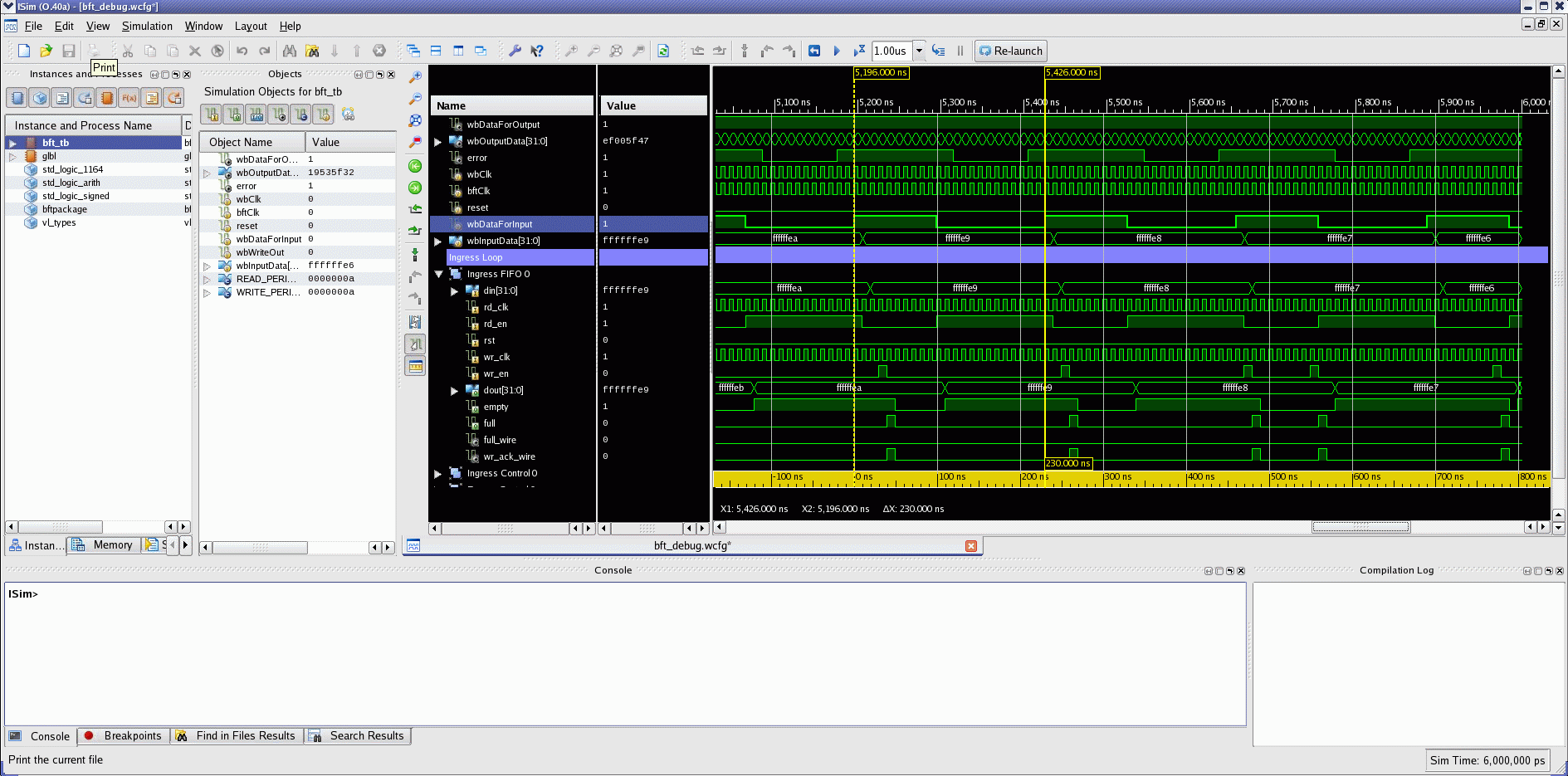


Ilustración 24. Interfaz gráfica de ISim

* + 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:

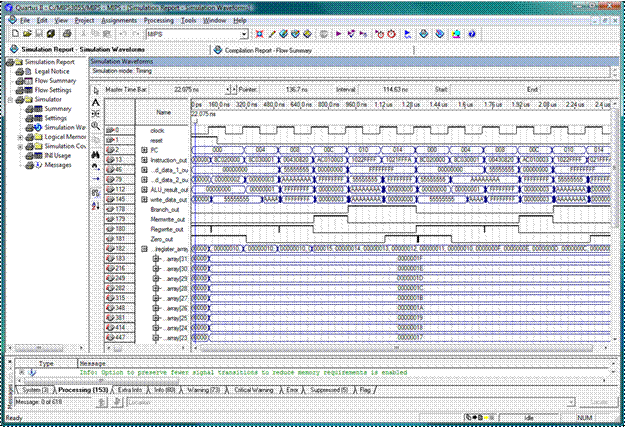


Ilustración 25. Interfaz de simulación de Quartus II

1. **CONCLUSIONES.**
2. **RECOMENDACIONES.**

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.

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

A quienes quieran continuar el proyecto: escriban código fácil de entender siempre que sea posible, nombren las variables con el amor con que nombrarían un hijo y nunca piensen en que serán las únicas personas que verán su código.

# **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: CARACTERISTICAS DE VERILOG ACEPTADAS.**
   2. **ANEXO B: TABLAS DE VERDAD DE COMPUERTAS LOGICAS.**
   3. **ANEXO C: ESTRUCTURAS XML.**
   4. **ANEXO D: ENCUESTA.**