Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Telecomunicaciones y Electrónica

TRABAJO DE DIPLOMA

Simulación de Circuitos Digitales con Software Libre

Autor: Yuniel Freire Hernández

Tutor: Ing. Erisbel Orozco Crespo

Santa Clara

2012

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Telecomunicaciones y Electrónica

TRABAJO DE DIPLOMA

Simulación de Circuitos Digitales con Software Libre

Autor: Yuniel Freire Hernández

Tutor: Ing. Erisbel Orozco Crespo

Profesor, Dpto. Telec. Y Electrónica

Santa Clara

2012

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Telecomunicaciones y Electrónica, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

_____

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del Tutor

Firma del Jefe de Departamento donde se defiende el trabajo

_____

Firma del Responsable de Información Científico-Técnica

PENSAMIENTO

No fracasé, sólo descubrí 999 maneras de cómo no hacer una bombilla.

Thomas Alva Edison

AGRADECIMIENTOS

A todos los profesores que brindaron sus conocimientos para mi formación.

A mi tutor por su paciencia, esmero y dedicación.

A mis amigos y compañeros por resistirme.

DEDICATORIA

A mis padres que siempre tuvieron la convicción de, que este sueño se convertiría en realidad. Gracias por darme el privilegio de ser su hijo.

TAREA TÉCNICA

1. Hacer una búsqueda bibliográfica sobre el tema del software libre para electrónica; específicamente para electrónica digital.
2. Realizar un estudio sobre los programas de software libre disponibles para simular circuitos digitales y caracterizarlos.
3. Elegir problemas de índole combinacional y secuencial diversos para simular con algunas de los programas estudiados.
4. Discutir sobre los resultados y las experiencias de las simulaciones.

Firma del Autor

Firma del Tutor

RESUMEN

Este trabajo contiene un estudio sobre simuladores de circuitos digitales con programas de software libre como TKGate y Qucs. Para ello se eligieron las versiones de estas herramientas presentes en la distribución de Linux Ubuntu 10.04 LTS. Fueron elegidos varios casos de circuitos combinaciones y secuenciales realizados a compuertas para su simulación. Además se modelaron y simularon diseños modulares simples, para conformar un criterio sobre las posibilidades de los programas en estudio para este tipo de tareas. Por último se simularon dos casos con lenguaje de descripción de hardware VHDL. Los resultados revelan de forma general que los programas son una alternativa viable para el diseño conceptual y el análisis de circuitos digitales. No obstante las limitaciones de estos programas, sobre todo del TKGate son considerables en comparación con productos de software propietario.

ÍNDICE

INTRODUCCIÓN

En Cuba se comenzó a dar seguimiento hace algunos años al tema del software libre, debido en gran medida, a que el país se ha visto imposibilitado de comprar las licencias en el mercado por estar en la lista de países embargados, como consecuencia del férreo bloqueo que Estados Unidos le ha impuesto a la Isla por más de 40 años. En la actualidad en la Universidad Central "Marta Abreu" de las Villas el software libre gana cada día más terreno y ya se encuentra diseminado en distintas aplicaciones y servicios. Muestra de ello es el uso de Apache en los servidores Web.

Quizás uno de los puntos neurálgicos afectados con este embargo, es la utilización de software propietario para la enseñanza. Hasta este punto se han desarrollado diversos programas con licencia GPL para el diseño digital como gEDA, Qucs, FreeHDL, TKgate y Confluence.

Este estudio está motivado por la necesidad de evaluar las posibilidades del software libre en el tratamiento de electrónica, sobre todo circuitos digitales. Además, es difícil encontrar una caracterización que abarque estas herramientas.

Se sabe sobre la importancia cada vez mayor que las herramientas informáticas con licencia GPL (Licencia Publica General) tienen, en particular en el ámbito educativo. Actualmente se pueden hacer las mismas tareas con ambos tipos de software, libre y privativo; aunque se califiquen uno mejor que otro indistintamente según el caso. El software libre, por lo general, tiene un menor consumo de recursos del ordenador, por lo que no se necesita el último ordenador del mercado ni el más costoso para utilizarlo.

Sería muy interesante disponer de herramientas maduras y completamente funcionales en plataformas de Linux, por el ahorro en coste que suponen para los que sepan aprovechar esta oportunidad. Haría falta responder ¿en qué medida estas herramientas de software están a la altura de las necesidades de los especialistas y estudiantes de la electrónica?

Se sabe de la existencia de software propietario con un prestigio tremendo en este campo. Tal es el caso del Cadence Encounter RTL Compiler el cual, junto con otros productos de Cadence, ha ganado varios premios de gran prestigio como herramientas EDA. Esto no es cuestionable, pero el acceso a este tipo de material no es posible en nuestro entorno por el momento y mucho menos de forma legal. Esto sucede de forma similar con otros productos. Entonces: ¿pueden las herramientas de software libre, en su estado actual, brindar alternativas factibles para la simulación de circuitos digitales?

De ser así: ¿cuáles herramientas de software libre para la electrónica digital se ajustan mejor y a qué tipo de tareas? ¿En qué medida pueden competir con el software propietario? ¿Cuáles programas son más apropiados y para qué casos? ¿Qué críticas podrían formularse sobre deficiencias medulares en estas herramientas?

Por eso el objetivo general de este trabajo es determinar el estado actual del software libre para su empleo como herramienta para electrónica digital, mediante la simulación de circuitos varios. Los objetivos específicos con:

- Describir el estado actual del diseño y el análisis de circuitos digitales con herramientas computacionales.
- Determinar el grado de desarrollo del software libre en el área de la Electrónica Digital.
- Caracterizar los distintos productos de software libre disponibles, en función de su idoneidad para ciertos casos de estudio.
- Obtener una evaluación experta, basada en el trabajo con diferentes casos de estudio, de cada una de las herramientas tratadas.

El cuerpo fundamental de la tesis está compuesto por tres capítulos, cuyo contenido se resume a continuación:

Capítulo 1: Software para electrónica

En este se expondrá el estado actual del software para electrónica digital, con especial atención al software libre. Se mencionarán algunos conceptos fundamentales sobre el software para la electrónica, su desarrollo a lo largo de los últimos años y los principales eventos al respecto en el mundo.

Capítulo 2: Software libre para electrónica digital

Se dedicará a la caracterización de los programas encontrados, según los materiales encontrados sobre los mismos en la bibliografía. Además expondrá algunos casos de estudio que serán simulados con algunos de las herramientas para su posterior evaluación.

Capítulo 3: Descripción del diseño y montaje

Este capítulo revelará los resultados de cada uno de los programas en virtud del caso de estudio escogido. A partir de estos resultados, se hará una evaluación de cada uno de las herramientas de simulación que permitirá: compararlos unos con otros, corroborar la caracterización del Capítulo 2, determinar su viabilidad para el empleo en la academia y en la práctica, etc.

El trabajo en general comenta la experiencia del autor como usuario de los programas para la simulación de circuitos digitales.

CAPÍTULO 1. SOFTWARE PARA ELECTRÓNICA

El desarrollo de un proyecto electrónico de cualquier tipo, tiene que estar acompañado de la utilización de software. Lo que lo hace imprescindible no es la incapacidad de los ingenieros, sino el hecho de contar con la herramienta adecuada. Una herramienta EDA presente en la metodología para el diseño o el análisis electrónico, a cualquier nivel de complejidad es fundamental para los resultados.

1.1 Concepto de EDA

Electronic Design Automation es la categoría de herramientas para el diseño y producción de sistemas electrónicos basados en circuitos impresos o *Printed Circuit Boards* (PCBs). En algunos casos se refiere al Diseño Electrónico Asistido por Computadora o *Electronic Computer Aided Design* (ECAD), o solo CAD. (Birnbaum, 2004)

1.1.1 Las décadas de los 70's y los 80's

Antes de las herramientas EDA, los circuitos integrados eran diseñados manualmente, y por supuesto esta técnica sucumbió ante el desarrollo de EDA. Algunas tiendas avanzadas usaron software geométrico para generar grabaciones en cinta magnética para el Gerber Photoplotter, pero las grabaciones no eran más que copias de componentes mecánicamente dibujados. El proceso era fundamentalmente gráfico, con la traducción manual de electrónica en gráficos. La compañía más relevante en este sentido fue Calma, que realizó el formato GDSII que existe en la actualidad. (Bozdoc, 2004)

A mediados de los 70's, los desarrolladores automatizaron no solo la construcción de PCBs, sino también el propio diseño del sistema electrónico. Los aportes más significativos están recogidos en la Conferencia de Diseño Automático (*Design Automation Conference*), celebrada en esta etapa.

A continuación tuvo lugar la publicación de *Introduction to VLSI Systems* por Carver Meady Lynn Conway,en 1980. El año 1981 marca los comienzos de EDA como empresa.

Por largos años, las mayores compañías electrónicas, como Hewlett Packard, Tektronix e Intel, habían buscado intensamente automatizar sus diseños. En 1981, algunos gerentes y desarrolladores se salieron de estas compañías para concentrarse en EDA como un negocio. (Birnbaum, 2004)

Al cabo de algunos años hubo muchas compañías especializadas en EDA, cada una con un énfasis ligeramente diferente. Algunas de estas fueron *Daisy Systems*, *Mentor Graphics* y *Valid Logic Systems*. La Conferencia de Diseño Automático de 1984, albergó quizás el primer acercamiento en cuanto al desarrollo de estas herramientas como producto.

1.1.2 Orientación del software EDA

Las principales áreas de desarrollo de las herramientas de software para el diseño electrónico se centraron en los siguientes puntos:

Síntesis de alto nivel para chips digitales (síntesis basada en comportamiento y síntesis algorítmica).

Síntesis lógica, que parte de un alto nivel de abstracción. Utiliza lenguajes de descripción de hardware como Verilog o VHDL para traducirlos en un arreglo discreto de compuertas lógicas.

La captura esquemática, quizás la más familiar para los no expertos en el tema. Se basa en la representación esquemática del sistema electrónico.

Síntesis automática de PCBs, por ejemplo: OrCAD Layout por Cadence, ARES en Proteus.

Por otro lado está el software de simulación. Puede parecer que la simulación está fuera del alcance del diseño electrónico, algo que es incorrecto. En la mayoría de los procedimientos de diseño más confiables, se simulan al menos secciones importantes con tal de analizar el comportamiento del diseño y evaluar su efectividad antes de la puesta en producción. Algunos tipos de simulación de circuitos electrónicos son:

Simulación de transistores, que no es más que una simulación de bajo nivel de un esquemático o *layout* cuyos componentes fundamentales son transistores. El alcance de precisión está dado por los modelos de dispositivos.

Simulación lógica: simulación del comportamiento lógico de un RTL o arreglo de compuertas lógicas.

Emulación de hardware: utilización de un hardware de propósito específico para emular un diseño dado. En muchos casos se adjunta al sistema en el lugar del circuito que será construido, para emular el funcionamiento; esto se denomina *in-circuit emulation*.

Technology CAD: simula la tecnología subyacente en cada proceso. Las propiedades eléctricas de los dispositivos, se derivan directamente de su modelo físico.

Solucionadores de campos electromagnéticos: resuelven el sistema de ecuaciones de Maxwell directamente para casos de interés de circuitos integrados y circuitos impresos. Son más lentos pero más exactos; mucho más convenientes para circuitos de RF de alta escala de integración que el software más convencional para extracción de *layout*.

Otros destinos que recibe el software EDA están dados en el análisis y verificación y en la asistencia para producción.

1.2 Flujo de diseño EDA

El flujo de diseño EDA es una combinación de herramientas de diseño automatizado con los procedimientos asociados para completar la realización de un circuito integrado. En gran medida el flujo de diseño está en función del propósito. Es común encontrar en la bibliografía el referente al flujo de diseño para la obtención de un circuito integrado, sin embargo está claro que existen otros procedimientos para el desarrollo con FPGA y PLDs en general por solo citar un ejemplo (Figura 1.1).

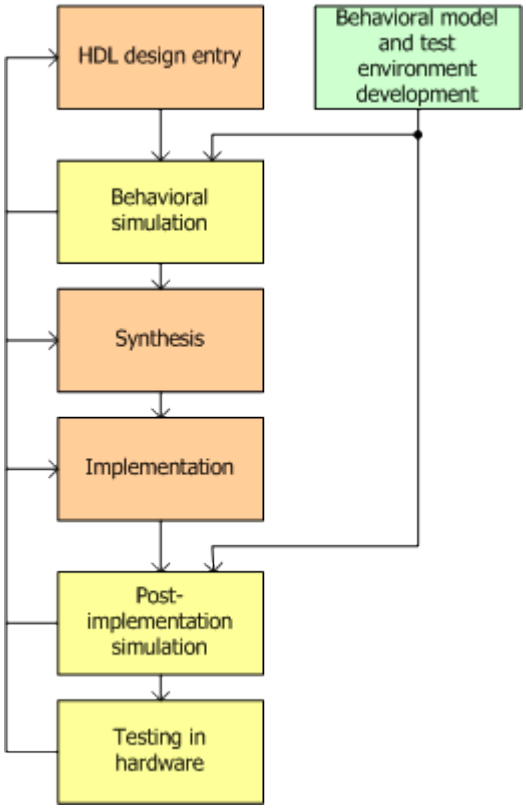


Figura 1.1 Ejemplo de un flujo de diseño con FPGA. Tomado de (TECHNOLOGIES, 2009).

Además de esto los pasos no siempre fueron los mismos. Por ejemplo, en flujo de diseño para llevar de RTL a GDSII, los pasos del año 1980 cambiaron varias veces hasta la actualidad, sobre todo en función del escalado de las tecnologías CMOS. Desafíos como fugas de energía, la variabilidad y la fiabilidad seguirán exigiendo cambios significativos en el proceso de terminación de circuitos integrados.

Otro aspecto a considerar es la herramienta en sí y el propósito, lo cual significa que de manera tangible los pasos no serían exactamente los mismos con programas distintos. En la Figura 1.2 se puede comprobar como el flujo de diseño con gEDA es particular pues en cada paso se especifica el módulo de programa implicado.

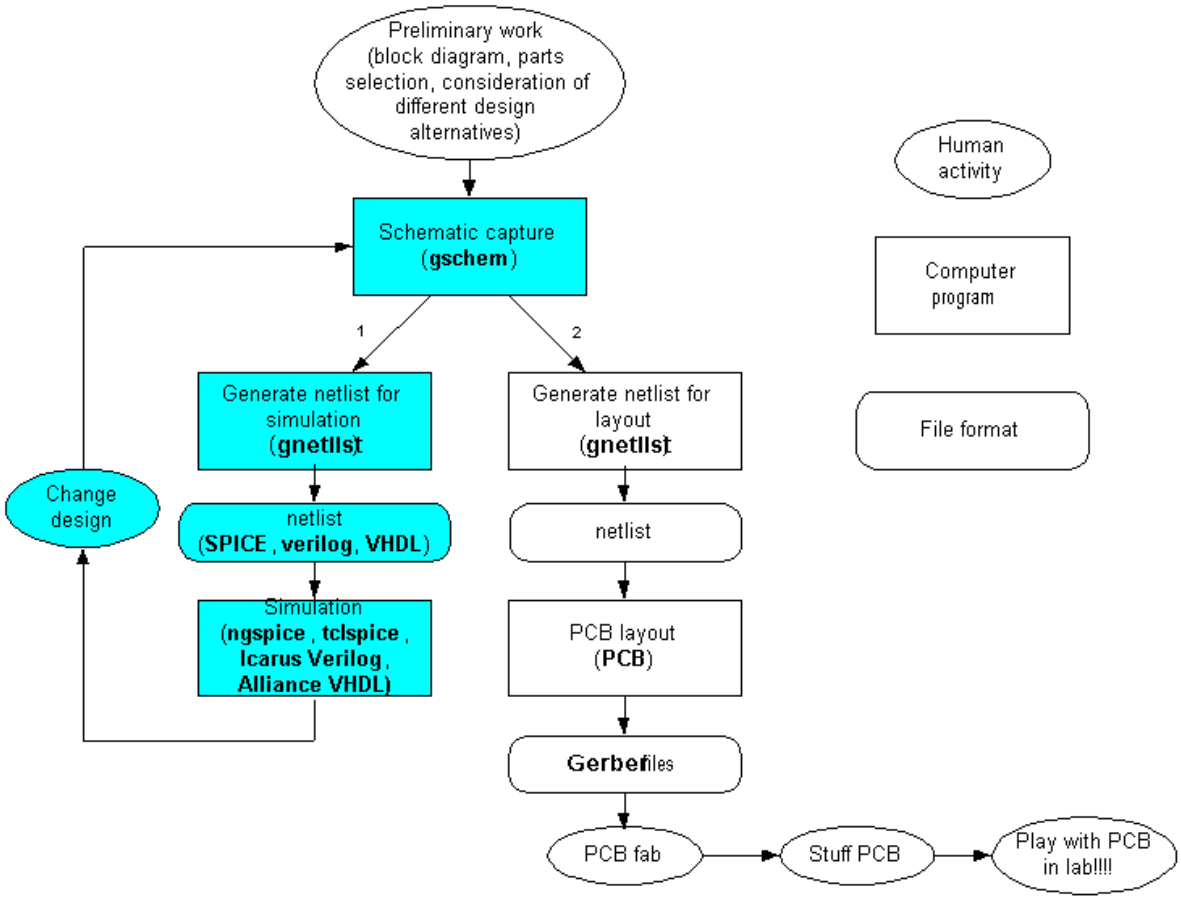


Figura 1.2 Flujo de diseño con gEDA. Tomado de (Proyect, 2011).

Hay muchos factores que describen el cambio del flujo de diseño desde un conjunto de pasos independientes a un enfoque totalmente integrado; así como los demás cambios que llegarán para hacer frente a los nuevos desafíos. En su discurso en la 40 edición de la *Design Automation Conference*, titulado *The Tides de EDA*, Alberto Sangiovanni-Vincentelli distingue tres períodos de EDA:

> La *era de la invención* fue el período durante el cual se construyen y definen términos como: ruteo, colocación, análisis de tiempo y síntesis lógica.
>
> Durante la *era de la implementación* los procedimientos fueron drásticamente mejorados mediante el diseño de mejores estructuras de datos y algoritmos sofisticados. Esto permitió que las herramientas se mantuvieran a la altura del rápido avance tecnológico. Sin embargo, debido a la falta de funciones de predicción de costos, se hizo casi imposible ejecutar un flujo de diseño conformado

de pasos discretos sin importar cuán eficientemente estaban implementados estos procedimientos.

La *era de la integración* es en la que el autor nos ubica. En esta etapa la mayoría de los pasos de diseño se ejecutan en un ambiente integrado, en el cual es determinante el papel de analizadores de costos.

Si se tuviese que explicar el papel transcendental de las funciones de análisis de costo en y dar un ejemplo de ello, la mejor opción sería utilizar la línea del desarrollo histórico de funciones como la generación del *bill of materials* o BoM y la propia conformación de PCBs.

1.3 Principales eventos internacionales sobre esta temática

La Convención de Automatización del Diseño o DAC combina una convención científico-tecnológica con un evento comercial; ambos especializados en la automatización del diseño electrónico.

DAC es la convención más vieja y más grande sobre diseño electrónico automatizado. Empezó como el taller de automatización del diseño SHARE en 1964. Hasta mediado de los 70´s, DAC tuvo sesiones en todos los tipos de automatización del diseño, incluyendo mecánico y arquitectónico. Después de eso, para todos los efectos, sólo los temas preocupados con diseño electrónico han sido incluidos. También hasta mediado de los 70´s, DAC fue estrictamente una convención técnica. Luego algunas compañías comenzaron a pedir espacio para mostrar sus productos, y dentro de algunos años que la porción de función de comercio de DAC se convirtió en el foco principal del acontecimiento. El primer DAC comercial fue realizado en junio de 1984. Una medida de la importancia de este evento comercial es que aproximadamente 5,500 personas asistieron al DAC en el 2005. (DAC, 2012)

Se reconoce al ICCAD (Conferencia Internacional en Diseño Asistido por Computadora), al menos igual de fuerte técnicamente pero sin la función de comercio, quizás por eso solo tuvo una décima parte de la asistencia del DAC en el 2005. Durante los últimos años la posición de la convención ha estado alternando entre San Diego, Anaheim y San Francisco. (Kuehlmann, 2003)

Otros datos de estos eventos pueden ser localizados fácilmente en Internet, pero la naturaleza de los mismos es quizás lo más interesante para este trabajo.

Otros ejemplos son: *Design Automation and Test in Europe* (DATE) y *Symposium on VLSI Circuits*.

1.4 Productos de software para electrónica

La comparación de productos de software para electrónica puede brindar una clara imagen de en qué estado está el desarrollo de este campo. Es difícil cubrir todos los aspectos y además sería bastante extenso. En lugar de ello se cita de la Wikipedia el recurso *Comparison of EDA software* que, por menos que guste, es el único de su tipo encontrado cuya información es acertada y útil.

Hasta el momento es evidente que el software propietario está mucho más desarrollado que el software libre en este sentido. Es por eso que se analizan por separado.

1.4.1 Software propietario

Entre las principales empresas que producen software para electrónica en la actualidad están: Cadence Design Systems, National Instruments, Labcenter Electronics, Altium, Xilinx, Altera y Logic Design Inc. Existen otras cuyo software tiene un espectro comercial más específico y no por eso son menos importantes.

A continuación se comentan algunos de los productos de estas empresas.

NI Multisim

Es una poderosa herramienta para el diseño electrónico. Fue diseñado para cubrir las necesidades de educadores y estudiantes, además de cumplir ampliamente con los requerimientos de los ingenieros y diseñadores a nivel profesional. Cuenta con herramientas técnicas como puntas de prueba industriales, intercambio de datos con instrumentos virtuales y reales, corrector de errores, sugerencias de cambios sobre el circuito, simulación integrada con micro controladores, etc.

NI Multisim es una de las herramientas más populares a nivel mundial para el diseño y simulación de circuitos eléctricos y electrónicos. Esta herramienta EDA proporciona avanzadas características que permiten ir desde la fase de diseño a la de producción.

Ofrece entradas esquemáticas, una amplia base de datos, simulación SPICE, entrada y simulación VHDL o Verilog, puede manejar circuitos de radiofrecuencia, realiza post procesamiento y es capaz de generar la placa PCB.

Multisim incluye una de las mayores librerías de componentes de la industria con más de 16.000 elementos. Cada elemento se complementa con los números de código de los fabricantes, símbolos para la captura esquemática, huellas para la realización del circuito impreso y parámetros eléctricos.

Las librerías están subdivididas: condensadores, resistencias, CMOS, multiplicadores, TTL, diodos, DMOS, etc., que incluyen todos los tipos de circuitos existentes en el mercado. Todos estos elementos están organizados en una completa base de datos que proporciona una forma sencilla y rápida de concreta de localizar los componentes. También dispone de una herramienta de simulación para circuitos de alta frecuencia (más de 100 MHz). Los resultados obtenidos por el programa pueden exportarse a formato gráfico o a formato de tablas incluyendo herramientas de visualización que incluyen editores para variar los tipos de letras, colores, etc.

Incluye, también, herramientas para la síntesis de FPGA/CPLD de la mayor parte de los fabricantes. El módulo de síntesis de Multisim ofrece una gran rapidez de síntesis y es la forma más fácil de transferir el diseño HDL al hardware. (Corporation, 2012)

Cadence OrCAD

Este es un conjunto de herramientas de software propietario empleada como EDA. Es empleada por muchos ingenieros y técnicos para la captura de esquemas electrónicos y la creación de PCBs. Su nombre es una combinación de dos palabras Oregon+CAD, con tal de reflejar los orígenes de este software.

Desde el 16 de Julio de 1999, la suite de OrCAD pertenece completamente a Cadence Design Systems. Algunos de sus productos han quedado descontinuados como el OrCAD Layout. En la última versión del OrCAD CIS se adjuntó la posibilidad de mantener una base de datos de componentes disponibles, sobre todo de circuitos integrados. Para ello, es posible conectar y actualizar esta base de datos desde los propios fabricantes como Analog Devices o Texas Instruments.

Para que se tenga una idea de lo mucho que es empleada esta herramienta, se puede citar que Intel ofrece PCBs de referencia de sus computadoras diseñados con Cadence PCB Tools y formatos de OrCAD Capture.

Es muy importante tener en cuenta que los principales premios otorgados en el 2011 a las herramientas de síntesis lógica fueron obtenidos por el Cadence Encounter RTL Compiler que, aunque no pertenece a suite del OrCAD, es otro producto de Cadence y se integra con el OrCAD en la síntesis y verificación de circuitos digitales. (Systems, 2012)

Altium Designer

Este es otro producto que ha revolucionado el mercado de EDA. Pertenece a la firma australiana Altium Ltd. Está diseñado para la creación de PCBs, diseño y simulación con FPGA y software empotrado. Desde el punto de vista de la captura esquemática, el producto permite: el manejo de bibliotecas de componentes, edición de esquemático, integración con muchos fabricantes para la búsqueda de componentes y sus datos técnicos, simulación de señal mixta con SPICE, análisis de integridad de señal, exportar Netlist, reporte con BoM, jerarquías de diseño y reutilización de esquemas. (Duncan, 2012)

Los módulos para la concepción de PCBs del Altium Designer permiten todas las posibilidades que sus competidores con una gran riqueza de herramientas y además: diseño interactivo en 3D con representación según la norma ISO 10303. Dicha norma establece lo relativo a la representación e intercambio en formato electrónico de la información de manufactura de un producto; se usa para referirse a ella la abreviatura STEP. (SCRA, 2006)

Otra de las notables y muy deseadas posibilidades que brinda este producto son las herramientas de desarrollo para FPGA. Esta es realmente una posibilidad que lo marca como un excelente producto para el co-diseño hardware-software. Permite:

Manejo del planeamiento de FPGA en PCB, con chequeo de la integridad de señales y del sincronismo.

Simulación y depuración de código en VHDL.

Softcore o *soft processor* en FPGA con todas las herramientas que implica: compilador, depurador y perfilador. Esto no es más que implementar un núcleo completo de un microprocesador con lenguaje de descripción de hardware,

mediante la síntesis lógica. Por solo citar un ejemplo, Altiumc Designer trae todas las herramientas para el co-diseño con los procesadores Microblaze de Xilinx. (Maxfield, 2006)

Xilinx ISE

En el diseño de sistemas digitales con FPGAs se utiliza el paquete informático Xilinx ISE (Integrated Software Environment). Está formado por un conjunto de herramientas que permiten diseñar circuitos digitales mediante esquemas lógicos o por medio de lenguajes de descripción de hardware como VHDL o Verilog. También permite simular el comportamiento de los circuitos diseñados, y sintetizarlos sobre dispositivos lógicos programables de Xilinx.

La herramienta consta de dos partes: el Navegador de Proyecto, donde se realizará el diseño del circuito, bien mediante un esquemático o utilizando un lenguaje específico de diseño y el ModelSim, donde podrá realizarse la simulación del funcionamiento del circuito y de este modo comprobar si funciona según las especificaciones.

El entorno de diseño ISE (Integrated Software Environment) de Xilinx consiste en una herramienta de software que permite realizar un diseño digital basado en lógica programable. Permite recorrer fácilmente las diferentes fases del proceso de desarrollo de un circuito lógico, desde el diseño hasta la implantación sobre una arquitectura reconfigurable. Cada una de las etapas del flujo de diseño puede realizarse dentro del entorno integrado o bien utilizar herramientas de terceros, algunas de esas herramientas también son integrables en ISE si están convenientemente instaladas. (Castelló and Valido, 2010)

ISE combina diferentes técnicas de diseño para facilitar la labor de descripción del diseño, mediante un conjunto de herramientas que permiten el diseño de circuitos digitales como esquemas lógicos, máquinas de estado o utilizando lenguajes de descripción de hardware como VHDL o Verilog. (Rosado and Bataller, 2003)

Este paquete permite simular el comportamiento de los circuitos diseñados y sintetizarlos sobre dispositivos lógicos programables de Xilinx. Llama de manera automática y desde su propio entorno de trabajo a las diferentes utilidades y herramientas suministradas dependiendo de la etapa de diseño, lo cual hace el trabajo más sencillo. El entorno de

desarrollo ISE Xilinx posee un aspecto similar al de los entornos de programación actuales como Visual Basic o Visual C, posee diversas ventanas para visualizar tareas específicas sobre cada una de ellas.

Dentro de las herramientas que usa está el llamado Xilinx ECS, del inglés (Engineering Capture System), la misma funciona tanto en el entorno integrado ISE como de forma autónoma. Esta herramienta posibilita la realización de esquemáticos, o sea que se puedan reproducir de forma gráfica aquello que haríamos sobre el papel, dibujando el circuito y conectando los componentes entre sí. Esto crea unos ficheros fuente con extensión ´´.sch´´. (Mendiguchía, 2007)

Xilinx ISE es un buen ejemplo de herramienta de software del fabricante. Es típico en los productores de PLDs. Otro ejemplo es el Quartus Plus de Altera.

1.4.2 Software libre

A diferencia del software propietario, el desarrollo de este grupo de programas no está siempre ligado a un propósito esencialmente comercial. Quizás también esto implique no sean tan sofisticados. No obstante la mayoría sustenta un interés en su desarrollo.

Dentro de este grupo están: gEDA, Kicad, FreePCB, Fritzing, Electric, Icarus Verilog, KTechLab, Magic, Oregano, OwlVision GDSII Viewer, Quite Universal Circuit Simulator, XCircuit y Verilator.

Solo pueden funcionar en una plataforma Windows: FreePCB, Fritzing, Kicad y Quite Universal Circuit Simulator. Por otra parte el único con licencia GPLv2 es el Kicad.

A continuación se comentan algunas de las herramientas de software libre, algunas de las cuales serán descritas con mayor profundidad en el Capítulo 2.

FreePCB

FreePCB es un programa de código abierto, un editor de PCB para Microsoft Windows, publicado bajo la Licencia Pública General de GNU. Con él se pueden crear circuitos electrónicos de forma sencilla. Fue diseñado para ser fácil de aprender y fácil de usar, aún en condiciones de trabajo de calidad profesional. No tiene *autorouter* incorporado, pero puede utilizar el FreeRoute.

Fritzing

Es un programa de automatización del diseño electrónico libre que busca ayudar a diseñadores para que puedan pasar de prototipos (usando, por ejemplo placas de pruebas) a productos finales.

Fritzing fue creado bajos los principios de Prossecing y Arduino, y permite a los diseñadores, investigadores y aficionados documentar sus prototipos basados en Arduino y crear esquemas de circuitos impresos para su posterior fabricación.

Además cuenta con un sitio web complementario que ayuda a compartir y discutir bosquejos, experiencias y a reducir los costos de fabricación.

Kicad

Kicad es un software de código abierto para la creación de diagramas esquemáticos electrónicos y circuitos impresos. Fue diseñado por Jean-Pierre Charras: investigador de la LIS (Laboratoire des Images et des Signaux) y profesor del IUT de Saint Martin d'Heres, Francia. Esta plataforma de trabajo para electrónica es gratuita y de código abierto (GPL). Es útil para todos los que trabajan en el diseño electrónico con diagramas esquemáticos y circuitos impresos de hasta 16 capas.

Este software utiliza wxWidgets y es multiplataforma pues puede funcionar en Linux y Windows XP o 2000. En la actualidad, la versión precompilada de Linux ha sido probada con Mandrake 9.2 y 10.0. En algún momento el software también se ha probado en otros sistemas operativos, específicamente FreeBSD y Solaris. (Charras, 2012)

Quite Universal Circuit Simulator

Es un simulador de circuitos electrónicos con una sencilla interfaz gráfica que puede simular el comportamiento de gran señal, de pequeña señal y el ruido del circuito. Una vez que la simulación termine puede ver sus resultados en una página de presentación o en una ventana. Es capaz de utilizar lenguajes de descripción de hardware como VHDL y Verilog, para ello se apoya en otro proyecto: el FreeHDL. QUCS es compatible con una creciente lista de componentes analógicos y digitales, así como circuitos SPICE. Está destinado a ser mucho más sencillo de usar y de manejar que otros simuladores de circuitos como gEDA o PSPICE.

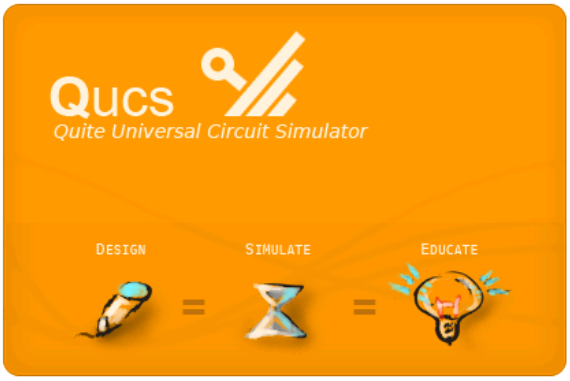


Figura 1.3 Logo de QUCS realizado por Dotan Nahum.

Es un programa de línea de comandos que se ejecuta por la interfaz gráfica de usuario con el fin de simular el esquema que previamente se preparó. Se necesita una lista de conexiones para realizar las acciones de simulación y, finalmente, producir un conjunto de datos. El editor de texto se utiliza para mostrar información de la simulación y para editar archivos incluidos por ciertos componentes. La aplicación de síntesis de filtro se puede utilizar para diseñar diferentes tipos de filtros. La calculadora de la línea de transmisión se utiliza para diseñar y analizar los diferentes tipos de líneas de transmisión como microstrips o cables coaxiales. Tiene modelos para dispositivos como transistores, diodos, puentes, amplificadores operacionales, etc. La herramienta de conversión es utilizada por la interfaz gráfica de usuario para importar y exportar bases de datos, listas de conexiones y esquemas desde y hacia otros CAD/EDA. (Team, 2011)

gEDA

Ngspice es un subproyecto del gEDA que tiene como objetivo crear una versión mejorada del spice (con algunos bugs arreglados) y con licencia GPL. El paquete viene con ngnutmeg, que es similar al nutmeg del spice, y sirve para graficar los datos de salida de las simulaciones.

El proyecto gEDA tiene como fin último el producir un conjunto de herramientas para el diseño automático en electrónica tipo GPL. Las herramientas que han sido desarrolladas por el momento se emplean para diversos propósitos, como el diseño de circuitos, la captura de esquemáticos, la simulación, la invención de prototipos y la producción. Actualmente, el proyecto gEDA ofrece un conjunto de aplicaciones de código abierto

maduras para el diseño electrónico como son: gattrib, geda, gerbv, gnetlist, gnucap, gschem, gsymcheck, GTKWave, Gwave, Icarus y Ngspice.

A pesar de los niveles de integración y los procedimientos bien probados, muchos autores aún no simpatizan con esta herramienta porque en varias etapas en el flujo de diseño se utilizan comandos. Por otra parte los resultados no son ricos en gráficos. Un análisis al respecto de los pros y los contras de utilizar una suite de programas en lugar de una aplicación integral se expone en (Proyect, 2011).

TKGate

Es un simulador de circuitos digitales, con una interfaz amigable e intuitiva, y varias características interesantes, como un compilador genérico de micro-código/macro-código para crear archivos de inicialización de memorias. (Hansen, 2012)

Figura 1.4 Logo de TKGate. Tomado de los archivos del programa.

1.5 El software para electrónica en Cuba

En las universidades cubanas, se emplea software propietario pirateado para la enseñanza de electrónica. Es indudable que esto da a lugar a un dilema de principios y legalidad, pero el hecho es que tales cuestiones no son penadas bajo el precepto de que es un país embargado económicamente. En cualquier caso las experiencias docentes son muy satisfactorias. Actualmente se emplean versiones de OrCAD, MultiSim, Proteus, plataformas de Xilinx y de Altera. Los estudiantes pueden llegar a dominarlos con profundidad, limitados solamente por la infraestructura.

Las empresas de desarrollo electrónico, cuentan también con software de acuerdo a sus esferas de trabajo. GEDISAI es un de ellas y algunos de sus grupos utilizan el Altium Designer, que ostenta ser una herramienta poderosa para el desarrollo de prototipos. No

consta tampoco el hecho de utilizar software libre en ninguna de las que fueron investigadas.

Por otra parte no hay propósito para una empresa cubana desarrolladora de software en desarrollar software de gran magnitud para electrónica, al menos no hasta ahora. Esto tiene que ver con el hecho de que no hay una necesidad en el país para ello, pues la producción de dispositivos electrónicos es escasa; dicho campo está confinado a las universidades, a empresas de soporte técnico y a algunas empresas como GEDISAI que no producen en gran escala. Las posibles necesidades dentro del país pueden ser suplidas con algunas de las variantes mencionadas. En cualquier caso sería más factible pagar las licencias del software propietario que pagar por el desarrollo de un nuevo software, que tendría mercado limitado.

CAPÍTULO 2. PREPARACIÓN DE LOS EJEMPLOS A RESOLVER

En el Capítulo anterior fueron presentados algunos ejemplos de herramientas de software libre para el trabajo con electrónica. Este capítulo enfocará directamente algunas de ellas con aplicaciones para electrónica digital que serán resueltas y simuladas. Se presentarán los ejemplos a emplear para la evaluación de dichos programas.

2.1 Caracterización de productos de software libre para electrónica digital

2.1.1 TKGate

El TKGate es un editor gráfico y simulador de circuitos digitales, desarrollado con Tcl/TK. Incluye componentes básicos como las puertas lógicas AND, OR y XOR; módulos estándares como sumadores, multiplicadores, registros y memorias e incluso transistores MOS.

Además permite desarrollar sistemas complejos, mediante el diseño de módulos y submódulos. Como por ejemplo: ALU's, Unidades de Control y Unidades de Ejecución. Otra de las ventajas es poder utilizar memorias para la simulación y asignar la memoria a un archivo binario.

Entre los circuitos ejemplo se incluye una CPU sencilla, programada para ejecutar el juego *Animals* (ver Anexo 2). TKGate es gratuito y se suministra con el código fuente bajo la Licencia Pública GNU (GPL). A continuación se destacan sus características claves:

Diseño gráfico de circuitos

Permite diseño jerárquico a través de módulos definidos por el usuario. La interfaz es fácil de usar, además puede aparecer en varios idiomas como Español, Catalán, Inglés, Francés, Alemán y Japonés. Se pueden crear de hiperenlaces para navegar dentro del circuito o cargar archivos. Se pueden manejar el formato de fichero Verilog.

Simulador lógico

El control del simulador se puede realizar a través del interfaz o a través de ficheros *scripts*. Este es apropiado para la simulación a nivel de transistor, puerta o registro. Tiene seis modelos de valores lógicos, que incluyen: 0, 1, flotante, desconocido, "bajo" y "alto". Soporta modelos de retraso personalizados. Tiene una ventana gráfica para los resultados de simulación. Tiene además puntos de control para el control de simulación por reloj. Otra de las funciones es el análisis estático de pasos críticos.

Componentes incluidos

Incluye compuertas básicas, transistores, buffers de tres estados, componentes para ALU (sumadores, registros de desplazamiento y multiplicadores) y elementos de memoria como registros, RAM y ROM. También contiene en su biblioteca de componentes elementos interactivos que permiten al diseño circuital interactuar con el usuario.

Soporta además herramientas muy importantes como un compilador de microcódigo/macrocódigo para asistir a la creación de grandes proyectos, tales como el diseño de microprocesadores. (Cuéllar, 2005)

Opciones en línea de comandos

La mayoría de las instrucciones de TKGate son ajenas a los usuarios debido a que predomina el empleo de la interfaz gráfica, lo cual es obvio. Pero realmente las operaciones existen también en el modo de comandos y el uso general es:

tkgate [-xqs] [-X script] [-l file] [-L lang] [-P printer] [-p file] [files ...]

Las opciones que aparecen en el comando anterior son las siguientes:

-X script: De forma automática comienza el simulador y ejecuta el script de simulación específico.

-l file: Lee el fichero especificado como si fuera una librería.

-x: Arranca de forma automática el simulador.

-q: Suprime los mensajes de arranque.

-P printer: Imprime el circuito especificado sin necesidad de arrancar el interfaz de usuario.

-p file: Imprime en un fichero sin necesidad de arrancar el interfaz de usuario.

-L lang: Especifica el lenguaje a emplearse. TKGate debe haber sido construido con soporte multilingüe con el fin de usar esta opción.

Interfaz de TKGate

La ventana de edición TKGate está constituida por una barra de menú en la parte superior. Las listas de módulos, redes o mallas y una lista de puertos, están todas a la izquierda. Tiene una barra de estado abajo y el área de edición principal en el centro. Las barras de desplazamiento pueden usarse para mover el circuito, para listar los módulos o listar las mallas.

La lista de módulos en la parte superior izquierda de la pantalla, muestra los módulos que son parte del circuito que está en edición. El módulo de nivel superior es indicado con un signo + tras su nombre.

La lista de redes ubicada abajo, corresponde a las redes correspondientes al módulo actual. En las redes den varios bits, estos se especifican a continuación del nombre. Los nombres de red pueden estar ocultos, en dependencia de si se muestran en la ventana del circuito o no. Las mallas ocultas son indicadas en la lista con el símbolo @ a continuación.

El símbolo X en la ventana de edición indica la posición de nuevos componentes (ver Figura 2.1).

Los comandos usados con mayor frecuencia pueden ser accedidos a través de la barra de botones de TKGate, al estilo de las herramientas basadas en Windows. Estos botones incluyen comandos para abrir, salvar e imprimir ficheros de circuitos, cambiar las herramientas de edición, abrir o cerrar módulos y controlar el simulador. La barra de botones también tiene un selector para elegir una tecnología por defecto.

La barra de estado de la parte inferior indica el fichero que está siendo editado y el módulo actual del fichero que está mostrado en la ventana de circuito. Una x tras el nombre del fichero indica que el buffer ha sido modificado desde la última vez que fue salvado. Bajo el nombre del fichero y del módulo está una barra de mensajes de información procedentes del programa. Estos mensajes de TKGate incluyen confirmación para ficheros que son cargados, salvados o información sobre el elemento seleccionado (ver Figura 2.1).

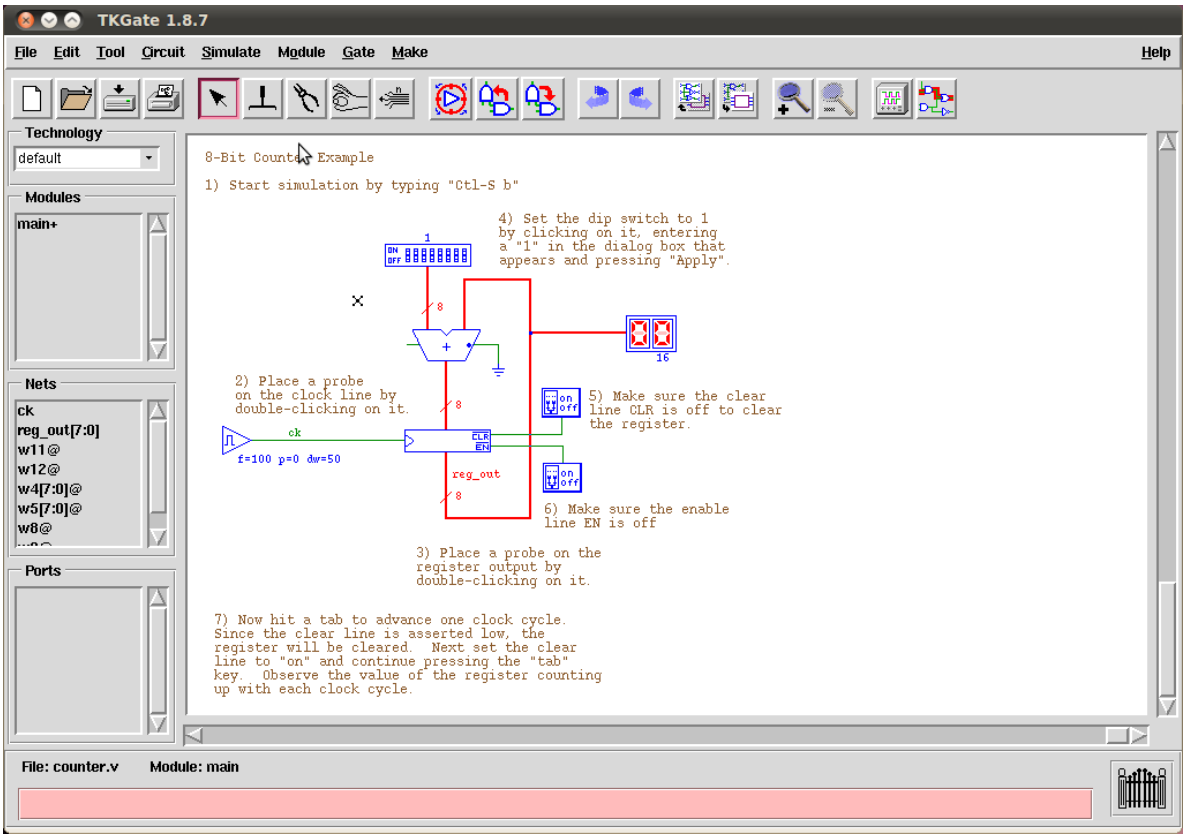


Figura 2.1 Ventana de la GUI de TKGate v1.8.7. Imagen tomada del propio software.

En la esquina inferior derecha está ícono de modo. Los íconos posibles son:

a) b) c) d)

Figura 2.2 Íconos de modo: a) modo de edición b) simulación detenida c) simulación en curso d) modo de análisis.

Para muchos de los elementos de la interfaz existen mensajes de ayuda que surgen con sólo colocar el ratón encima del elemento.

El resto de las herramientas de edición, características del trabajo con archivos, colores, ejemplos y más, puede ser accedido en (Cuéllar, 2005) y en (Hansen, 2012).

2.1.2 gEDA

La suite gEDA es la colección de las herramientas que forman parte, están asociadas o planean integrarse a dicho proyecto. Hasta el momento la suite gEDA incluye:

gEDA/gaf: captura de esquemático y *netlisting*.

ngspice: simulación con SPICE.

gnucap: simulación analógica.

gspiceui: GUI para ngspice/gnucap.

pcb: disposición de componentes y creación de PCB.

gerbv: visualizador de formatos Gerber.

Icarus Verilog: simulador para lenguaje Verilog.

GTKWave: visualizador de formas de ondas.

wcalc: análisis de estructuras de líneas de transmisión y campo electromagnético.

Aparenta ser una confederación de los programas un tanto independientes. Esto sucedió por razones de historia: Ales Hvezda inició el proyecto gEDA más o menos por su cuenta. La visión original era producir una suite de software de extremo a extremo para la creación de placas de circuito impreso, para que los aficionados a la robótica diseñaran sus propias placas. Sin embargo, con el avance del proyecto, la gran magnitud y el peso de esta tarea se hicieron notar; muchas de las aplicaciones propuestas aún no se habían programado.

Mientras tanto, otros desarrolladores de software, con sus propias aplicaciones independientes, encuentran la visión del proyecto gEDA convincente. Dichos autores se unieron a Ales y contribuyeron con sus aplicaciones. No es el código o una interfaz de usuario común lo que distingue a gEDA; es la visión compartida de un entorno de código abierto EDA es el hilo que sostiene el proyecto en conjunto.

Las aplicaciones se esfuerzan por operar juntas y tener éxito en general, pero los principios por separado de cada programa de la suite son aún marcados. Sin embargo, con un poco de trabajo, los diversos componentes de la suite son interoperables y muchos diseños bastante complejos han podido realizarse utilizando la suite de gEDA. (Proyect, 2011)

gEDA/gaf

gaf que significa *gschem and friends*, es el un subconjunto de la suite de programas, quizás el grupo de aplicaciones más conocidas. gEDA/gaf actualmente incluyen:

gschem: programa para la captura de diagramas esquemáticos (ver Figura 2.3).

gnetlist: programa para la generación de *netlist*.

gsymcheck: chequeador de sintaxis de símbolos esquemáticos.

gattrib: para manipular las propiedades de los símbolos en el esquemático.

libgeda: bibliotecas para gschem, gnetlist y gsymcheck.

gsch2pcb: herramienta para anotación desde el esquemático hacia el PCB.

Otras utilerías menores.

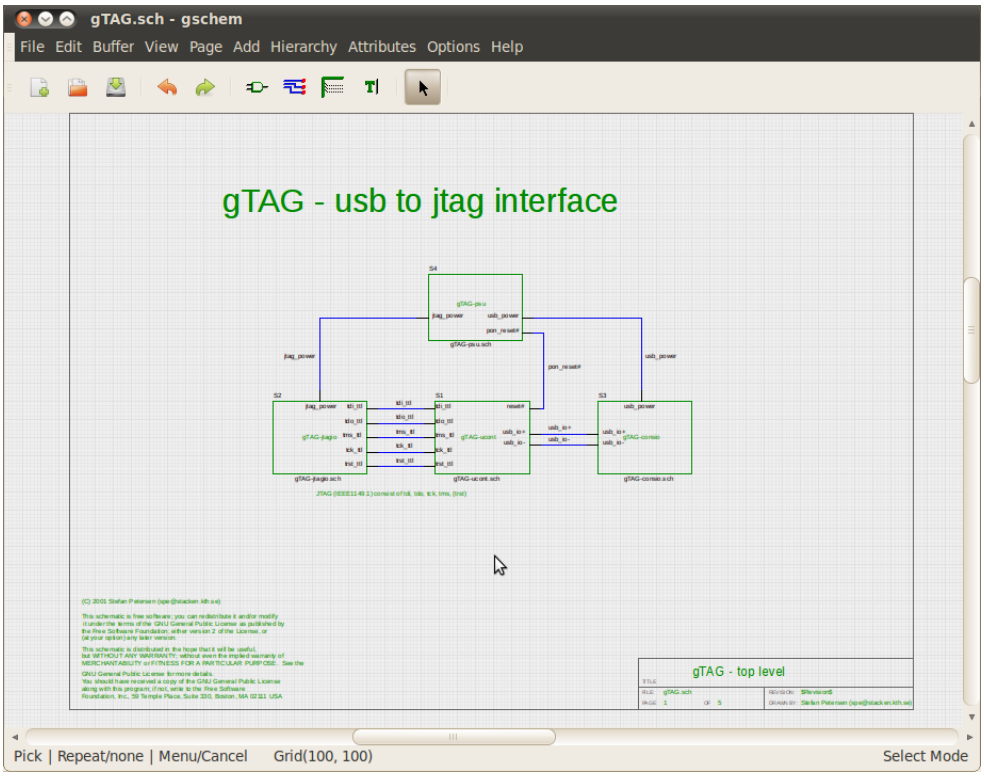


Figura 2.3 Ventana de la GUI de gschem. Tomado del propio programa instalado.

Los programas anteriores comparten un mismo formato de archivo (.sch) y una biblioteca de enlace común (libgueda.sch).

A pesar de que la suite permite la simulación de circuitos digitales, no está dotada de facilidades que permitan su fácil utilización. Es mucho mejor su utilización en proyectos de para la obtención de circuitos impresos, o de simulación de alta precisión con

2.1.3 *Quite Universal Circuit Simulator*

QUCS permite simulación analógica, simulación digital y optimización entre otras funciones.

Dentro de las simulaciones analógicas puede efectuar barrido DC, simulación con parámetros s.

La primera vez que arranca QUCS, se crea el directorio .Qucs dentro del directorio /home del usuario. Todos los archivos van por defecto a este directorio o a alguno de sus

subdirectorios. Después de cargado, la ventana principal tiene un aspecto similar al de la Figura 2.4. En el lado derecho hay un área de trabajo (6) que contiene los esquemas, visores de datos, etc. Si se usan las pestañas (5) sobre esa área, se puede cambiar con rapidez entre los documentos que estén abiertos. A la izquierda de la ventana principal hay otra área (1) cuyo contenido depende de la pestaña seleccionada en la parte superior: *Proyectos* (2), *Contenido* (3) y *Componentes* (4).

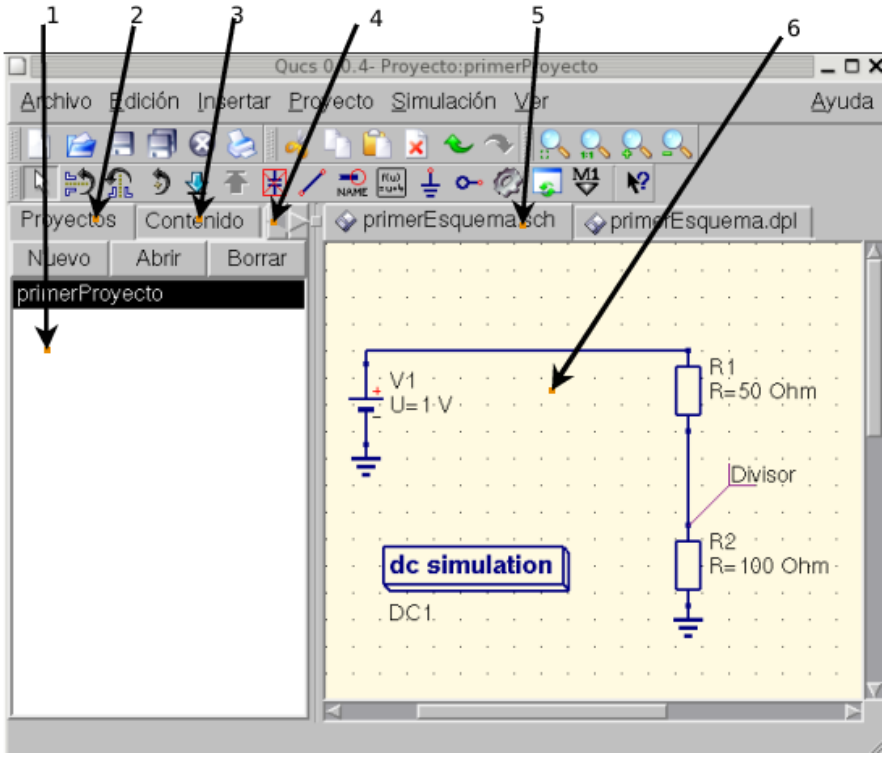


Figura 2.4 Ventana principal de QUCS. Tomado de la ayuda del programa.

Después de ejecutar QUCS, la pestaña *Proyectos* (2) se activa. La primera vez que se arranca el programa, esta área está vacía porque no tiene ningún proyecto. (Team, 2011)

Para hacer una simulación simple de corriente continua como la de la Figura 2.4. Primero se activa la pestaña *Componentes*. Ahí se observa una caja desplegable donde se puede escoger un grupo de componentes, y debajo, los componentes del grupo seleccionado. Una vez que se selecciona un elemento y se lleva a la hoja, al pulsar el botón derecho del ratón rota el símbolo, al pulsar el botón izquierdo coloca el componente en el esquema. Para editar los parámetros de los elementos, se hace doble clic sobre ellos. Para conectar los

componentes, se emplea el botón cablear de la barra de herramientas o por el menú principal: Insertar ->Cable.

Los subcircuitos se usan para dar más claridad a un esquema. Es muy útil en circuitos muy grandes o en circuitos en los que aparece un mismo bloque de componentes varias veces. En QUCS cada esquema que contenga una conexión de subcircuito es un subcircuito. Para conseguir una conexión de subcircuitos, se va a la barra de herramientas, la vista de componentes (en componentes sueltos) o al menú Insertar -> Insertar conexión. Después de colocar todas las conexiones del subcircuito hay que guardar el esquema.

Necesita varios programas que se instalan durante el proceso de instalación:

/usr[/local]/bin/Qucs - el interfaz gráfico
/usr[/local]/bin/Qucsator - el simulador (aplicación de consola)
/usr[/local]/bin/Qucsedit - un simple editor de texto
/usr[/local]/bin/Qucshelp - un programita para mostrar el sistema de ayuda

Todos los programas son aplicaciones independientes y pueden arrancarse por separado. El programa principal o interfaz gráfico llama a Qucsator para realizar una simulación, llama a Qucsedit para mostrar archivos de texto y llama a Qucshelp para mostrar el sistema de ayuda.

Además, se crean en la instalación los siguientes directorios:

/usr[/local]/share/Qucs/bitmaps - contiene todos los gráficos (iconos, etc.)
/usr[/local]/share/Qucs/docs - contiene documentos HTML para el sistema de ayuda
/usr[/local]/share/Qucs/lang - contiene los archivos con las traducciones

Otra característica notable a resaltar son las posibles funciones matemáticas que se pueden realizar. Dichas funciones van desde las operaciones más elementales hasta las trigonométricas, logarítmicas, exponenciales, con matrices, derivadas y otras relacionadas con vectores. (Team, 2011)

2.2 Multiplexor de 8 a 1

Un multiplexor es un conmutador digital que conecta los datos de las *n* entradas con la salida. En la Figura 6-59 de la página 433 de (Wakerly, 2006) se puede apreciar un diagrama genérico del mismo y a continuación un análisis de su expresión lógica general.

Los multiplexores comerciales típicos pueden tener $n = 1,2,4,8,16$ entradas y $b = 1,2,4$ salidas. Las entradas de selección s serían de acuerdo con $s = \log_2 n$.

Son dispositivos muy usados, sobre todo en aquellas aplicaciones en las que los datos deben ser conectados desde múltiples entradas hacia sus destinos. Un ejemplo común es en los sistemas con microprocesadores, en los dispositivos de entrada/salida o I/O; es típico que contengan registros para el almacenamiento de información de datos y control, cualesquiera de estos registros puede ser accedido para lectura por software. Este ejemplo también está explicado en (Wakerly, 2006).

Una variante estándar de multiplexor de 8 a 1 es la de la Figura 2.5. Este es el diseño es típico de componentes MSI tales como el 74151. La entrada de habilitación EN_L es activa en bajo, por lo que el componente se habilitará solo cuando se coloque un cero en dicha posición. El circuito tiene dos salidas: una Y_L con el bit lógico de entrada invertido y otra Y con el complemento.

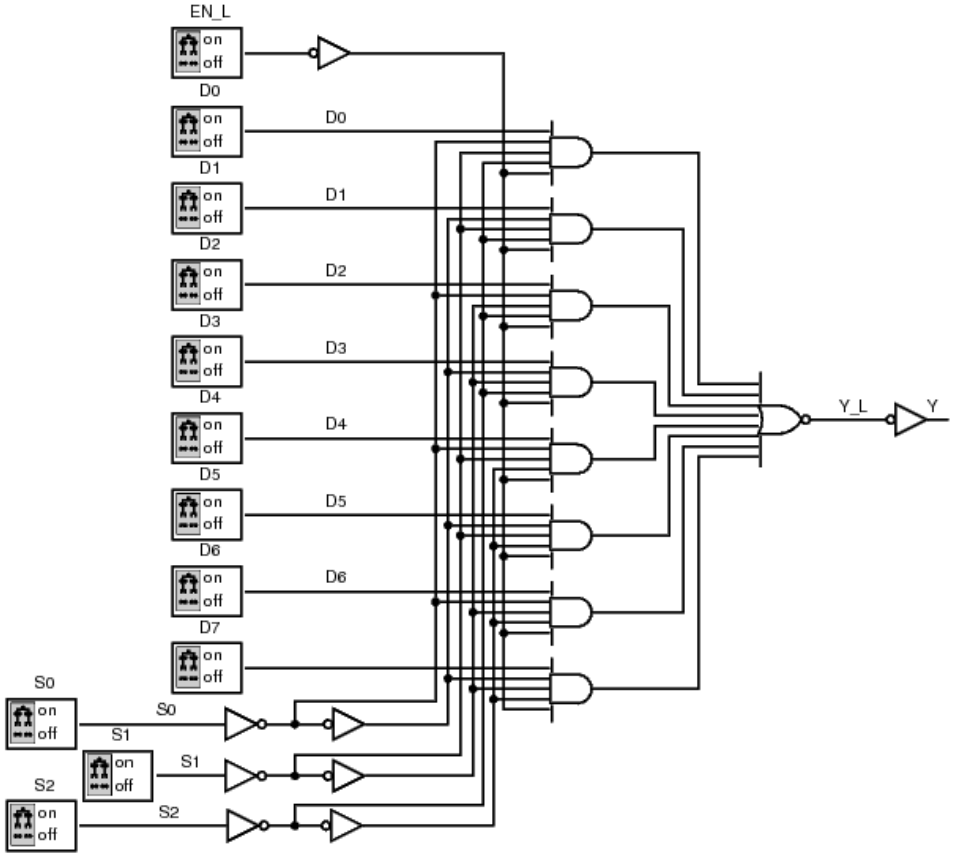


Figura 2.5 Esquema de un multiplexor de 8 a 1 conformado en TKGate. La figura fue obtenida a partir de la impresión en un archivo pdf.

La selección se realiza con las entradas S0, S1, y S2, tal y como se muestra en la tabla 6-42 de la página 435 del (Wakerly, 2006). Los interruptores a la izquierda en la Figura 2.5, en total 12, no constituyen en realidad parte del diseño sino que son una utilería del TKGate para realizar estimular las entradas con la combinación de señales deseada para la simulación.

Quizás uno de los aspectos más importantes que tienen estos circuitos desde su diseño es la posibilidad de expansión, es decir: varios circuitos como este multiplexor puedes ser apropiadamente instalados para conformar multiplexores más grandes.

2.3 Decodificador de 3 a 8

Los decodificadores binarios son circuitos digitales de *n* entradas y 2^n salidas, de tal forma que el código de la entrada se transforma en una salida singular de todas las 2^n salidas disponibles. Esto se hace evidente en la siguiente tabla de funcionamiento:

Tabla 2.1 Tabla de funcionamiento de un decodificador de 2 a 4.

Entradas			Salidas			
EN_L	I1	I0	Y3	Y2	Y1	Y0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

En la Figura 2.6 se presenta un esquema de un decodificador de 3 a 8 típico del componente MSI 74138. Las entradas G1, G2A_ y G2B_L son entradas de habilitación, la primera activa en alto y el resto en bajo. Todas las salidas son activas en 0. Una de las funciones de mayor interés de estas entradas es la posibilidad que brindan de crear decodificadores más grandes, o sea de un mayor número de entradas. Este principio lo comparten en gran medida la mayor parte de los circuitos MSI modulares. Ejemplos de este tipo de diseño escalar pueden encontrarse en (Wakerly, 2006).

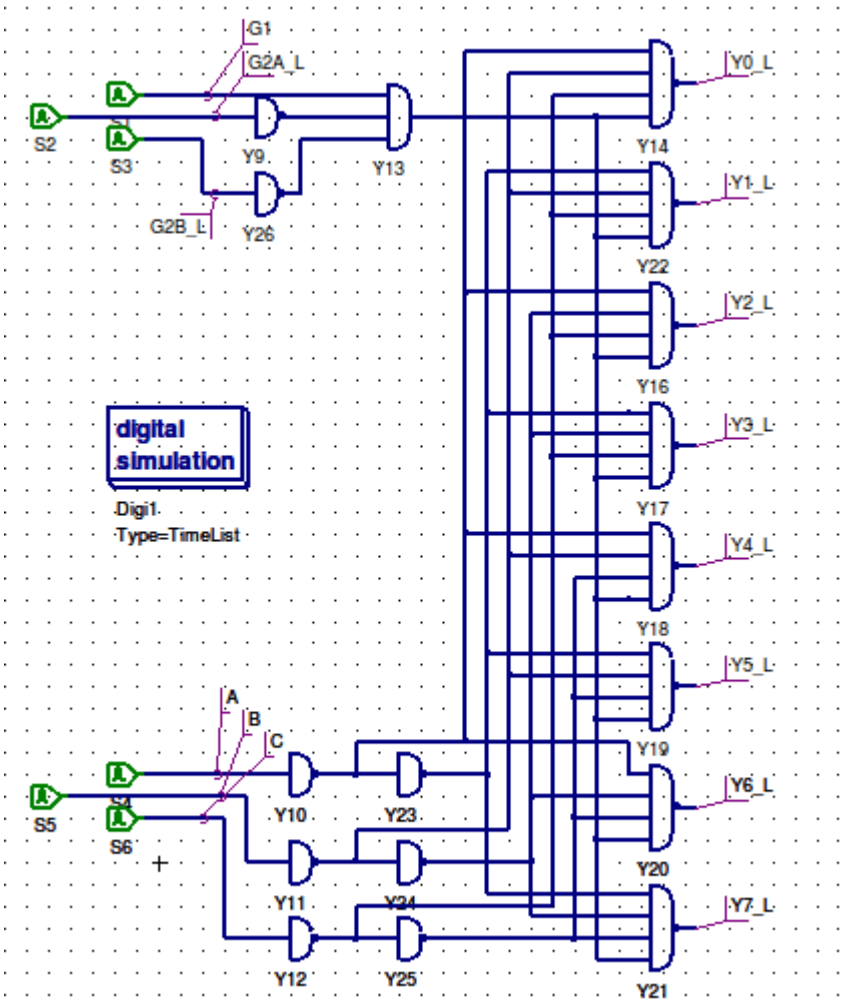


Figura 2.6 Esquema de un decodificador de 3 a 8 creado en el QUCS. La imagen se obtuvo mediante captura de pantalla.

2.4 Registro universal

Un registro de desplazamiento es un registro de *n* bits con la capacidad de desplazar cada uno de ellos en una posición, en un borde reloj. Si a esta funcionalidad se le añade hacer esta operación en los dos sentidos posibles, cargar una serie de datos de entrada y permanecer inactivo, entonces se obtiene lo que se conoce como registro universal.

En realidad un registro universal como el 74194 puede ser empleado entonces en aplicaciones de registro serie-serie, serie-paralelo, paralelo-serie y paralelo-paralelo. En la actualidad los componentes MSI como los antes mencionados son poco usados, debido a que cualquiera de las funciones deseadas puede ser lograda con dispositivos lógico-programables. Sin embargo, es muy útil mirar en el interior del diseño de un componente

como el 74194 para tener una idea de su complejidad y para mejor conocimiento sobre su empleo; sí porque la mayoría de los fabricantes de PLDs o FPGAs proveen de librerías con componentes MSI que permiten un diseño modular de circuitos digitales.

La tabla de funcionamiento de una 74194 es como la siguiente:

Tabla 2.2 Tabla de funcionamiento del 74194.

Función	Entradas		Estado siguiente			
	S1	S0	QA	QB	QC	QD
Inhibir	0	0	QA	QB	QC	QD
Desplazar de derecha	0	1	R_IN	QA	QB	QC
Desplazar de izquierda	1	0	QB	QC	QD	L_IN
Cargar	1	1	A	B	C	D

El modelo que aparece en (Wakerly, 2006) para este circuito es ligeramente diferente al de la Figura 2.7, debido a la adaptación realizada para que fuese compatible con los modelos de flip-flop D disponibles en QUCS.

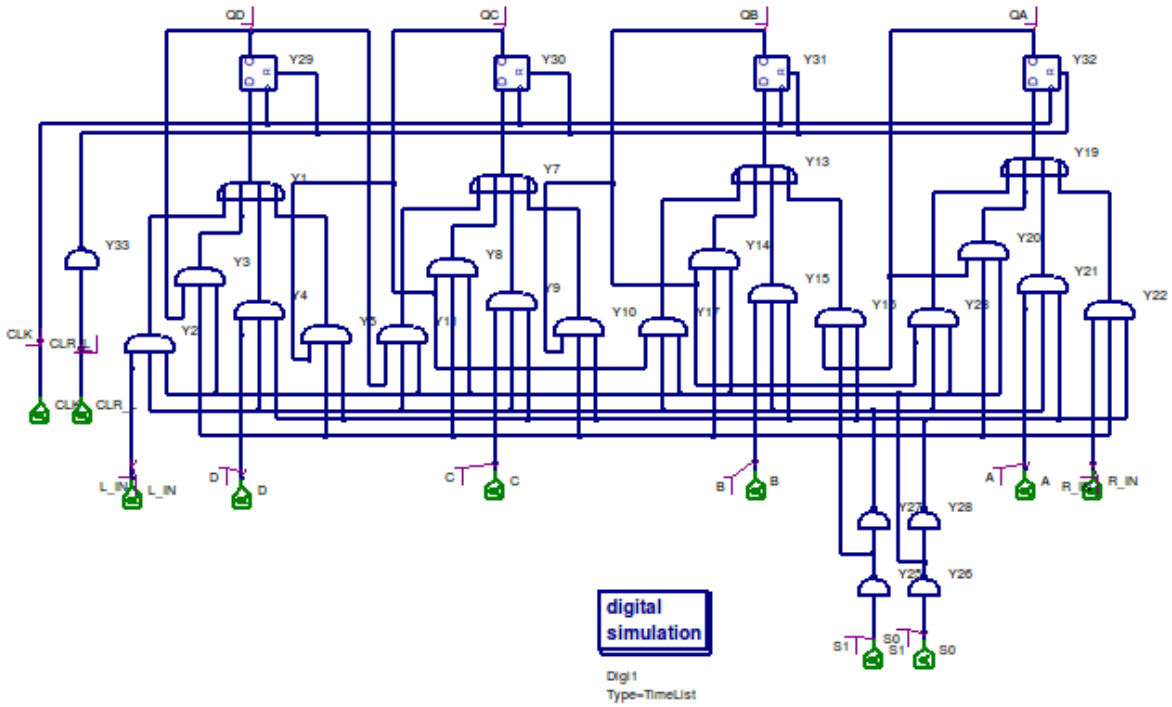


Figura 2.7 Esquema del circuito para el registro universal. La imagen se obtuvo mediante captura de pantalla.

En este caso fueron empleados flip-flops D con una entrada de *reset* asincrónica. Este modelo era muy conveniente pero hubo que adaptar el hecho de que dicha entrada era activa en alto, para lo cual se colocó el inversor Y33.

Otra implementación pudo realizarse con los mismo filp-flops D y cuatro multiplexores de 4 bits, más era conveniente hasta este punto realizar el esquema con compuertas ya que el diseño modular será abordado más adelante con otros ejemplos.

2.5 Contador binario módulo 16

Los circuitos denominados contadores son generalmente circuitos secuenciales sincrónicos cuyo diagrama de estados contiene un solo ciclo. El módulo de dichos contadores es el número de estados en el diagrama. En (Mano and Ciletti, 2004) se presentan otras formas de contadores como los contadores de rizado, pero estos, a pesar de tener menos componentes que cualquier otro tipo de contador, pagan el precio de ser un poco más lentos. En el peor caso, cuando el bit más significativo tiene que cambiar, la salida no es válida sino hasta después de $n \cdot t_{TQ}$ después del borde de activación del reloj, donde t_{TQ} es la demora de propagación desde la entrada hasta la salida en un flip-flop T. Así se pueden citar sucesivamente otros tipos de contadores con sus pros y contras, muchos de ellos pueden además encontrarse en (Wakerly, 2006) y en (Roth and Kinney, 2009).

Para interés de este trabajo se seleccionó una forma de contador sincrónico módulo 16 para su realización con el TKGate; el resultado del esquemático es el circuito de la Figura 2.8. Este circuito es similar al encontrado en (Wakerly, 2006) para el componente MSI 74163.

El contador usa flip-flops D en lugar de T para facilitar las funciones de carga y borrado. Cada entrada D está excitada por un multiplexor de dos entradas compuesto de dos compuertas OR y una AND. La salida del multiplexor es o si la entrada CLR_L es activada en bajo. Si por el contrario, la entrada LD_L es activada en bajo se pasa el dato de entrada que puede ser A, B, C o D. Si ninguno de estos dos casos se da, el multiplexor pasa la salida del XNOR, compuerta responsable de la función del conteo. La salida RCO (*ripple carry out*) indica el acarreo, o sea cuando todos los bits de salida son 1s y la entrada ENT está activa. (Wakerly, 2006)

Sin embargo le fueron realizadas algunas adaptaciones para obtener un correcto funcionamiento. Una de ellas tuvo que ver con el modelo presente en el software para el flip-flop D, dado a que el que el modelo presente en el TKGate utiliza entradas activas en

bajo CLEAR y ENABLE mientras que el presente en el libro utiliza las clásicas CLEAR y RESET.

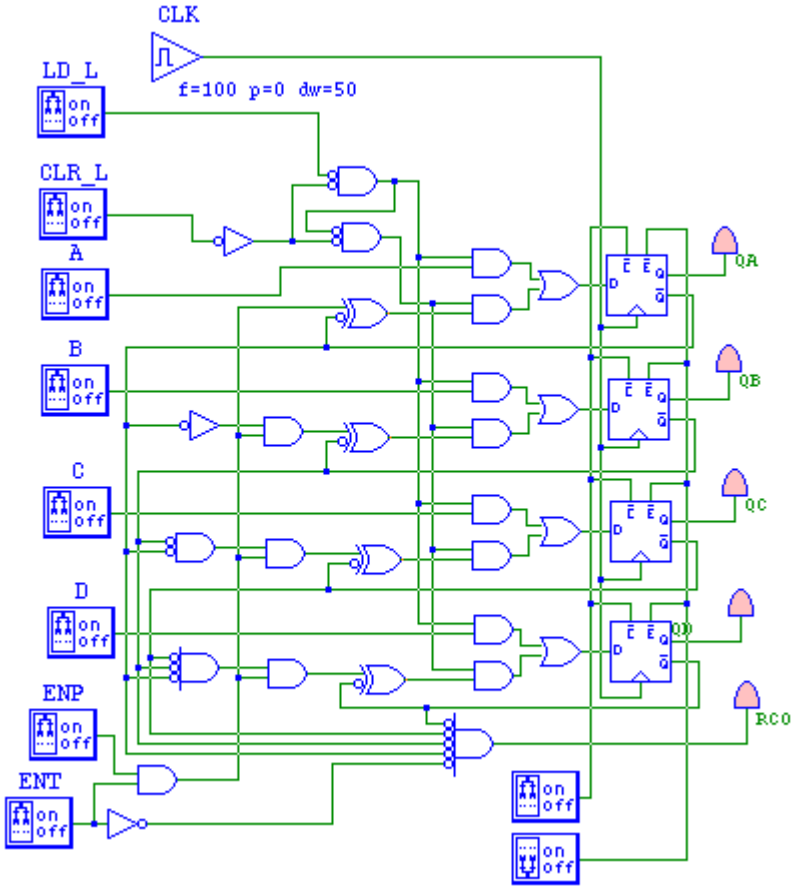


Figura 2.8 Esquema de un contador módulo 16 realizado en TKGate. La imagen se obtuvo mediante captura de pantalla.

Obsérvese además que le fueron colocados los mismos interruptores de la Figura 2.5 para la creación de los estímulos para la simulación, pero además le fueron colocados LEDs a las salidas para tener un indicativo del resultado. Otro elemento en el esquema es el reloj en la parte superior, cuyos ciclos tienen un ancho de 100, fase 0 y aprovechamiento de una 50%.

Otra de las diferencias con el modelo del 74163 es que la entrada dinámica de reloj no está disparada por el borde de caída, aunque esto no representa un gran inconveniente para realizarse en el TKGate. El software cuenta con una herramienta muy generosa para invertir cuyo ícono es el siguiente: 

2.6 Diseño y simulación con circuitos modulares

Raramente se utilizan circuitos complejos construidos a compuertas en la práctica ingenieril. En lugar de esto se realizan diseños modulares con componentes que originalmente se idearon para la mediana escala de integración. De hecho, el verdadero propósito de diseños como estos viene, en la mayoría de los trabajos, asociado a la modelación de un esquema digital para su utilización en circuitos de alta escala de integración; es decir, se modela con estos y se hace la síntesis en dispositivos lógico-programables.

A continuación se presentan varios casos de estudio que se modelaron y simularon en software libre. Los resultados aparecen en el capítulo tres. Se prepararon dos casos de estudio para circuitos combinacionales con diseño modular y el mismo número para secuencial. Las principales limitantes para la selección de los problemas fue el hecho de que los programas contaran con el modelo o no.

2.6.1 Casos de estudio de circuitos combinacionales

El problema commbinacional elegido para realizar con el TKGate fue el registro de desplazamiento de barrera. La razón para ello fue que el TKGate tiene en su versión 1.8.7 tiene pocos modelos de componentes MSI; solo cuenta con: multiplexor, decodificador, unidades para operaciones aritméticas y registros.

El problema del registro de desplazamiento de barrera de 8 bits consiste en, un circuito que cuyas salidas rotan a la izquierda la palabra de entrada, tantas veces como lo indiquen las entradas S2, S1 y S0. La solución es fácil de inferir con el empleo de multiplexores de 8 a 1, como el 74151 por citar un ejemplo.

El esquema realizado es el de la Figura 2.9. Las entradas están conectadas a un Switch DIP, componente que tiene un máximo de 8 interruptores para accionar. Las entradas de selección de los multiplexores están conectadas a otro Switch DIP, esta vez con un ancho de bus igual a 3 bits.

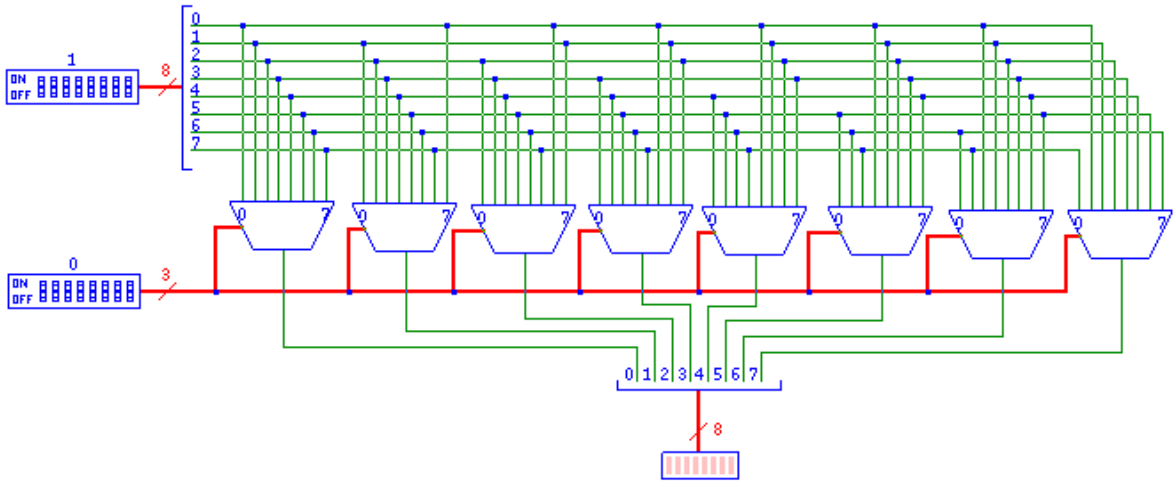


Figura 2.9 Esquema del registro de desplazamiento de barrera realizado en TKGate.

Los multiplexores empleados son bien genéricos, pero sirven para el propósito del problema. De hecho, el TKGate 1.8.7 no cuenta con modelos para componentes MSI que permitan escalabilidad. Por ejemplo: el multiplexor de 8 a 1 de la figura anterior, no permite la creación un multiplexor de 16 a 1 con solo conectarlos entre sí, pues no tiene entrada de habilitación ni salidas negadas, como de verdad tiene un circuito como el 74151.

A la salida se colocó una barra de LEDs para observar el comportamiento del desplazamiento de la entrada, en función de la selección en los multiplexores.

El segundo problema a resolver es bien sencillo y es un circuito comparador para dos valores de entrada de 4 bits, que entrega a la salida el mayor de ellos. Esta vez se realización con el deseado QUCS.

El problema hubiese sido fácil de resolver de haber contado con algún tipo de modelo de selector, similar al 74157, pero QUCS también carece de una vasta librería de componentes MSI. En su colección aparecen multiplexores, demultiplexores, semisumadores, sumadores completos, conversores de código Grey a binario y comparadores. No obstante, permite la creación de subcircuitos con tal de poder expresar el diseño en jerarquía de estos, y además poder crear componentes propios para enriquecer la librería de proyectos.

Como QUCS es amigable de usar y el trabajo con subcircuitos es sencillo, se ideó entonces el selector como un esquema independiente. Para ello se utilizaron cuatro multiplexores de 2 a 1 y puertos de entra o salida digital, tal y como se muestra en la siguiente figura:

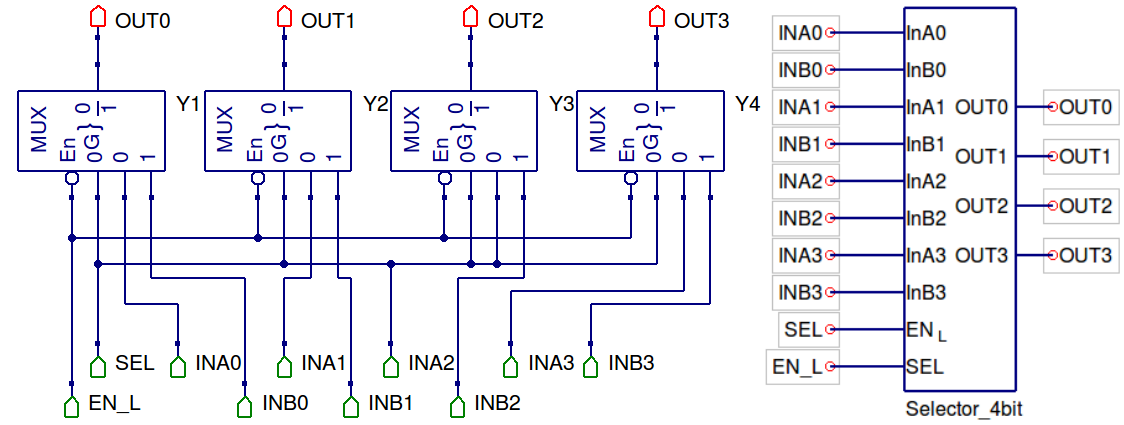


Figura 2.10 Selector de cuatro bits con entrada de habilitación. A la derecha el esquema y a la izquierda el símbolo. Tomado mediante captura de pantalla del proyecto del comparador.

Algo como esto es muy fácil de realizar. El autor Mike Brinson en (Brinson, 2006) afirma que en teoría no hay límites para la cantidad de niveles en la jerarquía en Qucs. No obstante la mayoría de los esquemas de circuitos jerárquicos se construyen con hasta 5 niveles.

Lo interesante de crear componentes a la medida es que pueden hacerse con el grado de especificación necesario para la tarea. Por ejemplo de haber tenido un modelo de comparador como el 7485, se hubiese podido extender este ejercicio a valores de A y B de tamaño n. De cualquier forma se considera una debilidad el no contar con una librería para el software.

Una vez que se tiene el selector, se coloca en otro esquema un comparador y un selector como el diseñado. El resultado es el de la Figura 2.11.

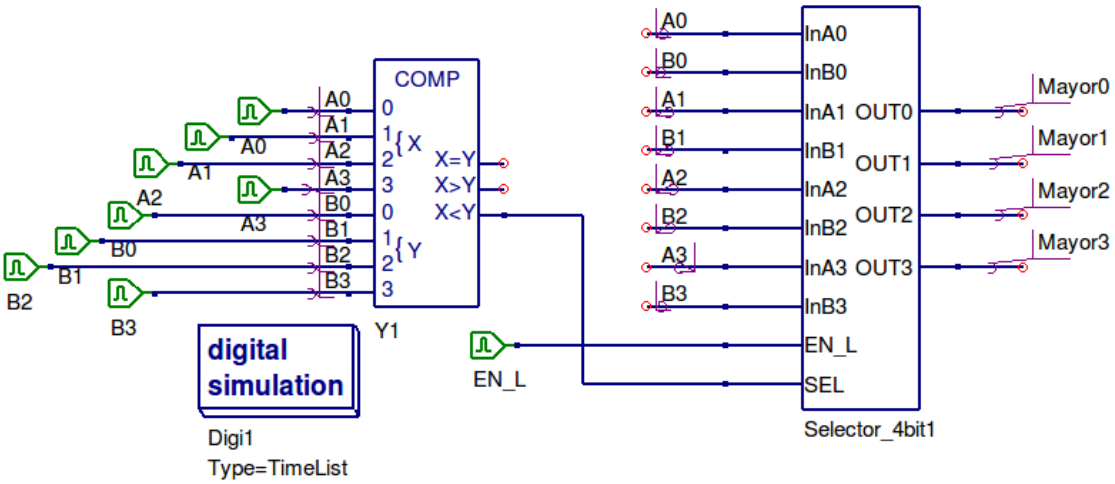


Figura 2.11 Esquema del comparador y selector.

2.6.2 Casos de estudio de circuitos secuenciales

El circuito elegido como ejemplo de diseño modular secuencial en TKGate es el contador de 8 bits de la Figura 2.12. Este caso fue el único tomado del conjunto de los ejemplos que vienen con el programa.

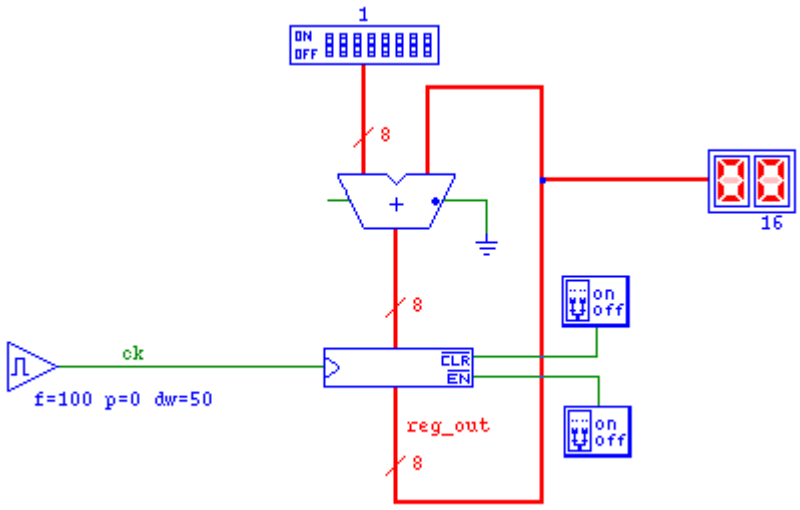


Figura 2.12 Esquema del contador con principio de suma y car en registro. Tomado de los ejemplo del software.

Este circuito es muy sugerente puesto a que TKGate no cuenta con un componente contador en su librería de circuitos MSI. El diseño cuenta con un registro simple con entradas *clear* y *enable* activas en cero; un sumador completo de 8 bits con la entrada de acarreo puesta en cero, una de las entradas de datos a un switch DIP puesto al valor de 1 en decimal y la otra entrada que viene de la salida del registro con el valor de la suma anterior. Básicamente lo que hace es sumar uno con cada borde de reloj. Para mostrar la salida tiene la pantalla de 7 segmentos de la derecha.

Una de las posibilidades de este circuito es que el contador podría tener módulos variables en función del valor del switch DIP. En otras palabras, el sumador puede incrementar en valores distintos de 1 y así el resultado es un código de salida que incrementa.

El otro caso de estudio secuencial trata de un problema en el que se entran datos sincrónicos de 4 bits y son comparados con un dato fijo de 4 bits. Si el dato de entrada es mayor, se incrementa un contador. Sin llegar a ser una máquina algorítmica, este problema se puede solucionar con un esquema básico de registro paralelo-paralelo, un comparador y contador sincrónico.

Como el caso anterior se implementó en TKGate, este se realizó en Qucs. Para ello se puede emplear el circuito del registro universal diseñado en el apartado 2.4 como un subcircuito, pero resultó más sencillo para el ejemplo el registro de la página 22 de (Brinson, 2006). El comparador ya viene definido en la librería del software. El contador por su parte fue adaptado de un ejemplo del software para convertirse en un subcircuito.

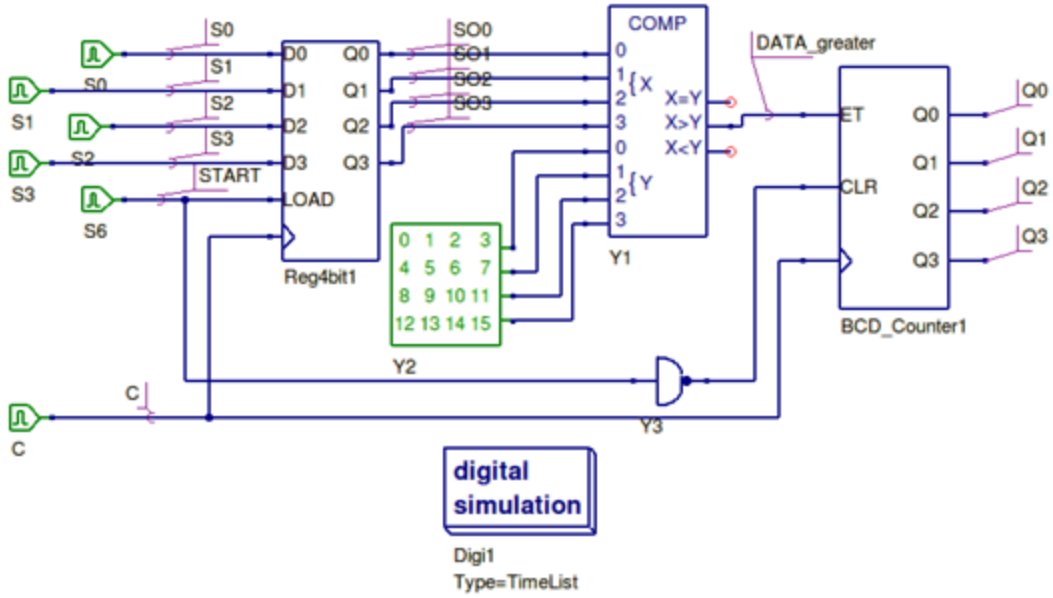


Figura 2.13 Esquema del caso del estudio secuencial para Qucs.

El modelo del registro pararalelo-paralelo funciona en carga sincrónica, es decir, si LOAD es 1 carga el dato en S0, S1, S2, S3 a Q0, Q1, Q2 y Q3 respectivamente. El modelo del contador es sincrónico BCD con una entrada de borrar activa en 1 y la entrada ET para mandar contar o no.

2.7 Utilización de VHDL en la simulación con Qucs

Las siglas VHDL vienen de VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. El VHDL es un lenguaje de descripción y modelado, diseñado para describir, en una forma que los humanos y las máquinas puedan leer y entender, la funcionalidad y la organización de sistemas hardware digitales, placas de circuitos, y componentes. Otros lenguajes de descripción hardware comúnmente empleados son el ABEL y el VERILOG. El ABEL permite la síntesis lógica (FSMs, funciones lógicas) pero en comparación con el VHDL, la sintaxis es demasiado limitada para la descripción de comportamiento.

El VHDL es un lenguaje que fue diseñado inicialmente para ser usado en el modelado de sistemas digitales. Es por esta razón que su utilización en síntesis no es inmediata, aunque lo cierto es que la sofisticación de las actuales herramientas de síntesis es tal que permiten implementar diseños especificados en un alto nivel de abstracción.

La síntesis a partir de VHDL constituye hoy en día una de las principales aplicaciones del lenguaje con una gran demanda de uso. Las herramientas de síntesis basadas en el lenguaje permiten en la actualidad ganancias importantes en la productividad de diseño. (M. et al., 2004)

Tanto Qucs como TKGate utilizan lenguajes de descripción de hardware. TKGate está especializado en Verilog, mientras Qucs puede utilizar tanto VHDL como Verilog. Desde este punto de vista y por otros puntos sobre este tema, Qucs vuelve a estar por encima de TKGate en cuanto a funcionalidad. Por interés en el VHDL entonces se trabaja con Qucs.

Qucs 0.0.8 se lanzó el 16 de enero del 2006 por el equipo de desarrollo de Qucs. Esta fue la primera versión del software que incluyó simulación basada en VHDL. El programa emplea FreeVHDL como motor de compilación del lenguaje (Scordilis et al., 2005). El trabajo a continuación en cuanto a la corrección de bugs y deficiencias del software ha sido continuo, lo cual se nota en su versión 0.015 del 2009 con la cual se simularon los siguientes casos de estudio.

El segundo en la simulación con Qucs, prácticamente transparente al usuario, es la conversión del esquemático del circuito en código VHDL que describe los componentes, sus conexiones y el *testbench* configurado. A continuación FreeVHDL convierte la descripción del circuito en código C++, que es compilado para formar una máquina ejecutable para la simulación del circuito digital. Qucs corre este ejecutable, colecta los datos de las señales en la medida en que los eventos aparecen y proyecta las formas de ondas o datos en formas de onda o tablas de verdad. (Scordilis et al., 2005)

El código VHDL generado por Qucs en su versión 0.0.8 está limitado en alcance por los siguientes factores:

- Las compuertas digitales son descritas mediante arquitectura *dataflow* concurrente.
- Los flip-flops y los generadores de señales se describen mediante procesos.

- Las conexiones o cables (señales), solo pueden ser de tipo bit como está definido en la librería estándar del VHDL.
- Las estructuras de bus no están soportadas.
- Las estructuras con subcircuitos se pueden anidar; tal funcionalidad ha sido probada hasta una profundidad de cuatro.

La Figura 5.10 de la página 97 de (Scordilis et al., 2005) presenta un diagrama de flujo con las rutas para la simulación con Qucs. En la página 54 de (Brinson, 2006) aparece otro diagrama en bloques detallado de mucho interés acerca de la compilación de código VHDL y procesamiento de resultados.

2.7.1 Estructuras de código VHDL

En su forma más básica, un código en VHDL está estructurado por un bloque *entity* y una arquitectura, o sea una bloque *architecture*. Los tipos de datos , funciones y operadores del paquete estándar son siempre visibles para el programa, por lo que no se necesita colocar referencias explícitas para su empleo. Sin embargo, si se emplean tipos de datos o funciones empleadas en otras librerías, como por ejemplo *std_logic* en la librería de IEEE, entonces es requerida una referencia a este recurso previo al par *entity-architecture*. Una forma general sería como sigue:

```
library ieee;
use ieee.std_logic_1164.all;
--
entity testbench is
        --entity body statements
end entity testbench;
--
architecture behavioural of testbench is
        -- architecture body statements
end architecture behavioural ;
```

La palabra de código *all* significa que todos los elementos en la librería estarán disponibles para su uso en el par *entity-architenture* que le sigue. Consecuentemente, si se necesitan más librerías, se añaden más sentencias para las mismas.

Códigos más completos consisten en más de un par *entity-architenture*. En tales casos el circuito *test bench*, que no es más que los vectores de prueba o estímulos, va al final.

CAPÍTULO 3. RESULTADOS DE LA EJECUCIÓN DE LOS EJEMPLOS PROPUESTOS

3.1 Preparación

El TKGate y el resto de los programas para la simulación de circuitos digitales utilizados en este trabajo fueron instalados y operados sobre Ubuntu 10.04 LTS. Esta distribución cuyo nombre es *Lucid Lynx*, fue la distribución con período de soporte prolongado anterior a la 12.04 disponible a partir de este año. Fue mucho más fácil el hecho de utilizar esta distribución de Linux pues todos los programas requeridos estaban al alcance, repositorios de software radicados en la propia Intranet UCLV.

Todas las operaciones fueron realizadas desde una máquina virtual instalada en VMWare Workstation versión 8.0.2 build-591240. Esto permitió salvar todo el trabajo: el sistema operativo, software instalado, configuraciones y los proyectos. Además permitió la portabilidad de todo el trabajo, es decir, si había que cambiar de máquina por alguna razón, bastaba con tener la máquina virtual y las características mínimas para continuar el desarrollo en otra computadora.

Para instalar los programas bastó con buscarlos en la base del repositorio e instalarlos, mediante el empleo de la herramienta con interfaz gráfica de usuario *Synaptic*. Este es un programa para instalar, eliminar o actualizar paquetes de programas que están dentro de la distribución. El resto de las configuraciones son pocas y serán mencionadas apropiadamente.

Todos los proyectos simulados fueron desarrollados desde cero, los ejemplos solo se usaron como apoyo para el trabajo. Los circuitos fueron cuidadosamente elegidos para cubrir varias áreas de la electrónica digital y ver la respuesta del software en ellas.

3.2 Montaje y simulación del multiplexor de 8 a 1 con el TKGate

El circuito de la Figura 2.5 fue montado en TKGate, como uno de los casos a analizar con esta herramienta. La colocación de las compuertas básicas NOR, NAND, inversores y buffers inversores fue relativamente fácil. El cableado, por otra parte se hace bastante engorroso al principio, aunque con un poco de dedicación y práctica se logra el resultado deseado.

Las compuertas lógicas son genéricas, en el sentido de que se coloca la compuerta y con clic derecho se le añaden entradas. Este programa permitió la colocación de buffers inversores, al contrario de QUCS; aunque esto no sea determinante en la simulación de los circuitos, resulta excelente para reproducir los modelos exactos de algunos de ellos.

La figura a continuación representa las formas de onda de las señales del multiplexor simulado. Obsérvese como el comportamiento del mismo es correcto en todo momento. La excitación representada en las entradas desde D0 hasta D7, S2, S1, S0 y EN_L, fue lograda por medio de interruptores sencillos.

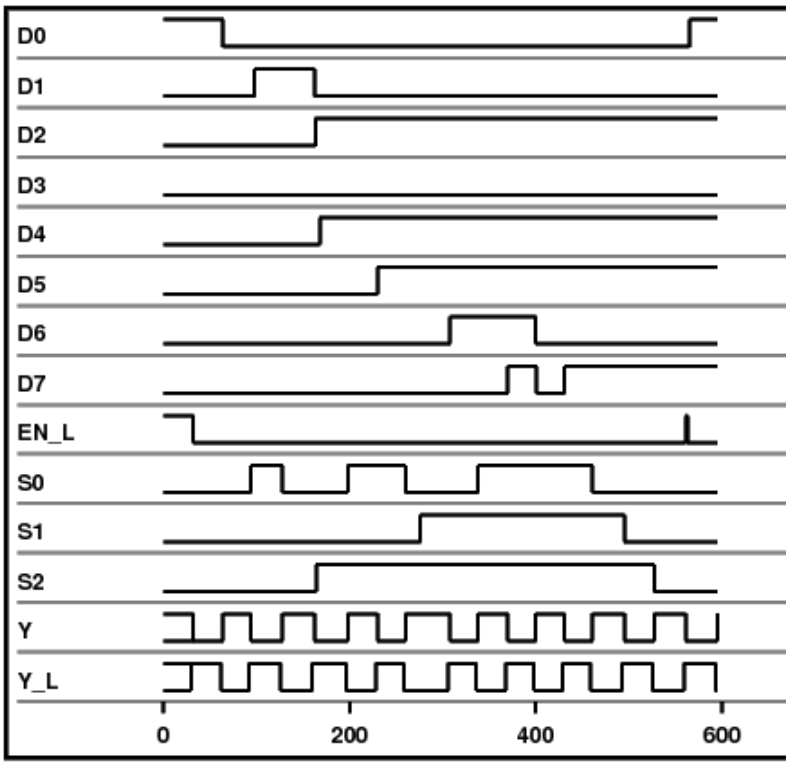


Figura 3.1 Trazas obtenidas con el TKGate del multiplexor de 8 a 1. La imagen se obtuvo mediante la impresión de las trazas en un archivo *PostScript*.

En tiempo de simulación, estos interruptores se pueden accionar lo cual introduce un cambio en las formas de onda de la ventana *scope* del simulador. Esto significa que las simulaciones en TKGate son interactivas, desde el punto de vista de que el usuario puede accionar componentes y encontrar resultados en otros como barras de LEDs y pantallas de 7 segmentos. Esto hace recordar a otros programas como el Proteus.

Las señales que aparecen en la ventana del simulador, son el resultado de la colocación de una punta de prueba mediante un doble clic en el cable deseado. El nombre que aparece en la venta será el del cable, es por eso que para mayor entendimiento del resultado es necesario renombrar los cables con la denominación apropiada. El nombre del cable puede hacerse visible o no en dependencia de como se quiera (ver Figura 3.2).

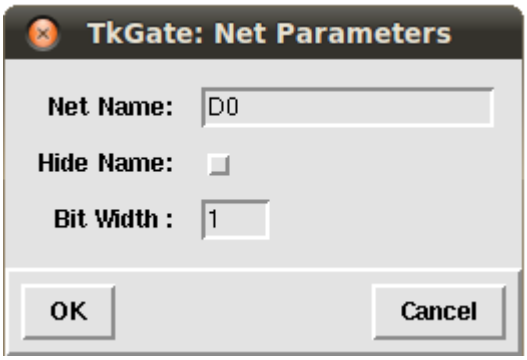


Figura 3.2 Cuadro de diálogo para los parámetros de los cables; véase el nombre, la cantidad de bits y el atributo de visibilidad. Tomado del proyecto del multiplexor mediante la captura de pantalla.

Cabe señalar que en la simulación de circuitos combinacionales, el resultado de un cambio en las señales solo se aprecia completamente a la salida cuando se realiza la próxima transformación. Es decir, si una de las entradas en el multiplexor anterior se cambia de forma tal que implique un cambio en la salida, dicho resultado solo se podrá apreciar completamente cuando se induzca el siguiente evento a la entrada. Esto tiene mucho sentido si se tiene en cuenta el concepto de la lógica combinacional y que las simulaciones no son en tiempo real.

Cabe señalar el mal sabor que dejó el software durante su operación. La GUI está basada en Tcl/Tk y tiene severas deficiencias en cuanto a los comandos que se le pasan. Por ejemplo: los comandos de teclado para copiar (Ctrl-X) y pegar (Ctrl-Y) elementos del esquemático no funcionan; solo pueden ser accionados desde el menú *Edit*. Además está el hecho de no poder salvar cómodamente los proyectos pues se presentaba un mensaje de archivo

corrupto. Tal mensaje no era cierto y constituye un *bug* oficial en distribuciones de Linux como Debian, la cual es muy parecida a la que se empleó. La evidencia de este planteamiento se puede encontrar en las páginas de reportes basadas en listas de distribución de correo electrónico. (kwantam@gmail.com, 2009)

Se piensa que una de las posibles formas de corregir estos problemas sería compilando la aplicación e instalándola paso a paso. Este software no da grandes pasos en su desarrollo, actualmente hay una versión 2.0 Beta 10 la cual promete en la erradicación de deficiencias como las anteriores.

Otro simple detalle en tiempo de simulación se presentó al no encontrar el camino del simulador. Este no fue un problema serio, de nuevo el paquete pre compilado para la distribución *Lucid Lynx* jugó un mala pasada. Fue resuelto por medio de la creación de un acceso directo o atajo mediante el comando siguiente:

```
# ln /usr/bin/gsim  /usr/share/tkgate-1.8.7/libexec/gsim
```

De cualquier forma, con un poco de trucaje fue posible salvar los archivos de los proyectos, cuya extensión es .v para su posterior utilización. Más adelante se ampliará sobre la utilización del TKGate con el resto de los escenarios simulados.

3.3 Montaje y simulación del decodificador de 3 a 8 en QUCS

El esquema de la Figura 2.6 fue montado en el QUCS como un proyecto completo. EL trabajo la GUI de este software es mucho más amigable que en el TKGate, al menos en la forma que se presenta este último en su versión 1.8.7. Los componentes colocados en la hoja corresponden al grupo de los componentes digitales, dado a que el software viene preparado con otros conjuntos de componentes para otros tipos de simulaciones.

Los comandos de teclado para copiar, cortar, pegar y cablear son funcionales. Además la vista por pestañas permite el trabajo con todos los archivos del proyecto a la vez.

En el circuito de la Figura 2.6 también se puede apreciar un bloque con el nombre *Digital Simulation*, el cual no es más que el componente que le indica el tipo de simulación que QUCS realizará. La simulación digital puede ser de tipo *TimeList* o *Truthtable*. Para este caso se seleccionó la primera con un tiempo de duración de 200ns (ver Figura 3.3)

Los estímulos de las entradas se configuran mediante las fuentes digitales en verde que se ven a la izquierda. Tal y como se muestra en la figura siguiente, los cambios de dichas fuentes parten de una valor lógico-digital especificado en *init* y cambian en cada tiempo especificado en *times*, separados por punto y coma.

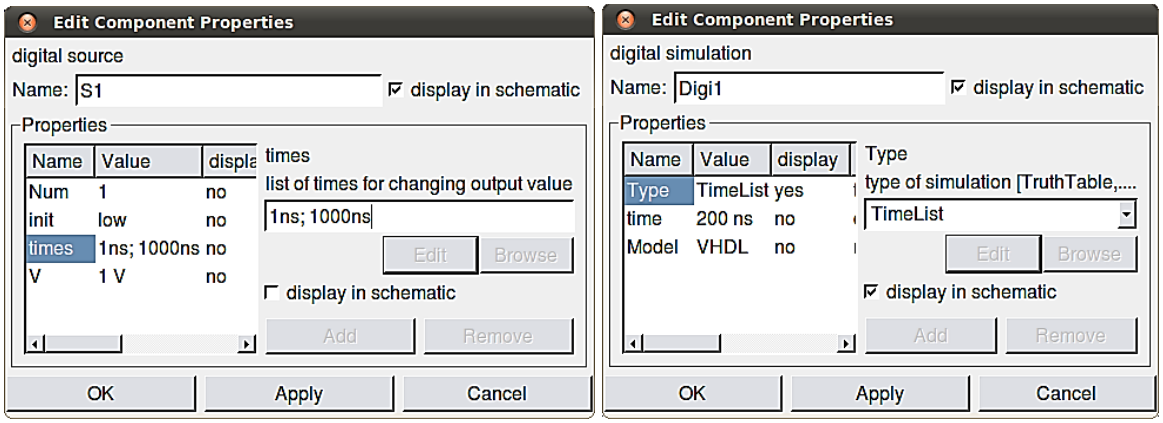


Figura 3.3 Cuadros de diálogos para la configuración de las fuentes digitales (izquierda) y para la configuración de las simulaciones (derecha). Tomado del proyecto del decodificador mediante captura de pantalla.

En la Figura 2.6 también se pueden apreciar las etiquetas de QUCS, colocadas intencionalmente en las entradas y en las salidas con tal de apreciar las señales en la simulación. Dichas etiquetas fueron: G1, G2A_L, G2B_L, A, B, C, Y0_L, Y1_L, Y2_L, Y3_L, Y4_L, Y5_L, Y6_L y Y7_L.

Una vez compilada la simulación, el resultado aparece en un archivo .dpl al cual se le pueden colocar diagramas de tiempo. El diagrama de tiempo correspondiente a la simulación del decodificado de 3 a 8 es el de la Figura 3.4.

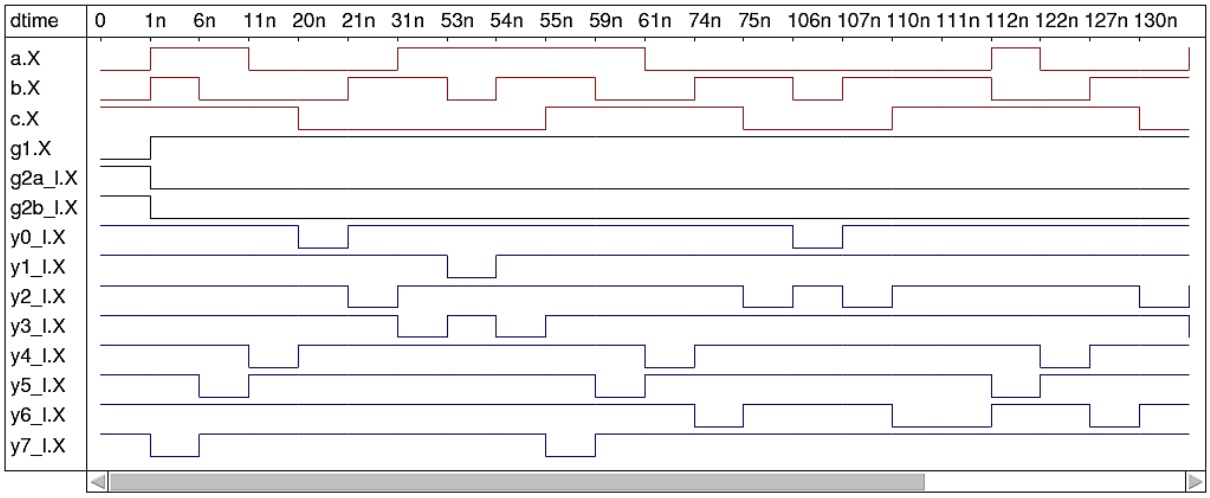


Figura 3.4 Formas de onda del decodificador de 3 a 8 realizado en QUCS. Imagen tomada mediante impresión en *PostScript*.

A diferencia del TKGate, el QUCS no permite la interacción con el circuito. Una vez que son establecidas las señales y compilado el circuito, se conforma un archivo de datos o *Dataset* que se mantiene hasta tanto no se compile la simulación otra vez.

Una de las cosas que más gustaron de QUCS es que el *netlist* se conforma en VHDL, a diferencia de TKGate que usa Verilog. El soporte para los lenguajes de descripción de hardware es muy bueno y pues además puede usar Verilog lo cual lo pone en ventaja desde el punto de vista de las posibilidades. Más adelante se presentarán otras simulaciones en el QUCS.

3.4 Registro universal de 4 bits

Para el registro de propósito general de 4 bits se montó el circuito presentado en el epígrafe 2.4. Este es el proyecto de circuito secuencial que fue concebido a compuertas para ser simulado en el QUCS, una de las principales razones para ello fue el hecho del relativamente grande grado de complejidad en el cableado de los componentes, así como su cantidad.

Los resultados reflejan un correcto funcionamiento del circuito. Es importante recordar que el modelo es ligeramente diferente con respecto al que aparece en (Wakerly, 2006). En la Figura 3.5 se muestran las formas de ondas de las entradas y salidas involucradas en el modelo del registro universal, similar al del 74194.

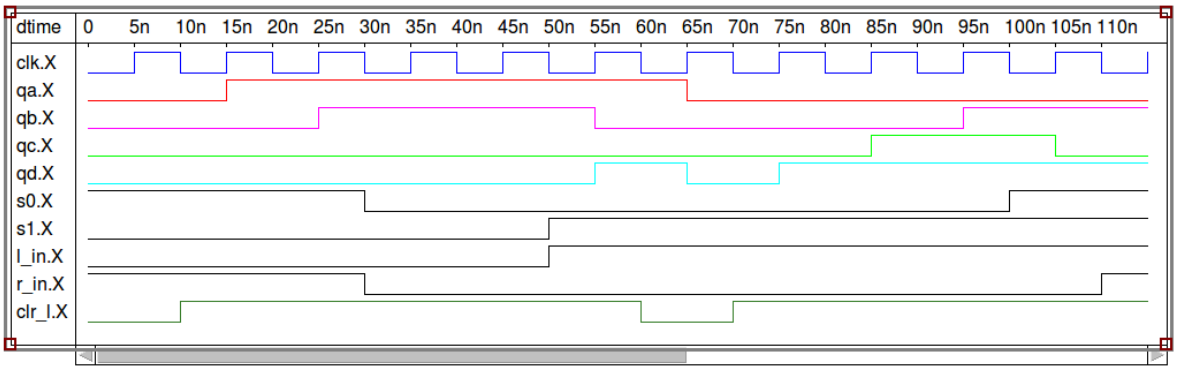


Figura 3.5 Resultado de la simulación del modelo similar al 74194 (captura de pantalla).

La duración de estas ventanas de señales, que se denominan *Timing Diagram* está determinada por el tiempo total de simulación configurado en el componente especial

Digital Simulation. Solo por esta vez se expondrá un ejemplo de la variante con tabla de verdad y la misma excitación que la Figura anterior.

	a.X	b.X	c.X	d.X	clk.X	clr_l.X	l_in.X	qa.X	qb.X	qc.X	qd.X	r_in.X	s0.X	s1.X
0111111001	1	1	1	1	1	0	0	0	0	0	0	1	0	1
0111111010	1	1	1	1	0	1	0	0	0	0	0	1	0	1
0111111011	1	1	1	1	1	1	0	0	0	0	0	1	0	1
0111111100	1	1	1	1	0	0	1	0	0	0	0	1	0	1
0111111101	1	1	1	1	1	0	1	0	0	0	0	1	0	1
0111111110	1	1	1	1	0	1	1	0	0	0	0	1	0	1
0111111111	1	1	1	1	1	1	1	0	0	0	1	1	0	1
1000000000	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1000000001	0	0	0	0	1	0	0	0	0	0	0	0	1	0
1000000010	0	0	0	0	0	1	0	0	0	0	0	0	1	0
1000000011	0	0	0	0	1	1	0	0	0	0	0	0	1	0
1000000100	0	0	0	0	0	0	1	0	0	0	0	0	1	0
1000000101	0	0	0	0	1	0	1	0	0	0	0	0	1	0

Figura 3.6 Fragmento de la tabla de verdad para el registro universal (captura de pantalla).

Para obtener una tabla de verdad o una simulación para formas de ondas es preciso cambiar la propiedad *Type* del componente *Digital Simulation* (ver Figura 3.3 derecha). Además es importante señalar que la compilación para la simulación del circuito demora más para la tabla de verdad, pues se trata de obtener todos los valores digitales de todas las variables y señales implicadas. Es por ello que cuando se quiera una tabla de verdad, debe ajustarse apropiadamente el tiempo de simulación a un valor no superior a los 50ms quizás. Además deben colocarse los estímulos o señales de entrada, de tal forma que se pueda observar lo suficiente como para satisfacer el objetivo de la simulación.

En este caso, nótese en la Figura 3.6 una barra de desplazamiento a la izquierda, con un índice que sugiere un gran tamaño. Esto es porque la tabla se calculó con el mismo tiempo de simulación que la lista de tiempo. Eso acarreó que la compilación demorara aproximadamente 4 minutos en el entorno de máquina virtual en que se corría todo el sistema.

3.5 Contador binario módulo 16

De vuelta al rústico pero interesante TKGate, se presentarán los resultados del contador sincrónico módulo 16 que modeló; este está presentado en el apartado 2.5 de este trabajo.

Con la simulación anterior con el TKGate solo se comentó sobre el hecho de las trazas de señales obtenidas. En este caso las formas de onda se pueden apreciar en la Figura 3.7, a continuación.

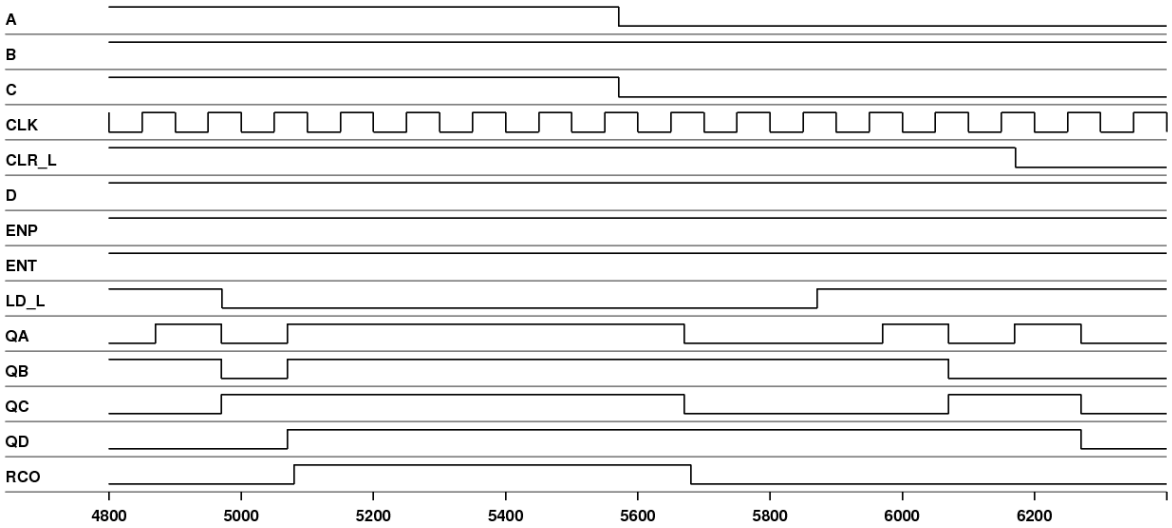


Figura 3.7 Comportamiento del contador ante una situación dada. Imagen obtenida con *PostScript*.

Una de las características mencionadas de este programa es su capacidad de interactuar con el usuario de forma didáctica. En la Figura 2.8 se mostró el esquema de este circuito. Para el mismo se le colocaron LEDs a la derecha, de forma tal que sin necesidad de ver la ventana de las señales se puede saber del comportamiento del contador. El resultado es fácilmente comprobable si, mediante el uso de la techa TAB, se adelanta paso a paso la simulación; se verá entonces los LEDs encenderse y apagarse en función de la configuración del circuito en ese momento. Otra variante pudo haber sido la utilización de una barra de LEDs o de una pantalla 7 segmentos, tal y como se muestran en los ejemplos de ayuda del software del contador de 3 bit y contador de 8 bits respectivamente.

3.6 Caso de estudio: registro de desplazamiento de barrera

En el epígrafe 2.6.1 se comentó sobre el problema del registro de desplazamiento de barrera realizado en el TKGate. Después del ya caracterizado como trabajoso proceso de cableado, se llegó a la simulación con las especificaciones siguientes:

Dato de entrada: $5_d = 00000101_b$

Valores para el Switch DIP de selección: 1, 2,3, 5, 2, 4 y 0

El resultado en la barra de LEDs fue el siguiente:

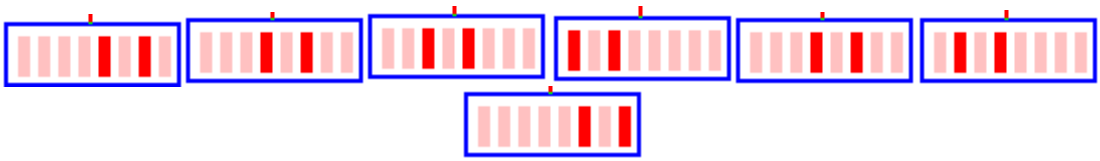


Figura 3.8 Comportamiento de la barra de LEDs ante la simulación del registro de desplazamiento de barrera. Obsérvese la secuencia esperada, de izquierda a derecha respectivamente.

En la imagen se aprecia claramente como el patrón de rojos activos coincide con la palabra dada. El desplazamiento está también en concordancia con el valor establecido en la entrada S de tres bits.

En la experiencia de la creación del trabajo se obtuvieron fenómenos indeseables con el programa que dificultaron aún más el avance. El programa cerró abruptamente después de un conjunto determinado de operaciones, esto evidencia su considerable inestabilidad. Si a ello se le suman los reportes de la otra simulación sobre el problema de salvar los archivos, el mal sabor es considerable.

3.7 Caso de estudio: comparador y selector

El problema del comparado y selector es bien sencillo, pero sirvió para comprobar la funcionalidad de Qucs con subcircuitos. Esta característica de este software es excelente y permite que el usuario ante un caso como este pueda flanquear la debilidad de con contar con una biblioteca ampliada de componentes modulares y la presencia de modelos no del todo buenos.

Para el ejemplo del apartado 2.6.1 se crearon estímulos y se simuló para obtener el resultado de la Figura 3.9. Para comprobar que el funcionamiento es el deseado basta con chequear que en *MAYOR* se contiene siempre el valor más alto de A y B.

Lo más relevante de este ejercicio fue el acto de creación del subcircuito y su utilización en el esquema general. El resto de las operaciones ya fueron examinadas en las simulaciones anteriores con Qucs.

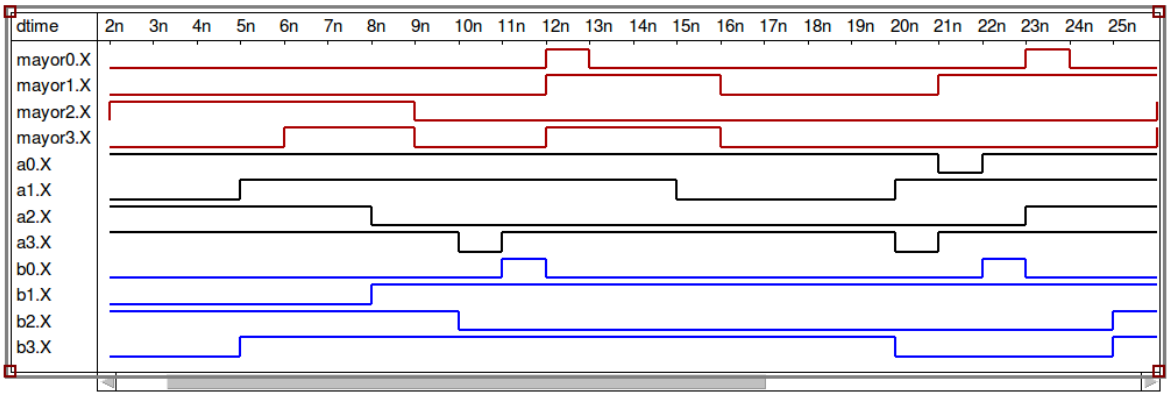


Figura 3.9 Comportamiento del circuito comparador selector.

3.8 Caso de estudio: contador realizado con sumador y registro

El circuito de este caso de estudio requiere de un conjunto de pasos para su funcionamiento. En el primer borde de la señal de reloj, el valor del registro aún no está determinado; es por eso que se requiere que el interruptor de *clear* esté accionado para inicializar el registro en todos 0s. Luego, con el switch de habilitación en 0, se acciona el interruptor de *clear* y en la pantalla comienzan a aparecer los valores del conteo incrementados.

Para poder apreciar bien el resultado, es conveniente ajustar la señal de reloj del ejemplo puesto a que la que trae consigo hace cambiar demasiado rápido la pantalla 7 segmentos. Otra posibilidad es realizar la simulación a saltos, el comando de teclado TAB.

Algo que no se ha expuesto hasta ahora es la aparición de valores de señales en las formas de onda. Es decir, en la ventana *scope* del simulador se pueden ver valores de varios bits, como se puede ver en la Figura 3.10.

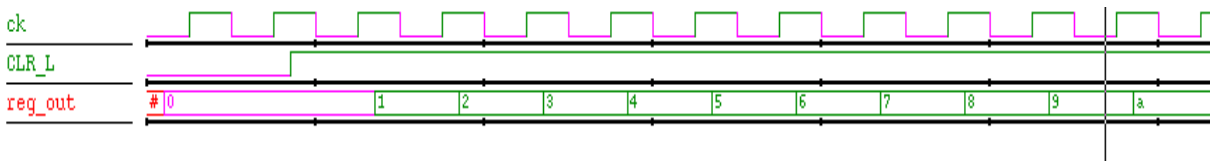


Figura 3.10 Valores de las trazas para el contador con un incremento de 1. Tomado mediante captura de pantalla.

3.9 Caso de estudio: contar si S es mayor que X

Este es el resultado de la simulación del segundo caso de estudio presentado en el epígrafe 2.6.2. Para ello se fijó un dato igual 4 y se editaron las señales de entrada para S0, S1, S2 y

S3. EL valor de reloj se puso a un valor de 100MHz y un ciclo útil de un 50%, es decir 1ns abajo y 1ns arriba. Es importante aclarar que los modelos de reloj para estas simulaciones no contemplan pendientes. Fue colocada también la señal de entrada START con una secuencia de 1ns;2ns;1000ns de la misma forma que en la Figura 3.3 izquierda.

El resultado es como sigue:

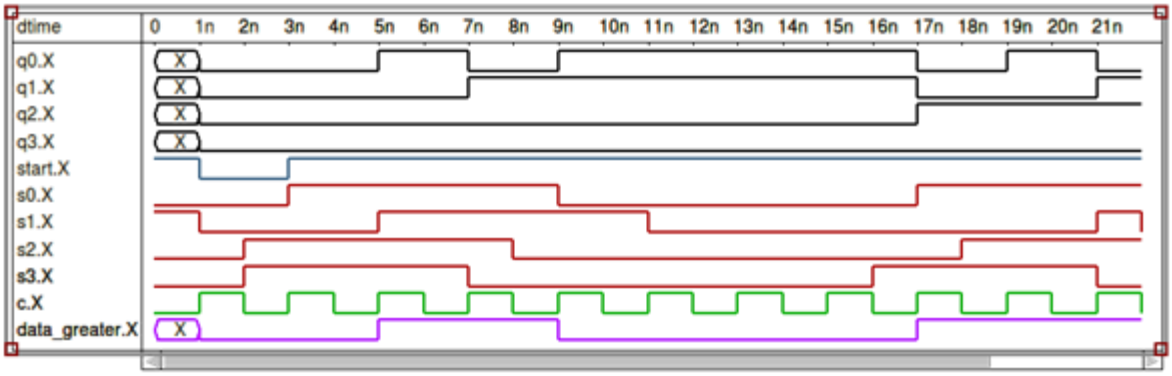


Figura 3.11 Resultado de la simulación del problema secuencial con diseño modular para Qucs.

Una debilidad importante que tiene el programa es, como se dijo antes, la no utilización de estructuras de bus. Esto dificulta interpretar un resultado como este.

Si se observa la figura, la señal *data_greater* indica muy bien cuando el registro tiene a la salida el dato mayor que cuatro. Claro, hay que inferir del gráfico que las señales S0, S1, S2 y S3 de entrada aparecen a la salida del registro hacia el comparador después de un ciclo completo de reloj. En cuanto *data_greater* es 1, el contador cuenta en cada borde de subida del reloj.

De nuevo el software evidencia su potencialidad en el diseño modular pues el contador y el registro fueron diseñados independientemente, colocados en un subcircuito y añadidos al proyecto por esta variante. El software permanece estable en todo momento.

3.10 Simulación con VHDL

Desde el momento que se encontró que el principio de simulación de Qucs para circuitos digitales recae totalmente en la traducción de los modelos a lenguaje VHDL, se supo que simular VHDL puro era una tarea más fácil en sí misma.

Como primer ejemplo se recreó, con el editor dedicado de Qucs, el ejemplo de la página129 de (Scordilis et al., 2005).

```
-- Avery basic test of data type integer.
entity testbench is
end entity testbench;
--
architecture behavioural of testbench is
signal A,B,C:integer:=0 ;
signal clk:bit;
begin
p0: process is    -- Generate clock signal.
        begin
                clk<='0'; wait for 10ns;
                clk<='1' ; wait for 10ns;
    end process p0;
--
p1: process (clk) is
        begin
                if (clk'event and clk='1') then
                        A <= A + 1;
                        B <= B + 2;
                end if ;
        end process p1;
        C <= A + B;
end architecture behavioural;
```

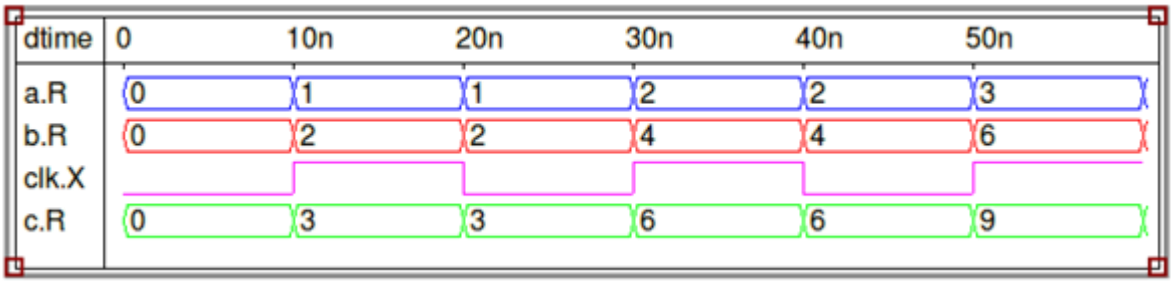


Figura 3.12 Formas de onda para el VHDL "Hello Integer".

El funcionamiento es correcto. El proceso p1 es un proceso básico secuencial. La declaración de entidad es la más básica, el precio es no poder utilizar el elemento como componente o subcircuito porque no tiene declarado los puertos.

Como este fueron recreados varios de los ejemplos encontrados en la bibliografía con éxito. A continuación se explican algunos de interés.

3.10.1 Representación de la lógica de bus

A pesar de que señales tipo bit son tan ampliamente usadas, una de las principales debilidades está en el hecho de que es difícil representar el bus con empleo de codificación con 1s y 0s. Por otra parte, la simulación de circuitos con contenciones de bus a menudo resulta en un fracaso. El paquete IEEE std_logic_1164 supera esta limitación mediante la

introducción de un sistema de lógica con varios valores que define nueve valores lógicos diferentes para representar los tipos de señales y los puntos fuertes de la señal. No sólo es el problema de contención de bus resuelto a través de la resolución de funciones lógicas, la lógica de varios valores permite la simulación de dispositivos construidos a partir de diferentes tecnologías de fabricación, al mismo tiempo, asegurando que el proceso de simulación refleje el circuito real desde su diseño.

El siguiente código VHDL fue recreado de (Scordilis et al., 2005) con el objetivo de comprobar esta funcionalidad.

```
library ieee;
use ieee.std_logic_1164.all;
--
entity testbench is
end entity testbench;
--
architecture behavioural of testbench is
signal clk: bit;
signal bv1: bit_vector (8 downto 0);
signal stdl1: std_logic_vector (8 downto 0);
signal INT1: integer:= 0;
signal INT2: integer:= 99;
signal R1: real:= 0.33;
signal R2: real:= 99.0;
signal R3: real:= 0.0;
signal R4: real:= 0.0;
begin
p0: process is
        begin
        clk <='0'; wait for 10 ns;
        clk <='1'; wait for 10 ns;
end process p0;
--
p1: process (clk) is
        variable v1:integer:=0;
        begin
        if (clk'event and clk='1') then
                v1:=v1+1;
                case v1 is
                when 1=>bv1<="000000000";
                stdl1<="000000000";
                when 2=>bv1<="000000001";
                stdl1<="000000001";
                when 3=>bv1<="000000011";
                stdl1<="00000001X";
                when 4=>bv1<="000000111";
                stdl1<="0000001XZ";
                when 5=>bv1<="000001111";
                stdl1<="000001XZU";
                when 6=>bv1<="000011111";
                stdl1<="00001XZUW";
                when 7=>bv1<="000111111";
                stdl1<="0001XZUWL";
                when 8=>bv1<="001111111";
                stdl1<="001XZUWLH";
                when 9=>bv1<="111111111";
                stdl1<="01XZUWLH-";
                when others=>v1:=0;
            end case;
        end if;
end process p1;
p3: process (clk) is
        begin
        if (clk'event and clk='1') then
                INT1<=INT1+1;
                INT2<=INT2-20;
        end if;
        --
        if (INT1>=9) then
                INT1<=0;
                INT2<=99;
        end if;
end process p3;
--
p4: process (clk) is
        Variable V2:real;
        begin
        if (clk'event and clk ='1') then
                R1 <= R1+1.0;
                R2 <= R2-20.0;
                R3 <= R1*R2;
                R4<=R2/(R1+0.0001);
        end if;
--
        if (R1>=20.0) then
                R1 <= 0.0;
                R2 <= 99.0;
        end if;
end process p4;
end architecture behavioural;
```

El resultado es como en la Figura 3.13 en la cual se observa como la conversión al formato de onda VCD (Value Change Dump) codifica las señales digitales en cuatro niveles lógicos diferentes: 0, 1, Z y X. La Tabla 3.1 muestra los nueve niveles lógicos admitidos en *ieee.std_logic* y su representación en VCD.

Tabla 3.1 Lógica IEEE de varios valore y su representación VCD.

Niveles de señal	VCD
'0' lógica 0 establecida	'0'
'1' lógica 1 establecida	'1'
'X' desconocido	'X'
'U' sin iniciar	'X'
'Z' alta impedancia	'Z'
'W' desconocido débil	'0'
'L' lógica 0 débil	'0'
'H' lógica 1 débil	'1'
'-' no importa	'X'

En la Figura 3.13 se puede comprobar como la transformación codifica las señales en cuatro niveles lógicos. Hasta que el estándar VCD sea revisado o mejorado, el paquete Qucs/FreeVHDL está restringido a mostrar la salida de las simulaciones empleando la codificación básica de la Tabla 3.1.

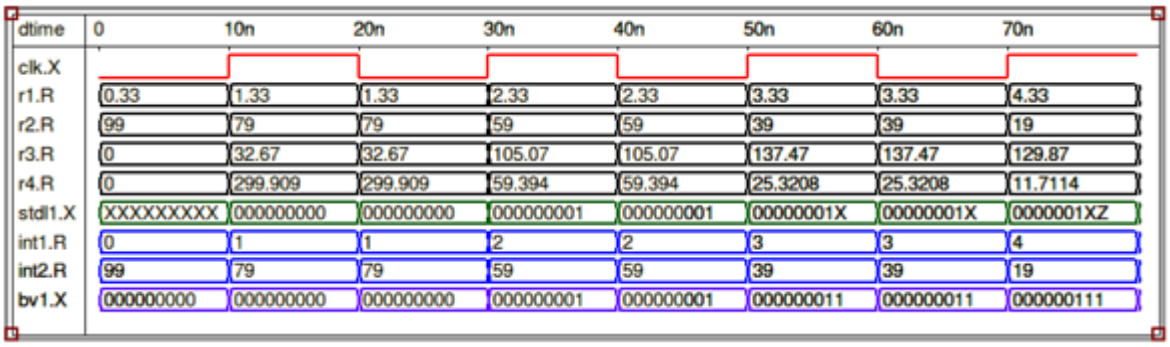


Figura 3.13 Representación de las señales con varios valores lógicos en formas de onda. El resultado fue idéntico al encontrado en la bibliografía.

En (Brinson, 2006) se presenta un código que simula dos buffers en tercer estado que pasan su salida a dos bus intermedios que se conectan a uno común. El bus intermedio asegura que la salida de los buffers en tercer estado se mantengan separadas antes de combinarse en el bus común. Esto permite que las salidas de las señales de los buffers en tercer estado y la

señal combinada sean representadas por separado. El resultado obtenido en la reproducción de este ejercicio fue idéntico al presentado en la fuente.

3.10.2 Multiplicador de 16 bits que demuestra las posibilidades de depuración

El lenguaje VHDL tiene un conjunto de características que permiten la depuración del código en tiempo de simulación. En esta sección las palabras reservadas del VHDL *assert*, *report* y *severity* se introducen y se explica su uso como código de depuración, apoyado en un ejemplo de diseño más elaborado de un multiplicador de 16 bits.

En el modo *estructural* un multiplicador de esta magnitud se hace engorroso. Para demostrar el potencial del VHDL, se preparó un diseño funcional. El diagrama en bloques para el caso de estudio aparece en la Figura 3.14.

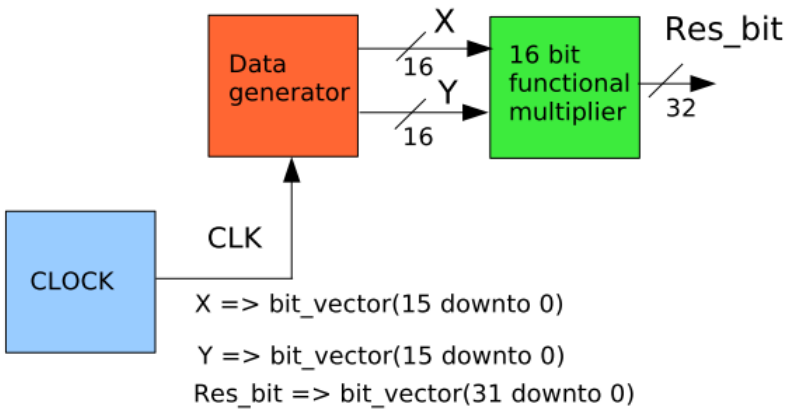


Figura 3.14 Diagrama en bloques de un multiplicador de 16 bits funcional. Tomado de (Brinson, 2006).

La unidad de generación de datos, genera dos secuencias de enteros por pulso de reloj. Estos son convertidos a *bit_vector* con tamaño 16 y aplicados a las entradas X e Y de la unidad de multiplicación. Finalmente la unidad multiplicadora censa X e Y para convertirlas a entero, multiplicarlos y convertirlos a *bit_vector* de tamaño 32. A pesar de que la aritmética de la librería estándar define operaciones aritméticas con enteros, no provee de funciones para la conversión de estos a *bit_vector*. El código del Anexo 2, recreado y adaptado de (Brinson, 2006), muestra la prueba completa incluyendo las funciones de conversión de entero a *bit_vector*.

Las sentencias de código de para depuración con las palabras reservadas *assert*, *report* y *severity* fueron añadidas al código de las arquitecturas *data_generator* y *functional_multiplier*.

Durante la simulación, estas cadenas de texto y el instante de tiempo cuando se accionan se escriben en el archivo *log.txt*, dando un registro de seguimiento de la actividad de simulación. En casos en los que se produce un error con nivel de gravedad *failure* la simulación terminará. FreeVHDL admite las sentencias *report* sin el acompañamiento de la sentencia *assert*, en concordancia con uno de los cambios realizados en 1993 del estándar IEEE VHDL 1076-1987.

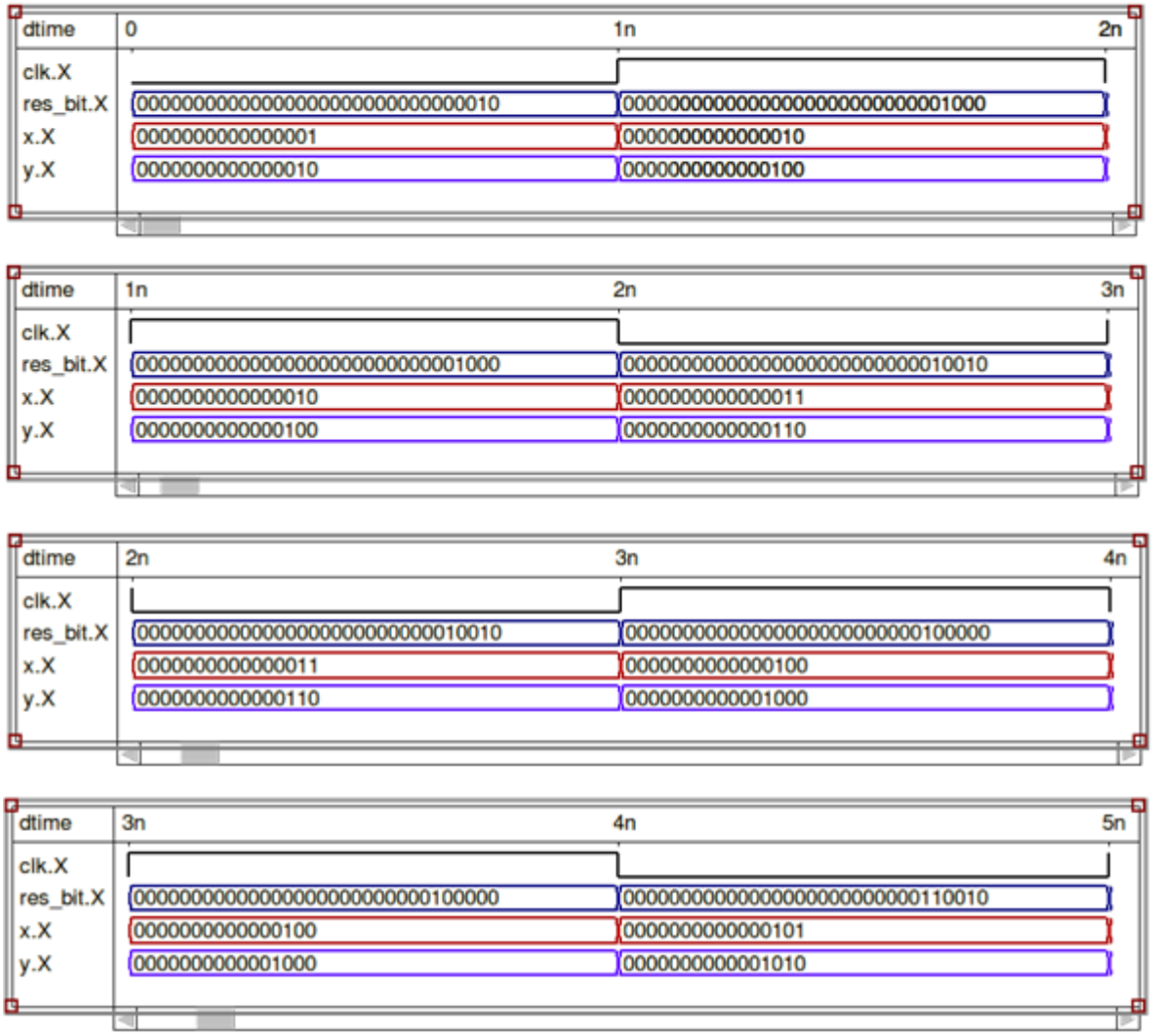


Figura 3.15 Simulación del multiplicador de 16 bits. La captura de pantalla evidencia el correcto funcionamiento del código.

3.10.3 Mensajes avanzados en Qucs

Mensajes de depuración más avanzados y tablas de resultados se puede escribir en el archivo *log.txt* de Qucs mediante el uso de las rutinas de manejo de datos predefinidas en el paquete de la biblioteca *textio*. Este paquete contiene las funciones de lectura y escritura de tipos de datos STD desde y hacia archivos. El siguiente segmento de código VHDL tomado y adaptado de (Brinson, 2006) ilustra cómo escribir una simple tabla de resultados en el archivo *log.txt*.

```
-- Test textio package.
--
library STD;
use STD.textio.all;
--
entity Qucs_write_test is
end entity Qucs_write_test;
--
architecture behavioural of Qucs_write_test is
begin
write_test : process is
        variable input_line, output_line: line;
        variable int1 : integer:= 10;
        begin
                write (output_line , string ' ( " " ) ) ;
                writeline (output, output_line) ;
                write (output_line, string'("String->log.txt"));
                writeline (output, output_line) ;
--
test_L1 :       for ic in 1 to 5 loop
                        int1:= int1 + 1 ;
                        write (output_line, string'("int1 = "));
                        write (output_line, int1);
                        write (output_line, string'("  int1*2 = "));
                        write (output_line, int1 * int1) ;
                        writeline (output, output_line);
                end loop test_L1 ;
                report "Finished test for loop.";
        end process write_test ;
end architecture behavioural ;
```

Esta es la salida en *log.txt* que se puede consultar directamente presionando F5:

Output:

Starting new simulation on Sun 17. Jun 2012 at 14:38:32

creating netlist... done.
running C++ conversion... done.
compiling functions... done.
compiling main... done.
linking... done.
simulating...

String->log.txt
int1 = 11 int1*2 = 121
int1 = 12 int1*2 = 144
int1 = 13 int1*2 = 169
int1 = 14 int1*2 = 196
int1 = 15 int1*2 = 225
0 fs + 0d: NOTE: Finished test for loop.
running VCD conversion... done.

Simulation ended on Sun 17. Jun 2012 at 14:38:33
Ready.

CONCLUSIONES

Como conclusiones del trabajo se tienen:

- Existen programas de software libre con diferente naturaleza que permiten la simulación y síntesis de circuitos digitales. Es un hecho que no están a la altura de los del software propietario, pero son una alternativa funcional. No todos se mantienen y mejoran en la misma medida, es por ello que no todos brindan la misma experiencia.
- Cuando de simulación de circuitos digitales se trata, se destacan los paquetes Qucs y TKGate. Existe otro: el Ktechlab que no fue probado por pertenecer a una distribución de Linux con interfaz distinta; de cualquier forma fue investigado a profundidad y en cierta medida no se mejora en la misma medida que Qucs, de hecho es referenciado poco. El gEDA es un paquete mucho más complejo y completo que, a pesar de poder simular circuitos digitales, tiene un alcance mucho mayor.
- Qucs y TKGate son los más amigables en cuanto a facilidades se trata. No obstante TKGate se presenta con dificultades en su programación y poca documentación al respecto. Quizás en su nueva versión 2.0beta se hayan mejorado algunos de sus errores, pero la velocidad de respuesta ante ellos es lenta. Qucs por su parte, es bueno y dentro de sus posibilidades funciona a la perfección. Su ciclo de desarrollo es mucho más corto, por lo que los errores se corrigen con frecuencia.
- TKGate se presenta con un modo interactivo; útil cuando se quiere apreciar una respuesta gráfica de la simulación.
- Todas las simulaciones realizadas fueron un éxito. Particularmente las que fueron realizadas con Qucs dieron menos problemas en su confección. Qucs es un paquete mucho más completo que tiene la hermosa funcionalidad de emplear lenguaje VHDL.
- Las bibliotecas de modelos de componentes son escasas en estos paquetes de programa. La funcionalidad modular de Qucs es muy superior y esto salva en la medida en que los usuarios sepan crear los modelos que necesitan para crear un diseño jerárquico y modular.

RECOMENDACIONES

Con el objetivo de dar seguimiento al presente trabajo se proponen las siguientes recomendaciones.

- Investigar sobre la existencia de programas avanzados para la síntesis y simulación de circuitos digitales sobre Linux, de código abierto o no. Esto está en función del buen desempeño de este tipo de herramientas en este sistema operativo.
- Utilizar Qucs en simulaciones avanzadas con VHDL de varios niveles de jerarquía, de forma que permita medir el verdadero potencial del software.
- Tratar de encontrar una forma sencilla de utilizar el paquete gEDA con simulaciones meramente digitales y con VHDL.

REFERENCIAS BIBLIOGRÁFICAS

BIRNBAUM, M. D. 2004. Essential Electronic Design Automation (EDA). Prentice Hall.

BOZDOC, M. 2004. *History of CAD/CAM* [Online]. Available: http://mbinfo.mbdesign.net/CAD-History.htm [Accessed 17-5-2012.

BRINSON, M. 2006. Qucs - A Tutorial - Getting Started with Digital Circuit Simulation.

CASTELLÓ, E. M. & VALIDO, M. R. 2010. Tutorial de Xilinx ISE. Available: http://webpages.ull.es/users/emagcas/index_archivos/apuntes_sed/mux2a1_tutorial.pdf [Accessed 25 febrero 2010].

CHARRAS, J.-P. 2012. *KICAD GPL PCB SUITE* [Online]. Available: http://iut-tice.ujf-grenoble.fr/kicad/ 17-5-2012].

CORPORATION, N. I. 2012. *What Is NI Multisim?* [Online]. National Instruments Corporation. Available: http://www.ni.com/multisim/whatis/ [Accessed 17-5-2012.

CUÉLLAR, R. V. B. 2005. Herramientas electrónicas para LINUX. Circuitos digitales. Tkgate 1.8 *Revista Digital "Investigación y Educación"*.

DAC. 2012. *Design Automation Conference* [Online]. MP Associates, Inc. Available: http://www.dac.com/ [Accessed 17-5-2012.

DUNCAN, S. 2012. *Altium Documentation* [Online]. Available: http://wiki.altium.com/display/ADOH/Altium+Designer [Accessed 17-5-2012.

HANSEN, J. P. 2012. *TkGate 1.8* [Online]. Available: http://www.tkgate.org/ [Accessed 17-5-2012.

KUEHLMANN, A. 2003. The Best of Iccad: 20 Years of Excellence in Computer-Aided Design. Springer Verlag.

KWANTAM@GMAIL.COM. 2009. *tkgate: 1.8.7 complains of corrupted files* [Online]. Debian-bugs-dist Navigation. Available: http://osdir.com/ml/debian-bugs-dist/2009-11/msg01186.html [Accessed 17-5-2012.

M., B., AL-HADITHI & MURO, J. S. 2004. NUEVAS TENDENCIAS EN EL DISEÑO ELECTRÓNICO DIGITAL: CODISEÑO HARDWARE/SOFTWARE *Tecnologí@ y Desarrollo - Revista de Ciencia, Tecnología y Medio Ambiente* [Online], II. Available: http://www.uax.es/publicaciones/archivos/TECELS04_001.pdf [Accessed 16-5-2012].

MANO, M. M. & CILETTI, M. D. 2004. Digital Design. 4th ed. Upper Saddle River: Prentice Hall.

MAXFIELD, C. 2006. FPGA Architectures from 'A' to 'Z'. *EE Times*.

MENDIGUCHÍA, M. N. 2007. Captura de esquemáticos con Xilinx. Available: http://dac.escet.urjc.es/docencia/ETC-ITIG_LADE/practicas-cuat1/practica1.pdf [Accessed 26 febrero 2010].

PROYECT, G. 2011. *gEDA Tool Suite on-line documentation* [Online]. gEDA Proyect. Available: http://geda.seul.org/wiki/geda:documentation [Accessed 17-5-2012.

ROSADO, A. & BATALLER, M. 2003. Práctica 1. Introducción al software Xilinx ISE Version 6. Available: http://www.uv.es/rosado/dcse/prac1XilinxISEintrod.pdf [Accessed 25 febrero 2010].

ROTH, C. H. & KINNEY, L. L. 2009. Fundamentals of Logic Design. *In:* GOWANS, H. (ed.) 6th ed.: Cengage Learning.

SCORDILIS, T., BRINSON, M., KRAUT, G., JAHN, S. & PITCHER, C. 2005. Qucs - Workgroup. Available: http://iut.geii.montp2.free.fr/telechargement/libre/Logiciels/qucs/workbook.pdf [Accessed 16-5-2012].

SCRA 2006. STEP Application Handbook: ISO 10303. *In:* SCRA (ed.) 3rd ed. North Charleston.

SYSTEMS, C. D. 2012. *Cadence OrCAD Solutions* [Online]. Cadence Design Systems Inc. Available: http://www.cadence.com/products/orcad/pages/default.aspx [Accessed 17-5-2012.

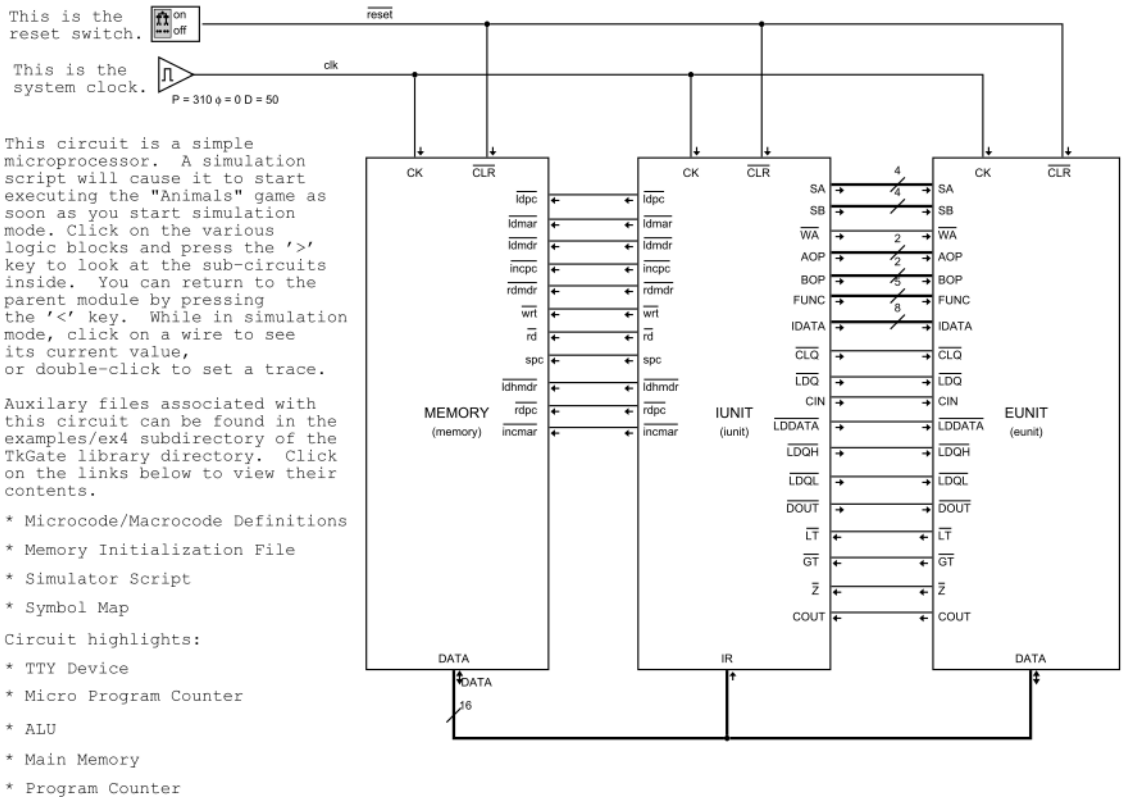
TEAM, Q. 2011. *Qucs project* [Online]. QUCS Team. Available: http://qucs.sourceforge.net/ [Accessed 17-5-2012.

TECHNOLOGIES, C. 2009. *FPGA design quick start guide* [Online]. CORE Technologies. Available: http://www.1-core.com/library/digital/fpga-design-tutorial/quick-start-guide.shtml [Accessed 17-5-2012.

WAKERLY, J. F. 2006. Digital Design. Principles and Practice. 4 ed. New Jersey: Prentice Hall.

ANEXOS

Anexo 1: Ejemplo *Menagerie CPU* de Tkgate.



Más información sobre este ejemplo se puede buscar en el propio software, en la versión 1.8.

Anexo 2: Código VHDL para el multiplicador de 16 bits.

```
-- 16 bit digital multiplier example.
-- Simulation trace using assert, report and severity statements.
--
entity clock is
        port (clk: out bit);
end entity clock ;
--
architecture behavioural of clock is
begin
p0: process is
        begin
        clk <='0' ; wait for 1 ns;
        clk <='1' ; wait for 1 ns;
end process p0;
end architecture behavioural;
--
entity data_generator is
port (clk: in bit;
        x,y: out bit_vector(15 downto 0));
end entity data_generator;
--
architecture behavioural of data_generator is
type mem_array_16 is array (1 to 8) of integer;
signal count: integer:= 0;
--
function integer_to_vector_16 (int_no: integer ) return bit_vector is
variable ni : integer;
variable return_value: bit_vector (15 downto 0);
begin
```

```
        assert (ni<0)
                report "Function integer_to_vector_32: integer number must be >= 0"
                severity failure;
        ni:= int_no;
        for i in return_value'Reverse_Range loop
                if ((ni mod 2)=1) then return_value (i):= '1';
                else return_value(i):= '0';
                end if ;
                ni:= ni/2;
        end loop ;
  return return_value ;
end integer_to_vector_16;
--
begin
p1: process (clk) is
variable xi: mem_array_16:= (1,2,3,4,5,6,7,8);
variable yi: mem_array_16:= (2,4,6,8,10,12,14,16);
variable xh,yh: integer;
variable counti: integer ;
begin
        counti:= count+1;
        if (counti>8) then
                counti:= 1;
        end if ;
        xh:= xi(counti);
        yh:= yi(counti);
        x<=integer_to_vector_16(xh);
        y<=integer_to_vector_16(yh);
        count<=counti;
        report "In process p1.data_generator.";
end process p1 ;
end architecture behavioural ;
```

```
--
--
entity functional_multiplier  is
port (x,y: in bit_vector (15 downto  0);
        res_bit: out bit_vector (31 downto  0));
end entity functional_multiplier;
--
--
architecture  behavioural  of functional_multiplier  is
--
function  vector_to_integer (v1: bit_vector) return  integer is
variable  return_value:  integer := 0;
alias v2: bit_vector (v1'length-1  downto  0) is v1;
begin
        for i in v2'high  downto  1 loop
                if (v2(i)='1')  then
                        return_value:=  (return_value  +1)*2;
                else
                        return_value:=  return_value*2;
                end if ;
        end loop ;
        if v2(0)='1' then return_value:=   return_value  +1;
        end if ;
        return return_value;
end vector_to_integer;
--
function  integer_to_vector_32 (int_no: integer) return  bit_vector is
variable  ni: integer;
variable  value: bit_vector (31 downto  0);
begin
        assert (ni<0)
                report  "Function  integer_to_vector_32: integer number  must be >= 0"
```

```
                severity failure;
        ni:=  int_no;
        for i in 0 to 31 loop
                if ((ni mod 2)=1) then value (i):='1';
                else value (i):='0';
                end if;
                if ni>0 then ni:= ni/2;
                else ni:= (ni-1)/2;
                end if ;
        end loop;
   return value;
end integer_to_vector_32 ;
--
begin
p0: process (x,y) is
variable xi, yi, prod_mult:  integer;
begin
        xi:=  vector_to_integer(x);
        yi:=  vector_to_integer ( y ) ;
        prod_mult:=  xi*yi;
        res_bit<=integer_to_vector_32(prod_mult);
        report  "In process p1.functional_multiplier";
end process p0;
end architecture  behavioural;
--
entity test_2_vhdl1 is
end entity test_2_vhdl1 ;
--
architecture  behavioural  of test_2_vhdl1 is
signal clk: bit;
signal x, y: bit_vector (15 downto  0);
signal res_bit: bit_vector (31 downto  0);
```

```
--
begin
d1 : entity work.clock  port map (clk);
d2 : entity work.data_generator  port map(clk,x,y);
d3 : entity work.functional_multiplier  port map (x,y,res_bit);
end architecture  behavioural  ;
```