Universidad Central "Marta Abreu" de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Telecomunicaciones y Electrónica

TRABAJO DE DIPLOMA

Propuesta de ejemplos integradores para la asignatura Diseño Digital VLSI.

Autor: *Lianet Guerra Morales*

Tutor: *Dr. Juan Pablo Barrios Rodríguez*

Santa Clara

2014

"Año 56 de la Revolución"

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Telecomunicaciones y Electrónica

TRABAJO DE DIPLOMA

Propuesta de ejemplos integradores para la asignatura Diseño Digital VLSI.

Autor: Lianet Guerra Morales

E-mail: lianetg@uclv.edu.cu

Tutor: Dr. Juan Pablo Barrios Rodríguez

E-mail: barrios@uclv.edu.cu

Santa Clara

2014

“Año 56 de la Revolución”

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 publicado 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

El amor de madre es el combustible

que le permite a un ser humano hacer lo imposible.

Marion C. Garretty.

DEDICATORIA

A mi madre la luz de mi vida por todos su sus sacrificios, amor y dedicación hasta el final de sus días. Por haberme convertido en la persona que soy y por estar en mi corazón dándome las fuerzas para seguir cuando el camino se vuelve cuesta arriba. Con todo mi amor y mis fuerzas todos los triunfos de mi vida serán para ti.

AGRADECIMIENTOS

A María Antonia Morales, mi madre y mi padre en estos 23 años que dedico su vida a la mía y hoy no me acompaña en el logro de este sueño.

A Lisset Morales, mi hermana de sentimiento por todo su apoyo, amor y comprensión aun en la distancia por ser mi sostén cuando todas las fuerzas flaquearon.

A mi hermana, mis tías (mis segundas madres), mis tíos, mis primos y toda mi familia que me apoya y ayuda a salir en estas circunstancias de mi vida cuando todo es tan difícil.

A mis amigas Lili y Sesy por quererme y soportarme siempre y ser incondicionales a mis reclamos.

A mi hermano Bufandis por ser esa personita tan especial que siempre encuentra la palabra precisa para hacerme feliz.

A mis compañeros de mil batallas, mis amigos para siempre Allen, Roberto y Yudis.

A mis compañeras de cuarto Any, que me dio la oportunidad de conocer mi provincia preferida Ciego de Avila, Sumito la más resabiosa de todas pero una excelente persona a todas, que me apoyaron en mis momentos de dolor y que me acompañaron en mis momentos de alegría.

A todos mis compañeros de aula por los magníficos momentos q compartimos juntos, por hacerme sentir orgullosa de la carrera que estudio.

A todos esos profesores que demostraron que más que buenos profesionales son excelentes personas.

A todas las personas que no menciono pero que han formado parte de mi vida.

A todos gracias

TAREA TÉCNICA

Para confeccionar el presente trabajo y alcanzar los resultados esperados, fue necesario elaborar las tareas técnicas siguientes:

- Descripción de las tendencias tecnológicas internacionales relacionadas con el diseño digital VLSI.
- Descripción de las habilidades generales básicas e invariantes de contenido relacionadas con el Diseño Digital VLSI.
- Análisis del Programa Analítico de la asignatura Diseño Digital VLSI.
- Desarrollo de un conjunto de ejemplos prácticos que integren el sistema de contenido y habilidades de la asignatura.
- Comprobación de los resultados en el kit de desarrollo Nexys 2.
- Elaboración del informe final del Trabajo de Diploma.

Firma del Autor	Firma del Tutor

RESUMEN

La presente investigación se dedica al desarrollo de un conjunto de ejemplos integradores para las prácticas de laboratorio de la asignatura Diseño Digital VLSI, que permiten abarcar los procedimientos y habilidades fundamentales del diseño digital de alta escala de integración. Con este fin se hace un análisis de la evolución histórica y de los componentes actuales del diseño electrónico digital, haciendo énfasis en los lenguajes de descripción de hardware y en los dispositivos lógicos programables. Se presenta una metodología general de diseño y las herramientas de software utilizadas en el desarrollo y comprobación de los ejemplos propuestos. Con la realización de este trabajo se demostró la relevancia de los ejemplos seleccionados y su desarrollo en prácticas de laboratorio pues permiten que los estudiantes dominen una metodología muy cercana a la que es empleada internacionalmente en el diseño digital, vinculando los procesos de modelación, descripción, simulación, implementación y análisis de resultados que forman parte de la habilidad compleja “diseñar”.

TABLA DE CONTENIDOS

INTRODUCCIÓN

Dentro de las fundamentaciones del Plan de Estudios D de la carrera de Telecomunicaciones y Electrónica se plantea que: "Los cambios constantes y cada vez más rápidos en las esferas culturales, políticas y socio-económicos en Cuba han traído un desarrollo impetuoso en las Telecomunicaciones y la Electrónica produciéndose un salto tecnológico con la introducción de las técnicas más actuales … Se está produciendo, igualmente, una explosión en la transmisión de datos, la utilización de las redes públicas y privadas de prácticamente todos los organismos" (MES, 2007).

Las exigencias del país en los campos de las telecomunicaciones, la electrónica, la automatización y la informatización han ido en incremento. En ese sentido los Lineamientos de la Política Económica y Social de Cuba plantean: "Actualizar los programas de formación e investigación de las universidades en función de las necesidades del desarrollo económico y social del país y de las nuevas tecnologías (Lineamiento 152)", así como: "elevar el rigor y la efectividad del proceso-docente educativo para incrementar la eficiencia del ciclo escolar" (Lineamiento 151 (2011).

Internacionalmente la Electrónica (y la Informática), ha dejado de ser una disciplina que tenía sus mayores aplicaciones en su propio campo o en investigaciones de élite, para convertirse en una disciplina de gran horizontalidad que sirve de base al desarrollo de múltiples aplicaciones tales como la informática, las comunicaciones, la automatización, la televisión y la gran industria del ocio.

A lo anterior se añade el proceso de migración, casi total, de la electrónica del campo analógico al campo digital, por las múltiples potencialidades conocidas del segundo. Por lo que se puede afirmar que la electrónica digital es el basamento común de casi todas las aplicaciones electrónicas existentes y futuras.

El Plan de Estudios D establece, dentro de su currículo base, la impartición de dos asignaturas vinculadas a los fundamentos teórico-prácticos de la electrónica digital, las cuales se nombran Electrónica Digital I (ED I) y Electrónica Digital II (ED II).

Dichas asignaturas cumplen como objetivo fundamental que los estudiantes desarrollen habilidades de diseño (principal) y análisis de sistemas digitales (combinacionales y secuenciales) típicos, que le permitan implementar aplicaciones de media a alta complejidad a partir de conocer los principios y modelos lógicos de descripción de éstos.

En el departamento de Telecomunicaciones y Electrónica de nuestra facultad existe experiencia en el desarrollo de aplicaciones digitales, utilizando tanto la lógica cableada (Electrónica Digital con componentes MSI y LSI), como la lógica programada (Microprocesadores y Microcontroladores).

Esta experiencia ha venido evolucionando a tono con el desarrollo de las tecnologías de fabricación de circuitos integrados digitales, las que actualmente se clasifican en VLSI y UVLSI y en las que se encuentran un amplio espectro de dispositivos digitales configurables tales como los CPLDs (Complex Programmable Logic Devices) y las FPGAs (Field Programmable Gate Arrays).

En respuesta a las necesidades económicas y sociales antes citadas se ha venido perfeccionando el Plan de Estudios D, en las múltiples actividades de la Comisión Nacional de Carrera, introduciéndose en el mismo un grupo de asignaturas optativas que satisfagan la formación tecnológica integral de los graduados de Telecomunicaciones y Electrónica en consonancia con dichas necesidades y con las tendencias mundiales en este perfil.

Tal es el caso de la asignatura optativa: “Diseño Digital VLSI”, la cual tiene como objetivo, profundizar en el diseño electrónico digital (hardware configurable) como complemento a los objetivos que se alcanzaron en las asignaturas ED I y ED II.

La asignatura “Diseño Digital VLSI” (DDVLSI) abarca una temática que está en la frontera de conocimiento e investigación de le electrónica mundial.

Las exigencias de reducción de costes, tiempo de establecimiento en el mercado (time to market, en inglés), fiabilidad, portabilidad, mantenimiento y reactualización de las aplicaciones, han llevado a que los procesos de diseño electrónico digital sean cada vez más

cambiantes y estén muy relacionados, en una sinergia total, con los medios computacionales.

La acelerada evolución tecnológica en el campo de la microelectrónica ha conllevado a un déficit cada vez mayor entre las capacidades de integración que permite la tecnología de circuitos integrados y las capacidades de realización que los diseñadores pueden incorporar en dichos circuitos.

Para reducir este déficit es necesario reemplazar las técnicas tradicionales de diseño por las llamadas técnicas avanzadas de diseño de sistemas electrónicos digitales complejos, también conocidas como técnicas de diseño digital de alto nivel.

Estas técnicas conforman un paquete tecnológico que agrupa a los lenguajes de descripción de hardware, las técnicas de reusabilidad con énfasis en la utilización de módulos de Propiedad Intelectual, la utilización de dispositivos lógicos programables, las técnicas de codiseño hardware/software y el desarrollo de los sistemas electrónicos empotrados (Savage et al., 2000). Aunque algunas de estas técnicas pueden ser utilizadas de forma independiente para la solución de determinadas aplicaciones simples, su verdadera potencialidad en el desarrollo de sistemas electrónicos digitales complejos radica en la utilización conjunta de las mismas. Así, mediante la utilización de estas novedosas técnicas se posibilita el incremento de la productividad de los diseñadores y se reduce el déficit relativo a la capacidad de integración.

El desarrollo de la habilidad profesional diseño electrónico digital (diseñar) exige un estudio descriptivo-comparativo del tratamiento de la misma en el currículo de universidades foráneas y nacionales, visto este último como proyecto y proceso (Tyler, 1971, Sanz, 2004), reflejo de las tres fuentes básicas: la sociedad, la especialidad y los alumnos.

En algunos casos aparece como: Electrónica Digital, Sistemas Digitales y en otros como Diseño Digital, lo que revela tanto la complejidad de esta temática como la finalidad de la misma.

En la mayoría de los planes de estudio analizados de diferentes universidades, la Electrónica Digital se imparte como asignatura obligatoria en carreras de ingeniería, entre las que se encuentran: Telecomunicaciones, Telecomunicaciones y Electrónica,

Electrónica, Biomédica, Industrial (modelo europeo), Mecatrónica (Latinoamérica), y en Ciencias de la Computación. Se imparte como una sola asignatura de duración anual en el sexto y séptimo semestre, aunque también se puede encontrar como dos asignaturas-semestre (Electrónica Digital I y Electrónica Digital II) e incluso, de forma separada las asignaturas de fundamentos teóricos y las dedicadas netamente al desarrollo de los laboratorios.

Es por todo lo anterior que esta asignatura no puede quedarse estática antes estos cambios y exige una sistemática actualización de su sistema de contenidos y habilidades, así como de la preparación del colectivo de profesores que la imparten.

Existen dos necesidades que justifican la realización del presente trabajo. La primera está relacionada con el dominio y actualización tecnológica sobre la temática de los procesos de diseño digital VLSI, a nivel internacional, en particular con las FPGAs de Xilinx, líder a nivel mundial en la fabricación de estos dispositivos que tienen gran impacto en múltiples aplicaciones (controladores de comunicaciones, controladores de buses, procesamiento digital de señales, etc.).

La segunda está relacionada con la necesidad académica de sistematizar los conocimientos tecnológicos anteriores en la asignatura Diseño Digital VLSI (DDVLSI), de manera que, a partir de la identificación de las invariantes de contenidos y habilidades (Alvarez, 1986) se puedan satisfacer los objetivos de formación del profesional de Telecomunicaciones y Electrónica que se declaran en el nuevo Plan de Estudios.

Un aporte a estas dos necesidades de perfeccionamiento de la asignatura Diseño Digital VLSI lo constituye el desarrollo de un conjunto de ejemplos integradores que permitan abarcar los procedimientos y habilidades fundamentales del diseño digital de alta escala de integración.

Las prácticas de laboratorio son fundamentales en cualquier disciplina, pero en la objeto de estudio revierten particular importancia por cuanto permiten que los estudiantes dominen una metodología muy cercana a la que es empleada internacionalmente en el diseño digital, vinculando los procesos de modelación, descripción, simulación, implementación y análisis de resultados que forman parte de la habilidad compleja “diseñar” (Barrios, 2005)

Por lo que se formula el siguiente problema científico:

Problema Científico:

¿Cómo lograr que los estudiantes de Telecomunicaciones y Electrónica adquieran conocimientos y habilidades en el diseño electrónico digital con dispositivos digitales de alta escala de integración que estén a tono con las metodologías mundiales actuales en este campo?

La problemática fundamental del presente Trabajo de Diploma se concentra entonces en el siguiente objetivo:

Objetivo General:

Desarrollo de un módulo de ejemplos prácticos integradores para consolidar las habilidades de diseño en la asignatura Diseño Digital VLSI

Objetivos Específicos:

- Describir las tendencias tecnológicas internacionales relacionadas con el diseño digital VLSI y UVLSI.
- Describir las habilidades generales básicas e invariantes de contenido relacionadas con el DDVLSI.
- Analizar el Programa Analítico de la asignatura Diseño Digital VLSI.
- Proponer un conjunto de ejemplos prácticos que integren el sistema de contenido y habilidades de la asignatura.
- Comprobar los resultados en el kit de desarrollo Nexys 2.

Para satisfacer los objetivos planteados en la tarea se ha decidido dividir el trabajo en Introducción, tres capítulos, y conclusiones y recomendaciones.

CAPITULO I: Caracterización del diseño electrónico digital moderno.

CAPITULO II: Análisis de la asignatura Diseño Digital VLSI y propuesta de prácticas de laboratorio integradoras.

CAPITULO III: Diseño, simulación e implementación de los ejemplos integradores.

En las conclusiones y las recomendaciones se pretende hacer una valoración del alcance de este trabajo y su posible expansión hacia futuras investigaciones.

CAPÍTULO 1. CARACTERIZACIÓN DEL DISEÑO ELECTRÓNICO DIGITAL MODERNO.

En el presente capítulo se realiza un análisis de la evolución histórica del diseño electrónico digital partiendo del concepto de electrónica hasta llegar al objeto de estudio de la Electrónica Digital. Se definen las características básicas de los lenguajes de descripción de hardware y de los dispositivos lógicos programables. Además, se analizan los componentes y tendencias actuales del diseño electrónico digital.

1.1 Objeto de la Electrónica. La Electrónica Digital.

La Electrónica ha transitado por importantes cambios desde sus inicios como una rama de la Física. De hecho, la denominación de "Electrónica" en la actualidad no resulta esclarecedora, al no expresar ni el objetivo ni el objeto de esta materia; mas bien alude a los medios que utiliza: dispositivos basados en el comportamiento del electrón.

Una definición esclarecedora de Electrónica, en cuanto su objetivo, campo de acción y medios, es la siguiente: "La Electrónica es una técnica de manejo de la información, codificada en señales eléctricas, utilizando dispositivos que aprovechan las propiedades de los electrones" (Pollán, 2003). De aquí el carácter de tecnología horizontal de la Electrónica (Mandado and Valdés, 2004) que contribuye al desarrollo de otras tecnologías y, fundamentalmente a la de la Informática y las Comunicaciones.

A partir de la definición anterior es que la Electrónica puede dividirse, atendiendo a su objeto de estudio, en dos grandes campos: la Electrónica Analógica y la Electrónica Digital. La primera se dedica al manejo de la información, codificada como señales eléctricas continuas en el tiempo. La Electrónica Digital tiene como objetivo el diseño y análisis de sistemas procesadores de información, codificada como señales eléctricas no continuas en

el tiempo, de dos posibles valores solamente: alto y bajo (1 ó 0) y que tiene sus fundamentos teóricos en el Álgebra de George Boole (Barrios, 2005).

Los dispositivos básicos con los que la Electrónica Digital realiza sus funciones son las compuertas lógicas (AND, OR, NOT) que, con el desarrollo alcanzado por la tecnología de semiconductores, permiten integrar millones de estas en un solo circuito integrado o *chip*, capaz de procesar de muy diversas maneras la información digitalizada. Tanto para la Electrónica Analógica como para la Electrónica Digital una de sus actividades profesionales fundamentales es el diseño de sistemas procesadores de la información (Barrios, 2005).

La elevación de la complejidad de los sistemas creados por el ser humano, producto del desarrollo tecnológico en diferentes áreas, hizo necesario desarrollar métodos de diseño que permitiesen garantizar el correcto funcionamiento del sistema una vez implementado, así como la eficiencia y la repetitividad de dicho proceso. Dichos métodos han tenido además que ir cambiando, conforme se ha elevado la complejidad de los sistemas.

1.2 Evolución Histórica del diseño electrónico digital

El diseño electrónico digital se puede definir, en términos generales como el proceso planificado y organizado mediante el cual se implementa un sistema electrónico digital concreto, con vista a solucionar un problema práctico. El diseño es, además de un proceso tecnológico, el modo de actuación de los especialistas de la electrónica, es por tanto una habilidad profesional compleja.

La necesidad de construir circuitos digitales cada vez más complejos es patente día a día. Ya en el siglo XXI se tiene la capacidad de construir microprocesadores de muy altas prestaciones que están compuestos por millones de unidades funcionales (transistores) que realizan tareas de gran responsabilidad en la sociedad.

En la práctica, el 100% de la electrónica de control y supervisión de los sistemas, elaboración de datos y transferencia de los mismos se realiza mediante circuitos integrados digitales, constituidos por una gran cantidad de transistores: son los llamados circuitos integrados de muy alta escala de integración, o VLSI (Mandado et al., 2002).

Si en los años cincuenta y sesenta, en los albores de la electrónica integrada los circuitos eran esencialmente analógicos, en los que el número de elementos constituyentes de los circuitos no pasaba de la centena, en la actualidad el hombre dispone de tecnologías de integración capaces de producir circuitos integrados con millones de transistores a un coste no muy elevado, al alcance de una PYME. A mediados de los años sesenta Gordon E. Moore ya vaticinaba un desarrollo de la tecnología planar en el que cada año la escala de integración se doblaría, y de la misma manera aumentaría la capacidad de integrar funciones más complejas y la velocidad de procesamiento de esas funciones. Las predicciones de Moore se han cumplido con gran exactitud durante los siguientes 30 años, y la tendencia continuará durante los próximos 20 (Zemva, 1998).

En la Electrónica Digital moderna, caracterizada por su alto desarrollo tecnológico, coexisten un conjunto de conceptos (invariantes de contenido) interrelacionados que presentan además diversas alternativas de representación y realización. Tanto la representación (modelación) como la realización de un problema, solucionable dentro del campo de la Electrónica Digital, se combinan en el proceso llamado diseño.

El incremento de la complejidad de los sistemas creados por el ser humano, producto del desarrollo tecnológico en diferentes áreas, hizo necesario desarrollar métodos de diseño que garantizaran el correcto funcionamiento del sistema una vez implementado, así como la eficiencia y la repetitividad de dicho proceso. Dichos métodos han tenido, además, que ir cambiando conforme se ha elevado la complejidad de los mismos.

Han sido numerosos los autores que han propuestos modelos del proceso de diseño, aplicables a cualquier tecnología, que permitan el desarrollo de un método para llegar a su realización. Entre ellos se destaca el modelo propuesto por la asociación de ingenieros alemanes a través de la norma VDI 2221 (ver figura 1.1) (Mandado, 2000).

En la misma se concibe la estructura del diseño como un sistema jerárquico en el que cada componente constituye un módulo separado. La utilización de esta estrategia en el diseño de sistemas digitales complejos tiene la ventaja de que los módulos pueden ser realizados para diseñar diferentes sistemas (reutilización), con la consiguiente reducción del costo de diseño.

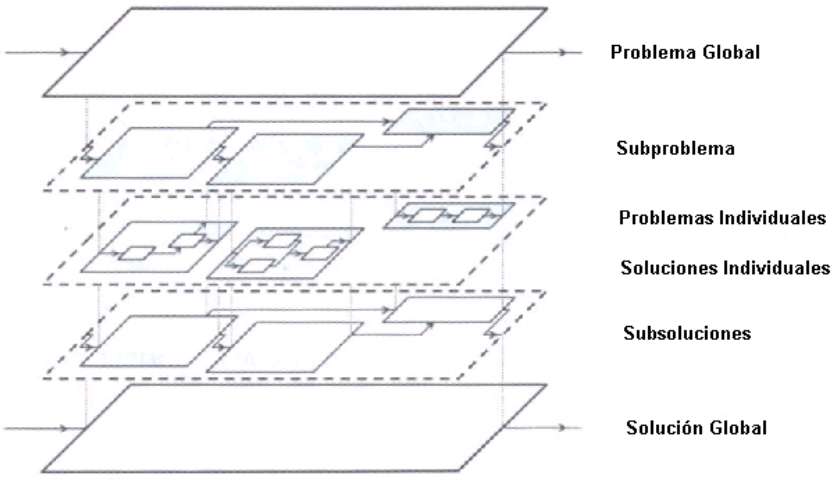


Figura 1.1. Descripción gráfica del modelo de diseño VDI 2221.

Los sistemas digitales complejos se definen como aquellos compuestos por un número de componentes elementales (compuertas) superior a cien y se caracterizan porque en su diseño no es posible utilizar los métodos manuales "clásicos" (ecuaciones del Álgebra de Boole, relaciones de entrada salida en forma de tabla de verdad, entre otras).

Lo anterior permite caracterizar al diseño digital como resultado (implementación física) y como proceso o sucesión de fases, con un método o estrategia propia.

A manera de resumen histórico, la tabla 1.1 muestra la relación entre el contenido del diseño de los sistemas digitales y los métodos y medios para solucionar el mismo:

Tabla 1.1. Resumen del desarrollo histórico del diseño electrónico digital.

Período	**Problemas a Diseñar (Contenido)**	**Métodos y Medios (Procesos)**
1970	Memorias, Microprocesadores de 8 y 16 bits.	- Metodología de diseño ascendente (bottom-up). - Herramientas de ayuda al diseño muy rudimentarias. - Tecnología bipolar y NMOS. - Diseño centrado en las fábricas específicas de semiconductores.

1980	Diseño de Circuitos Integrados de Aplicación Específica (ASICs), circuitos integrados de alta densidad programables (FPGAs y CPLD), microprocesadores de 32 bits y procesadores digitales de señales.	- Se separan los procesos de diseño y fabricación. - Metodologías de diseño ascendentes (bottom-up) y descendentes (top-down). - Surgimiento del lenguaje de descripción de hardware VHDL. - Tecnologías CMOS y BiCMOS. - Desarrollo de herramientas computacionales de ayuda al diseño.
1990	Predominio de los FPGAs y CPLDs sobre los ASICs, desarrollo de aplicaciones en un solo chip (System on a Chip), Sistemas Complejos de Co-diseño Hardware-Software. Diseño de procesadores Pentium, Interfaces para redes y Multimedia.	- Desarrollo profundo de la tecnología CMOS. - Herramientas computacionales de ayuda al diseño muy poderosas. - Predominio de la metodología de diseño top-down. - Desarrollo y actualización del VHDL. - Abundantes bibliotecas de módulos complejos reutilizables. - Desarrollo de Co-Diseño Hardware-Software (Co-Design).

2000	Predominio de los System on a Chip (SoC), Diseños completos en un chip (mixtos), procesadores con propiedad intelectual.	- Consolidación del Co-Design y la convergencia Hardware-Software, Analógico-Digital, Diseño-Encapsulado. - Fortalecimiento de la metodología de diseño descendente (top-down). - Poderosas herramientas de ayuda al diseño por computadora (CAD-EDA)*. - Lenguajes de descripción de hardware con síntesis automatizada. - Fuerte independencia entre el diseño y la tecnología. - Trabajo en equipo con amplio uso de las TIC. - Tecnología submicrónica.

* CAD: Computer Aided Design (Diseño Asistido por Computadora)

EDA: Electronic Design Automation tools (Herramientas de Diseño Electrónico Automatizadas).

Las características más destacadas de las tendencias actuales en diseño de sistemas electrónicos se pueden resumir en un incremento creciente de la complejidad de los sistemas a desarrollar, ciclos de vida y de diseño cada vez más reducidos, así como la necesidad de incluir un cierto grado de flexibilidad que permita afrontar posibles modificaciones o actualizaciones futuras" (Moreno, 2005).

La consolidación en la década de los noventa de los lenguajes de descripción de hardware (HDLs, *Hardware Description Languages*) ha supuesto, por otro lado, la implantación progresiva de la denominada metodología de diseño '*Top-Down*' (descendente) que, en contraposición a la metodología ascendente (Bottom-Up), permiten la descripción del

sistema al más alto nivel. En el diseño top-down la dependencia de la implementación final es prácticamente inexistente en las etapas de definición funcional, y se irá concretando en las sucesivas fases de diseño, hasta llegar a la síntesis de nuestro sistema sobre cualquiera de las tecnologías existentes (fulI-custom, ASICs, FPGAs, CPLDs,...). Hoy en día los HDLs están ampliamente difundidos y estandarizados bajo la IEEE, por lo que su aprendizaje, más que aconsejable, es una necesidad en la pequeña y gran industria. (Al-Hadithi et al., 2004)

Además, los lenguajes de descripción de hardware al formar parte de las herramientas EDA permiten el trabajo en equipo. Así, al estructurar el desarrollo del proyecto, cada integrante del equipo de diseño puede trabajar en subproyectos antes de integrar todas las partes del sistema.

Dicha tendencia se ha generalizado dentro del diseño digital moderno (Sagahyroon and Assim, 2000). El trabajo por niveles de jerarquía, la descripción mediante lenguajes de hardware de alto nivel y el uso de herramientas de software en prácticamente en todas las etapas del proceso de diseño, se integran en entornos informáticos favorables al intercambio entre diseñadores y la reutilización de diseños. En el diseño digital tiene una gran importancia la comunicación y la documentación.

Actualmente las firmas con mayor impacto en el campo de los sistemas digitales (Altera, Xilinx, Actel, Lattice, Cypress, Intel, entre otras) incluyen dentro de sus herramientas de ayuda al diseño electrónico de una u otra forma esta estrategia general, diferenciándose en las interfaces hombre-máquina, en herramientas específicas y/o en las formas de entrada de las descripciones para cada una de las fases de diseño.

A todo lo anterior se añade la tendencia al uso de herramientas de modelación y simulación matemático-lógicas, como Matlab-Simulink y de lenguajes de programación de alto nivel, como C, que permiten una mayor nivel de abstracción en la descripción de los diseños y dejan la fase de implementación o traducción a un lenguaje de hardware a un proceso automatizado que realizan los propios sistemas informáticos.

1.3 Definiciones básicas acerca de los lenguajes de descripción de hardware.

Los lenguajes de descripción de hardware (HDL) son lenguajes de programación, establecidos como normas internacionales, que permiten describir en un ordenador el funcionamiento de un circuito digital de cualquier nivel de complejidad.

Lo que diferencia los HDL de los lenguajes de programación para computadoras (C, Pascal, Basic, Fortran, etc.) es que mientras que estos se ejecutan de forma secuencial (línea a línea), los HDL, por simular el funcionamiento de un sistema digital como un todo se ejecutan simultánea o concurrentemente.

El surgimiento de los HDL fue el resultado de un esfuerzo internacional por tener un lenguaje común de intercambio de información de los diseños digitales que permitiera el desarrollo de proyectos en equipo. Dentro de los más difundidos en la actualidad se encuentran el VHDL y el Verilog, ambos normas internacionales de la IEEE (Instituto de Ingenieros Eléctricos y Electrónicos). El VHDL ó IEEE-1076/87 tiene una mayor difusión en el contexto europeo, mientras que el uso del Verilog se encuentra mayormente en Norteamérica y Japón (Pollán, 2003)

Las siglas del **VHDL** proceden del inglés: **V**ery high speed integrated circuit **H**ardware **D**escription **L**anguage que significan lenguaje de descripción para hardware de circuitos intregados de alta velocidad. Desde su surgimiento en 1987 el mismo ha venido perfeccionándose y ampliándose su utilidad que inicialmente era para la descripción funcional (sin interesar la implementación física del modelo) y la documentación de los proyectos.

VHDL fue desarrollado como un lenguaje para el modelado de sistemas digitales. Proporciona una sintaxis amplia y flexible que permite el modelado estructural, en flujo de datos y de comportamiento de hardware. VHDL al ser un estándar de la IEEE, favoreció su adopción en la industria lo que se ve reflejado en las constantes mejoras en las herramientas. Debido a su estandarización, un código en VHDL puede ser portado a diferentes herramientas y también, puede ser reutilizado en diferentes diseños (Gutiérrez, 2004).

En la actualidad cada ingeniero diseñador en la electrónica debe conocer y usar un lenguaje de descripción de hardware. Usando VHDL se pueden escribir y sintetizar rápidamente

circuitos de 10 o 20 mil compuertas. Diseños equivalentes descritos a través de esquemas o ecuaciones lógicas en el ámbito de transferencia de registros podrían requerir algunos meses de trabajo por una persona. Además VHDL proporciona las siguientes capacidades:

Flexibilidad: VHDL es un poderoso lenguaje mediante el cual se pueden describir circuitos complejos con múltiples niveles de descripción del diseño. Está soportado por bibliotecas y la creación de componentes que pueden ser reutilizados. Esto proporciona la posibilidad del diseño jerárquico a través del diseño modular. VHDL es un lenguaje para la síntesis y la simulación.

Diseño independiente del dispositivo: VHDL permite implementar el diseño sin tener que seleccionar primero el dispositivo donde será implementado. Luego se puede realizar una simulación funcional del diseño que incluye solamente el comportamiento lógico del circuito.

Factibilidad: Este permite que se pueda simular la misma descripción del diseño que se ha sintetizado. Simular una descripción del diseño de un circuito de varios miles de compuertas antes de la síntesis en un dispositivo determinado puede ahorrar un tiempo considerable. El flujo de diseño puede ser rectificado en esta etapa antes de su implementación.

Bajo costo y corto tiempo de llevar el diseño al mercado: El diseño digital con VHDL para la síntesis usado lógica programable aumenta la velocidad del proceso de diseño. VHDL permite le descripción del proceso rápidamente.

Actualmente permite, a partir de esta descripción funcional o de comportamiento (tal y como si se estuviese describiendo en lenguaje natural), la síntesis o compilación automática sobre un circuito integrado digital, liberando a los diseñadores de esta tediosa tarea.

1.4 Dispositivos Lógicos Programables.

Los dispositivos lógicos programables PLD, del inglés (Programmable Logical Device), son circuitos integrados digitales que no tienen una función predefinida por el fabricante. Su función puede ser definida o programada por el usuario. Permiten reemplazar grandes diseños digitales que antes se implementaban con componentes discretos como compuertas

y flip-flops por un solo integrado. Debido a la gran capacidad lógica que tienen los dispositivos modernos, sistemas completos pueden desarrollarse sobre un solo circuito integrado. (López and Ayala, 2004)

Los dispositivos actuales (CPLD y FPGAs) tienen una capacidad lógica de hasta millones de compuertas, incluyen interfaces programables para varios estándares de interface eléctrica y tienen bloques de funciones especiales embebidos entre la lógica programable tales como memoria, multiplicadores o CPUs completas.

Tradicionalmente los dispositivos lógicos programables son personalizados fuera del sistema, o cargan su personalización durante la inicialización. Se les puede cambiar su programación a medida que se va refinando el diseño de un prototipo. Esto da a los diseños con FPGAs suficiente flexibilidad para adaptarse a cambios en los requerimientos en etapas avanzadas del diseño, lo que los hace muy atractivos para realizar implementaciones tempranas de estándares no totalmente definidos (Compton and Hauck, 2002).

Un enfoque más reciente en la utilización de las FPGAs aprovecha el hecho de que estos dispositivos permiten "infinitas" reprogramaciones en forma dinámica y que el tiempo que lleva la reconfiguración de los chips es muy pequeño. De esta manera se puede redefinir el hardware en tiempo real. Este nuevo enfoque ha dado en llamarse "Lógica Reconfigurable" o "configurable". Las mismas celdas lógicas pueden ser en determinado momento un contador, luego una ALU o un filtro digital. Cada compuerta puede realizar una función durante un tiempo y cuando esa función ya no es necesaria ser utilizada para realizar otra(Compton and Hauck, 2002).

1.5 Componentes actuales del diseño electrónico digital.

Una caracterización de gran vigencia (Gajski, 1988) con relación a las formas de representación de un diseño se muestra en la Figura 1.2.

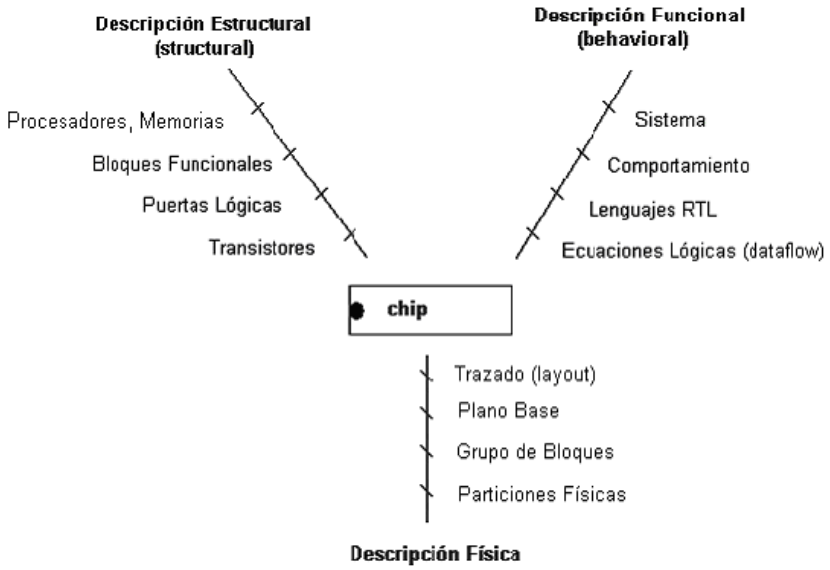


Figura 1.2 Formas de descripción de los sistemas digitales complejos (Gajski-Kuhn).

En la actualidad, producto de la separación entre los procesos de diseño e implementación del chip, los modos de descripción estructural y de comportamiento están orientados y se encuentran disponibles en las herramientas EDA ó CAD para los profesionales dedicados al diseño, mientras que los relacionados con la realización física del chip están disponibles en las herramientas propias de los fabricantes de circuitos integrados y poseen una alta complejidad y costo.

Las exigencias actuales del mercado, han impulsado el desarrollo de herramientas de diseño electrónico (herramientas EDA, *Electronic Design Automation en inglés*) con facilidades para la transformación de una idea inicial a una realización de forma rápida, eficiente, fiable y, con un bajo costo. Tales exigencias han revolucionado la forma clásica de desarrollo de sistemas, y en especial los digitales. Lo que habitualmente era un ciclo de diseño basado en prototipos que se iban mejorando hasta conseguir un producto final, está dando paso a concentrar los esfuerzos en el nivel conceptual o de modelación y la verificación previa de la funcionalidad del sistema sin detallar excesivamente en sus partes (Zemva, 1998).

La mayoría de los diseños electrónicos digitales, a partir del 2004, se caracterizan en su fase inicial por descripciones en un lenguaje de alto nivel (HLL, *High Level Language*) (Gartner, 2000).

Los análisis anteriores permiten concluir que, visto como un sistema, los pilares en que se basa el diseño electrónico digital moderno son: Los fundamentos teóricos, la tecnología electrónica y las tecnologías de la información y la comunicación (Figura 1.4).

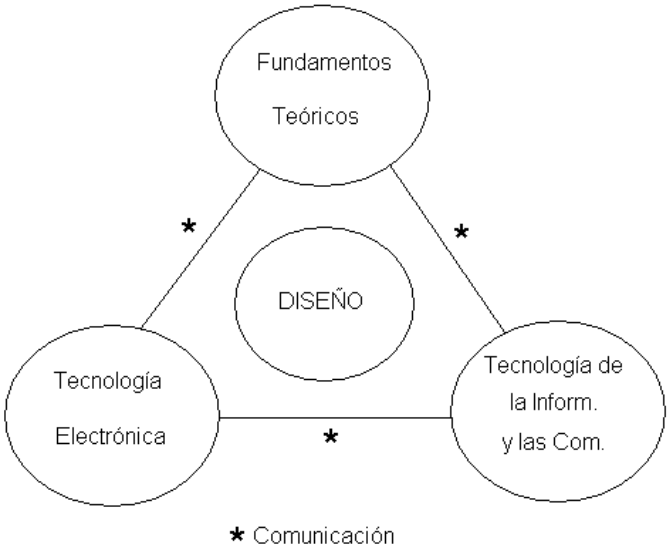


Figura 1.4 Componentes generales del diseño digital.

La tecnología electrónica abarca el desarrollo alcanzado en la fabricación de circuitos integrados (CMOS, AsGa, etc.), las herramientas de diseño asistido por computadora (EDA, CAD-CASE) que facilitan el mismo y lo ponen a tono con el desarrollo de las nuevas estructuras de estos chips (ASICs, FPGAs, CPLDs), las interfaces para la programación y verificación, así como las tarjetas de desarrollo (kits). En resumen, todos los medios técnicos que permiten la realización física del diseño.

El componente Fundamentos Teóricos, está constituido por las invariantes de contenido y las metodologías de trabajo que caracterizan el diseño digital en la actualidad. En cuanto a invariantes de contenido se pueden señalar:

1- Fundamentos del Álgebra de Boole.
2- Los sistemas de representación binaria de datos.
3- Circuitos combinacionales típicos.
4- Circuitos secuenciales típicos.
5- Fundamentos del procesamiento paralelo de datos.
6- Fundamentos del procesamiento secuencial de datos.
7- Subconjunto funcionalmente completo de un lenguaje de descripción de hardware.

En cuanto a las metodologías de trabajo:

1- La estrategia de diseño por niveles de jerarquía descendente (top-down), utilizando herramientas EDA.
2- El uso, cada vez más intensivo de herramientas de modelación y simulación matemáticas que incorporan módulos predefinidos de funciones digitales complejas y la generación automática de la implementación física.

Las TIC son el soporte donde concurren y se realizan los otros dos componentes. Entre las TIC y la tecnología electrónica se produce una sinergia, razón del actual desarrollo vertiginoso y simultáneo de la Computación, la Electrónica y las Comunicaciones.

CAPÍTULO 2. ANÁLISIS DE LA ASIGNATURA DISEÑO DIGITAL VLSI Y GENERALIDADES DE LAS PRÁCTICAS DE LABORATORIO INTEGRADORAS

En el presente capítulo se abordan los aspectos relacionados con el Programa Analítico de la asignatura optativa Diseño Digital VLSI (DDVLSI) de la carrera de Telecomunicaciones y Electrónica, destacándose la estructura técnica de la habilidad diseñar y la metodología general para desarrollar la misma.

A partir de los análisis anteriores se describen las características que deben tener las prácticas de laboratorio integradoras para esta asignatura con el objetivo de lograr un mayor nivel de motivación de los estudiantes y de la integración de los conocimientos del año mediante ejemplos atractivos.

2.1 Programa Analítico de la asignatura Diseño Digital VLSI.

La asignatura Diseño Digital VLSI (DDVLSI) se imparte como asignatura optativa en el tercer año de la carrera de Telecomunicaciones y Electrónica, del Plan de Estudios D. Un resumen (modelo P1) de los aspectos esenciales del contenido y las formas de actividades docentes se muestra en el Anexo 4.

La asignatura DDVLSI se enmarca dentro de la disciplina Electrónica y requiere como precedentes a las asignaturas ED I y ED II, que proveen las bases teórico-prácticas para la modelación, descripción, simulación e implementación de sistemas digitales típicos de mediana complejidad.

El Objetivo General de DDVLSI es que los estudiantes dominen las metodologías, estrategias y herramientas modernas del diseño digital con dispositivos de alta escala de integración que le permitan diseñar ejemplos complejos típicos de los sistemas electrónicos.

La misma se imparte en el segundo semestre del tercer año de Telecomunicaciones y Electrónica, en el cual los estudiantes han concluido prácticamente todo el sistema de conocimientos y habilidades relacionados con la disciplina Electrónica y comienzan a relacionarse con los fundamentos de las comunicaciones electrónicas.

Por el propio objetivo general de la asignatura se concluye que la misma tiene un alto nivel de integración con las ED I y ED II, y sirve para complementar todo el sistema de conocimientos y habilidades de éstas.

A lo anterior se adiciona que el Plan de Estudios D prevé, dentro de la Disciplina Integradora, la ejecución de un Proyecto de Curso que abarque y consolide los objetivos generales del año, fundamentalmente relacionados con la Electrónica.

De todo lo anterior se puede proponer que la asignatura DDVLSI pueda contribuir, mediante el desarrollo de ejemplos integradores, al cumplimiento de los objetivos de la asignatura Telecomunicaciones y Electrónica III (de la Disciplina Integradora).

En el momento en que se desarrollan estas asignaturas los estudiantes han adquirido un grupo de conocimientos relacionados con bloques típicos de la electrónica analógica y digital que forman parte de múltiples sistemas y aplicaciones de diversas esferas de la producción y los servicios, a saber:

1. Amplificadores.
2. Osciladores.
3. Fuentes de voltaje.
4. Convertidores AD y DA.
5. Unidades de procesamiento aritmético-lógico.
6. Registros (serie y paralelo) para la trasmisión de datos en banda base.
7. Memorias ROM, RAM, LIFO, FIFO.

8. Máquinas de Estado y de Estado Algorítmico.
9. Fundamentos del hardware de los microprocesadores.

Por tanto, cualquier propuesta de integración como la antes señalada, puede abarcar uno o varios de estos subsistemas típicos que de modo atractivo para los estudiantes, los motive a consolidar la metodología y las estrategias de diseño de la asignatura DDVLSI.

Para poder enfrentar esta actividad compleja (diseñar) es que se plantea en el Programa Analítico de DDVLSI un conjunto de actividades destinadas al dominio de los conceptos y contenidos de los circuitos digitales de alta escala de integración (CPLDs y FPGAs), la metodología de diseño digital actual, las estrategias de descripción de sistemas digitales y la explotación de herramientas de ayuda al diseño.

Es por ello que se pasan a resumir los aspectos fundamentales antes señalados y que pueden servir como referencia inicial para los estudiantes en esta asignatura.

2.2 Metodología General de Diseño Digital

Al desarrollar cualquier sistema digital es importante seguir ciertas pautas de trabajo y tener en cuenta factores muy diversos para que el diseño pueda terminarse a tiempo y funcione correctamente. A medida que los diseños se hacen más complejos, la necesidad de seguir un método ordenado para lograr buenos resultados se hace más importante. Los lenguajes de descripción de hardware, si se utilizan correctamente, pueden utilizarse en varios pasos del proceso de desarrollo, no solo para diseñar sino también para especificar y documentar el sistema que se quiere desarrollar.

Las grandes capacidades de los PLDs y herramientas de diseño disponibles permiten que los diseños implementados sobre FPGAs sean cada vez más complejos. En muchos casos varias personas pueden estar trabajando sobre el mismo producto, incluso en localidades separadas. Para poder atacar el problema del diseño de sistemas digitales complejos (ya sea para desarrollos sobre FPGAs, ASICs etc) es importante tener una metodología de trabajo que permita planificar y ordenar el trabajo.

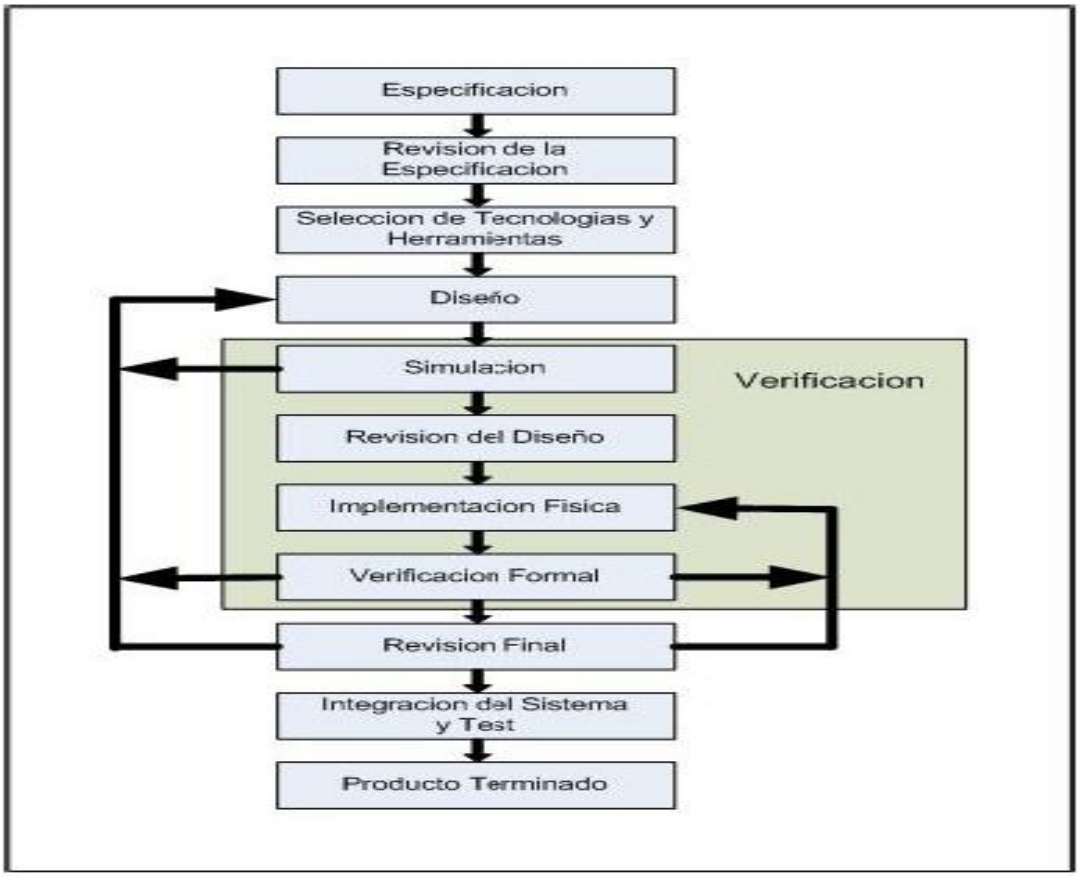


Figura 2.1 Metodología General de Diseño Digital.

Como ejemplo, en la figura 2.1 se muestra un esquema de un método que permite ordenar el trabajo de diseño. Este permite:

- Diseñar un dispositivo libre de defectos de manufactura, que funciona de manera adecuada y se integra con el sistema.
- Diseñar el dispositivo de manera eficiente, sin malgastar recursos ni tiempo.
- Planificar el diseño de manera eficiente, crear un cronograma razonable y asignar los recursos necesarios para las diferentes tareas de manera ordenada.

El ciclo comienza con un conjunto de requerimientos para la fabricación de un dispositivo o sistema. Estos requerimientos pueden venir de un cliente, de otro grupo de trabajo dentro de la misma empresa o del mismo grupo de trabajo que necesita desarrollar una parte de un sistema más grande.

Especificación y Diseño: Una especificación permite que todas las personas involucradas en un proyecto comprendan cual es el dispositivo que se va a desarrollar.

Las especificaciones deberían describir la solución y la manera de cumplir con los requerimientos que se piden para el dispositivo. Al haber una especificación formal, las bases sobre las que trabajar en un diseño quedan preestablecidas y se minimizan los errores por diferencias de apreciación o entendimiento entre los participantes.

Una especificación debería comprender los siguientes puntos:

- Diagrama en bloques del sistema externo, que muestra como y donde encaja el dispositivo dentro del sistema completo.
- Diagrama en bloques interno que muestra los principales bloques funcionales.
- Descripción de las entradas/salidas, incluyendo interfaces lógicas, eléctricas y protocolos de comunicación.
- Estimaciones de tiempos que se deben cumplir, incluyendo tiempos de establecimiento y mantenimiento para las entradas/salidas y frecuencias de reloj.

- Estimación de la complejidad y/o magnitud del dispositivo, dado en número de compuertas equivalentes o número de circuitos integrados necesarios.
- Especificación física del dispositivo. Tamaño, empaquetamiento, conectores, etc.
- Estimación del consumo de potencia del dispositivo.
- Precio estimado del dispositivo.
- Procedimientos de verificación y validación para el dispositivo.

Después de escribir las especificaciones es importante hacer una revisión con todos los miembros del equipo. De esta revisión podrán surgir cosas que no se tuvieron en cuenta individualmente y que produzcan modificaciones. La especificación también incluye la metodología de verificación del dispositivo. Estas muchas veces se dejan para el final del proyecto y no se definen ni llevan a cabo de manera adecuada.

Una vez que se escribe la especificación se puede utilizar para seleccionar componentes y tecnologías que se utilizarán para el proyecto. El diseño deberá hacerse siguiendo métodos

aceptados y confiables. El proceso de diseño es en general un ciclo, e incluye varios pasos intermedios.

Verificación: la verificación engloba varios pasos menores, y al revisar pueden surgir cosas que obligan a volver atrás hacia pasos anteriores. Dependiendo del dispositivo y tecnología utilizada, pero en general sigue los siguientes pasos:

- Simulación: es en general un proceso continuo, ya que al simular se pueden encontrar problemas que hacen volver sobre el diseño y hacer cambios. Las simulaciones se hacen sobre pequeñas partes del sistema y sobre el sistema completo. Se debe llevar a cabo una simulación funcional, pero también puede incluir simulaciones de temporizado, consumo de potencia y otros parámetros.
- Revisión: En este paso se revisan los resultados de la simulación y se analiza el comportamiento del dispositivo. Es importante que participen ingenieros externos al proyecto y personas que conozcan el sistema en su totalidad.
- Implementación Física: Una vez que se ha aceptado el diseño se lleva a cabo la implementación física final del dispositivo.
- Verificación Formal: En este paso se verifica la implementación física para asegurarse que su funcionamiento coincide con las simulaciones hechas anteriormente. En este paso se deben también evaluar los tiempos, consumo de potencia y cualquier otro parámetro de importancia.

Pasos Finales: si todos los pasos se siguieron correctamente la revisión final debería ser solo una formalidad. Se verifica que el dispositivo está listo para ser entregado o integrado al sistema. La integración y verificación en el contexto del sistema general es muy importante. Si los pasos se siguieron correctamente, cualquier modificación que surja de esta integración será en general pequeña y no requerirá grandes cambios. Cualquier problema o falla que se encuentre debe ser documentarse y analizada para poder corregirla en una próxima versión del dispositivo y evitarla en el futuro.

2.2.1 Flujo de diseño

Cuando se diseña con lógicas programables, cualquiera sea el método usado para diseñar el circuito (HDLs, esquemáticos, etc.), el proceso desde la definición del circuito por el desarrollador hasta tenerlo funcionando sobre un FPGA implica varios pasos intermedios y en general utiliza una variedad de herramientas. A este proceso se lo denomina ciclo o flujo de diseño. Para cada uno de estos pasos se utilizan herramientas de software diferentes que pueden o no estar integradas bajo un ambiente de desarrollo.

A continuación se describe cada uno de los pasos del ciclo de diseño. Se agrega su denominación en inglés entre paréntesis, ya que estos son los términos que se encontrarán en las herramientas de desarrollo.

Descripción del Diseño: este es el paso en el que se describe el diseño, muchas veces usando un lenguaje de descripción de hardware como el VHDL. Muchas herramientas permiten ingresar el diseño no solo como HDLs sino también como un diagrama esquemático o estructural, una representación gráfica de una máquina de estados o una tabla de entrada-salida. Estas herramientas simplifican en gran medida el diseño y simplifican mucho la tarea del diseñador. El código HDL puede ingresarse utilizando cualquier editor de texto, pero se recomienda uno que tenga coloreado automático de sintaxis ya que ayuda y hace más fácil esta etapa (Güichal, 2005a).

Generación o Traducción (Generate, Translate): Este paso tiene sentido cuando el diseño se hace mediante algunas de los métodos mencionados anteriormente en vez de en VHDL. En este paso se traducen todos los módulos a VHDL. Para los módulos ingresados como VHDL, las herramientas generalmente hacen una copia a una librería interna. En esta etapa se hace un análisis del VHDL para verificar la sintaxis y semántica de los módulos. También se hace una elaboración de los archivos, que consiste en replicar los componentes que se utilizan más de una vez en el diseño para hacer copias únicas y definir un conexionado adecuado(Güichal, 2005a).

Compilado: Los simuladores actuales compilan el código VHDL a un formato que permite una simulación más rápida y eficaz. Esto se hace en este paso.

Simulación y verificación: En este paso se simula el comportamiento del diseño y se evalúa su comportamiento. La simulación puede hacerse en tres etapas diferentes del diseño. La primera es sobre el código VHDL original para verificar el correcto funcionamiento del diseño. La segunda es después de sintetizar el circuito, y se simulan la implementación real sobre el FPGA, ya sea con o sin la anotación de tiempos. La tercera etapa en la cual se puede simular el diseño es después de la ubicación e interconexión. Esta es la más exacta y la más engorrosa y lenta, ya que incluye la información final lógica y temporal del diseño sobre el PLD (Güichal, 2005a).

Síntesis: En este paso se traduce el VHDL original a su implementación con lógica digital, utilizando los componentes específicos del FPGA que va a utilizarse. Esta traducción puede llegar hasta el nivel más básico de elementos lógicos (CLBs, LUTs, etc.) o hasta un nivel superior, en el que el diseño se presenta en módulos básicos estándar provistos en una librería por el proveedor del dispositivo (Güichal, 2005a).

Ubicación e Interconexión (Place and Route): El FPGA está compuesto por muchos bloques idénticos, como se presentó en las secciones anteriores. En este paso, cada componente del diseño sintetizado se ubica dentro del FPGA específico. También se interconectan los componentes entre sí y con los pines de entrada-salida. (Güichal, 2005a)

Tareas Adicionales: Las tareas adicionales dependen del fabricante y las herramientas. Puede definirse la interconexión de las señales con los pines físicos del FPGA ingresar condiciones de entorno físico para guiar a la herramienta de Ubicación e Interconexión, seleccionar áreas del FPGA para ubicar los bloques lógicos, etc(Güichal, 2005a).

Anotación de Retardos: Como en todo circuito digital, las señales tendrán un retardo de propagación que influirá sobre su comportamiento. Con estos retardos puede anotarse el diseño compilado para una simulación que incluya información de temporizado mas cercana a la implementación real. Una vez sintetizado el diseño, puede hacerse una estimación de los retardos de propagación que habrá entre las señales. Después de la ubicación e interconexión, el cálculo de retardos es mucho más exacto (Güichal, 2005a).

Generación de Binarios: Después de la ubicación e interconexión se genera algún archivo ya sea para poder utilizar el sistema en un diseño más complejo o para programar un FPGA.

Configuración de PLD: Con el archivo binario generado puede configurarse directamente un FPGA a través de alguna de las opciones de configuración. Estas opciones dependerán de las herramientas y del dispositivo que se esté utilizando.

Programación de Memoria (PROM): Muchas FPGA no pueden configurarse de manera permanente y requieren algún tipo de memoria para leer la configuración cuando se les aplica tensión de alimentación. En la etapa de producción deben configurarse memorias PROM y conectarlas al FPGA en el circuito impreso.

Verificación (automática): Una vez integrado el FPGA con su programación al sistema debe hacerse una verificación para controlar que el diseño sobre el FPGA funciona bien (las FPGAs pueden tener fallas internas) y que el FPGA se integra bien al sistema en el que está. Pueden usarse técnicas y herramientas de verificación automáticas para evaluar si el dispositivo y el diseño están funcionando como debieran.

2.2.2 Flujo de diseño con ISE XILINX

Al usar la herramienta ISE, la cual integra en un mismo ambiente de desarrollo todas las etapas de flujo de diseño digital, hay que seguir una serie de pasos para llevar a cabo un diseño hasta la implantación en el dispositivo programable. Estos pasos son los siguientes:

- Descripción
- Simulación funcional
- Síntesis
- Implementación
- Simulación temporal
- Programación

En la Figura 2.2 se puede ver un diagrama en bloque de dichos pasos de diseño y otras opciones que se pueden ejecutar.

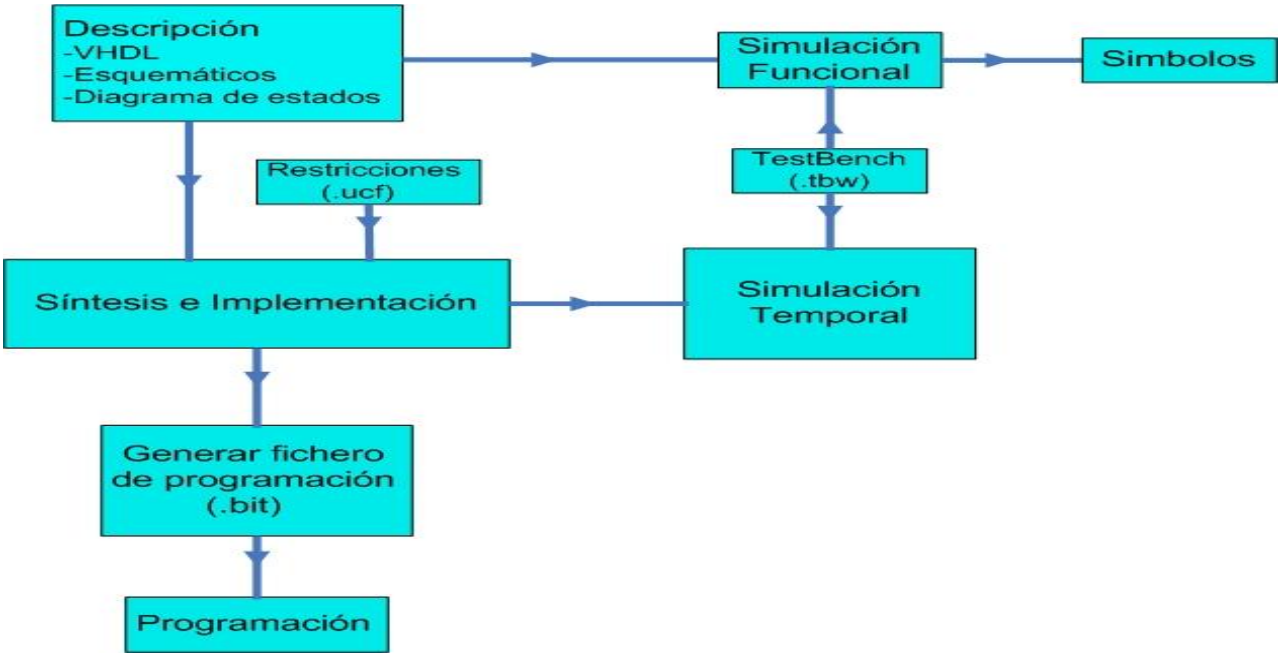


Figura 2.2 Flujo de diseño con ISE Xilinx.

2.3 Herramientas de software

La integración de las herramientas de diseño electrónico en entornos de desarrollo basados en modelos ha impulsado el auge de nuevas técnicas de diseño para dispositivos reconfigurables, facilitando la implementación de sistemas de procesamiento modulares y autónomos. Xilinx es pionera en el desarrollo de herramientas de este tipo, que facilitan el diseño de sistemas de procesamiento digital en una amplia variedad de FPGAs producidos por esta misma compañía (XILINX, 2010).

Xilinx ISE

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 implementación sobre una arquitectura reconfigurable. Cada una de las etapas del flujo de diseño puede realizarse dentro del entorno integrado (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).

Es posible llamar de manera automática y desde su propio entorno de trabajo a las diferentes utilidades y herramientas suministradas dependiendo de la etapa de diseño en la que se encuentre el proyecto, lo cual hace el trabajo más sencillo. El entorno posee un aspecto similar al de los entornos de programación actuales como Visual Basic o Visual C, con diversas ventanas para visualizar tareas específicas sobre cada una de ellas (Arias, 2010).

Matlab/Simulink

Matlab es un programa de cálculo numérico y visualización de datos para la resolución de problemas complejos planteados en la realización y aplicación de modelos matemáticos de ingeniería. Este software posee una extraordinaria versatilidad y capacidad para resolver problemas de matemática aplicada, física, química, ingeniería, finanzas y otras aplicaciones. Su base la constituye el cálculo matricial e integra análisis numérico, procesamiento de señales y visualización gráfica. Se utiliza en escuelas y centros universitarios, así como en departamentos de investigación y desarrollo de muchas compañías.

Matlab dispone de herramientas adicionales que expanden sus prestaciones como Simulink, un software para modelar, simular y analizar sistemas dinámicos. Simulink proporciona una interfaz de usuario gráfica para construir los modelos como diagramas de bloques, utilizando operaciones con el ratón del tipo pulsar y arrastrar. Después de definir un modelo, se puede simular e interactuar con dicha simulación a través de los diferentes menús que ofrece el programa. Además, es posible cambiar los parámetros y ver de forma inmediata lo que sucede tras el cambio; pudiendo transferir los resultados de la simulación al espacio de trabajo del programa Matlab, para su posterior procesamiento y visualización (MathWorks, 2009).

Simulink es un entorno de programación visual, que permite construir y simular modelos de sistemas físicos y de control mediante diagramas de bloques. El comportamiento de dichos sistemas se define mediante funciones de transferencia, operaciones matemáticas, elementos de Matlab y señales predefinidas (MathWorks, 2009).

Simulink proporciona una interface gráfica de usuario GUI (*Graphical User Interface*) para realizar modelos como diagramas en bloques, permitiendo dibujarlos de la misma forma

que se realizarían con lápiz y papel. Además incluye una librería de bloques de fuentes, componentes lineales y no lineales, conectores; aunque también permite crear bloques definidos por los usuarios. El GUI simplifica el proceso de modelado, eliminando la necesidad de realizar complicadas tareas como por ejemplo la elaboración de ecuaciones diferenciales en lenguaje de programación (MathWorks, 2009).

System Generator

System Generator es una herramienta de diseño de alto nivel desarrollada por Xilinx que permite el diseño basado en modelos en el entorno Matlab/Simulink para el desarrollo de sistemas de procesamiento digital en FPGAs. Se integra a Simulink como un toolbox al instalarse en la computadora Xilinx ISE.

Como todos los *blocksets* de Simulink, System Generator se integra como una biblioteca de bloques que pueden ser conectados para crear modelos funcionales de un sistema dinámico. De esta forma, permite modelar, simular y analizar sistemas de procesamiento complejos y de alto rendimiento para una plataforma hardware específica, mediante un entorno flexible, robusto y fácil de utilizar. System Generator no puede considerarse como un sustituto de los lenguajes de descripción de hardware hasta la fecha, pero sí un potente complemento (XILINX, 2010).

Los diseños desarrollados con System Generator pueden componerse de una gran variedad de elementos: bloques específicos de System Generator, código de un lenguaje de descripción de hardware tradicional como VHDL o Verilog y funciones derivadas del lenguaje de programación M de Matlab. Así, todos estos elementos pueden ser usados simultáneamente, simulados en conjunto y sintetizados para obtener todo un sistema de procesamiento de señales sobre un FPGA (XILINX, 2010).

Digilent Adept

Digilent Adept es una potente aplicación que permite la transferencia de datos de una configuración a los dispositivos lógicos de Xilinx.

Este software se utiliza para descargar el fichero de programación en el FPGA, es totalmente compatible con la tarjeta. A pesar de que el software Xilinx ISE posee la opción para descargar la configuración mediante la herramienta iMPACT no se cuenta con el cable necesario para realizar esta tarea. El software Digilent Adept brinda al usuario una interfaz sencilla para la programación del kit Nexys2 mediante el puerto USB.

Para trabajar con el software, una vez instalado se puede acceder en computadoras con sistema operativo Windows desde el menú Inicio. Después de abierto, el software detecta automáticamente la tarjeta si está conectada a un puerto USB de la computadora. En este punto se selecciona dónde se va a cargar la configuración, si directamente en el FPGA y/o en la memoria ROM de la tarjeta con el objetivo de que dicha configuración no se pierda.

En la ventana del software se especifica la ruta del fichero de programación y se selecciona la opción Initialize Chain. Una vez terminado el proceso de descarga se puede verificar el diseño en el kit de desarrollo.

2.4 Las Prácticas de Laboratorio

Las clases se clasifican sobre la base de los objetivos que se deben alcanzar y sus tipos principales son: la conferencia, la clase práctica, el seminario, la clase encuentro, **la práctica de laboratorio** y el taller.

En cada modalidad de estudio, el profesor debe utilizar adecuadamente las posibilidades que brinda cada tipo de clase para contribuir al logro de los objetivos educativos formulados en el programa analítico de la asignatura y del año académico en que se desarrolla.

2.4.1 Caracterización e importancia de las prácticas de laboratorio

La práctica de laboratorio se define como el tipo de clase que tiene como objetivos que los estudiantes adquieran las habilidades propias de los métodos y técnicas de trabajo y de la investigación científica; amplíen, profundicen, consoliden, generalicen y comprueben los fundamentos teóricos de la disciplina mediante la experimentación, empleando para ello los medios necesarios.

Este tipo de forma de enseñanza garantiza un aprendizaje significativo, pues da la oportunidad a los estudiantes de enfrentar en la práctica la solución de problemas científicos y el análisis de fenómenos aplicando los presupuestos teóricos, métodos y procedimientos de la disciplina así como los medios y herramientas disponibles.

El proceso de realización de las prácticas de laboratorio constituye parte integrante del trabajo independiente de los estudiantes, el cual está constituido por tres etapas (Iglesias, 2008).

a. Preparación previa a la práctica.
b. Ejecución de la práctica.
c. Conclusiones de la práctica

La preparación previa se desarrolla fundamentalmente sobre la base del estudio teórico orientado por el profesor como fundamento de la práctica, así como el estudio de las técnicas y procedimientos de los experimentos correspondientes. El desarrollo o ejecución de la práctica se caracteriza por el trabajo de los estudiantes con el material de laboratorio (utensilios, instrumentos, aparatos, y reactivos), la aplicación de las técnicas y métodos propios de cada actividad, el análisis de los fenómenos deseados, el reconocimiento de los índices característicos de su desarrollo, la anotación de las observaciones, entre otras tareas docentes.

Durante las conclusiones el estudiante deberá analizar los datos de la observación y arribar a las conclusiones y generalizaciones que se derivan de la práctica en cuestión.

En las prácticas de laboratorio predominan la observación y la experimentación, lo que exige la utilización de métodos y procedimientos específicos para el trabajo.

En relación con esto, son significativas la observación, explicación, comparación, elaboración de informes, entre otras.

La preparación de las prácticas de laboratorio exige del profesor una atención especial a los aspectos organizativos, ya que su realización se basa fundamentalmente, en la actividad individual o colectiva de los alumnos, de manera independiente.

Al igual que en otros tipos de clases, es necesario durante su preparación tener en cuenta respetar las etapas del proceso de enseñanza-aprendizaje (Labarrere, 1996):

- Motivación.
- Orientación.
- Ejecución.
- Evaluación.

Durante el proceso organizativo de la práctica se deben determinar con precisión las características de la actividad de los estudiantes y las habilidades que se van a desarrollar, garantizar las condiciones materiales que exige el cumplimiento de los objetivos propuestos y diseñar la estructura metodológica de la práctica de laboratorio.

Resulta necesario además, determinar una secuencia de pasos que faciliten la dirección, por el profesor, de la realización de la práctica de laboratorio, entre los que se encuentran las siguientes:

- Orientación de los objetivos y las tareas fundamentales a desarrollar y las técnicas operatorias básicas que se utilizarán.
- Distribución de materiales.
- Trabajo independiente de los estudiantes.
- Discusión colectiva de los resultados obtenidos.

2.4.2 Tipos de prácticas de laboratorio en las Ciencias Tecnológicas

Los laboratorios se clasifican en función de dos criterios: (1) La forma de acceder a los recursos (local o remota) para propósitos de experimentación y (2) la naturaleza del sistema físico (real o virtual), con lo que los entornos de experimentación quedarían clasificados en (Dorrmido, 2004):

- Locales y reales: Laboratorios presenciales con plantas reales.
- Locales y virtuales: Laboratorios presenciales con plantas simuladas.
- Remotos y reales: Teleoperación de una planta real.
- Remoto y virtual: Laboratorios remotos con plantas simuladas

En la contextualización de esta clasificación a la Electrónica Digital, las prácticas de la asignatura son básicamente locales-virtuales o locales-reales, en las que el computador, además de ser un medio de enseñanza, es un instrumento de trabajo para la compilación, comunicación, programación y verificación de las diferentes placas de circuitos de VLSI, entiéndase “kits” de desarrollo. De ahí que los ejemplos propuestos en este trabajo se desarrollarían en prácticas de laboratorio de naturaleza virtual y presencial. De manera que a continuación se alude a sus características e importancia en el proceso de enseñanza-aprendizaje (Dorrmido, 2004).

Los laboratorios virtuales se pueden utilizar como alternativa a los laboratorios presenciales y remotos. En este caso se usan los ordenadores para simular el comportamiento de los sistemas a estudiar haciendo uso de modelos matemáticos. Aunque en este caso no se interacciona con plantas reales, la experimentación con modelos simulados es comparable

siempre que se cumplan las siguientes premisas: (1) Se usen modelos matemáticos realistas que representen al alumno los detalles importantes del sistema a analizar y (2) se complementen las gráficas que muestran la evolución temporal de los sistemas con animaciones que permitan a los alumnos visualizar y entender mejor el comportamiento del sistema (Dorrmido, 2004).

Dentro de las ventajas reconocidas a los laboratorios virtuales se encuentran las siguientes(Dorrmido, 2004):

1) Dado que un laboratorio virtual se basa en modelos matemáticos que se ejecutan en ordenadores, su configuración y puesta a punto es mucho más sencilla que la configuración y puesta a punto de los laboratorios reales.

2) Presentan un grado de robustez y seguridad mucho más elevado ya que al no haber dispositivos reales éstos no pueden causar problemas en el entorno.

2.5 Propuesta de estrategia de las Prácticas de Laboratorio como facilitadora del desarrollo de ejercicios integradores.

Teniendo en cuenta que los ejercicios integradores de cualquier temática relacionada con la electrónica y las comunicaciones está compuesto por uno o varios subsistemas típicos de estas disciplinas, y que los mismos han sido abordados en su mayoría por los estudiantes del tercer año de la carrera, es que se propone que la planificación y ejecución de las prácticas de laboratorio de DDVLSI sean consecuentes con la metodología de diseño top-down y bottom-up (Al-Hadithi et al., 2004).

La propuesta es que la selección y diseño de los ejemplos integradores se desarrolle utilizando el diseño de bloques jerárquicos y que, finalmente, integrando los mismos se puedan configurar dichos ejemplos.

De esta manera en cada práctica se abordaría un subsistema en particular, preparando para ello previamente en las conferencias, clases prácticas y seminarios la fundamentación teórica y modelación de los mismos.

La estrategia consistiría entonces en concebir el ejemplo integrador, modelarlo, y diseñarlo siguiendo la metodología top-down, simularlo y configurar la FPGA, comprobando su

funcionamiento. Tarea que debe ser ejecutada por el profesor como preparación previa de la asignatura.

Luego, para el desarrollo de las actividades docentes, la propuesta, en particular la de los laboratorios, sería utilizar la metodología bottom-up, desarrollando en cada uno de ellos diferentes subsistemas. Es aprender haciendo.

Todo lo anterior exige que la selección de los ejemplos, además de ser abarcadores de subsistemas fundamentales de la electrónica y motivantes, no sean tan disímiles en sus bloques funcionales. Todo ello con el objetivo de no provocar frustración cognitiva y recarga de información en los estudiantes (Barrios, 2005).

Por todo lo anterior se considera proponer los siguientes ejemplos integradores:

- Controlador VGA.
- Generador de caracteres.
- Juego de Ping Pong.
- Ascensor de n pisos.

2.6 Análisis teórico de los ejemplos desarrollados

A continuación se presentan las principales consideraciones teóricas de las aplicaciones confeccionadas en este trabajo.

2.6.1 Controlador VGA

VGA (*Video Graphics Array*) es un estándar para mostrar video introducido a finales de la década de 1980 en las computadoras IBM. El VGA original soportaba una resolución de 640x480 con 16 colores en modo gráfico. El modo VGA sigue siendo el estándar utilizado en el arranque de los ordenadores, hasta que se hacen cargo del control de los gráficos los controladores de la tarjeta de video instalada. Se trata del último estándar de video impuesto por IBM, y a partir de él se empezaron a desarrollar modelos que cada vez ofrecían más calidad y prestaciones, movidos en gran medida por el auge de los juegos de ordenador (Chu, 2008).

Un circuito controlador VGA como se observa en la Figura 2.3 debe generar las señales de temporización vertical y horizontal y coordinar el envío de los datos de video basado en el reloj de píxel. El reloj de píxel define el tiempo disponible para representar la información de un píxel en la pantalla. Los datos de video vienen típicamente de una memoria de video, con uno o más octetos asignados a cada localización del píxel. El controlador debe poner un índice en la memoria cuando los haces se mueven a través de la pantalla, recuperar y aplicar estos datos a la misma en el tiempo exacto que el haz electrónico se está moviendo a través de un píxel dado (Chu, 2008).

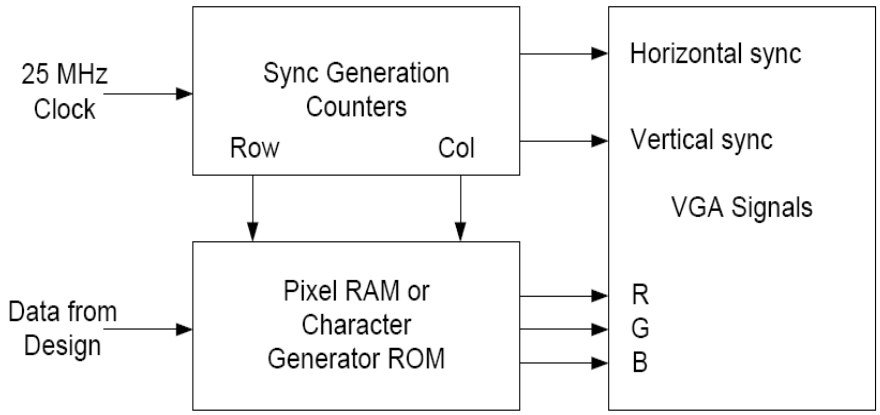


Figura 2.3 Diagrama en bloques de un controlador VGA.

Un monitor VGA común puede ser controlado utilizando solo cinco líneas: dos de sincronismo y tres señales analógicas correspondientes a los colores rojo, verde y azul (RGB). Las señales analógicas correspondientes a cada color permiten variar la intensidad de los mismos simplemente variando su voltaje. Combinando correctamente el valor de tensión aplicado a las tres líneas analógicas se puede obtener el color deseado.

El circuito de sincronización de video genera la señal de sincronismo horizontal que especifica el tiempo que demora escanear una línea horizontal, y la señal de sincronismo vertical que especifica la frecuencia de refrescamiento del monitor. Para una resolución de 640x480 con un reloj de píxel de 25MHz y una frecuencia de refrescamiento de 60Hz, se derivan las señales de sincronismo mostradas en las Figuras 2.4 y 2.5 para el circuito horizontal y vertical respectivamente.

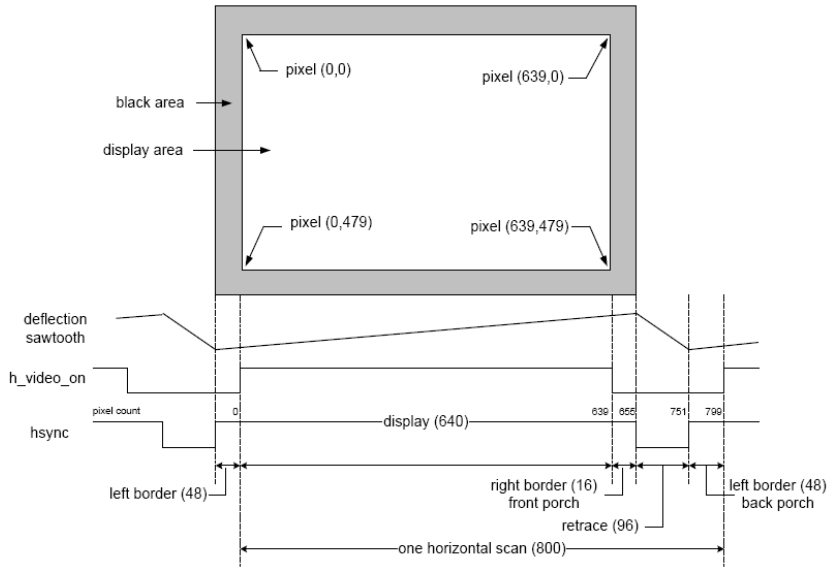


Figura 2.4 Diagrama de tiempo para una línea horizontal.

Un circuito decodifica la salida de un contador de sincronismo horizontal controlado por el reloj del píxel, para generar la señal de sincronismo *hsync* (Figura 2.4). Asimismo, la salida de un contador de sincronismo vertical que incrementa con cada pulso de *hsync* se utiliza para generar la señal de sincronismo *vsync* (Figura 2.5). Ambos contadores pueden ser usados para localizar un píxel en cualquier posición de la pantalla y formar una dirección en la memoria RAM de vídeo cuya información se corresponda con el color del píxel. De esta forma se logra disminuir notablemente la complejidad del circuito (Chu, 2008).

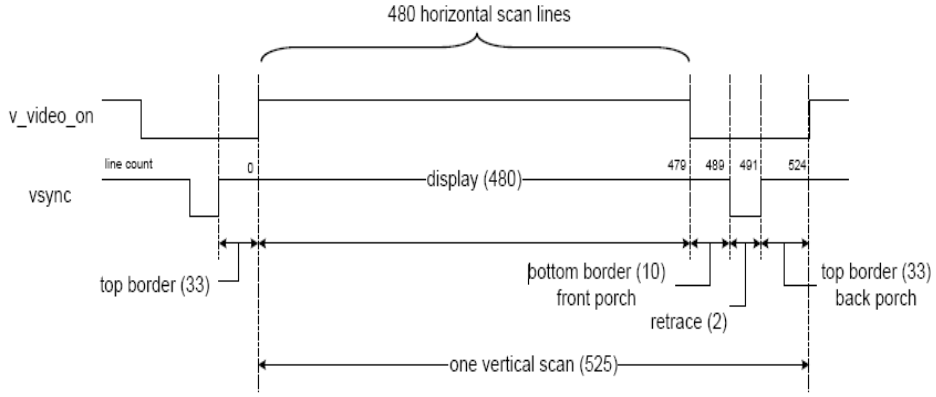


Figura 2.5 Diagrama de tiempo para la exploración vertical .

2.6.2 Generador de caracteres

Para la mejor comprensión de este diseño es necesario tener en cuenta los elementos explicados anteriormente en el ejemplo del controlador VGA, pues son la base del funcionamiento de un circuito generador de caracteres.

Un circuito de generación de píxel genera la señal de *rgb* de 3 bits para el puerto VGA, las señales de control externo y de datos especifican el contenido de la pantalla, y las señales *pixel-x* y *pixel-y* del circuito vgasync proveen las coordenadas actuales del píxel. Este circuito se divide en tres amplias categorías (Chu, 2008):

- Esquema *Bit-mapped*
- Esquema *Tile-mapped*
- Esquema *Object-mapped*

El display de texto discutido en este ejemplo se basa en el esquema *Tile-mapped.* En este esquema, se agrupa una colección de bits para formar un cuadro y tratar a cada cuadro como una unidad de display. Por ejemplo se puede definir una cuadratura de píxeles de 8*8 como un cuadro.

Al aplicar un esquema *tile-mapped*, se trata a cada caracter como un cuadro. El valor de un cuadro representa el código de un patrón específico. Para el display de texto, se usa el código ASCII de 7 bits para los cuadros de carácter.

Los patrones de caracteres se almacenan en una ROM y cada patrón requiere 2^4 * 8 bits. . La memoria patrón se conoce como fuente de la ROM. El conjunto de fuente original consta de 256 patrones, incluyendo dígitos, letras mayúsculas y minúsculas, símbolos de puntuación, y muchos símbolos gráficos de propósito especial. Para acomodar dicho set es necesario, 2^7 * 2^4 * 8 bits de ROM. Es usualmente configurado como una ROM de 2^11*8 bits (Chu, 2008).

Cuando se realiza la impresión en la ROM, el set de impresión se implementa con los 128 caracteres del código de ASCII, q se muestra en el Anexo II. En esta ROM, los 7 bits más significativos (MSBs) de los 11 bits de la dirección se usan para identificar el caracter, y los cuatro bits menos significativos (LSBs) de la dirección se usan para identificar la fila dentro de un patrón de carácter (Chu, 2008).

El circuito de generación del pixel genera sus valores según las coordenadas actuales del pixel (dadas por las señales *pixel-x* y *pixel-y*) y las señales de control externa y de datos. La generación de pixel se basa en un esquema de mapeado de cuadros que involucra dos etapas. La primera etapa usa los bits superiores de las señales *pixel-x* y *pixel-y* para generar el código de un cuadro, y la segunda etapa usa este código y los bits inferiores para generar el valor del pixel (Chu, 2008).

El circuito de generación de texto sigue este método, y el diagrama básico es mostrado en la Figura 3.6. En la primera etapa, las señales *pixel-x* (9 downto 3) y *píxel-y* (8 downto 4) proveen las coordenadas ***x*** y ***y*** de la posición actual del cuadro. El circuito de generación de caracteres usa estas coordenadas, combinadas con otros datos externos, para generar el valor de este cuadro (char-addr), que corresponda al caracter del código de ASCII.

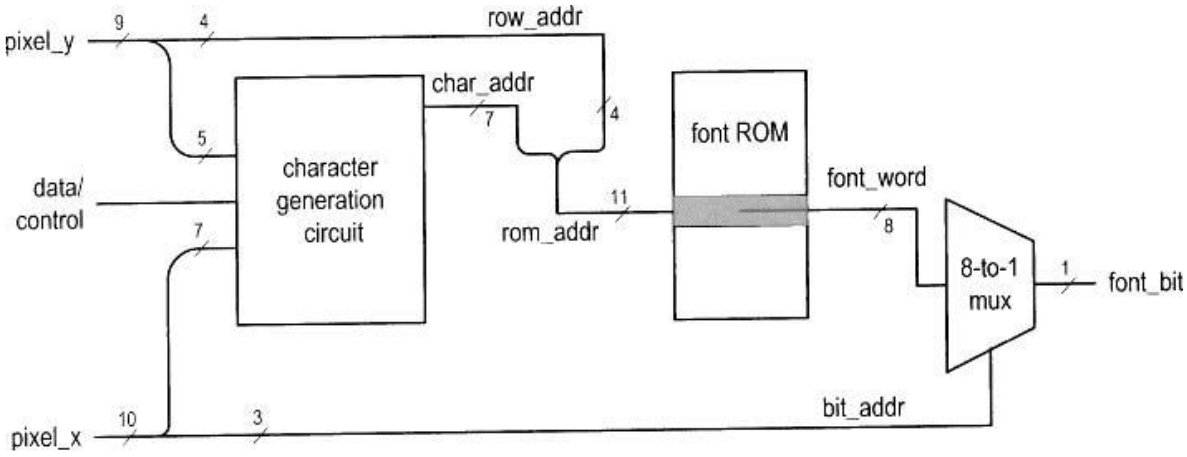


Figura 3.6 Diagrama básico del circuito de generación de texto.

En la segunda etapa, los siete MSBs del código ASCII se convierten en la dirección del set de impresión de la ROM y especifican la posición del patrón actual. Esto se concatena con los cuatro LSBs de la coordenada ***y*** de la pantalla para formar la dirección completa del caracter. La salida del caracter de la ROM corresponde a una fila de 8 bits en el patrón. Los tres LSBs de la coordenada ***x*** de la pantalla especifican la posición deseada del pixel, y un multiplexor de 8 a 1, la ruta del pixel para la salida.

2.6.3 Juego Ping Pong

En el diseño de este juego se utiliza un circuito gráfico de movimiento libre y se diseña una FSM de control de nivel superior que integra los subsistemas gráficos y coordina el funcionamiento del circuito global.

El diseño se construye de la siguiente forma:

1. Crear una pantalla sencilla con objetos rectangulares.
2. Dar a conocer la animación.
3. Crear un circuito de control de nivel superior.

El diagrama conceptual de un circuito de generación de pixel asignado a objetos contiene tres objetos que se muestran en la Figura 3.7. El esquema consta de tres circuitos de generación de objetos y un circuito de selección y encaminamiento especial, etiquetado **rgb mux**. Un circuito de generación de objetos realiza las siguientes tareas:

- Mantiene las coordenadas del objeto en cuestión y lo compara con la exploración actual proporcionada por las señales *pixel-x* y *píxel-y*.
- Si la ubicación de la exploración actual cae dentro de la región, se activa la señal *obj-i-on* para indicar que la localización de la exploración actual está dentro de la región del objeto i-ésimo y este debe ser "activado".
- Se especifica el color deseado en la señal *obj-i-rgb*.

El circuito **rgb mux** realiza la multiplexación de acuerdo con un esquema de priorización interno. Examina varias señales *obj-i-on* y determina que señal *obj-i-rgb* debe ser encaminada a la salida **rgb**. El esquema de priorización prioriza el orden de las pantallas cuando múltiples señales *obj-i-on* se confirman al mismo tiempo. Corresponde la selección de un objeto para el primer plano.

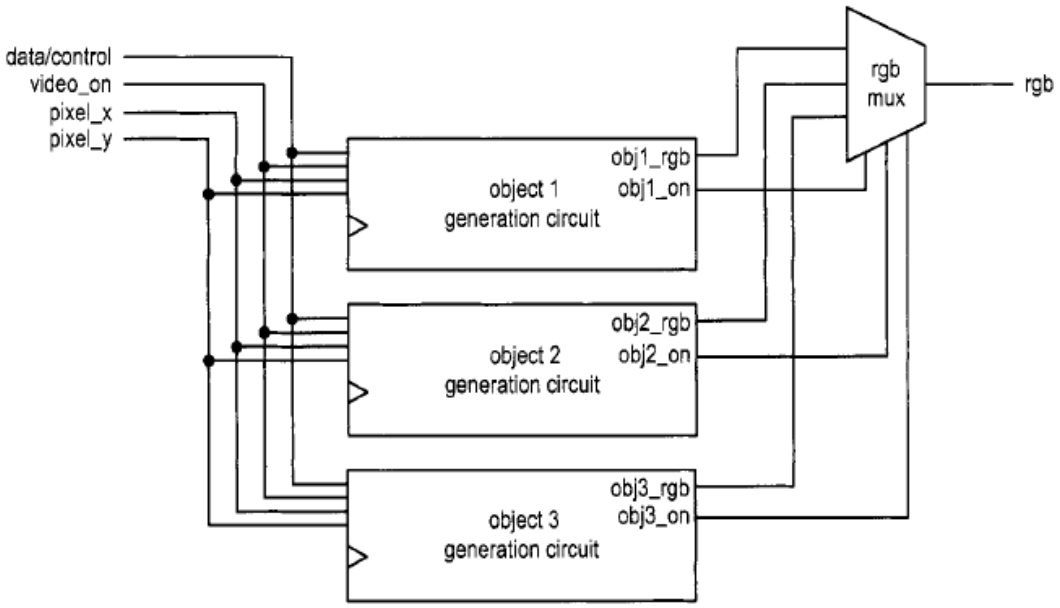


Figura 3.7 Diagrama conceptual de un circuito de generación de pixel orientado a objeto.

Cuando un objeto cambia su ubicación gradualmente en cada exploración, crea la ilusión de movimiento y llega a ser animada. Para lograr esto, se pueden utilizar registros para almacenar los límites de un objeto y actualizar su valor en cada exploración. En el juego de Pong, la paleta es controlada por dos botones y puede moverse hacia arriba y hacia abajo, y la pelota puede moverse y saltar en todas direcciones.

Para que la paleta pueda adaptarse a las cambiantes coordenadas del eje y, se reemplazan las constantes con dos señales, *bar-y-t* y *bar-y-b*, para representar los límites superior e inferior, y se crea un registro, **bar-y-reg**, para almacenar la ubicación actual de los ejes del límite superior. Si se pulsa uno de los botones, **bar-y-reg** aumenta o disminuye una cantidad fija. La cantidad se define por una constante, BAR-V, que representa la velocidad de la barra. (Chu, 2008).

El diseño de la pelota es más complicado. Hay que sustituir las cuatro constantes de frontera con cuatro señales y crear dos registros, **ball-x-reg** y la **ball-y-reg**, para almacenar los cambios de las coordenadas **x** y **y**. La bola se mueve generalmente a una velocidad constante. Puede cambiar de dirección cuando golpea la paleta, o la parte inferior o superior de la pantalla. Se descompone la velocidad en una componente **x** y una componente **y**, cuyos valores pueden ser una constante de valor positiva valor, Ball-vp, o una constante

de valor negativo, Ball-vn. Los valores de los dos componentes se almacenan en los registros **x-delta-reg** y **y-delta-reg** (Chu, 2008).

2.6.4 Asencor de n pisos

Este ejemplo se dedica a la elaboración de una metodología que permita generalizar el algoritmo de hardware para el control digital de un elevador de cualquier cantidad de pisos y se obtiene como resultado el circuito digital que cumple con estas funciones. Con este ejemplo se realiza una integración de los contenidos de las asignaturas Electrónica Digital I y II con la utilización de Tablas de Verdad, Máquinas de Estado Algorítmico (ASM), Diagramas de Estado y el lenguaje de descripción de hardware (VHDL), elementos los cuales se complementan entre sí para lograr el correcto funcionamiento del diseño implementado.

Un ascensor o elevador es un sistema de transporte vertical diseñado para movilizar personas o bienes entre diferentes alturas. Se conforma con partes mecánicas, eléctricas y electrónicas que funcionan conjuntamente para lograr un medio seguro de movilidad. Puede ser utilizado, ya sea para ascender o descender en un edificio, o una construcción subterránea.

Elementos constitutivos de un ascensor(Jiménez. et al., 2012):

- *Cabina:* es el elemento portante del sistema de ascensores. Está formada por dos partes: el bastidor o chasis y la caja o cabina. Cuenta además con las puertas que se abren o cierran según el caso.

- *Grupo tractor:* está formado normalmente por un motor acoplado a un reductor de velocidad, en cuyo eje de salida va montada la polea acanalada que arrastra los cables por adherencia. Es el encargado de poner en movimiento el ascensor.

- *Maniobras de control:* el control de los sistemas de ascensores funciona mediante sistemas electrónicos, encargados de hacer funcionar la dirección de movimiento de la cabina y seleccionar los pisos en los que esta deba detenerse. Actualmente, los controles de ascensores funcionan con microprocesadores electrónicos que mediante algoritmos de inteligencia artificial determinan la forma de administrar la respuesta a los pedidos de llamadas coordinando los

distintos equipos para trabajar en conjunto. (En el caso nuestro solo se utiliza lógica cableada VLSI). Las maniobras de control pueden ser internas y externas a la cabina.

- *Sensores:* detectan la posición actual del ascensor. De esta forma, la información que proveen es mostrada mediante los circuitos de visualización y su información es utilizada también en otros procesos de control (Moreno, 1999).

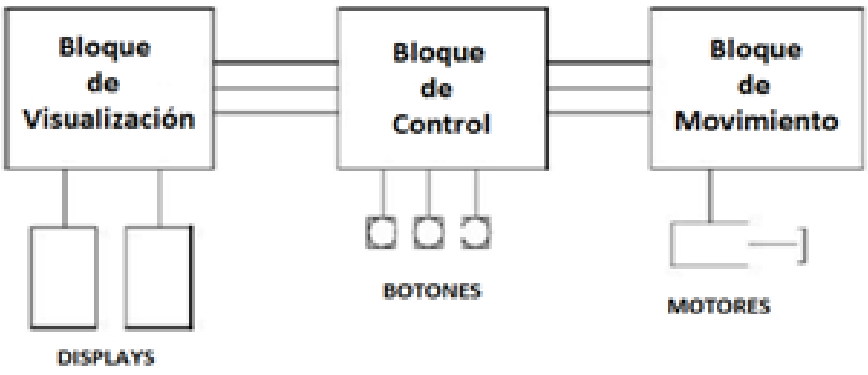


Figura 3.9 Estructura general de un elevador.

La Figura 3.9 muestra la estructura general de un elevador. El bloque de visualización de la información es en esencia un bloque de lógica combinacional que se ocupa de transformar ciertas señales provenientes de los sensores en datos útiles para el usuario siendo visualizados en los displays internos y externos a la cabina . Por otra parte, se encuentra el bloque de movimiento que está compuesto por el Grupo tractor, y el bloque de Control que es la parte central del trabajo y que dada la activación de alguna o algunas de las entradas del circuito, tomará una decisión sobre qué movimiento debe tener la parte mecánica . La respuesta del sistema será un conjunto de señales digitales que alimentarán el bloque de movimiento dándole la orden de cómo actuar (Güichal, 2005b).

CAPÍTULO 3. DISEÑO, SIMULACIÓN E IMPLEMENTACIÓN DE LAS PRÁCTICAS INTEGRADORAS.

En este capítulo se desarrollan una serie de diseños digitales con el objetivo de validar la metodología propuesta. Es importante decir que dichos ejemplos abarcarán tanto lógica combinacional como secuencial, y la descripción se realiza tanto con esquemáticos como con modelos VHDL. En algunos ejemplos se centrará la atención en el trabajo con las herramientas de software, ISE Xilinx, System Generetor, MatLab Simulink y el trabajo con el kit. También se representará un caso donde se integraran los conocimientos de las asignaturas Electrónica Digital I y II (Tablas de verdad, ASM, diagramas de estado, etc...). Para una mejor comprensión de los mismos en el Capítulo 2, se realiza una introducción donde aparecen aspectos como el principio de funcionamiento y la utilidad práctica. El grado de dificultad varía de uno a otro, de manera que se va profundizando en el diseño. En todos los ejemplos se realiza el proceso de asignación de pines con el objetivo de verificar su funcionamiento mediante el kit de desarrollo y el FPGA que este posee además de una simulación temporal luego de los procesos de síntesis e implementación para verificar su correcto funcionamiento.

3.1 Confección de un controlador VGA.

El esquema diseñado en System Generator se muestra en la figura 3.1. Para comprobar el funcionamiento se utilizó la interfaz disponible en el kit Nexys2 la cual utiliza 10 señales para crear un puerto VGA con ocho bits de color y dos señales estándares de sincronismo.

Las etapas de sincronismo se implementaron utilizando dos contadores, uno que contara hasta 800 utilizando el reloj de píxel para generar el sincronismo horizontal y otro que

contara 525 líneas para el barrido vertical. Un circuito decodificador es el encargado de generar las señales de sincronismo y de habilitación de video.

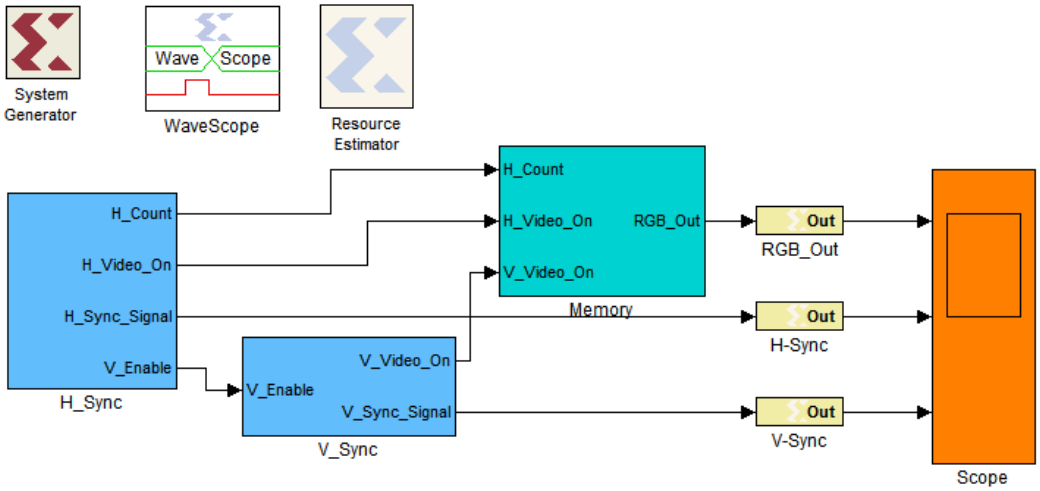


Figura 3.1 Controlador VGA desarrollado con System Generator.

Para verificar el funcionamiento del controlador se generó un patrón de barras verticales que muestra ocho colores en orden decreciente de luminancia desde el blanco (máxima luminancia) hasta el negro (mínima luminancia) como se puede observar en la figura 3.5. El patrón de pruebas es uno de los elementos más utilizados por los ingenieros en el control de los parámetros que determinan la calidad de la imagen reproducida. El uso de estos patrones permite realizar una inspección visual inmediata de la calidad de la imagen reproducida sobre la pantalla del monitor de imagen.

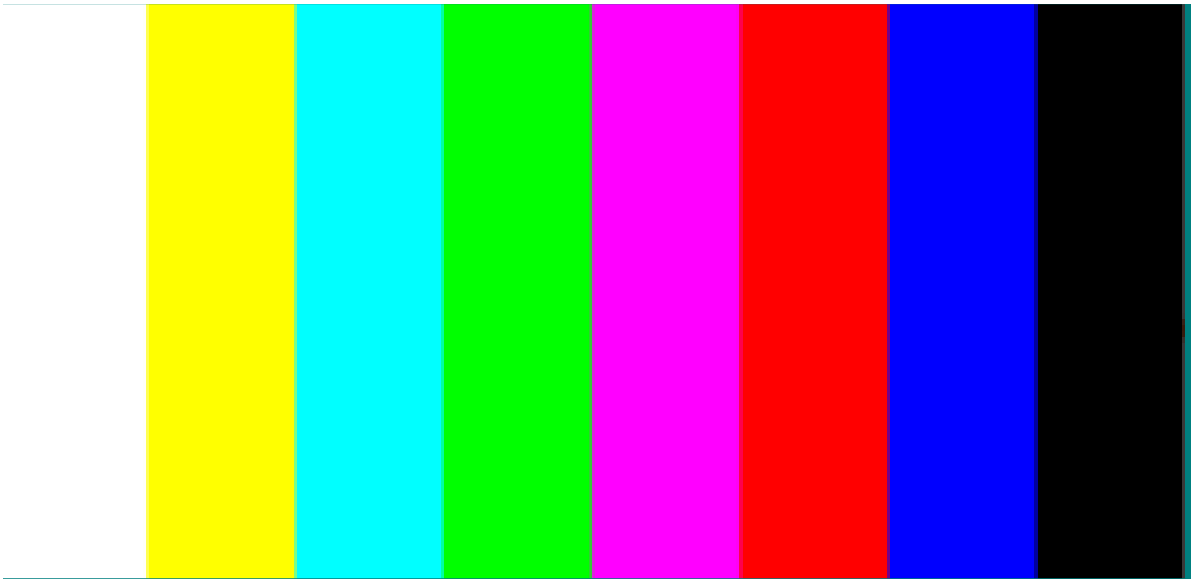


Figura 3.2 Patrón de pruebas utilizado.

Una vez finalizada la simulación del proyecto e implementado en el FPGA Spartan-3E disponible en la tarjeta de prototipos, se puede observar (Anexo IV) que al conectar el monitor al puerto VGA se visualiza en la pantalla el patrón tal como se muestra en la Figura 3.2

3.2 Generador de caracteres de texto

El código que rige este programa es el controlador de VGA (Video Graphics Adapter) donde se declaran las bibliotecas q se van a utilizar, la entidad con todas las entradas y salidas necesarias: **clk** y **reset** de entrada, **hsync** y **vsync** de salida y el vector **rgb** de tres cifras significativas. En la arquitectura del programa, es donde se implementan todas las funciones que este va a desarrollar. (Ver anexo II)

Después se pasa a la implementación del código del bloque VGA_SYNC que es el encargado del control de la sincronización del display VGA para que este se desempeñe correctamente. Luego se desarrolla el código de lo que sería el circuito de generación de pixelado que es el último bloque funcional del controlador y por último el código que se encarga de guardar en la ROM los caracteres que se quieren mostrar, en este caso particular los caracteres guardados formarán la palabra FIE y a continuación un rayo en representación a la facultad.

Luego de finalizada la programación se procede a establecer la relación entre las entradas y salidas del controlador de VGA y los pines de la tarjeta que serán utilizados en la aplicación, a través de la herramienta PlanAhead que posee el Xilinx ISE. La distribución de los pines se realiza con el preciso conocimiento de las funcionalidades de cada uno de estos para el buen desempeño de la aplicación. Después de creado el fichero .bit se descarga en la tarjeta con el programa Digilent Adept y se conecta un monitor por el puerto VGA. Los resultados se muestran en la Figura 3.3.

Figura 3.3 Comprobación del generador de texto.

3.3 Ping Pong

Este es el diseño de un juego que consta de tres objetos, dos paletas y una pelota cuadrada, donde su utiliza el puerto PS2 para introducir los datos mediante un teclado y el puerto VGA del kit Nexys2 para visualizar el juego. El diseño visualiza PONG en el display de cuatro dígitos del kit, recibe los datos serie del puerto PS2, los procesa y según los mismos varia la señal de salida VGA.

Este diseño es tomado de uno de los ejemplos del libro *FPGA prototyping by VHDL example* escrito por Pong P. Chu. Todos los módulos son descritos mediante VHDL, estos se conectan en un esquemático mediante símbolos de los mismos y se completa el diseño hasta la descarga en la placa como se observa en la Figura 3.4.

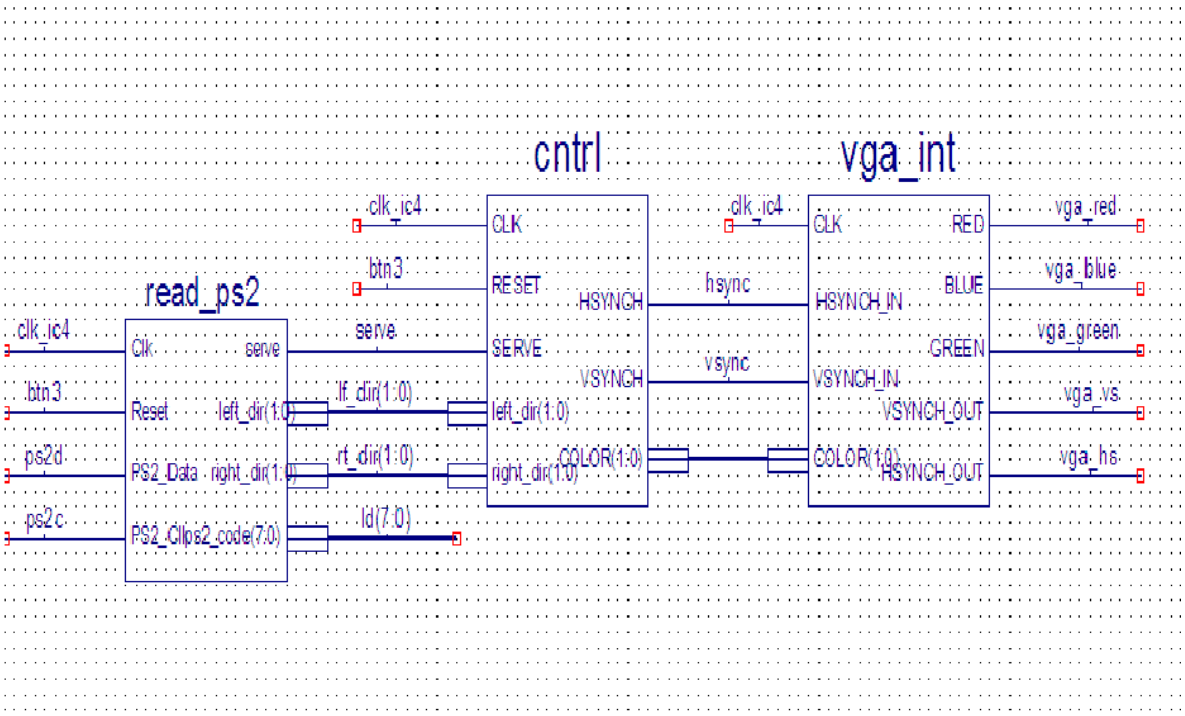


Figura 3.4 Esquemático de los bloques del juego Ping Pong.

Los resultados obtenidos tras la comprobación en el kit de desarrollo del ejemplo implementado se muestran en las imágenes siguientes.

Figura 3.5 Visulización en el display.

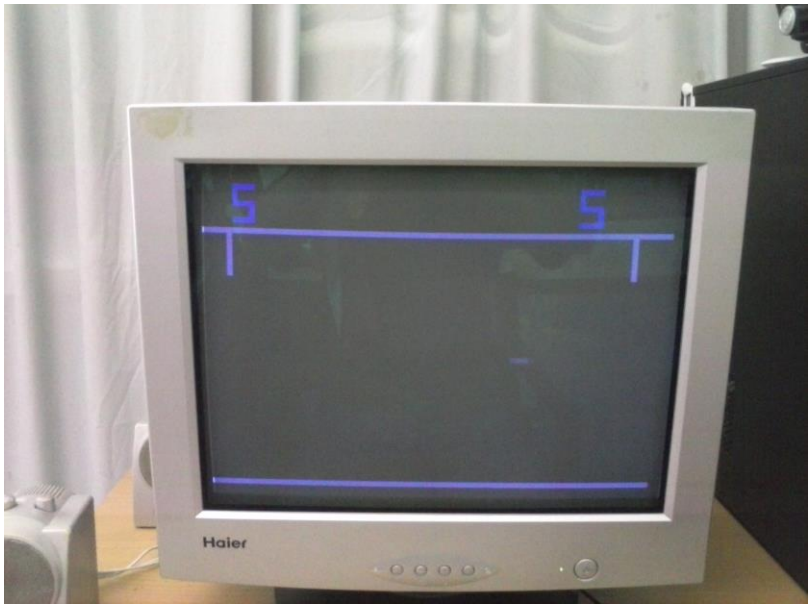


Figura 3.6 Imagen del juego de Ping Pong.

3.4 Ascensor de n pisos

Para el diseño del circuito que permita el control de un elevador de ***n*** pisos, se partirá de un ejemplo de un ascensor de 4 pisos y se plantearán los cambios a realizar cuando se quiere otra cantidad. Se elegirá un procesador de datos acorde con las características deseadas y se procederá a diseñar la lógica de control, posterior a lo cual se culminará el diseño del procesador de datos.

3.4.1 Espacios de entrada y salida de datos

Para el diseño del circuito se puede partir del símbolo mostrado en la Figura 3.7 donde se observa la señal de sincronismo CLK y los espacios de entrada y salida (Güichal, 2005b).

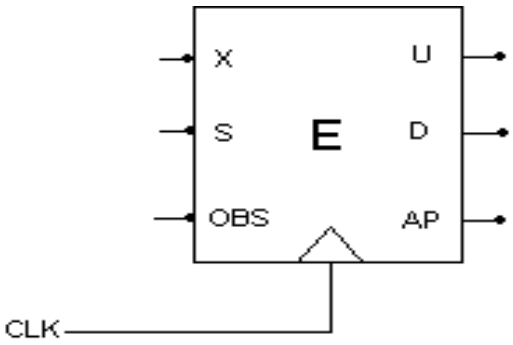


Figura 3.7 Símbolo.

El espacio de entrada está compuesto por (Jiménez. et al., 2012):

X → Señal de llamada del ascensor. Señal de llamada del ascensor. En el siguiente diseño, las llamadas interiores a la cabina (que se denomina ***a***) y las peticiones desde fuera (que se denomina s ***b***) tienen igual prioridad, o lo que es lo mismo $X = a \ or \ b$.

S → Señal proveniente de los sensores. Señal proveniente de los sensores. Los valores de S están en correspondencia con el piso en que se encuentre el elevador, como se muestra para el caso particular de 4 pisos, en la Tabla 3.3. Para un elevador de ***n*** pisos (desde Piso **0** hasta Piso ***n-1***), el valor de S se corresponde con la representación en binario del valor del piso.

OBS → Señal para detener la puerta.
OBS=’0’ cuando, cerrándose la puerta no ocurre interrupción.
OBS=’1’ cuando, cerrándose la puerta ocurre una interrupción.

Tabla 3.3 Valor de S según la posición del ascensor.

PISO	SEÑAL *S*
0	00
1	01
2	10
3	11

El espacio de salida está compuesto por:

U y D → Señales que determinan el movimiento del ascensor.
AP → Señal para abrir o cerrar la puerta.
AP='0' puerta cerrándose o cerrada.
AP='1' puerta abriéndose o abierta.

Tabla 3.4 Movimiento del ascensor según U y D

U	D	MOVIMIENTO DEL ASCENSOR
0	0	No se mueve (no se activan los motores)
0	1	Se mueve hacia abajo
1	0	Se mueve hacia arriba
1	1	No concebido

3.4.2 Diseño del Procesador de Datos y la Logica de Control

El procesador de datos está compuesto por los siguientes componentes sincrónicos y combinacionales que se pueden observar en la Figura 3.8:

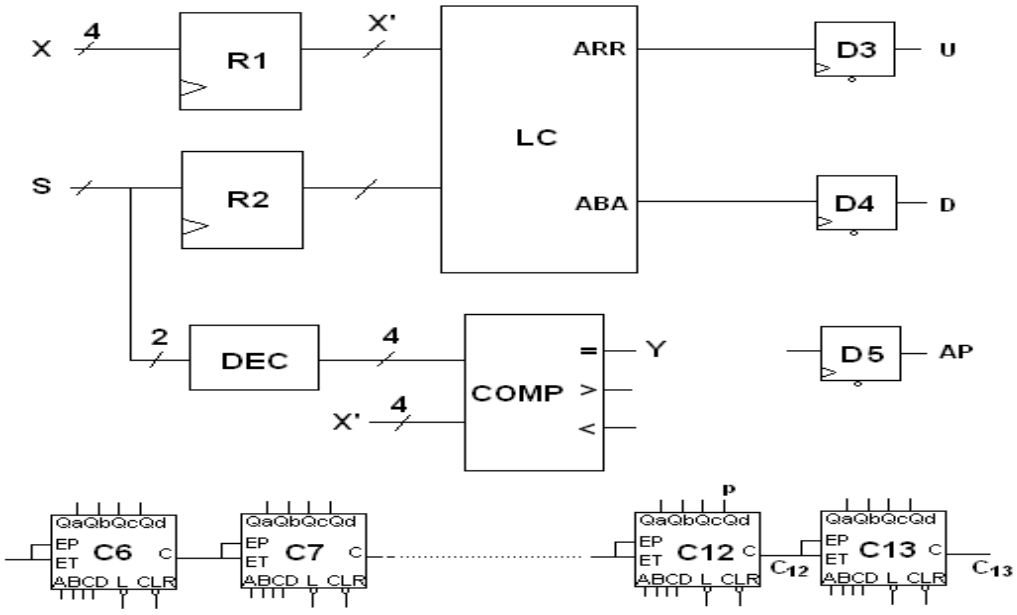


Figura 3.8 Componentes del Procesador de Datos

Componentes sincrónicos:

- Dos contadores 74xxx162 (R1 y R2).
- Tres biestables D (D3, D4 y D5)
- Ocho contadores 74xxx162 (C6, C7,…, C12 y C13)

Componentes Combinacionales:

- Decodificador **2** a **4** (DEC).
- Comparador 74xxx85 (COMP)
- Lógica combinacional (LC)

Después de analizar e implementar el funcionamiento de cada bloque, se procede a realizar la Máquina de Estado Algorítmico (ASM), como se muestra en el Anexo 1 y que posee 7 estados (Jiménez. et al., 2012).

- **T0** → Se establecen las condiciones iniciales del elevador: elevador detenido y con la puerta cerrada y contadores con el valor inicial requerido. Se procede a cargar en R1 y R2 los valores de ***X*** y ***S*** de modo que los componentes combinacionales determinen hacia donde debe ir dirigido el movimiento del ascensor, este proceso consume un determinado tiempo por lo que es necesario pasar al siguiente estado para continuar.
- **T1** → Se muestra a la salida del circuito la decisión tomada por LC de subir, bajar o no moverse. Si el ascensor no tiene que moverse regresa al estado anterior de lo contrario pasa al siguiente.
- **T2** → Durante este estado se comprueba que el ascensor haya llegado al piso destino, si no lo ha hecho se queda en este estado hasta que lo haga, y entonces pasa al estado siguiente.
- **T3** → Cuando se ha llegado al piso destino es necesario detener el elevador, comenzar a abrir la puerta y empezar a contar los 10s en que la puerta permanece abierta. Cuando transcurran los 10s se pasa al estado siguiente.
- **T4** → En este estado ya han transcurrido los 10s entonces hay que comenzar a cerrar la puerta. Esto toma un tiempo de 3s. Si durante este tiempo no hay una obstrucción el ascensor regresa a su estado inicial (T0), de lo contrario pasa al

estado siguiente.

- **T5 →** En este estado se restablecen los valores de los contadores para el próximo conteo en el estado siguiente.
- **T6 →** Debido al obstrucción causada se comienza a abrir la puerta nuevamente, esta vez en un tiempo menor (5s) que una vez concluido regresa a T4 donde se comienza a cerrar la puerta repitiéndose el mismo proceso.

A partir del ASM descrito se puede realizar el diagrama de estados correspondientes a la lógica de control, mostrado en la Figura 3.9 que está estrechamente relacionado con los bloques de toma de decisión del ASM.

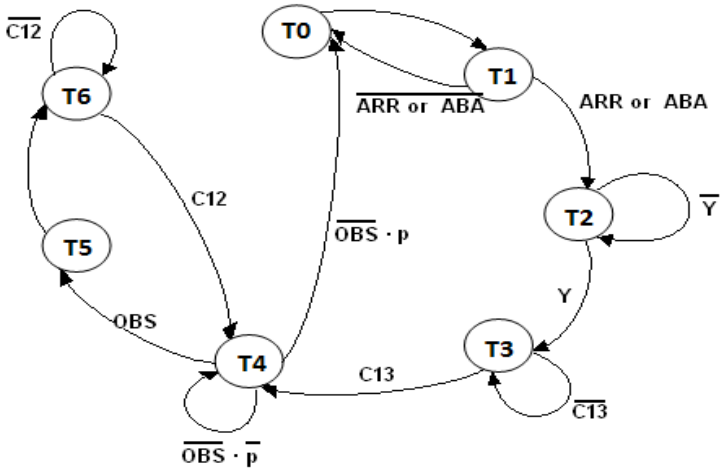


Figura 3. 9 Diagrama de Estados de lo Lógica de Control

Teniendo en cuenta el diagrama de estados de la Figura 3.9 se pueden plantear las ecuaciones correspondientes a la lógica de control y a partir del ASM descrito se puede culminar el diseño del procesador de datos, estableciendo cuáles deben ser las conexiones de los distintos pines de cada uno de los componentes, con todo lo anterior se puede confeccionar el circuito para el control de un elevador de ***n*** pisos, con la utilización de lógica cableada VLSI para cada bloque de lógica combinacional. Un esquema con el circuito en bloques puede ser consultado en la Figura 3.10 (Jiménez. et al., 2012).

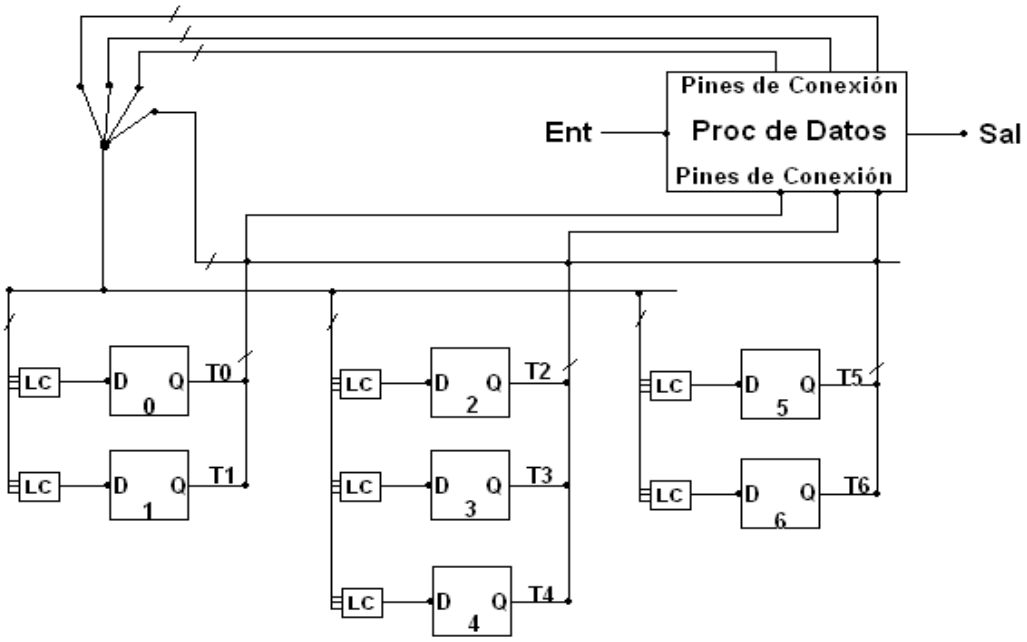


Figura 3.10 Esquema general del circuito

Para la comprobación de los resultados obtenidos se lleva a cabo la elaboración del archivo que se introduce en la tarjeta. La descripción del circuito se implementa en el lenguaje VHDL y luego se hace una declaración de pines, de modo que exista una correspondencia ente las variables de entrada y salida del programa y los pines de la tarjeta (Perdomo, 2001).

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

1. El desarrollo de la Electrónica en general, y de la Electrónica Digital en particular, ha modificado el contenido y los métodos de diseñar. El uso de herramientas computacionales de ayuda al diseño, las estrategias de diseño descendente (top-down), la utilización de los lenguajes de descripción de hardware (HDL) y el trabajo con dispositivos lógicos programables, caracterizan el estado actual de esta ciencia aplicada, lo que exige que en el proceso de enseñanza-aprendizaje de la misma se realicen las modificaciones pertinentes para preparar a los profesionales en el dominio de estos contenidos y habilidades.
2. Se considera que el objetivo fundamental del presente trabajo se ha alcanzado, por cuanto se proponen un conjunto de ejemplos integradores que vinculan conocimientos de diferentes asignaturas del año y que por su aplicabilidad resultan motivantes para los estudiantes.
3. En el desarrollo de los ejemplos integradores se han tenido en cuenta las tres estrategias fundamentales del diseño electrónico digital moderno: descripción esquemática por jerarquía de niveles, empleo de herramientas de descripción matemática de alto nivel, descripción en lenguaje de hardware. Cada una de las cuales tiene su finalidad dentro de la preparación integral de un diseñador electrónico digital.
4. Los ejemplos desarrollados han sido diseñados con la particularidad de contener subsistemas típicos de la electrónica en diferentes perfiles de la misma, lo que facilitará la asimilación gradual por parte de los estudiantes de los fundamentos teóricos y la comprobación práctica de los mismos en las diferentes actividades

docentes de la asignatura DDVLSI.

5. Los ejemplos desarrollados han sido simulados y comprobados en el kit de desarrollo NEXYS 2, con excelentes resultados.

Recomendaciones

1. Proponer al colectivo de las asignaturas ED I, ED II y DDVLSI que se realice el análisis metodológico pertinente para el perfeccionamiento de la asignatura DDVLSI a partir de los ejemplos integradores propuestos.
2. Sugerir que el desarrollo de las prácticas de laboratorio de DDVLSI se realicen a partir del diseño y simulación de los diferentes subsistemas que componen los ejemplos integradores.
3. Proponer al Colectivo de 3er Año que valore la posibilidad de vincular estos y otros ejemplos integradores como Proyectos de Curso de la Disciplina Integradora.
4. Continuar trabajando en la búsqueda de otros ejemplos integradores que tengan utilidad en la producción y/o los servicios, como elemento motivador para los estudiantes.

REFERENCIAS BIBLIOGRÁFICAS

2011. Lineamientos de la Politica Económica y Social del Partido y la Revolución. 6to Congreso del Partido: PCC.

AL-HADITHI, BASIL, M., SUARDÍAZ, Z. & MURO, J. 2004. Nuevas tendencias en el diseño electrónico digital: codiseño hardware/software. *Tecnologí@ y Desarrollo.* Madrid: Universidad Alfonso X el sabio.

ALVAREZ, Z. 1986. Elementos de didáctica de la Educación Superior.

ARIAS, R. 2010. Metodología para el diseño de aplicaciones medianas en FPGAs de

Xilinx. Universidad Central "Marta Abreus" de las Villas. .

BARRIOS, J. P. 2005. *Estrategia didáctica para el desarrollo de la habilidad diseño electrónico digital en estudiantes de Ingenieria en Telecomunicaciones y Electrónica.* Tesis presentada en opción al grado científico de Doctor en Pedagogía, Universidad Central "Marta Abreu" de las Villas.

CASTELLÓ, E. M. & VALIDO, M. R. 2010. Tutorial de Xilinx ISE.

CHU, P. P. 2008. FPGA prototyping by VHDL example. John Wiley & Sons.

.

COMPTON, K. & HAUCK, S. 2002. "Recunfigurable Computing: a Survey of Systems and Software," *ACM Computing Surveys (CSUR)* 34**,** 171- 210.

DORRMIDO, S. 2004. *Control Learning: Present and Future*, Annual Reviews in control.

GAJSKI, D. 1988. Silicon Compilation.: Addison-Wesley.

GARTNER, D. 2000. *"Electronic Design Automation Worlwide 2000"DoD VHDL Project.*

GÜICHAL, G. 2005a. Diseño Digital Utilizando Lógicas Programables Universidad Tecnológica Nacional Facultad Regional Bahía Blanca.

GÜICHAL, G. 2005b. Diseño digital utilizando lógicas programables. Universidad Tecnológica Nacional, Facultad Regional Bahía Blanca.

GUTIÉRREZ, M. 2004. *Introducción a la tecnología FPGA* [Online]. Available: http://ccc.inaoep.mx/fpgacentral/pdf/introfpga.pdf . [Accessed 20/5/14 2014].

IGLESIAS, C. 2008. Fundamentos teóricos para la implementación de la didáctica en el proceso enseñanza aprendizaje. Universidad de Cienfuegos "Carlos Rafael Rodríguez".

JIMÉNEZ., A. A., MORALES., L. G., MOREJÓN., R. J. G., FLORES., L. C. & DARIAS, L. L. 2012. Metodología, diseño y comprobación de un algoritmo para el control digital de un elevador de n pisos

Facultad de Ingeniería Eléctrica. Universidad Central "Marta Abreu "de las Villas.

LABARRERE, A. 1996. *Análisis y autorregulación de la actividad cognoscitiva de los alumnos,*, La Habana.

LÓPEZ, M. L. & AYALA, J. L. 2004. FPGA: Nociones básicas e implementación. Available: http://www.miky.com.ar/fpga_2004.pdf [Accessed 3 febrero 2010].

MANDADO, E. 2000. *"Sistemas Digitales, principios y aplicaciones",* España.

MANDADO, E. & VALDÉS, M. 2004. "Sistema integrado para la enseñanza/aprendizaje de la Electrónica". *VI Congreso TAEE*

MANDADO, E., VALDÉS, M. & ÁLVAREZ, J. 2002. *Dispositivos Lógicos Programables y sus aplicaciones*, España, Ed. Thomson.

MATHWORKS, T. 2009. *Simulink Getting Started Guide* [Online]. Available: http://www.mathworks.com/access/helpdesk/help/pdf_doc/simulink/sl_gs.pdf

MES 2007. Plan de Estudios C (modificado). Ciudad de La Habana: Ministerio de Educación Superior.

MORENO, A. 2005. Técnicas de diseño digital y sistemas configurables (FPGAs). 9ª ed. España: Universidad Politécnica de Cataluña.

POLLÁN, T. 2003. Sistemas Combinacionales. *Electrónica Digital. I.* Universidad de Zaragoza: Prensas Universitarias de Zaragoza.

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

SAGAHYROON, A. & ASSIM, A. 2000. From AHPL to VHDL: A Course in Hardware Description Languages. IEEE Transactions on Education. VOL 43. NO. 4

SANZ, T. 2004. El curriculum. Su conceptualización. *Revista Pedagogía Universitaria.* Universidad de La Habana: CEPES.

SAVAGE, W., CHILTON, J. & CAMPOSANO, R. 2000. IP reuse in the system on a chip era, . *Proceedings 13th International Symposium on System Synthesis*

TYLER, J. 1971. El curriculum. Su conceptualización *Revista Pedagogía Universitaria.* Universidad de La Habana: CEPES.

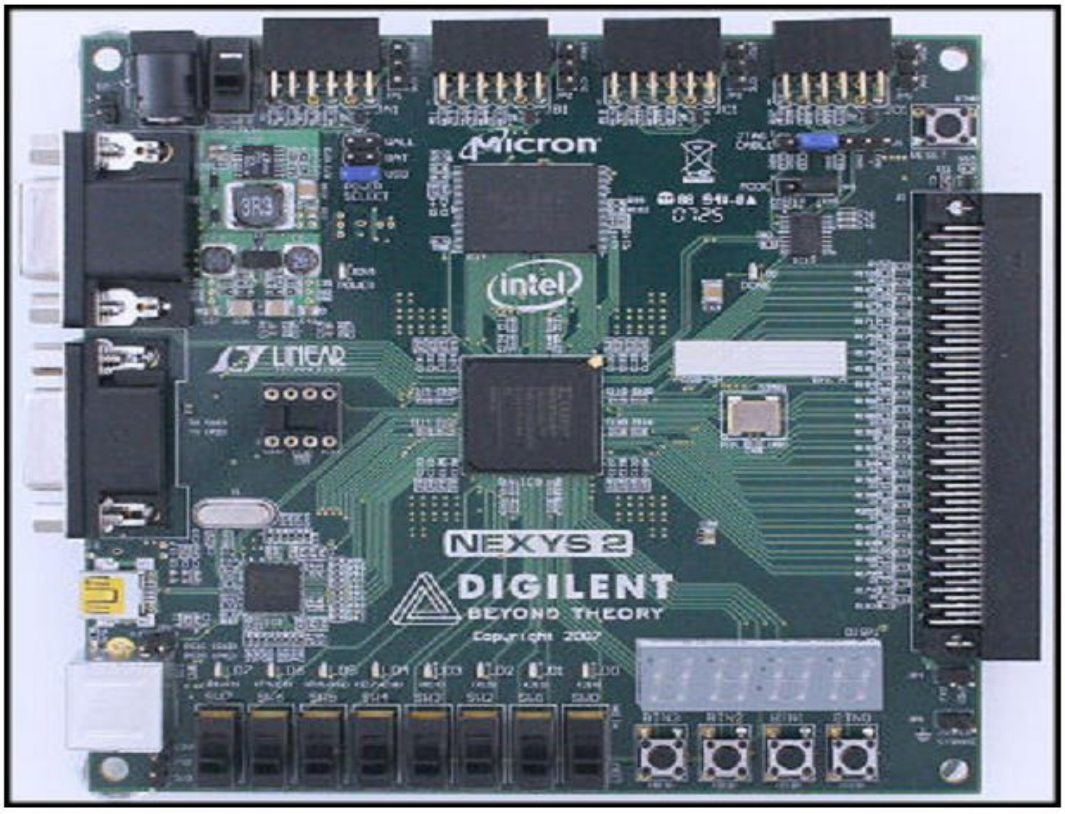
XILINX. 2010. *System Generator for DSP user guide* [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/sysgen_user.pdf

ZEMVA, A. 1998. A Rapid Prototyping Environment for Teaching Digital Logic Design. IEEE TRANSACTIONS ON EDUCATION.

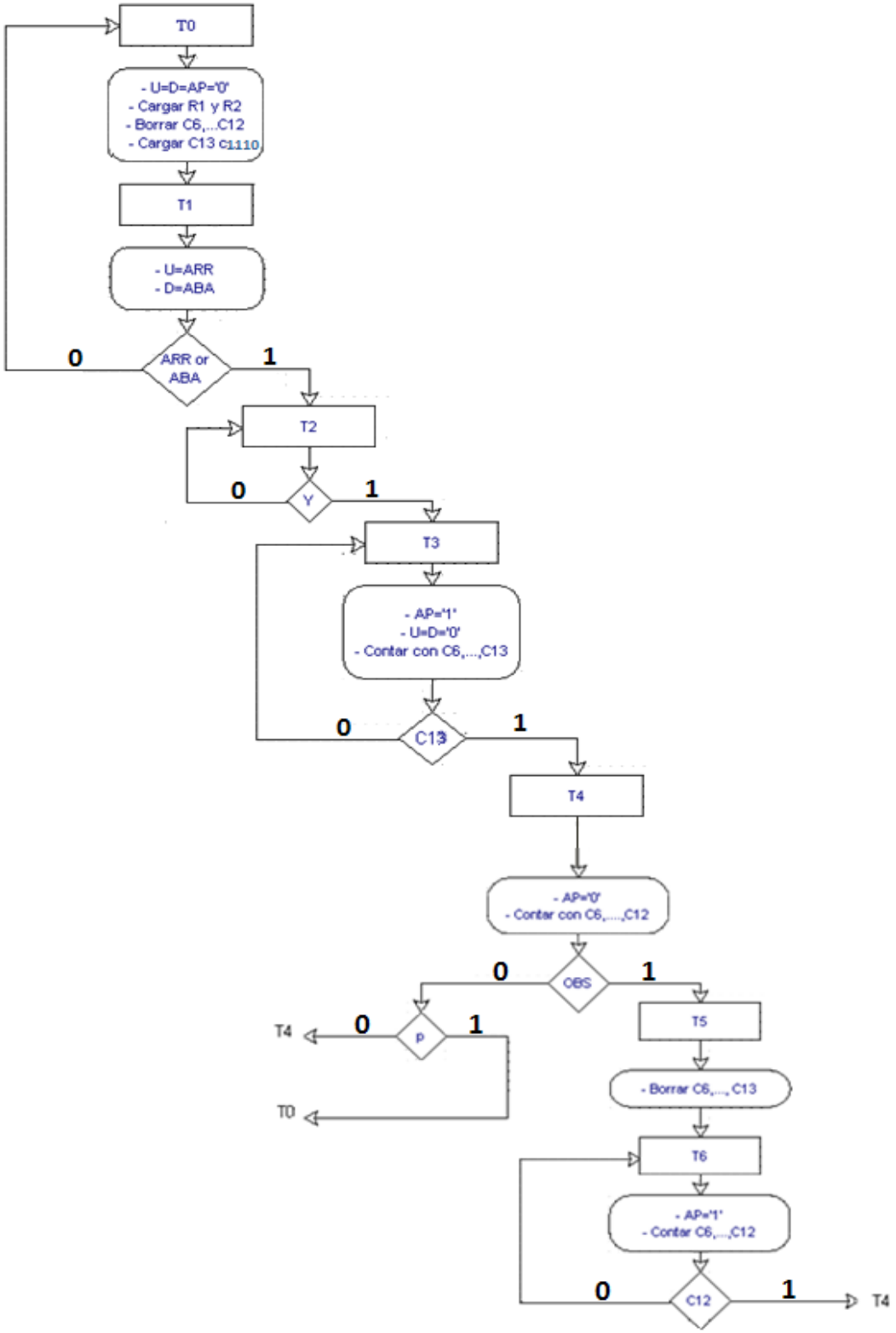
ANEXOS

Anexo I

a) Kit de desarrollo Digilent S3E.



b) Diagrama ASM.



b) Conexiones del procesador de datos.

R1	R2	D3
$EP = ET = nc$ $CLR =' \mathbf{1}'$ $LOAD = \overline{T\mathbf{0}}$ $D, C, B, A = X$	$EP = ET = nc$ $CLR =' \mathbf{1}'$ $LOAD = \overline{T\mathbf{0}}$ $B, A = S$ $D, C = nc$	$D = ARR.T\mathbf{1}$ $PR =' \mathbf{1}'$ $CLR = \overline{T3 + T\mathbf{0}}$
D4	**D5**	**Contadores**
$D = ARR.T\mathbf{1}$ $PR =' \mathbf{1}'$ $CLR = \overline{T3 + T\mathbf{0}}$	$D = T3 + T\mathbf{6}$ $PR =' \mathbf{1}'$ $CLR = \overline{T4 + T\mathbf{0}}$	C6 $EP = ET = T3 + T4 + T\mathbf{6}$ C6-C12 $CLR = \overline{T\mathbf{0} + T\mathbf{5}}$ El resto *nc* C13 $A, B, C, D = 111\mathbf{0}$ $CLR = \overline{T\mathbf{5}}$

c) Declaración de pines de entrada y salida

ENTRADA Y SALIDA	PINES DE LA TARJETA
X(3)	B18 (Buttons)
X(2)	D18 (Buttons)
X(1)	E18 (Buttons)
X(0)	H13 (Buttons)
S(1)	G18 (Switches)
S(0)	H18 (Switches)
U	J14 (LED)
D	J15 (LED)
OBS	R17 (Switches)
AP	R4 (LED)
CLK	B8

d) Código VHDL del ascensor

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
```

```
library UNISIM;
use UNISIM.Vcomponents.ALL;
entity asmelevavhd is
  port ( clk : in    std_logic;
       OBS : in    std_logic;
       S   : in    std_logic_vector (1 downto 0);
       X   : in    std_logic_vector (3 downto 0);
       AP  : out   std_logic;
       D   : out   std_logic;
       U   : out   std_logic);
end asmelevavhd;
architecture BEHAVIORAL of asmelevavhd is
type state is (T0, T1, T2, T3, T4, T5, T6);
signal ps: state;
signal S_OBS: std_logic;
signal S_S: unsigned (1 downto 0);
signal S_X: unsigned (3 downto 0);
signal S_AP: std_logic;
signal S_U: std_logic;
signal S_D: std_logic;
signal X_reg: unsigned (3 downto 0);
signal S_reg: unsigned (1 downto 0);
signal S_dec: unsigned (3 downto 0);
signal ARR: std_logic;
signal ABA: std_logic;
signal ABAorARR: std_logic;
signal Y: std_logic;
signal cnt3:  integer := 0;
signal cnt5:  integer := 0;
signal cnt10: integer := 0;
begin
-- type conversion
S_OBS <= OBS;
S_S <= unsigned (S);
S_X <= unsigned (X);
AP <= S_AP;
D <= S_D;
```

```
U <= S_U;
process(S_S, S_dec,ABA, ARR, S_reg, X_reg)
begin
case S_S is
  when "00"=>
    S_dec<="0001";
  when "01"=>
    S_dec<="0010";
  when "10"=>
    S_dec<="0100";
  when others =>
    S_dec<="1000";
end case;

if S_dec=X_reg then
  Y<='1';
else
  Y<='0';
end if;
ABAorARR<=ABA or ARR;
------Logica Combinacional para determinar el movimiento del ascensor
case S_reg is
   when "00"=>
    case X_reg is
   when "0010"=>
  ARR<='1';
  ABA<='0';
   when "0100"=>
 ARR<='1';
ABA<='0';
 when "1000"=>
ARR<='1';
ABA<='0';
 when others=>
ARR<='0';
ABA<='0';
            end case;
```

```
when "01"=>
    case X_reg is
when "0100"=>
 ARR<='1';
 ABA<='0';
when "1000"=>
 ARR<='1';
 ABA<='0';
when "0001"=>
 ARR<='0';
 ABA<='1';
 when others=>
  ARR<='0';
  ABA<='0';
         end case;
 when "10"=>
     case X_reg is
 when "0001"=>
   ARR<='0';
   ABA<='1';
when "0010"=>
   ARR<='0';
   ABA<='1';
when "1000"=>
   ARR<='1';
   ABA<='0';
when others=>
   ARR<='0';
   ABA<='0';
            end case;
when others =>
           case X_reg is
when "0001"=>
   ARR<='0';
   ABA<='1';
when "0010"=>
   ARR<='0';
```

```
   ABA<='1';
when "0100"=>
   ARR<='0';
   ABA<='1';
when others=>
  ARR<='0';
   ABA<='0';
                end case;
      end case;
end process;
process(clk)
begin
 if clk='1' and clk'event then
                case ps is
when T0=>
 X_reg<=S_X;
 S_reg<=S_S;
 S_U<='0';
 S_D<='0';
 S_AP<='0';
 cnt3<=0;
 cnt5<=0;
 cnt10<=0;
  ps<=T1;
when T1=>
 S_U<=ARR;
 S_D<=ABA;
if ABAorARR='0' then
  ps<=T0;
  else
   ps<=T2;
            end if;
when T2=>
  if Y='0' then
   ps<=T2;
  else
   ps<=T3;
```

```
            end if;
when T3=>
  S_AP<='1';
  S_U<='0';
  S_D<='0';
  cnt10<=cnt10+1;
if cnt10=500000000 then        --para contar 10s
  ps<=T4;
else
  ps<=T3;
            end if;
when T4=>
 S_AP<='0';
  cnt3<=cnt3+1;
if S_OBS='1' then
 ps<=T5;
  else
if cnt3=150000000 then       --para contar 3s
  ps<=T0;
 else
  ps<=T4;
             end if;
             end if;
when T5=>
 cnt5<=0;
 ps<=T6;
when T6=>
  S_AP<='1';
  cnt5<=cnt5+1;
if cnt5=250000000 then        --para contar 5s
  ps<=T4;
else
  ps<=T6;
               end if;
         end case;
 end if;
  end process;
```

```
end BEHAVIORAL;
```

Anexo II

a) Código ASCII.

Code	Char	Code	Char	Code	Char	Code	Char
00	(nul)	20	(sp)	40	@	60	‘
01	(soh)	21	!	41	A	61	a
02	(stx)	22	"	42	B	62	b
03	(etx)	23	#	43	C	63	c
04	(eot)	24	$	44	D	64	d
05	(enq)	25	%	45	E	65	e
06	(ack)	26	&	46	F	66	f
07	(bel)	27	’	47	G	67	g
08	(bs)	28	(	48	H	68	h
09	(ht)	29	)	49	I	69	i
0a	(nl)	2a	*	4a	J	6a	j
0b	(vt)	2b	+	4b	K	6b	k
0c	(np)	2c	,	4c	L	6c	l
0d	(cr)	2d	-	4d	M	6d	m
0e	(so)	2e	.	4e	N	6e	n
0f	(si)	2f	/	4f	O	6f	o
10	(dle)	30	0	50	P	70	p
11	(dc1)	31	1	51	Q	71	q
12	(dc2)	32	2	52	R	72	r
13	(dc3)	33	3	53	S	73	s
14	(dc4)	34	4	54	T	74	t
15	(nak)	35	5	55	U	75	u
16	(syn)	36	6	56	V	76	v
17	(etb)	37	7	57	W	77	w
18	(can)	38	8	58	X	78	x
19	(em)	39	9	59	Y	79	y
1a	(sub)	3a	:	5a	Z	7a	z
1b	(esc)	3b	;	5b	[	7b	{
1c	(fs)	3c	<	5c	\	7c	|
1d	(gs)	3d	=	5d	]	7d	}
1e	(rs)	3e	>	5e	^	7e	~
1f	(us)	3f	?	5f	_	7f	(del)

b) Asignación de pines

SEÑALES	PINES
"clk"	B8
"hsync"	T4
"rgb[0]"	R8
"rgb[1]"	P6
"rgb[2]"	U4
"rgb[2]"	U3

c) Código base del generador de caracteres.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity font_test_top is
    Port ( clk, reset : in  STD_LOGIC;
           hsync, vsync : out  STD_LOGIC;
           rgb : out  STD_LOGIC_VECTOR (2 downto 0));
end font_test_top;

architecture arch of font_test_top is
    signal pixel_x, pixel_y: STD_LOGIC_VECTOR(9 downto 0);
    signal video_on, pixel_tick: STD_LOGIC;
    signal rgb_reg, rgb_next: STD_LOGIC_VECTOR(2 downto 0);
begin
    --instantiate VGA sync circuit
    CSVGA: entity work.CSVGA(arch)
    port map(clk => clk, reset => reset, hsync => hsync,
                         vsync => vsync, video_on => video_on,
                         pixel_x => pixel_x, pixel_y => pixel_y,
                         p_tick => pixel_tick);
    --instantiate font ROM
    font_gen_unit: entity work.font_test_gen(arch)
    port map(clk => pixel_tick , video_on => video_on,
                         pixel_x => pixel_x, pixel_y => pixel_y,
                         rgb_text => rgb_next);
```

```
    --rgb buffer
    process (clk)
    begin
            if (clk' event and clk = '1') then
                    if (pixel_tick = '1') then
                            rgb_reg <= rgb_next;
                    end if;
            end if;
    end process;
    rgb <= rgb_reg;
end arch;
```

d) Código del circuito de generación de pixelado.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity font_test_gen is
    Port ( clk : in  STD_LOGIC;
           video_on : in  STD_LOGIC;
           pixel_x, pixel_y : STD_LOGIC_VECTOR (9 downto 0);
           rgb_text : out  STD_LOGIC_VECTOR (2 downto 0));
end font_test_gen;


architecture arch of font_test_gen is
    signal rom_addr: STD_LOGIC_VECTOR(10 downto 0);
    signal char_addr: STD_LOGIC_VECTOR(6 downto 0);
    signal row_addr: STD_LOGIC_VECTOR(3 downto 0);
    signal bit_addr: STD_LOGIC_VECTOR(2 downto 0);
    signal font_word: STD_LOGIC_VECTOR(7 downto 0);
    signal font_bit, text_bit_on: STD_LOGIC;
begin
    --instantiate font ROM
    font_unit: entity work.font_rom
            port map( clk => clk, addr => rom_addr, data => font_word);
    --font ROM interfase
    char_addr <= pixel_y(5 downto 4) & pixel_x(7 downto 3);
    row_addr <= pixel_y(3 downto 0);
    rom_addr <= char_addr & row_addr;
    bit_addr <= pixel_x(2 downto 0);
    font_bit <= font_word(to_integer(unsigned(not bit_addr)));
    --on region limited to top-left corner
    text_bit_on <=
            font_bit when pixel_x(9 downto 8) = "00" and
                                        pixel_y(9 downto 6) = "0000" else
            '0';
    --rgb multiplexing circuit
    process(video_on, font_bit, text_bit_on)
    begin
            if video_on = '0' then
                    rgb_text <= "000"; --blank
            else
                    if text_bit_on = '1' then
                            rgb_text <= "010"; --green
                    else
```

```
                        rgb_text <= "000"; --black
                end if;
            end if;
    end process;
end arch;
```

e) Código de sincronización del VGA.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity CSVGA is
    Port ( clk, reset : in  STD_LOGIC;
           hsync, vsync : out  STD_LOGIC;
           video_on, p_tick : out  STD_LOGIC;
           pixel_x, pixel_y : out  STD_LOGIC_VECTOR (9 downto 0)
    );
end CSVGA;

architecture arch of CSVGA is
    --VGA 640-by-480 sync parameters
    constant HD: integer:=640; --horizontal display area
    constant HF: integer:=16; --h.front porch
    constant HB: integer:=48; --h.back porch
    constant HR: integer:=96; --h.retrace
    constant VD: integer:=480; --vertical display area
    constant VF: integer:=10; --v.front porch
    constant VB: integer:=33; --v.back porch
    constant VR: integer:=2; --v.retrace
    --mod-2 counter
    signal mod2_reg, mod2_next: STD_LOGIC;
    --sync counters
    signal v_count_reg, v_count_next: UNSIGNED(9 downto 0);
    signal h_count_reg, h_count_next: UNSIGNED(9 downto 0);
    --output buffer
    signal v_sync_reg, h_sync_reg: STD_LOGIC;
    signal v_sync_next, h_sync_next: STD_LOGIC;
    --status signal
    signal h_end, v_end, pixel_tick: STD_LOGIC;
begin
    --registers
    process (clk, reset)
    begin
            if reset='1' then
                    mod2_reg <= '0';
                    v_count_reg <= (others => '0');
                    h_count_reg <= (others => '0');
                    v_sync_reg <= '0';
                    h_sync_reg <= '0';
            elsif (clk' event and clk = '1') then
                    mod2_reg <= mod2_next;
                    v_count_reg <= v_count_next;
                    h_count_reg <= h_count_next;
                    v_sync_reg <= v_sync_next;
```

```
                h_sync_reg <= h_sync_next;
        end if;
end process;
--mod-2 circuit to generate 25 MHz enable tick
mod2_next <= not mod2_reg;
--25 MHz pixel tick
pixel_tick <= '1' when mod2_reg = '1' else '0';
--status
h_end <= --end of horizontal counter
        '1' when h_count_reg = (HD+HF+HB+HR-1) else --799
        '0';
v_end <= --end of vertical counter
        '1' when v_count_reg = (VD+VF+VB+VR-1) else --524
        '0';
--mod-800 horizontal sync counter
process ( h_count_reg, h_end, pixel_tick )
begin
        if pixel_tick = '1' then --25 MHz tick
                if h_end = '1' then
                        h_count_next <= (others => '0');
                else
                        h_count_next <= h_count_reg + 1;
                end if;
        else
                h_count_next <= h_count_reg;
        end if;
end process;
--mod-525 vertical sync counter
process (v_count_reg, h_end, v_end, pixel_tick)
begin
        if pixel_tick = '1' and h_end = '1' then
                if (v_end = '1') then
                        v_count_next <= (others => '0');
                else
                        v_count_next <= v_count_reg + 1;
                end if;
        else
                v_count_next <= v_count_reg;
        end if;
end process;
--horizontal and vertical sync, buffered to avoid glitch
h_sync_next <=
        '1' when (h_count_reg >= (HD+HF))          --656
                 and (h_count_reg <= (HD+HF+HR-1)) else --751
        '0';
v_sync_next <=
        '1' when (v_count_reg >= (VD+VF))          --490
                 and (v_count_reg <= (VD+VF+VR-1)) else --491
        '0';
--video on/off
video_on <=
        '1' when (h_count_reg < HD) and (v_count_reg < VD) else
        '0';
--output signal
hsync <= h_sync_reg;
vsync <= v_sync_reg;
```

```
    pixel_x <= STD_LOGIC_VECTOR(h_count_reg);
    pixel_y <= STD_LOGIC_VECTOR(v_count_reg);
    p_tick <= pixel_tick;
end arch;
```

f) Código de impresión en la ROM.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity font_rom is
    Port ( clk : in  STD_LOGIC;
           addr : in  STD_LOGIC_VECTOR (10 downto 0);
           data : out  STD_LOGIC_VECTOR (7 downto 0));
end font_rom;

architecture arch of font_rom is
    constant ADDR_WIDTH: integer :=11;
    constant DATA_WIDTH: integer :=8;
    signal addr_reg: STD_LOGIC_VECTOR(ADDR_WIDTH-1 downto 0);

type rom_type is array (0 to 80)
              of STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0);
    --ROM definition
    constant ROM : rom_type:= ( --2^11-by-8
    --code x00(F)
    "00000000",
    "00000000",
    "11111110",
    "11111110",
    "11000000",
    "11000000",
    "11111000",
    "11111000",
    "11000000",
    "11000000",
    "11000000",
    "11000000",
    "11000000",
    "00000000",
    "00000000",
    "00000000",
    "00000000",
    --code x01(I)
    "00000000",
    "00000000",
    "11111100",
    "11111100",
    "00110000",
    "00110000",
    "00110000",
    "00110000",
    "00110000",
    "00110000",
    "11111100",
    "11111100",
    "00000000",
```

```
    "00000000",
    "00000000",
    "00000000",
    --code x02(E)
    "00000000",
    "00000000",
    "11111110",
    "11111110",
    "11000000",
    "11000000",
    "11111000",
    "11111000",
    "11000000",
    "11000000",
    "11111110",
    "11111110",
    "00000000",
    "00000000",
    "00000000",
    "00000000",
    --code x03(blank space)
    "00000000",
    "00000000",
    "00000000",
    "00000000",
    "00000000",
    "00000000",
    "00000000",
    "00000000",
    "00000000",
    "00000000",
    "00000000",
    "00000000",
    "00000000",
    "00000000",
    "00000000",
    "00000000",
    "00000000",
    --code x04(Thunder)
    "00000000",
    "00000000",
    "00001111",
    "00011110",
    "00111100",
    "01111000",
    "11111110",
    "00011100",
    "00111000",
    "01110000",
    "01100000",
    "01000000",
    "00000000",
    "00000000",
    "00000000",
    "00000000"
    );
begin
```

```
--addr register to infer block RAM
process (clk)
begin
        if (clk' event and clk = '1') then
                addr_reg <= addr;
        end if;
end process;
data <= ROM(to_integer(unsigned(addr_reg)));
                                                end arch;
```

Anexo III

a) Código VHDL del Ping Pong

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity cntrl is Port (

        CLK :       in std_logic;

        RESET:      in std_logic;

        left_dir:           in std_logic_vector(1 downto 0);

        right_dir:  in std_logic_vector(1 downto 0);

        SERVE:      in std_logic;

        HSYNCH:  out std_logic;

        VSYNCH: out std_logic;

        COLOR:      out std_logic_vector( 1 downto 0 )

        );
```

```
end cntrl;

architecture static_display of cntrl is

component vgacore is Port (
        CLK :       in std_logic;
        RESET:      in std_logic;
        HSYNCH:  out std_logic;
        VSYNCH: out std_logic;
        HBLANK: out std_logic;
        LINE:       out std_logic_vector(5 downto 0);
        PIXEL:      out std_logic_vector(6 downto 0)
        );
end component;

component testram is Port (
        address: in std_logic_vector( 6 downto 0 );
        data:  out std_logic_vector( 3 downto 0 )
        );
end component;

signal        number_data: std_logic_vector( 3 downto 0 );
signal   number_address: std_logic_vector( 6 downto 0 );
signal        enable: std_logic;
```

```
signal      LINE:              std_logic_vector( 5 downto 0 );
signal      PIXEL:     std_logic_vector( 6 downto 0 );
signal      HBLANK: std_logic;

signal      next_HSYNCH:  std_logic;
signal      next_VSYNCH:  std_logic;
signal      VCLK: std_logic;
signal   next_COLOR:  std_logic_vector( 1 downto 0 );

constant ball_height: integer := 1;
constant ball_width: integer := 2;

constant paddle_height: integer := 8;
constant paddle_heighta: integer := 4;     -- defines top 1/4 of paddle
constant paddle_heightb: integer := 6;     -- defines middle 1/2 of paddle
constant paddle_heightc: integer := 8;     -- defines bottom 1/4 of paddle
constant paddle_width: integer := 2;       -- defines width of paddle

constant wall_height: integer := 1;
constant wall_top: integer := 9;
constant wall_bottom: integer := 58;
constant right_wall: integer := 77;
constant left_wall: integer := 2;
```

```
constant right_score_x: integer:= 64;
constant right_score_y: integer:= 4;
constant left_score_x: integer:= 16;
constant left_score_y: integer:= 4;
constant score_width: integer := 4;
constant score_height: integer := 4;

constant right_x: integer := 72;
constant left_x: integer := 8;

signal ball_xdir, ball_ydir: std_logic;
signal ball_x: integer range 0 to 80;
signal ball_y: integer range 0 to 60 := 30;
signal ball_yrate: integer range 0 to 3;

signal left_y: std_logic_vector(5 downto 0);
signal right_y: std_logic_vector(5 downto 0);
signal nextleft_y: std_logic_vector(5 downto 0);
signal nextright_y: std_logic_vector(5 downto 0);

signal delay: std_logic_vector(2 downto 0);          -- Delay vector is used to slow down
the speed of the ball

signal lscore, rscore: integer range 0 to 9 := 0;
```

```
begin



-- VGA CORE Instantiation
VGA1: vgacore port map (
        CLK => CLK,
        RESET=> RESET,
        HSYNCH=> next_HSYNCH,
        VSYNCH=> next_VSYNCH,
        HBLANK=> HBLANK,
        LINE=> LINE,
        PIXEL=> PIXEL
        );

-- Character generator memory instantiation
CGEN1: testram port map (
        address      => number_address ,
        data         => number_data
        );

-- Pipeline the control signals to account for Game Delay
pipeline: process ( clk, pixel, line)
begin
```

```
        if ( clk = '1' and clk'event) then
                VSYNCH <= next_VSYNCH;
                VCLK <= next_VSYNCH;
                HSYNCH <= next_HSYNCH;
                if ( HBLANK = '1' ) then
                        color <= "00";
                else
                        color <= next_COLOR;
                end if;
        end if;
end process;

-- Code to display the ball and paddles
display: process (clk, line, pixel, left_y, right_y, ball_x, ball_y, lscore, number_data, rscore)
begin

        number_address(2 downto 0) <= line(2 downto 0);

        -- Display Background Color
        next_COLOR <= "00";

        -- Display the playing field top bar
                if ( line = wall_top  ) then
                                next_COLOR <= "11";
```

```
        end if;


-- Display the playing field bottom bar
        if ( line = wall_bottom ) then
                next_COLOR <= "11";
        end if;


-- Display the left Paddle:
        if (( pixel = left_x -1)  ) then
            if ( (line >= left_y ) and (line <= (left_y + paddle_height)) ) then
                next_COLOR <= "11";
            end if;
        end if;


-- Display the right Paddle:
        if ( pixel = right_x + 1 ) then
            if ( (line >= right_y) and (line <= (right_y + paddle_height)) ) then
                next_COLOR <= "11";
            end if;
        end if;


-- Display the Ball:
        if ( (pixel = ball_x) ) then
            if ( line = ball_y ) then
```

```
                    next_COLOR <= "11";
                end if;
            end if;

        -- Display the Left Score ( Using Std_Logic_Vectors instead of integers )
        if ( clk = '1' and clk'event) then
            if ((pixel >= "0001000" ) and ( pixel <= "0001011" )) and
                    ((line >= "000000" ) and (line <= "000111" )) then
                        number_address(6 downto 3) <=
CONV_STD_LOGIC_VECTOR(lscore,4);
            elsif ((pixel >= "01000000" ) and ( pixel <= "01000011" )) and
                    ((line >= "000000" ) and (line <= "000111" )) then
                        number_address(6 downto 3) <=
CONV_STD_LOGIC_VECTOR(rscore,4);
            else
                    number_address(6 downto 3) <= "0001";
            end if;
        end if;

            if ((pixel >= "0001000") and (pixel <= "0001011" )) and
                ((line >= "000000") and (line <= "000111" ) ) then
                    case pixel( 1 downto 0 ) is
                        when "00" =>
                            if ( number_data(3) = '1' ) then
                                next_COLOR <= "10";
```

```
                    end if;
                when "01" =>
                    if ( number_data(2) = '1' ) then
                        next_COLOR <= "10";
                    end if;
                when "10" =>
                    if ( number_data(1) = '1' ) then
                        next_COLOR <= "10";
                    end if;
                when "11" =>
                    if ( number_data(0) = '1' ) then
                        next_COLOR <= "10";
                    end if;
                when others => NULL;
            end case;
        end if;

-- Display the Right Score

        if ((pixel >= "01000000") and (pixel <= "01000011" )) and
            ((line >= "000000") and (line <= "000111")) then
              case pixel(1 downto 0) is
                when "00" =>
                    if (number_data(3) = '1') then
```

```
                                    next_COLOR <= "10";
                                end if;
                            when "01" =>
                                if (number_data(2) = '1') then
                                    next_COLOR <= "10";
                                end if;
                            when "10" =>
                                if (number_data(1) = '1') then
                                    next_COLOR <= "10";
                                end if;
                            when "11" =>
                                if (number_data(0) = '1') then
                                    next_COLOR <= "10";
                                end if;
                            when others => NULL;
                        end case;
                end if;

end process;

-- Game play logic
moving_paddles: process (VCLK, reset)
  begin
        if (reset = '1') then
```

```
        left_y <= "001001";
        right_y <= "001001";
        nextleft_y <= "001001";
        nextright_y <= "001001";
    elsif (VCLK = '1' and VCLK'event) then

        if (left_dir = "10") then
            nextleft_y <= nextleft_y + 1;  -- move up
        elsif (left_dir = "01") then
            nextleft_y <= nextleft_y - 1;   -- move down
        else
            nextleft_y <= nextleft_y;                -- don't move
        end if;

        if (right_dir = "10") then
            nextright_y <= nextright_y + 1;      -- move up
        elsif (right_dir = "01") then
            nextright_y <= nextright_y - 1;      -- move down
        else
            nextright_y <= nextright_y;          -- don't move
        end if;

        if (nextleft_y < 9) then
            left_y <= "001001";                          -- stop at top of screen
```

```
                nextleft_y <= "001001";
            elsif(nextleft_y > 50) then
                left_y <= "110010";                    -- stop at bottom of screen
                nextleft_y <= "110010";
            else
                left_y <= nextleft_y;
            end if;

            if ( nextright_y < 9 ) then
                right_y <= "001001";                   -- stop at top of screen
                nextright_y <= "001001";
            elsif( nextright_y > 50 ) then
                right_y <= "110010";                   -- stop at bottom of screen
                nextright_y <= "110010";
            else
                right_y <= nextright_y;
            end if;
        end if;
end process;

moving_ball: process (VCLK, ball_xdir, ball_ydir, ball_x, ball_y, reset)
begin

        if (reset = '1') then
```

```
            ball_x <= left_wall;
            ball_y <= 32;
            ball_yrate <= 1;
            lscore <= 0;
            rscore <= 0;
            enable <= '0';
            delay <= "000";
        elsif (VCLK = '1' and VCLK'event) then
          if (delay >= "100") then      --This value may be increased or decreased to adjust the speed of the ball
             delay <= "000";
                if (SERVE = '1') then
                    if (enable = '0' ) then
                        ball_yrate <= 0;
                        ball_y <= 16;
                    end if;
                    ball_y <= 32;
                    enable <= '1';
                end if;

                if (ball_xdir = '1') then                    -- Horizontal Movement ( 1 = right )
                    if (enable = '1') then
                        ball_x <= ball_x + 1;
                    end if;
```

```
-- check for hit on upper 1/4 of right paddle
if  ((ball_x = right_x) and ( ball_y >= right_y - ball_height ) and ( ball_y < right_y + paddle_heighta) ) then
    ball_xdir <= '0';
    -- if ball is going down
    if ( ball_ydir = '0' ) then
        if ( ball_yrate > 0 ) then
            ball_yrate <= ball_yrate - 1;
        else
            ball_yrate <= 1;
            ball_ydir <= '1';
        end if;

    -- if ball is going up
    else
        if ( ball_yrate < 2 ) then
            ball_yrate <= ball_yrate + 1;
        else
            ball_yrate <= 2;
        end if;
    end if;

elsif  ((ball_x = right_x) and ( ball_y >= right_y + paddle_heighta) and ( ball_y < right_y + paddle_heightb) ) then
```

```
                ball_xdir <= '0';


            -- check for hit on lower half of right paddle
            elsif  ((ball_x = right_x) and ( ball_y >= right_y + paddle_heightb) and ( ball_y <= right_y + paddle_heightc) ) then
                ball_xdir <= '0';
            -- if ball is going down
            if (ball_ydir = '0') then
                if (ball_yrate < 2) then
                    ball_yrate <= ball_yrate + 1;
                else
                    ball_yrate <= 2;
                end if;
            -- if ball is going up
            else
                if (ball_yrate > 0) then
                    ball_yrate <= ball_yrate - 1;
                else
                    ball_ydir <= '0';
                    ball_yrate <= 1;
                end if;
            end if;


            -- Score for left team
            else
```

```
                if (ball_x = right_wall) then
                    ball_xdir <= '0';
                    if (enable = '1') then
                        if (lscore = 9) then
                            lscore <= 0;
                        else
                            lscore <= lscore + 1;
                        end if;
                    end if;
                    enable <= '0';
                end if;
            end if;

        -- Ball going left
        else
            -- in middle of playing field
            if ( enable = '1' ) then
                ball_x <= ball_x - 1;
            end if;

            if ( ball_x = left_x ) then

            -- upper portion of paddle
                if (( ball_y >= left_y ) and ( ball_y < left_y + paddle_heighta
)) then
```

```
                ball_xdir <= '1';
                -- if ball is going down
                if ( ball_ydir = '0' ) then
                        if ( ball_yrate > 0 ) then
                                ball_yrate <= ball_yrate - 1;
                        else
                                ball_ydir <= '1';
                                ball_yrate <= 1;
                        end if;
                -- if ball is going up
                else
                        if ( ball_yrate < 2 ) then
                                ball_yrate <= ball_yrate + 1;
                        else
                                ball_yrate <= 2;
                        end if;
                end if;


        -- lower portion of paddle
        elsif  ( ( ball_y >= left_y + paddle_heightb) and ( ball_y <=
left_y + paddle_heightc) ) then
                ball_xdir <= '1';
                -- if ball is going down
                if ( ball_ydir = '0' ) then
                        if ( ball_yrate < 2 ) then
```

```
                    ball_yrate <= ball_yrate + 1;
                else
                    ball_yrate <= 2;
                end if;
            -- if ball is going up
            else
                if ( ball_yrate > 0 ) then
                    ball_yrate <= ball_yrate - 1;
                else
                    ball_ydir <= '0';
                    ball_yrate <= 1;
                end if;
            end if;
        elsif (( ball_y >= left_y + paddle_heighta) and ( ball_y < left_y + paddle_heightb)) then
            ball_xdir <= '1';
        end if;

    -- Score for right team
    else
        if ( ball_x = left_wall ) then
            ball_xdir <= '1';
            if ( enable = '1' ) then
                if ( rscore = 9 ) then
                    rscore <= 0;
```

```
                                    else
                                        rscore <= rscore + 1;
                                    end if;
                                end if;
                            enable <= '0';
                            end if;
                        end if;
                    end if;

                    -- Vertical Movement ( 1 = up )
                    if ( ball_ydir = '1' ) then
                        if (ball_y <= wall_top) then
                            ball_ydir <= '0';
                        else
                            ball_y <= ball_y - ball_yrate;
                        end if;
                    else
                        if (ball_y >= wall_bottom) then
                            ball_ydir <= '1';
                        else
                            ball_y <= ball_y + ball_yrate;
                        end if;
                    end if;
                else
```

```
                delay <= delay + '1';
                ball_y <= ball_y;
                ball_yrate <= ball_yrate;
                ball_x <= ball_x;
                ball_ydir <= ball_ydir;
                ball_xdir <= ball_xdir;
        end if;
  end if;
end process;


end static_display;
```

Anexo IV

Modelo P1.

CENTRO DE EDUCACIÓN SUPERIOR Universidad Central "Marta Abreu" de Las Villas		**PLAN CALENDARIO DE LA ASIGNATURA** **P1** Diseño Digital VLSI	
FACULTAD: Ingeniería Eléctrica		DEPARTAMENTO: Telecomunicaciones	
ESPECIALIDAD: Telecomunicaciones y Electrónica		DISCIPLINA: Electrónica	
AÑO: 3ero	TIPO DE CURSO: Diurno	CURSO ACADÉMICO 2013-2014	SEMESTRE: 2do
ELABORADO POR:		JEFE DE	FECHA

Dr. Juan Pablo Barrios Rodríguez.	DEPARTAMENTO: Dr. Vitalio Alfonso	D	M	A
CATEGORÍA DOCENTE Profesor Titular	FIRMA:	17	1	14

Sem Nº	Act. Nº	Contenido	Forma de docencia	Observaciones
1	1.	Conferencia 1. Introducción a los circuitos VLSI. Clasificación General. Comparación entre lógica cableada y programable. Comparación entre ASICs, CPLDs y FPGAs.		
2	2.	Conferencia 2. Estrategia de Diseño General con circuitos VLSI del tipo CPLDs y FPGAs.		
3	3.	Seminario 1. Características actuales del diseño digital VLSI. Ejemplos de aplicaciones.		Búsqueda en Internet
4	*4.*	Conferencia 3. Los lenguajes HDL en el diseño VLSI (generalidades del VHDL y el Verilog).		
5	5.	Conferencia 4. Familia de chips VLSI de Altera.		
6	6.	Conferencia 5. Familia de chips VLSI de Xilinx		
7	7.	Seminario 2. Análisis comparativo entre los dispositivos VLSI de Altera y Xilinx		Búsqueda en Internet
8	8.	Conferencia 6. Introducción a la herramienta de ayuda al diseño ISE Xilinx Foundation (I)		
9	9.	Conferencia 7. Introducción a la herramienta de ayuda al diseño ISE Xilinx Foundation (II)		

11	10.	Laboratorio 1. Diseño y simulación de ejemplos sencillos con ISE Xilinx Foundation		
12	11.	Conferencia 8. Orientación de los proyectos.		
13	12.	Laboratorio 2. Trabajo con proyectos.		
14	13.	Conferencia 9. Características de la tarjeta Digilent Spartan 3E. Programación y ejemplos.		
15	14.	Laboratorio 3. Trabajo con proyectos.		
16	15.	Laboratorio 4. Trabajo con proyectos.		
16	16.	Evaluación 1: Discusión de los Proyectos.		

UNIVERSIDAD CENTRAL "MARTA ABREU" DE LAS VILLAS

FACULTAD DE INGENIERÍA ELÉCTRICA

Departamento de Telecomunicaciones y Electrónica.

1. Distribución:

Sem **Act**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**T**
Conf	1	1		1		1		1	1	1		1		1			9
C Prác																	
Lab											1		1		1	1 +1(Ev)	5
Sem			1				1										2
Eval																	
Total																	32

IMPORTANTE: Los laboratorios deben planificarse en el 224.

2. Profesores: Dr. Juan Pablo Barrios Rodríguez, Ing. Yakdiel Rodríguez Gallo, Erisbel Orozco Crespo.

3. Total de horas: 32

4. Software que utilizará: ISE Xilinx Foundation

5. Estudiantes por computadoras: _4_

6. Bibliografía:

Textos básicos;

- Digital Design with CPLD Applications and VHDL (2000)-DUECK.
- FPGA.Prototyping.by.VHDL.Examples.Xilinx.Spartan.3.Version.Feb.2008 (formato digital).
- Design Warriors Guide to FPGA Design (Clive Maxfield) (formato digital).
- Advanced Xilinx Fpga Design With Ise. (formato digital).
- Programmable Logic Design Quick Start Handbook (Xilinx 2002) (formato digital).

Textos complementarios: Materiales de la Intranet e Internet.

7. Observaciones modificaciones del Plan de Estudio y limitaciones de profesores: