TRABAJO ESPECIAL DE GRADO

SELECCIÓN DE DISPOSITIVOS LÓGICOS PROGRAMABLES DEL TIPO "FIELD PROGRAMMABLE GATE ARRAY" Y DE LA PLATAFORMA DE SOPORTE, PARA SER UTILIZADOS EN LAS ACTIVIDADES DE DOCENCIA, INVESTIGACIÓN Y DESARROLLO DE LA ESCUELA DE INGENIERÍA ELÉCTRICA.

Presentado ante la Ilustre
Universidad Central de Venezuela
por el Br. Bertsch S., Antonio J.
Para optar al título de
Ingeniero Electricista

Caracas, Junio de 2011

TRABAJO ESPECIAL DE GRADO

SELECCIÓN DE DISPOSITIVOS LÓGICOS PROGRAMABLES DEL TIPO "FIELD PROGRAMMABLE GATE ARRAY" Y DE LA PLATAFORMA DE SOPORTE, PARA SER UTILIZADOS EN LAS ACTIVIDADES DE DOCENCIA, INVESTIGACIÓN Y DESARROLLO DE LA ESCUELA DE INGENIERÍA ELÉCTRICA.

Tutor Académico: Prof. Nunes F., Joao L. G.

Presentado ante la Ilustre
Universidad Central de Venezuela
por el Br. Bertsch S., Antonio J.
Para optar al título de
Ingeniero Electricista

Caracas, Junio 2011

CONSTANCIA DE APROBACIÓN

Caracas, 7 de Junio de 2011

Los abajo firmantes, miembros del jurado designado por Consejo de Escuela de Ingeniería Eléctrica, para evaluar el trabajo Especial de Grado presentado por el Bachiller Bertsch S., Antonio J.

"SELECCIÓN DE DISPOSITIVOS LÓGICOS PROGRAMABLES DEL TIPO FIELD PROGRAMMABLE GATE ARRAY Y DE LA PLATAFORMA DE SOPORTE, PARA SER UTILIZADOS EN LAS ACTIVIDADES DE DOCENCIA, INVESTIGACIÓN Y DESARROLLO DE LA ESCUELA DE INGENIERÍA ELÉCTRICA".

Consideran que el mismo cumple con los requisitos exigidos por el plan de estudios coducente al título de Ingeniero Electricista en la mención de Electrónica, Computación y Control, y sin que ello signifique que se hacen solidarios con las ideas expuestas por el autor, lo declaran APROBADO.

Prof. Pinto Pedro P.
Jurado

Prof. Servando E. Alvarez Ch.
Jurado

Prof. Nunes F., Joao L. G.
Tutor Acádemico

DEDICATORIA

Este trabajo de grado va dedicado a mis padres Carmen Alicia y Antonio jose, a mis hermanas Annalisse, Carolina y Vanessa y a mi novia Imelda Pernia quienes a pesar de observar todas las dificultades a lo largo de mi carrera siempre me dierón animo y nunca cuestionarón que si lo podia lograr.

RECONOCIMIENTOS Y AGRADECIMIENTOS

Quiero expresar mi agradecimiento al Prof. Pedro Pinto, su apoyo y consejo en los momentos criticos a lo largo de mi carrera ha sido crucial y nunca se olvidara, asi mismo deseo expresar mi agradecimiento a mi Tutor y amigo Prof. Joao Nunes quien me dio la oportunidad de realizar el presente trabajo, y quien pacientemente siempre estuvo dispuesto a escuchar y a brindar consejo durante su realización.

Deseo expresar mi agradecimiento a mi amigo Leonardo Araujo por hacer prestamo de equipos, osciloscopio y libros, a mi amigo Javier Lozada en la logistica del traslado de la tarjeta de desarrollo, a mi novia Imelda Pernia que participo no solo emocionalmente brindando su apoyo y amor en todo momento impulsando la realización de este trabajo, sino ademas en toda la logistica que implica su realización.

Mi agradecimiento a mis supervisores Guillermo Villegas y Yahinira Ordoñez de CANTV quienes otorgarón permisos en los momentos criticos de la realización de este trabajo de grado sin realizar cuestionamientos.

Bertsch S., Antonio J.

SELECCIÓN DE DISPOSITIVOS LÓGICOS PROGRAMABLES DEL TIPO "FIELD PROGRAMMABLE GATE ARRAY" Y DE LA PLATAFORMA DE SOPORTE, PARA SER UTILIZADOS EN LAS ACTIVIDADES DE DOCENCIA, INVESTIGACIÓN Y DESARROLLO DE LA ESCUELA DE INGENIERÍA ELÉCTRICA.

Tutor académico Prof. Nunes F., Joao L. G. Tesis. Caracas. U.C.V. Facultad de Ingeniería. Escuela de Ingeniería Eléctrica. Ingeniero Electricista. Opción: Electrónica, Computación y Control. 2011. 102 h. + Anexos

Palabras Claves: FPGA; Selección; Plataforma de desarrollo; Filtro; FIR; Ascensor; Memoria; VHDL**.**

Resumen. El presente trabajo busca seleccionar una plataforma de desarrollo basada en Field Programmable Gate Array (FPGA) con el fin de implementarla en la Escuela de Ingeniería Eléctrica (EIE) de la Universidad Central de Venezuela (UCV) en sus materias afines de pregrado para proyectos en general. Se realizó la selección y se propone la tarjeta de desarrollo Spartan 3E Starter Board con la FPGA XC3S1600E, luego se realizó la implementación de dos experiencias prácticas con dicha tarjeta, en la primera experiencia se realizó una unidad de control para ascensor de 4 pisos con memoria y en la segunda se implemento un filtro digital tipo FIR (Finite Impulse Response) haciendo uso de los periféricos disponibles en la tarjeta. Ambas experiencias prácticas se realizaron haciendo uso del lenguaje de descripción de hardware VHDL.

INDICE GENERAL

INDICE DE TABLAS

INDICE DE FIGURAS

SIGLAS

ADC: Analog Digital Converter (Conversión analógico digital).

ASIC: Application Specific Integrated Circuit (Circuito Integrado de Aplicaciones Especificas).

CAD: Computer Aided Design. (Diseño Asistido por Computador).

CBL: Configurable Logic Block (Bloque lógico Configurable).

CMOS: Complementary Metal–Oxide–Semiconductor.

CPLD: Complex Programmable Logic Device (Dispositivos Lógicos Programables Complejos).

DAC: Digital Analog Converter (Conversión digital analógica).

DCM: Digital Clock Managers (Manejadores de reloj digital).

DLL: Delay-Locked-Loops.

DSP: Digital Signal Processor (Procesamiento digital de señales).

EDA: Electronic Design Automation (Diseño electrónico automatizado).

EIE: Escuela de Ingeniería Eléctrica.

E^2PROM: Electrically Erasable Programmable Read-Only Memory (Memoria Programable de solo lectura eléctricamente borrable).

FIT: Factor Incident Time (factor de fallas en el tiempo).

FIR: Finite Impulse Response (Respuesta finita al impulso).

FPGA: Fielg Programmable Gate Array (Arreglo de compuertas de campo programable).

GA: Gate Array (Arreglo de Compuertas).

GAL: Generic Array Logic (Lógica de Arreglo Genérico).

HDL: Hardware Description Language (Lenguaje de descripción de hardware).

IEEE: Institute of Electrical and Electronics Engineers.

ISP : In System Programmable (Programable en el sistema).

IP: Intelectual Property (Propiedad Intelectual).

I2C: Inter-Integrated Circuit (Circuito Inter-Integrado).

LAB: Logic Array Block.

LC: Logic Cell (Celda Lógica).

LCD: Liquid Chrystal Display (Pantalla de Cristal Líquido).

LE: Logic Element (Elemento Logico).

LED: Light Emitter Diode (Diodo Emisor de Luz).

LUT: Lookup Table (Tablas de Busqueda).

OTP: One-Time Programmable (Programable solo una vez).

PAL: Programmable Array Logic.

PLA: Programmable Logic Array.

PLL: Phase-Locked Loops.

R: Resistencia.

RAM: Random Access Memory (Memoria de Acceso Aleatorio).

ROM: Read Only Memory (Memoria de Sólo Lectura).

SOC: System-on-a-chip (Sistema en un chip).

SPI: Serial Peripheral Interface (Interfaz Periférica Serial).

SPLD: Simple Programmable Logic Device (Dispositivo Lógico Programable Simple).

SRAM: Static Random Access Memory (Memoria de Acceso Aleatorio Estatico).

UART: Universal Asynchronic Receiver Transmitter (Receptor Transmisor Asincrónico Universal).

V: Volt.

VHSIC: Very High Speed Integrated Circuit (Circuitos integrados de muy alta velocidad).

VLSI: Very Large Scale Integration (Escala de integración muy larga).

Ω: Ohm.

INTRODUCCIÓN.

La constante evolución en las ultimas décadas, tanto en el desarrollo de tecnologías de fabricación de circuitos integrados, como en avances informáticos en la creación de nuevas y poderosas herramientas de software especializado y metodologías eficientes de diseño, han permitido la mejora continua de los dispositivos lógicos programables, convirtiéndolos día a día en una alternativa viable para la creación de sistemas a bajo costo, con alta complejidad y con cortos ciclos de desarrollo, que de otra manera serian prácticamente inviables con metodologías de diseño tradicionales discretas, por lo cual se hace evidente la necesidad de involucrar a estudiantes y profesores en el uso de la mismas para poder disfrutar de sus ventajas.

En este trabajo de grado se busca básicamente seleccionar una plataforma de desarrollo basada en Field Programmable Gate Array (FPGA) con el fin de implementarla en la Escuela de Ingeniería Eléctrica (EIE) de la Universidad Central de Venezuela (UCV) en sus materias afines de pregrado para proyectos en general, así mismo se desarrollan dos experiencias practicas haciendo uso de dicha plataforma de desarrollo, para ponerla a prueba y ganar experiencia en el uso de dicha tecnología.

Para llevar a cabo dicho objetivo partimos de una breve reseña histórica en el Capítulo I para introducirnos en cómo surge dicha tecnología, cuáles son sus ventajas y qué problemas resuelve en el ámbito del diseño y prototipado.

Luego en el Capítulo II se realizo un levantamiento de información a modo general de las características relevantes de las FPGAs, su terminología, sus recursos en cuanto a sus características internas y una idea de su funcionamiento, para luego

poder en el Capítulo III tener la capacidad de realizar la selección de la plataforma de desarrollo, de sus periféricos y de su FPGA.

Una vez establecida dicha selección en el Capítulo IV se pone a prueba dicha plataforma realizando para ello dos experiencias practicas utilizando las nuevas herramientas tanto de hardware como de software propuestas, en una primera experiencia se lleva a cabo una practica de diseño digital en la cual se realiza una unidad de control para ascensor de 4 Pisos con Memoria, con el fin de aprender a utilizar estructuras Combinacionales y Secuenciales en el ámbito de un lenguaje de descripción de Hardware. En la experiencia número dos se realiza una práctica que nos introduce al mundo del procesamiento digital de señales, en la cual se realiza la implementación de un filtro FIR (Finite Impulse Response) haciendo uso de los periféricos disponibles en la tarjeta de desarrollo seleccionada.

CAPÍTULO I

RESEÑA HISTÓRICA DE LOS DISPOSITIVOS LÓGICOS PROGRAMABLES

Desde finales de los años 70 se vienen realizando esfuerzos para lograr construir dispositivos que integren muchas compuertas de lógica estándar en un mismo chip.

El primero de estos dispositivos son los llamados "PLA" (Programmable Logic Array), el cual tenia una arquitectura provista de dos planos programables que permitían la implementación de cualquier combinación de compuertas AND y OR (figura 1), esta arquitectura era flexible pero tenia algunos inconvenientes como altos retardos de propagación (Tpd) y bajos niveles de integración (tecnología de 10μm).[4]

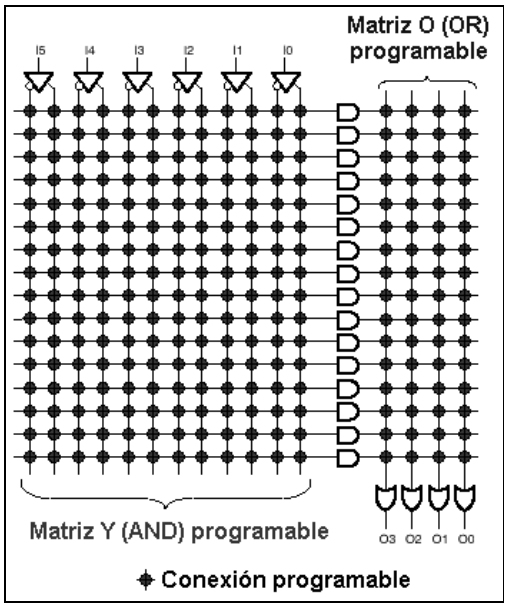


Figura 1. PLA (Programmable Logic Array)

Luego una empresa llamada MMI (Monolithic Memories Incorporated, comprada luego por AMD) en 1978 realizó una modificación de la arquitectura "PLA" fijando uno de los planos programables (el plano OR), lo que dio como resultado los denominados "PAL" (Programmable Array Logic). Este dispositivo logró mejoras respecto a los "PLA" en cuanto a los retardos de propagación y avances con respecto a la complejidad del software de programación (ver figura 2).[4]

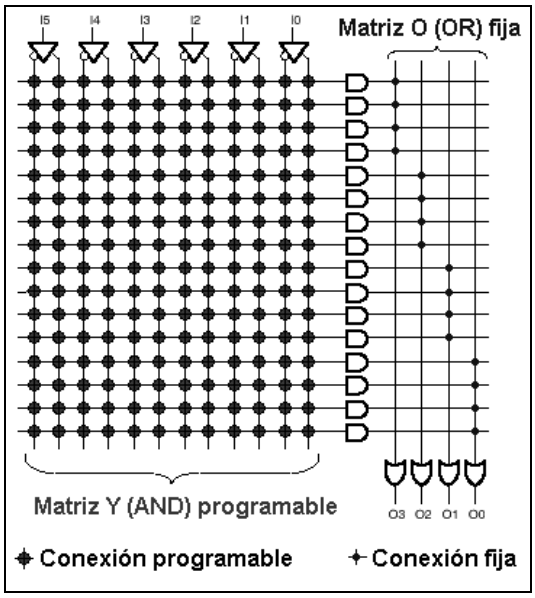


Figura 2. PAL (Programmable Array Logic)

En los años siguientes continuaron las mejoras, con la aparición de los dispositivos GAL (Generic Array Logic). Inventados por la compañía Lattice semiconductor en 1985, este componente tiene las mismas propiedades lógicas y arquitectura de las PAL pero pueden ser borrados y reprogramados, siendo muy útiles en el estudio y diseño de prototipos. Todos estos dispositivos basados en PAL constituyen la familia de los denominados SPLD (Simple Programmable Logic Device), los cuales proveen alrededor de 50 veces más compuertas que un paquete de lógica discreta.[3]-[4]-[5]

Continuando con la búsqueda de integrar más y más dispositivos lógicos y dados los crecientes niveles de integración surgen los CPLD (Complex Programmable Logic Device), los cuales básicamente están compuestos por conjuntos de bloques de SPLD (Ver figura 3) con una serie de dispositivos de interconexión de uso general que permiten la comunicación entre los mismos, pudiéndose implementar en ellos complejas funciones lógicas, además de poder prever fácilmente retardos de propagación en la lógica involucrada. Con la aparición de los CPLD surgieron mejores y potentes herramientas para el diseño CAD (Computer Aided Design), con un enfoque de alto nivel, y algoritmos eficientes de síntesis que permiten el desarrollo de prototipos complejos de una manera más rápida y eficiente. [3]-[4]

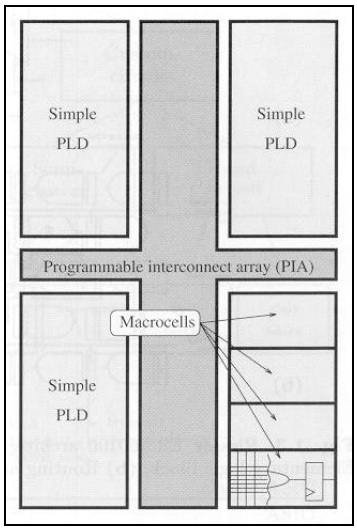


Figura 3. Estructura básica de un CPLD (Extraido de Meyer-Baese, U. Digital signal processing with field programmable gate arrays, 2001)

Todas estas estructuras de arquitectura PLD buscaban un mayor nivel de integración, facilidades a la hora de diseñar prototipos, reprogramación, mejoras en el factor FIT (Failure in Time o Falla en el Tiempo) y principalmente una reducción de costo con respecto a otras tecnologías como lo son los GA (Gate Array) y los ASIC (Application Specific Integrated Circuit), en los cuales el fabricante del circuito

integrado directamente se encarga de conectar bloques de compuertas mediante capas de metal o mascaras en el proceso de fabricación.[4]

En la Figura 4 se puede observar una clasificación de los VLSI (Very Large Scale Integration) en donde se denota el gran campo de acción de los ASIC, nótese que los dispositivos GA son implementados con ASIC.

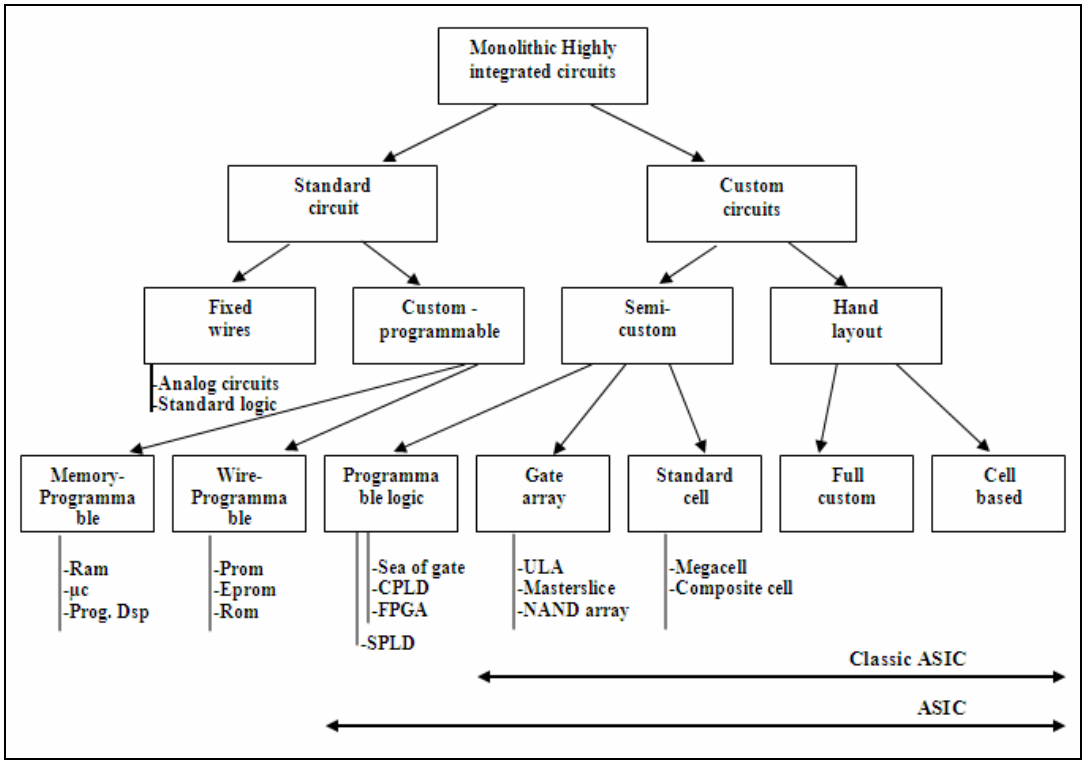


Figura 4. Clasificación de los VLSI.(Extraido de Meyer-Baese, U. Digital signal processing with field programmable gate arrays, 2001)

En general los ASIC implican altos costos debido al desarrollo de una línea de producción dedicada (ver figura 6), y sólo se justifican para grandes volúmenes de producción, por lo cual se generó una brecha a medida que los niveles de integración fueron aumentando acorde con la Ley de Moore (La cual indica que el número de transistores por chip se incrementa al doble cada 18 meses, como se muestra en la figura 5), ya que no se contaban con estructuras de hardware altamente densas y

reprogramables donde fácilmente se pudieran desarrollar y poner a prueba prototipos sin tener que implementar una línea de producción. [2]-[3]

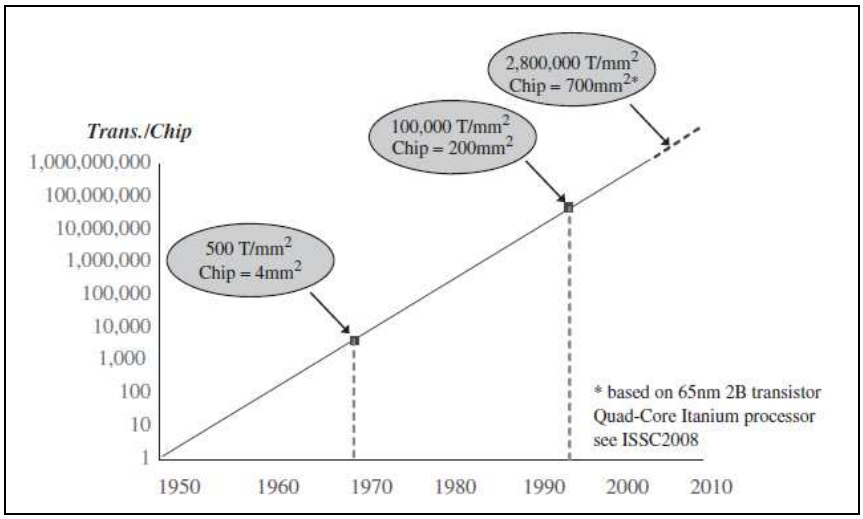


Figura 5. Ley de Moore (1965) (Woods, Roger. FPGA-based Implementation of Signal Processing Systems. 2008)

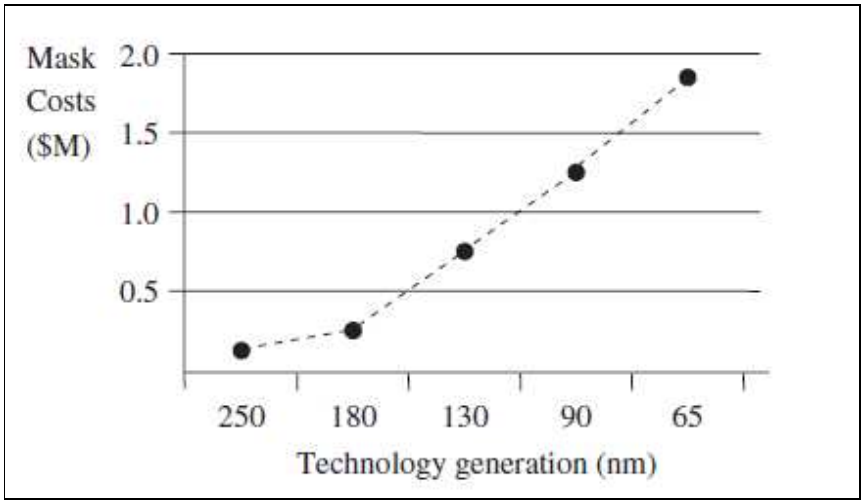


Figura 6. Costos de Mascara ASIC Vs Tecnología de generación (Extraido de Woods, Roger. FPGA-based Implementation of Signal Processing Systems. 2008)

Este conjunto de avances dio lugar al nacimiento de los dispositivos FPGA (Field Programmable Gate Array) en 1985 por la empresa Xilinx, los cuales buscan combinar el control con que cuenta el usuario sobre los dispositivos CPLD con las altas densidades, costos y beneficios de los arreglos lógicos de Compuertas GA.

PLANTEAMIENTO DEL PROBLEMA

Cada día se hace más complicado elaborar en nuestra escuela prácticas, proyectos y prototipos con estructuras de hardware discreto ya que los mismos son descontinuados en su mayoría, se requieren nuevas estructuras de hardware y software que permitan realizar diseños con mayor complejidad y alta densidad.

Existen una serie de características que presentan el uso de los dispositivos FPGA en el diseño digital que no podemos dejar pasar desapercibidas y que ya han sido adoptadas en algunas de las universidades a nivel nacional y en gran parte de las universidades a nivel internacional para adaptarse a los crecientes niveles de integración y nuevas herramientas de diseño existentes en la actualidad, las FPGAs nos ofrecen entre otros:

- Potentes herramientas CAD-EDA para diseño, simulación, síntesis y programación de los dispositivos.
- Facilidades para diseñar sistemas complejos mediante el uso de algoritmos de síntesis asociados a lenguajes de descripción de hardware en alto nivel (por ejemplo Verilog o VHDL), ya que cuando el diseño de un circuito alcanza tamaños en el orden de 10.000 compuertas el uso de diagramas esquemáticos tradicionales no es práctico. Lenguajes de descripción de Hardware (HDL) hacen manejable proyectos de alto grado de complejidad.
- Ciclos más cortos de desarrollo de prototipos.
- Bajos costos de desarrollo, debido a mejores herramientas de diseño y la facilidad de poner a prueba prototipos sin tener que implementar un ASIC directamente.
- Altos niveles de integración, es decir, alta densidad de dispositivos lógicos por unidad de área.

- Reprogramabilidad. En un mismo dispositivo FPGA se pueden implementar ilimitados diseños (hardware virtual) sin realizar cambio físico alguno.
- Mejoras en el FIT (factor de fallas en el tiempo), menos partes involucradas en un diseño permiten una mayor calidad y ser menos propenso a fallas en el tiempo.
- Reducción de costos por espacio e inventario. No se requiere un stock de componentes discretos de diversas funciones lógicas y lo concerniente a su almacenamiento ya que todas las partes lógicas pueden ser implementadas en la misma FPGA.
- Kit de desarrollos que integran en PCB, la FPGA y dispositivos de uso común (conversores A/D, D/A, memorias, microcontroladores, reloj, displays, módulos E/S, codecs, etc.).
- Uso de librerías llamadas IP CORE (Intellectual Property Core), las cuales implementan funciones altamente complejas previamente programadas y ya probadas que se pueden integrar fácil y modularmente por el diseñador.
- Fácil transición a la implementación en dispositivos ASIC. El código HDL es un lenguaje Standard de los fabricantes de ASIC, por lo que luego del diseño, simulación y puesta a prueba en FPGA facilita su implementación en dispositivos arquitectura similar.

En el presente proyecto se busca realizar un estudio de los distintos dispositivos FPGA producidos en el mercado, sus kits de desarrollo, herramientas, características, versatilidad y costos con el fin de disponer de alguno de ellos e implementarlo en futuros proyectos y prácticas de laboratorio en la Escuela de Ingeniería Eléctrica (EIE) de la UCV y familiarizar a estudiantes y profesores en su uso, el cual eleva a un nivel superior el conjunto de herramientas de diseño digital de que se dispone, proporcionando una nueva plataforma de desarrollo que permitiría realizar diseños complejos no limitados físicamente por hardware.

JUSTIFICACIÓN

En asignaturas del Pensum de Estudios de la EIE de la UCV, tales como Sistemas Digitales I, Microprocesadores I y II, y proyectos en general, su aplicación a nivel académico puede ser muy útil, ya que los dispositivos FPGA permiten implementar desde sistemas muy simples, como por ejemplo circuitos lógicos combinacionales, hasta sistemas de gran complejidad, como podrían ser núcleos de microprocesadores, aplicaciones DSP o en general un SOC (System-on-a-chip). Dadas sus características de reprogramabilidad (Basada en RAM), estos equipos actúan como un protoboard o mesa de trabajo virtual con la gran ventaja de que tanto el estudiante como la Escuela no tienen que adquirir componentes discretos para la realización de prácticas, lo que le permite experimentar libre y espontáneamente. Además, cabe destacar el ahorro económico que implica su reutilización constante.

El estudio propuesto en este trabajo con miras a desarrollar conocimientos en este tipo de tecnología permitiría a los Estudiantes y al Personal Docente de la Escuela familiarizarse con nuevas herramientas y posibilidades, que elevan a un nuevo nivel el diseño digital y proveen una plataforma de desarrollo que daría paso a innumerables estudios y proyectos de desarrollo en nuestra Escuela.

OBJETIVO GENERAL

Realizar un estudio de las alternativas existentes en el mercado de los dispositivos lógicos programables de tipo “Field Programmable Gate Array” FPGA, para su adquisición e implementación en asignaturas y proyectos en la Escuela de Ingeniería Eléctrica de la UCV.

OBJETIVOS ESPECÍFICOS

A fin de lograr el objetivo general planteado se describen los siguientes objetivos específicos:

- Realizar una investigación sobre los requerimientos de la Escuela de Ingeniería Eléctrica de la Universidad Central de Venezuela con miras a la adquisición de equipos y herramientas de desarrollo basadas en dispositivos FPGA.
- Realizar una investigación que contemple la disponibilidad de equipos existentes ofrecidos por los fabricantes y distribuidores de herramientas de desarrollo basadas en FPGA a nivel nacional e internacional.
- Seleccionar una herramienta o kit de desarrollo que se adapte a los requerimientos de las asignaturas afines a la implementación de proyectos con dispositivos FPGA dictadas en la Escuela de Ingeniería Eléctrica de la UCV.
- Diseñar prácticas de materias afines en donde se utilicen los dispositivos mencionados.
- Verificar el funcionamiento de las prácticas diseñadas.

METODOLOGÍA

A modo general, para la realización del presente Trabajo Especial de Grado se dividió la metodología de trabajo en dos partes. La primera de ellas abarcó una investigación y selección de una plataforma de desarrollo, mientras que la otra básicamente fue dedicada al diseño, simulación e implementación de prácticas demostrativas que corroboraran la factibilidad de la realización de proyectos y prácticas de laboratorio en la Escuela de Ingeniería Eléctrica de la Universidad Central de Venezuela.

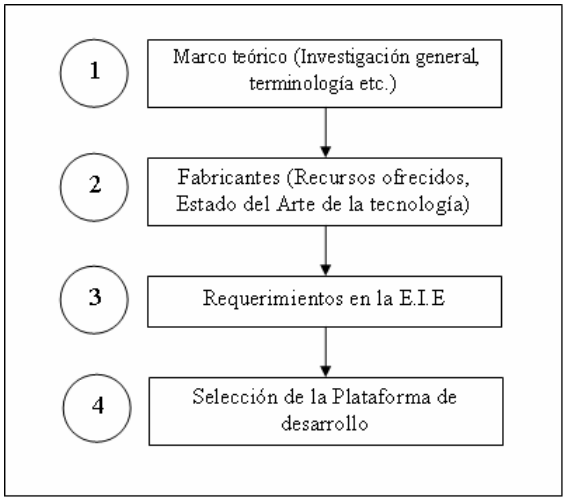


Figura 7. Metodología para la selección de la plataforma de desarrollo.

Lo primero que se llevó a cabo fue una investigación general sobre la tecnología FPGA, sus clasificaciones, terminología, ciclo de diseño entre otros. Una vez realizada esta investigación, se procedió a observar a grandes rasgos los productos ofrecidos por los principales fabricantes en el mercado, con la finalidad de tener una perspectiva de qué existe en la actualidad y cuál es el estado del arte de la

tecnología. Así mismo, se investigó sobre el uso de dicha tecnología en diversas universidades a nivel nacional e internacional. Una vez establecido dicho perfil sobre la tecnología y sus fabricantes, se procedió a determinar cuáles son los requerimientos en la EIE para asignaturas de pre-grado en las áreas que estén mayormente involucradas con la aplicación de esta tecnología. Por último, se realizó la selección de una plataforma de desarrollo acorde con dichos requerimientos.(Ver figura 7)

En la segunda parte de este trabajo se llevaron a cabo los siguientes pasos para poder llevar desarrollar las experiencias demostrativas del uso de dicha tecnología. En la figura 8 se puede observar un resumen de los mismos resumido en tres pasos.

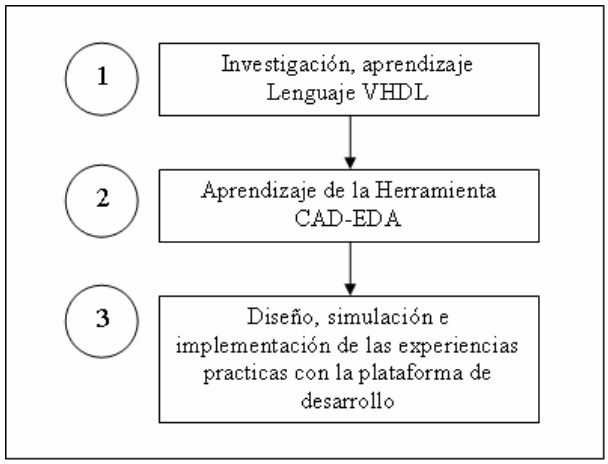


Figura 8. Metodología en la realización de experiencias prácticas.

CAPÍTULO II

1. GENERALIDADES, CARACTERÍSTICAS Y CONCEPTOS BÁSICOS DE LOS DISPOSITIVOS FPGA.

En general, una FPGA está formada por una estructura que ofrece internamente un gran número de bloques lógicos que contienen lógica combinacional programable e independiente con una serie de registros para fijar datos de entrada o salida, mientras que en su periferia existen a su vez un conjunto de bloques que se pueden configurar como entrada/salida o alta impedancia. [5]

Su arquitectura, disposición y/o constitución de todos los bloques lógicos, registros y bloques de entrada/salida varía según cada fabricante, aunque sin embargo estos bloques se pueden interconectar mediante programación para implementar virtualmente cualquier circuito o hardware, dando un completo control al diseñador para realizar cambios durante el desarrollo de proyectos. [3]-[5]

Cualquier circuito de aplicación específica (ASIC) puede ser implementado en una FPGA, siempre y cuando se disponga de los recursos necesarios. Las aplicaciones donde más comúnmente se utilizan los FPGA, incluyen el amplio campo de los DSP (Digital Signal Processor) en sus diversas áreas, como por ejemplo en el campo de las comunicaciones en celulares, a nivel de redes de área local o de banda ancha, enrutadores inalámbricos, aplicaciones de procesamiento de imágenes, procesamiento de sonido, reconocimiento de voz, Transformada Rápida de Fourier (FFT), Filtros Digitales, estimación espectral, en sistemas de control discreto, controladores, periféricos de PCs, codificadores/decodificadores, microprocesadores a la medida, creación de prototipos de circuitos VLSI entre tantos otros. [1]-[2]

Cabe destacar que tradicionalmente se ha hecho énfasis en el uso de microprocesadores y/o microcontroladores en donde el hardware es fijo y todo el esfuerzo es concentrado en el desarrollo de código que hace que el hardware trabaje bajo las especificaciones requeridas del sistema. En este tipo de arquitecturas las operaciones son procesadas secuencialmente lo cual facilita el desarrollo y diseño de software de programación, pero en aplicaciones DSP de gran demanda hoy en día esta característica limita severamente el desempeño, sobre todo en aplicaciones que requieran altos niveles de paralelismo y en los que demanden procesamiento y operaciones con datos de forma independiente. Por tales motivos, un procesador DSP con arquitectura fija no es adecuado para ciertas aplicaciones, y de nuevo entra la importancia de los dispositivos FPGA que puedan proporcionar un hardware completamente dedicado y hecho a la medida. [2]

2. CLASIFICACIÓN DE LAS FPGAS.

Las FPGAs se pueden clasificar de varias maneras, ya sea según su granularidad y según el tipo de tecnología de la que están fabricadas sus celdas lógicas, tal y como se puede ver en el esquema de la figura 9. [2]-[3]

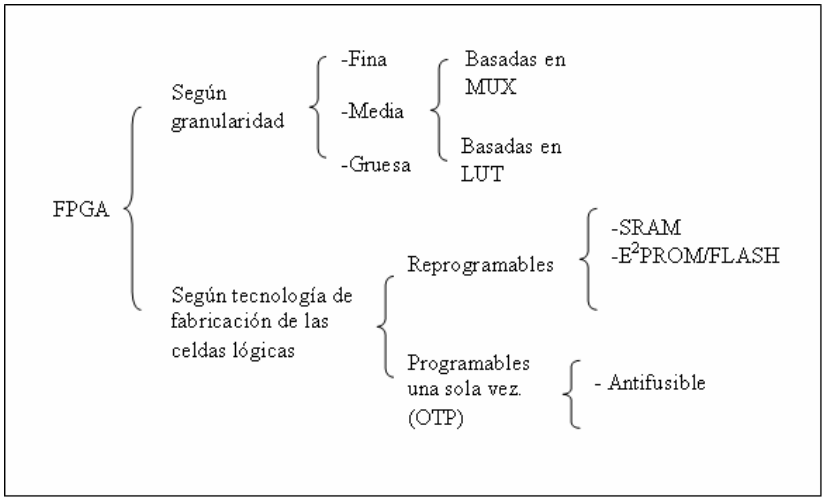


Figura 9. Clasificaciones de las FPGAs

2.1. Clasificación según la granularidad.

Como ya se comentó en las FPGAs, su estructura interna está formada por una red de bloques lógicos interconectados. La granularidad en este tipo de dispositivos se refiere a que tan complejos son estos bloques lógicos constituyentes. Una granularidad fina indica que sus elementos constitutivos están formados por elementos simples de lógica primitiva, (ejemplo compuertas NAND) y por ende cada uno puede implementar funciones lógicas muy simples. A medida que se necesite mayor complejidad se deben realizar un gran número de interconexiones para poder interactuar con otros bloques lógicos y así poder implementar la aplicación deseada. Una granularidad media implica que cada bloque lógico puede implementar por sí mismo un nivel de lógica intermedia (como por ejemplo una FPGA común cuenta con 4 LUT de 4 entradas, 4 Mux, 4 F/F tipo D, y lógica para acarreos). Una granularidad gruesa indica que cada bloque lógico tiene capacidad para implementar funciones u algoritmos complejos (destacan la FFT o alguna otra operación DSP) actuando como nodos independientes interconectados, como se muestra en la figura 10.

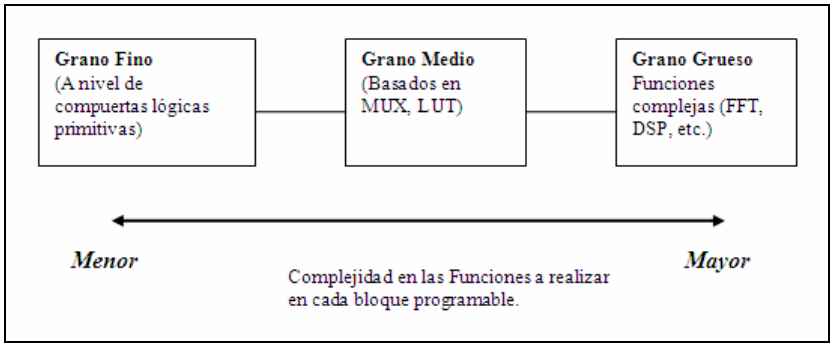


Figura 10. Clasificación según granularidad.

En las FPGAs de granularidad media existe una sub-clasificación de acuerdo a la forma en que ellas implementan una función lógica:

2.1.1. FPGA basadas en “Mux” (Multiplexores)

En este tipo de estructuras los bloques lógicos constitutivos están formados por multiplexores los cuales se basan en que mediante los mismos se pueden implementar funciones lógicas arbitrarias. En el ejemplo de la figura 11 veremos la misma función lógica anterior implementada con multiplexores. [3]

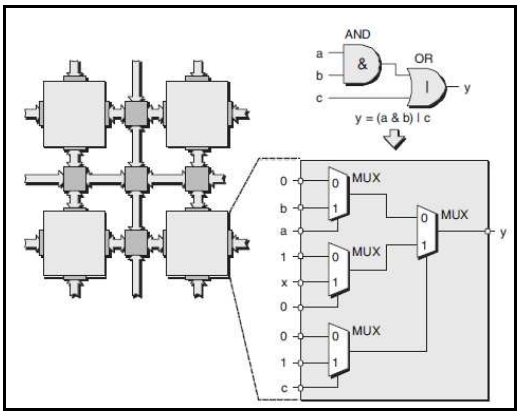


Figura 11. Bloque lógico basado en multiplexor. (Extraido de Maxfield, Clive. The design warrior´s guide to fpga´s, 2004)

2.1.2. FPGA basadas el “LUT” (Lookup Table)

Este tipo de bloque lógico está basado como su nombre lo indica en tablas de búsqueda similares a las utilizadas en memorias. Un grupo de señales de entrada son usadas como puntero para realizar dichas búsquedas [3]. En la figura 12 se observa la estructura de un LUT basado en compuertas bilaterales y sus celdas SRAM asociadas implementando una función lógica arbitraria.

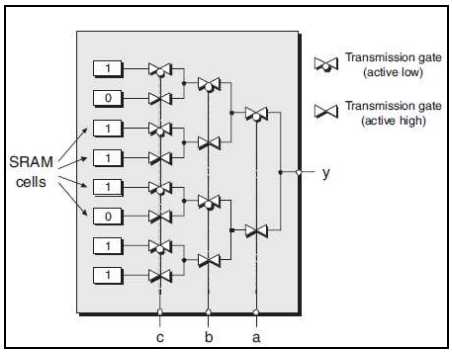


Figura 12. Estructura de un LUT basado en compuertas bilaterales. (Extraido de Maxfield, Clive. The design warrior´s guide to fpga´s, 2004)

Como se puede apreciar, dando valores a las entradas (A, B, C) obtendríamos a la salida "Y" la función lógica combinacional deseada, siempre y cuando los valores preprogramados en las celdas SRAM sean los correctos. [3] Cabe destacar que en este caso se asumió que las celdas son SRAM, aunque la misma idea aplica si las celdas fuesen de cualquier tecnología (anti-fusible, E^2prom/Flash, etc.) [3]

En la actualidad las FPGA están basadas en el uso de LUT debido a que presentan mejor desempeño en aplicaciones de procesamiento aritmético, mejor propagación de acarreos y mayor velocidad de respuesta que su contra parte basada en Mux. [3] El número de entradas de un LUT generalmente está establecido entre 3 y 6 siendo el standard 4 en la mayoría de los dispositivos FPGA, aunque recientemente desde el 2008 los principales fabricantes están haciendo uso en sus dispositivos de Lut de 6 entradas. [2]

Otra gran ventaja de trabajar con LUTs es que como su núcleo está basado en celdas SRAM, un mismo LUT puede ser utilizado como un pequeño bloque de RAM o como un registro de cambio (Shift Register) por lo que un LUT puede ser considerado multifacético (Ver figura 13). No obstante, la herramienta de síntesis según el fabricante sacará provecho de estas características a la hora de hacer la implementación en la FPGA. [3]

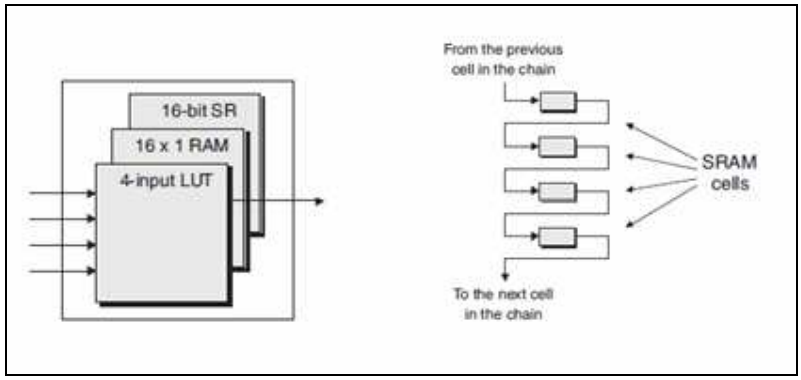


Figura 13. Multifacetas del LUT. (Extraido de Maxfield, Clive. The design warrior´s guide to fpga´s, 2004)

2.2. Clasificación según la Tecnología de fabricación de las celdas lógicas.

Las FPGAs se pueden clasificar según el tipo de tecnología en la cual estén fabricadas sus celdas lógicas, las cuales pueden ser de tipo SRAM (Static Random Access Memory), E^2PROM/FLASH (Electrically Erasable Programmable Read-Only Memory) y Antifusibles. Luego según lo antes comentado, estas se pueden subclasificar en reprogramables y programables una sola vez (conocido como OTP, One-Time Programmable). [1]-[3]-[4]

2.2.1. Dispositivos basados en tecnología SRAM

La gran mayoría de los dispositivos FPGA están basados en el uso de celdas configurables de tecnología SRAM, las cuales hacen uso de la tecnología CMOS (Complementary Metal–Oxide–Semiconductor) común en Memorias, las cuales pueden ser reprogramadas una y otra vez, siendo esta su principal ventaja, sobre todo en el campo de diseño y prototipado. Otra utilidad interesante a nivel práctico es que en el inicio pueden ser programadas para cumplir una función tal como auto-test o prueba del sistema usando el mismo hardware, para luego ser reprogramadas y cumplir su principal tarea encomendada. [1]-[3]-[4]

La desventaja de esta tecnología es que cada vez que el sistema es encendido se debe reconfigurar nuevamente por ser volátil, por lo que es requerida una memoria externa donde se almacene la configuración a cargar y se necesita algo de tiempo en cada reprogramación (algunos milisegundos). Estos dispositivos pueden ser programados fuera de línea haciendo uso de un programador o pueden incluir versiones ISP (In System Programmable). [3]

Otra consideración a tomar en cuenta en los dispositivos basados en SRAM es que es más difícil proteger la propiedad intelectual o IP (Intelectual Property), dado que el archivo de configuración está guardado en una memoria externa y es transferido en el arranque. [3]

2.2.2. Dispositivos basados en tecnología Anti-fusible.

A diferencia de los dispositivos SRAM, los cuales son programados mientras el archivo de configuración es residente en el sistema, en los dispositivos basados en tecnología anti-fusible una vez realizada la programación, el archivo de configuración es no volátil, lo cual indica que el sistema estará inmediatamente disponible en el arranque o encendido del mismo. [3]

La tecnología anti-fusible emplea una delgada barrera de silicio amorfo entre dos conductores metálicos. Cuando un voltaje suficientemente alto se aplica a través del silicio amorfo éste se convierte en una aleación de silicio policristalino de metal que posee una baja resistencia, y establece una unión permanente. [2]

Dentro de las ventajas que presenta este tipo de tecnología se destaca el no requerir de una memoria externa como en el caso de la tecnología SRAM para guardar el archivo de configuración, ahorrando costos y espacio en la tarjeta. Por otra parte, una notable ventaja de esta tecnología sobre las demás es que las celdas en tecnología anti-fusible están interconectadas de modo natural (Rad Hard) en hardware

haciéndolas relativamente inmunes a los efectos de la radiación, lo cual es de particular interés en aplicaciones militares y aeroespaciales. Por otro lado es prácticamente imposible realizar ingeniería inversa a una FPGA con tecnología anti-fusible ya que el archivo de configuración está auto contenido, luego de la programación haciendo uso de un anti-fusible especial que sella la extracción de datos de la misma. [3]

Su principal desventaja, es que soló puede ser programada una única vez, lo cual implica que no puede recibir actualizaciones en el caso de un bug (error), ni realizar tareas de auto-test o reconfiguración. [3]

2.2.3. Dispositivos basados en tecnología E^2PROM/FLASH

Los dispositivos FPGA basados en E^2PROM ó FLASH son similares a su contraparte SRAM en que las celdas de configuración son conectadas juntas a lo largo de un gran Shift Register (Registro de cambio). Estos dispositivos pueden ser programados fuera de línea haciendo uso de un programador, o algunas incluyen versiones ISP, pero su tiempo de programación es alrededor de 3 veces mayor que el de una FPGA basada en SRAM. [1]- [3]

Las FPGAs basadas en esta tecnología no requieren de dispositivos de memoria externos para almacenar el archivo de configuración. Una vez programadas son no volátiles, lo cual indica que pueden trabajar directamente al instante de ser energizadas. Por otro lado una vez programadas, al estar auto contenido el archivo de configuración son más seguras en términos de ingeniería inversa y protección del IP. [1]- [3]

2.2.4. Dispositivos Hibridos Flash-SRAM.

En estos dispositivos cada elemento de configuración está formado por una combinación de una celda Flash (E^2PROM) y una celda SRAM. Las primeras pueden

ser programadas y proporcionan la no volatilidad, ya que al momento de encender el sistema el contenido de todas las celdas Flash es copiado masivamente en forma paralela a su correspondiente celda SRAM asociada.

En este tipo de dispositivos se busca obtener la no volatilidad de los bloques Anti-fusible (lo cual indica que el sistema estará disponible inmediatamente al encender el equipo), pero a diferencia de los mismos las celdas SRAM permiten que sean reconfigurables mientras el archivo de configuración es residente aun en el sistema con el sistema encendido. De igual manera pueden ser reprogramados usando las celdas flash vía programador. [3]

A continuación se presentan las principales características de cada tecnología en la siguiente tabla comparativa:

Tabla 1. Tabla resumen comparativa de las diferentes tecnologías de fabricación de las FPGAs.

Característica	**SRAM**	**Antifusible**	**E^2PROM/FLASH**
Reprogramable	SI (en sistema ISP)	NO	SI (en sistema ISP ó fuera de línea)
Velocidad de programación	Rápida	---	3 veces mas lenta que SRAM
Volatilidad	SI	NO	NO (puede aplicar si se requiere)
Buena para prototipado	SI (Muy Buena)	NO	SI (Regular)
Encendido al Instante	NO	SI	SI
Seguridad IP	Aceptable (usando encriptación en Bitstream en Mem. Ext)	Muy Buena	Muy Buena

Tamaño celda de configuración	Grande	Muy pequeño	Mediano
Consumo de potencia	Medio	Bajo	Medio
Protección Radiación (Rad Hard)	NO	SI	No realmente

3. TERMINOLOGÍA BÁSICA EN LA ARQUITECTURA DE LOS DISPOSITIVOS FPGA.

A continuación se expondrán algunos conceptos básicos referentes a la arquitectura interna de las FPGA. Dichos conceptos son generales pero ayudan a tener una visión sobre cómo es la estructura y su forma de funcionamiento así como que se debe poder realizar. Cabe aclarar que según el fabricante la terminología cambia pero la idea fundamental es la misma. A lo largo de esta aplicación se usará terminología del fabricante Xilinx y se hará referencia a terminología equivalente para Altera. Xilinx (inventor de esta tecnología en 1985) y Altera son los principales fabricantes de dicha tecnología.

3.1. Celda lógica, LC (Logic Cell terminología Xilinx)

Una celda lógica básicamente comprende un LUT (el cual como se indico puede actuar como un bloque RAM 16x1 o como un shift register de 16 Bit), un multiplexor y un registro o flip-flop, tal y como se muestra en la figura 14. [3]

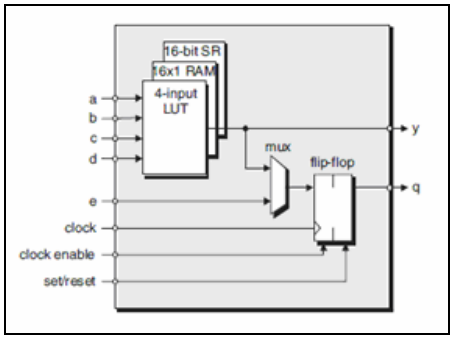


Figura 14. Estructura de una celda logica (Extraido de Maxfield, Clive. The design warrior´s guide to fpga´s, 2004)

El registro puede actuar como flip-flop o como Latch, la polaridad del reloj puede ser configurada (activo por flanco de subida o por flanco de bajada), así como también es configurable el habilitador de reloj (clock enable) o si la entrada de set-reset es activa en alta o en baja. En terminología del fabricante Altera un concepto similar seria LE (Logic Element). [3]

3.2. Slice (Terminología Xilinx).

Subiendo en jerarquía tenemos los slices como se aprecia en la figura 15. Un Slice sólo contiene una agrupación de celdas lógicas almacenadas en su interior, en este caso dos LC. Se hace referencia a este nivel de jerarquía ya que las herramientas CAD-EDA hacen referencia a estos como una medida de los recursos utilizados en un diseño dado. [3]

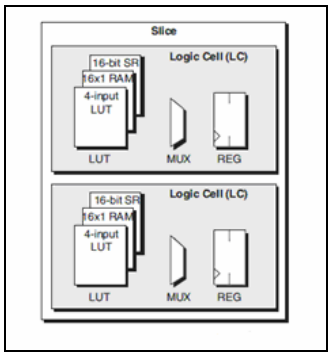


Figura 15. Estructura basica de un Slice (Extraido de Maxfield, Clive. The design warrior´s guide to fpga´s, 2004)

3.3. Bloque de lógica configurable CBL (Configurable Logic Block)

Continuando en la jerarquía se tiene los CBL, los cuales están formados por 4 o más Slices dependiendo de la familia. En la figura 16 se muestra un ejemplo de este tipo de bloques. En terminología del fabricante Altera un concepto similar seria el LAB (Logic Array Block). [3]

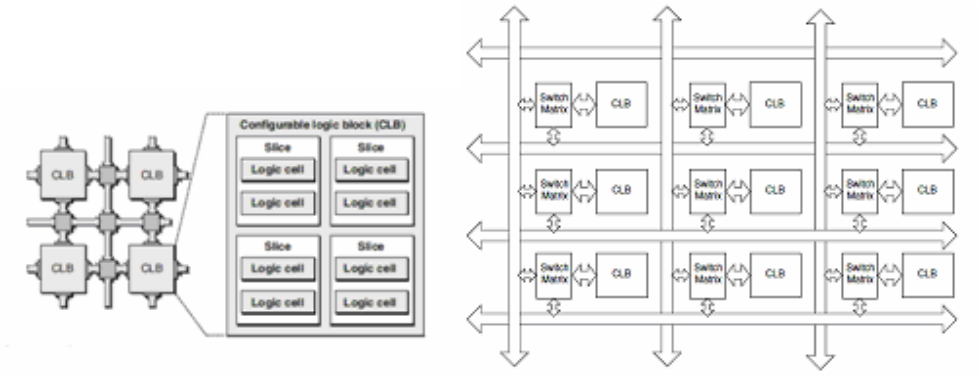


Figura 16. Estructura y disposición de los CBL

Este nivel de jerarquía es apropiado para ver el interior del dispositivo FPGA como un mar de bloques lógicos programables o CBL (LAB en Altera) unidos por bloques de lógica de interconexión programable que se encargan dado el caso mediante programación comunicar un CBL con otro CBL vecino. [3]

3.4. Ram distribuida y shift registers

Se observa que los LUTs tienen carácter multifacético y se pueden comportar como un pequeño bloque de RAM (ejemplo 4 LUTs se pueden comportar como un bloque RAM de 16x1, o un 16 shift register de 16 bits), una vez observados los niveles de jerarquía básicos descritos anteriormente, en donde a modo general un CBL está formado por una cantidad de slices, un slice está formado por una cantidad

de LCs y a su vez estos por una cantidad de LUTs. En un mismo CBL es posible usar todos los LUTs contenidos y obtener nuevas y diversas configuraciones, de acá otra de las ventajas del uso de LUTs. [3]

En el caso del ejemplo descrito de LUTs de 4 entradas se podrían obtener lo siguiente:

- Single-port 16 × 8 bit RAM
- Single-port 32 × 4 bit RAM
- Single-port 64 × 2 bit RAM
- Single-port 128 × 1 bit RAM
- Dual-port 16 × 4 bit RAM
- Dual-port 32 × 2 bit RAM
- Dual-port 64 × 1 bit RAM

De igual manera es posible configurar un CBL para que implemente un Shift Register que contenga hasta 128bits.

3.5. Administradores de reloj, DCM (Digital Clock Managers).

Si bien es posible realizar en una FPGA circuitos asíncronos, estas prácticas no son recomendadas por los fabricantes por tener más complicaciones y se debe evitar (por ejemplo carreras, problemas en la codificación de estados, etc). En general se trabaja con circuitos sincronizados con reloj, haciendo uso de los elementos de registro F/F, etc provistos para ello en cada LC. Para esto la FPGA necesita la entrada de señales de reloj en varios pines especiales provistos para ello y dichos relojes son generados de forma externa a la misma.

Cada fabricante ofrece dentro de su arquitectura una serie de árboles o ramificaciones de cómo se distribuyen los relojes por toda la superficie de la FPGA (ejemplo en la Figura 17, se observa la distribución del reloj para la familia Spartan 3 de Xilinx), esto además ayuda a garantizar que todos los flip-flop puedan observar la misma versión de la señal de reloj tanto como sea posible, ya que al distribuir el reloj en una pista larga se ira impulsando todos los flip-flops, uno tras otro, y entonces el flip-flop más cercano al pin de reloj va a ver la señal del reloj mucho antes que el del final de la cadena. Esto se conoce como skew, y puede causar todo tipo de problemas. [3]

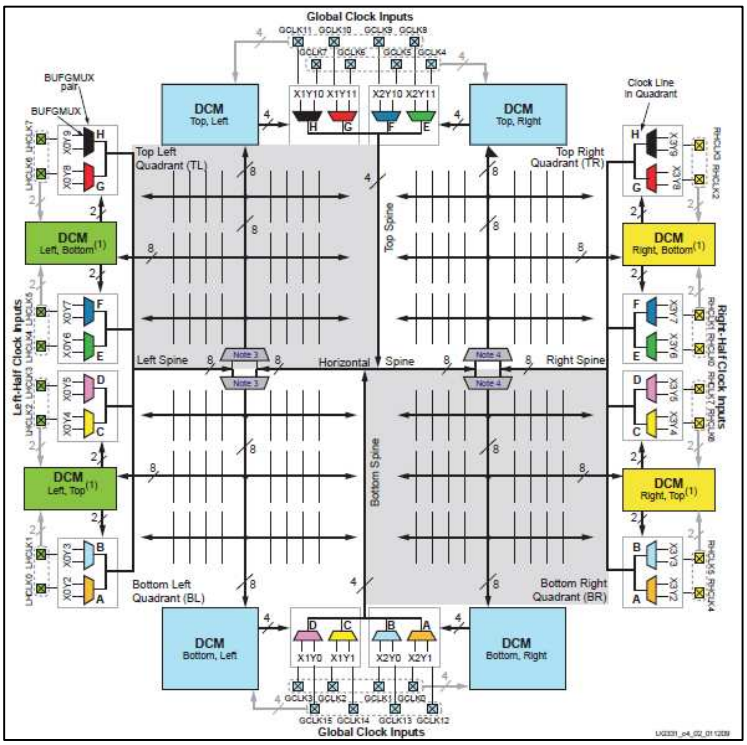


Figura 17. Estructura de distribución de reloj y DCMs para la familia Spartan 3 de Xilinx.(Extraido de Spartan-3 Generation FPGA User Guide)

Por otro lado es posible a su vez tener varios dominios de reloj en una misma FPGA, Para operar adecuadamente y llevar a cabo estas tareas, dentro de la FPGA existen estructuras altamente especializadas en gestionar los relojes tales como los

DCM, como se puede apreciar en la figura 18, los cuales ejecutan varias tareas importantes tales como:

- Generación de relojes hijos, tanto para uso interno en los árboles de la arquitectura, como para uso externo hacia otros dispositivos fuera de la FPGA.
- Eliminación de Jitter, depurando las distintas señales de reloj.
- Auto-Correccion del Skew, permitiendo el ajuste de los relojes hijos, corregir los retardos lógicos causados tanto por la lógica implementada como por las distancias de las pistas internas y que todas las ramificaciones internas de la FPGA observen el mismo reloj. Esto se realiza mediante realimentación.
- Síntesis de frecuencia, en donde el administrador de reloj puede ser usado para multiplicar o dividir la señal de reloj original.
- Desplazamiento de fase: permite realizar operaciones de cambios de fase de una señal hija respecto a otra, aspecto muy importante en aplicaciones de modulaciones digitales por ejemplo.

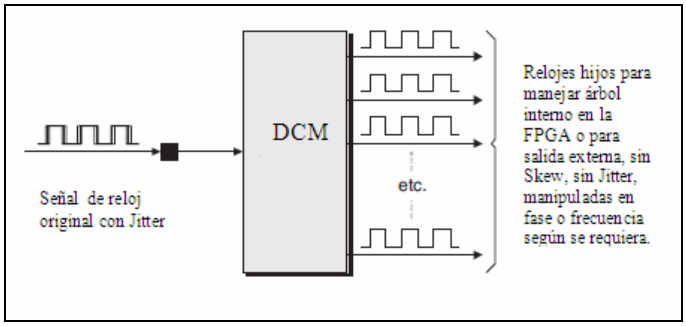


Figura 18. Esquema del DCM.

Para llevar a cabo estas operaciones en la red de distribución del reloj los DCM pueden estar basados en PLL (Phase-Locked Loops), o pueden estar basados en DLL (Delay-Locked-Loops) el cual es la alternativa digital del primero. Obsérvese el

lazo de realimentación para poder realizar la corrección de skew, como se muestra en la figura 19.

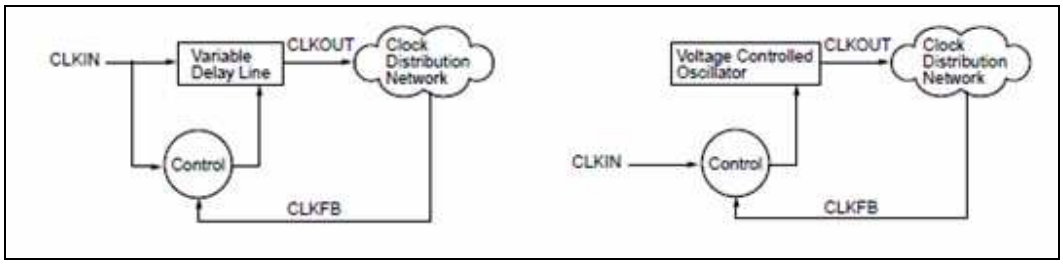


Figura 19. DLL(Izquierda) y PLL(Derecha). (Extraido de Spartan-3 Generation FPGA User Guide)

3.6. Dispositivos E/S (Entrada/Salida) de propósito general.

En la actualidad existen una gran variedad y cantidad de estándares para representar los estados a nivel lógico de una señal digital. Es por esto que los fabricantes de dispositivos FPGAs integran modulos E/S multifuncionales que permiten mediante programación escoger dichos estándares, dando libertad al usuario en la comunicación con gran cantidad de periféricos.

De igual manera las FPGAs permiten el uso de resistencias de terminación internas, usadas para evitar reflexiones y realizar acoplamientos de impedancia, cuyos valores pueden ser configurados por el usuario para dar cabida a diferentes entornos de E / S según los estándares.

Algunos de estos estándares comúnmente soportados son:

- LVTTL (Low-Voltage TTL) el cual es un estándar de propósito general para aplicaciones de 3,3V el cual es evolución de la popular familia TTL (Transistor- Transistor Logic).

- LVCMOS (Low-Voltage CMOS) el cual es un estándar de propósito general para aplicaciones de 1,2V a 3,3V el cual es evolución natural de la popular familia CMOS (Complementary metal oxide semiconductor).
- PCI (Peripheral Component Interface) el cual es un estándar específico para el desarrollo en aplicaciones de buses de datos en 33 MHz y 66 MHz.
- GTL(Gunning Transceiver Logic Terminated) : el cual es un estándar para aplicaciones en buses de alta velocidad inventado por Xerox.

3.7. Dispositivos Embebidos en la FPGA.

Existen ciertas funciones y dispositivos de uso general que son ampliamente usados por los diseñadores y es conveniente tenerlas directamente a disposición en hardware, si bien estos bloques pueden ser construidos haciendo uso de los bloques lógicos internos de la FPGA (ejemplo CBL en el caso Xilinx, LAB caso Altera) estos se comportarían más lentos que su contraparte en hardware, y como son herramientas de uso común los fabricantes los colocan ya directamente en la FPGA en conjunto con la lógica de interconexión asociada formando parte de la arquitectura interna de la misma proporcionando un mayor performance.

Por ejemplo, muchas aplicaciones DSP requieren de bloques de RAM para implementación de funciones FIFO (first-in first-out), así como también Multiplicadores, Sumadores, Funciones Multiplica y Acumula (MAC) (muy usado en arquitecturas DSP como por ejemplo en filtros digitales), Microcontroladores, Microprocesadores, Transceiver diferenciales especiales para comunicación a altas velocidades, dispositivos Ethernet, entre otros. (Ver figura 20)

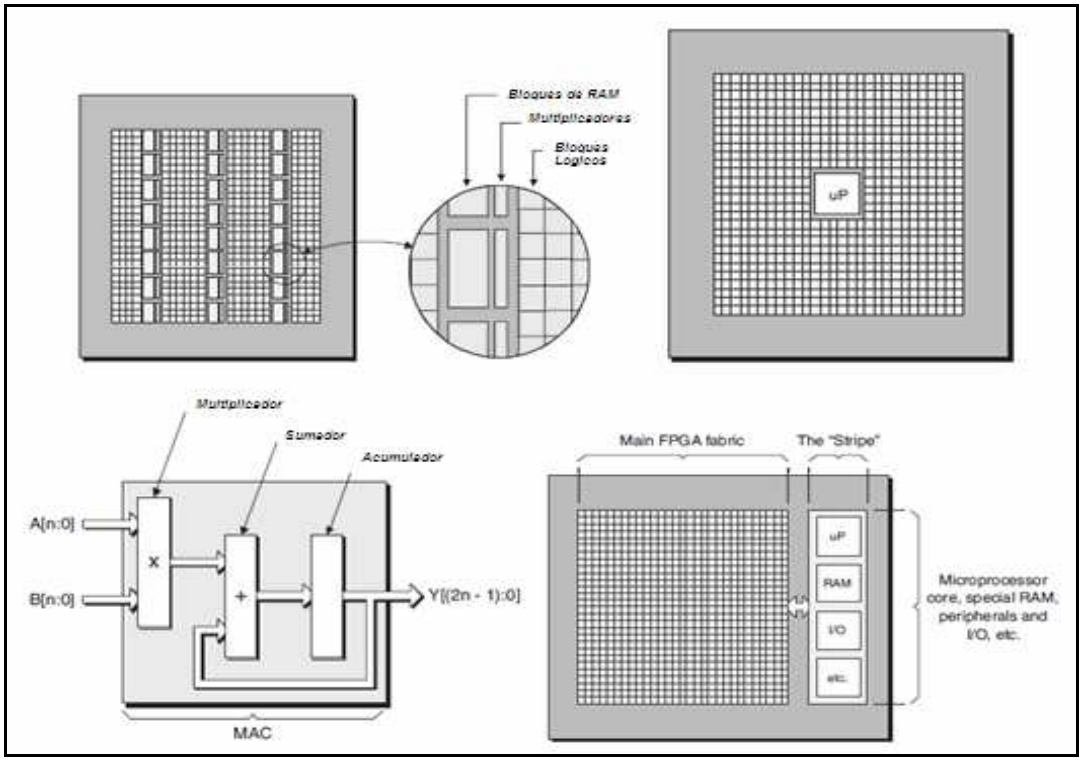


Figura 20. Distintos dispositivos embebidos en una FPGA (Extraido de Maxfield, Clive. The design warrior´s guide to fpga´s, 2004)

Es Importante aclarar, como se comentó anteriormente que estas funciones pueden ser agregadas vía software o contenidas vía hardware. Por ejemplo en el caso de microcontroladores existen los Soft-cores y Hard-cores, las primeras como su nombre indica implementadas vía software ya sea por diseño del mismo usuario (Vía HDL, Hardware Description Language) o vía Librerías IP (Intelectual Property) ambos haciendo uso de los recursos internos de los que dispone una FPGA en particular. [1]-[3]

3.8. Flujo de diseño.

En la figura 21 observamos el flujo de diseño a seguir para desarrollar un prototipo en general. Cabe destacar que gran parte del proceso de diseño se lleva a cabo en base a herramientas CAD – EDA (Computer Aided Design- Electronic

Design Automation), el software generalmente está asociado a cada fabricante y conforma un conjunto de herramientas dedicadas para cumplir cada etapa del flujo mencionado. Estas son de gran importancia y en general proporcionan en gran medida las herramientas de diseño y la eficiencia en la implementación.

A continuación una breve descripción de cada etapa:

- El diseño comienza con la introducción del esquema del circuito de forma gráfica (esquemático o FSM) o en formato texto. La descripción gráfica mediante esquemáticos implica la incorporación al diseño de elementos o componentes de librerías prediseñadas por el fabricante o por el usuario (custom), tales como, puertas lógicas, sumadores, contadores, etc. El uso de las FSM permite un alto nivel de abstracción en el diseño. Por último, la entrada de texto implica la utilización de lenguajes de descripción de circuitos como VHDL, Verilog, etc. Estos lenguajes permiten describir un circuito eléctrico o digital. La descripción puede ser estructural, donde se muestra la arquitectura del diseño; o bien comportamental, donde se describe el comportamiento del circuito en vez de los elementos de los que está compuesto.
- Una vez el diseño ha sido introducido se procede a la verificación del mismo mediante una simulación funcional. Este tipo de simulación comprueba el funcionamiento de circuitos digitales de forma funcional; es decir, a partir del comportamiento lógico de sus elementos (sin tener en cuenta problemas eléctricos como retrasos, etc) se genera el comportamiento del circuito frente a unos estímulos dados. Cabe destacar que no todas las estructuras descritas en HDL son sintetizables a un circuito por lo que hay que tener cuidado en comó se escribe dicho código de manera tal que sea interpretado correctamente por la herramienta, siempre teniendo en mente que no se está diseñando software si no hardware.
- El siguiente paso es la síntesis o traducción del código HDL a un posible circuito lógico. El resultado de la síntesis es una lista de conexiones en formato específico del fabricante. En el proceso de síntesis se fijan

restricciones de diseño (topológicas o temporales). En esta etapa se lleva a cabo una simulación lógica donde se corrobora que el código escrito fue correctamente traducido e inferido a hardware.

- Finalmente, el módulo de implementación tiene la misión de plasmar el diseño de entrada sobre una tecnología concreta. El módulo de implementación genera un archivo para el análisis temporal teniendo en cuenta la localización y conexionado de los recursos hardware utilizados, permitiendo la simulación temporal del diseño (Timing simulation), la cual es como la simulación funcional, con la diferencia de que se tienen en cuenta retrasos en la propagación de las señales digitales. Es una simulación muy cercana al comportamiento real del circuito y prácticamente garantiza el funcionamiento correcto del circuito a realizar.

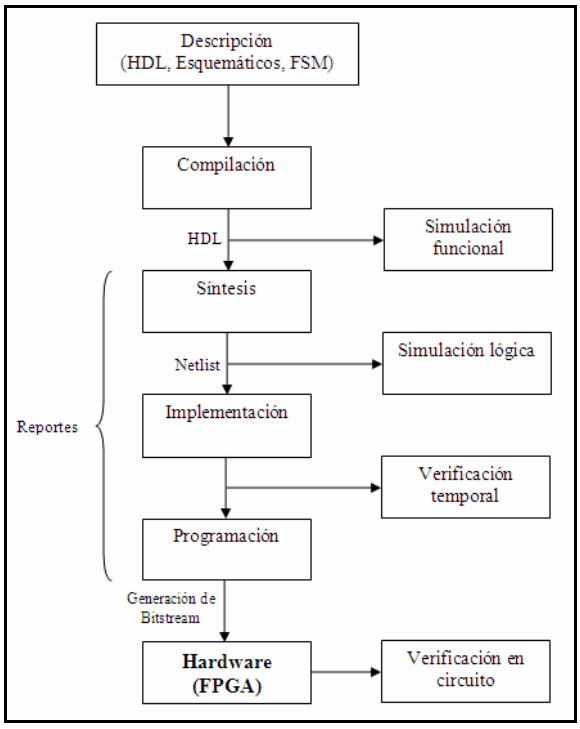


Figura 21. Flujo de diseño mediante FPGAs.

4. PRINCIPALES FABRICANTES DE DISPOSITIVOS FPGAS.

En la actualidad existen en el mercado muchos fabricantes dedicados a la fabricación de esta tecnología, cada uno de los cuales tiene características propias en cuanto a su arquitectura, tecnología de fabricación, recursos ofrecidos, herramientas de programación, facilidades de implementación entre otros.

Algunos fabricantes son:

Tabla 2. Principales fabricantes de dispositivos FPGAs.

Fabricante	Dirección	Sitio WEB
Achronix	333 West San Carlos Street Suite 1050, San Jose, CA 95110 USA	http://www.achronix.com
Actel	2061 Stierlin Ct., Mountain View, CA 94043, USA	http://www.actel.com
Altera	101 Innovation Drive, San Jose, CA 95134, USA	http://www.altera.com
Atmel	2325 Orchard Parkway, San Jose, CA 95131, USA	http://www.atmel.com
Cypress	198 Champion Ct., San Jose, CA 95134 USA	http://www.cypress.com
Chengdu S.M	High-tech Zone, Chengdu, Sichuan Province, Gao Peng Road No. 11 Hi-tech Zone Industrial Park, Block D. China	http://www.csmsc.com
Lattice	5555 N.E. Moore Court Hillsboro, Oregon 97124-6421, USA	http://www.latticesemi.com
QuickLogic	1277 Orleans Drive, Sunnyvale, CA 94089-1138 USA	http://www.quicklogic.com
Silicon Blue	505 N Mathilda Ave Suite 110 Sunnyvale, CA 94085 USA	http://www.siliconbluetech.com
Tabula	3250 Olcott St., Santa Clara, CA 95054 USA	http://www.tabula.com
Tier Logic	2975 Scott Blvd, Suite 215, Santa Clara, CA 95054 USA	http://www.tierlogic.com
Xilinx	2100 Logic Drive San Jose, CA 95124-3400, USA	http://www.xilinx.com
eASIC	2585 Augustine Drive, Suite 100, Santa Clara, CA 95054 USA	http://www.easic.com

Cada uno de estos fabricantes tiene repertorios de familias de dispositivos que tratan de adaptarse a las necesidades del diseñador.

Algunas características generales son:

Xilinx:

-Ofrece en general dispositivos basados con tecnología SRAM.

-Ofrece una amplia gama de productos, básicamente los más comunes son clasificados en tres familias: una familia de CPLD llamada CoolRunner, y dos gamas de FPGAs, una de bajo costo llamada Spartan y otra de alto costo y alta calidad llamada Virtex. Recientemente ha agregado nuevas familias que buscan a futuro sustituir las series actuales, una de bajo costo llamada Artix , una familia de rango medio Kintex, y una de gama alta Zynq el cual integra procesadores ARM Cortex-A9 (hardcore) basada en tecnología de 28-nm, también ofrece tecnología resistente a la radiación en sus productos.

- Ofrece un conjunto de herramientas de diseño como el Webpack ISE (con soporte para las plataformas Windows y Linux) de licencia libre, para diseño y análisis de sistemas en general y otro conjunto de herramientas para facilitar el diseño de DSPs (System Generators for DPS), los cuales están integrados con Matlab, tiene el Xilinx's Embedded Developer's Kit (EDK) para diseño y puesta a punto de sistemas embebidos, igualmente existen herramientas de programación de más alto nivel ejemplo Labview FPGA.

- Ofrece IPs y diseños de referencia, algunos de estos de diseño propio optimizado para sus productos como el microcontrolador PicoBlaze (8 bit, arquitectura RISC, softcore similar a un PIC), MicroBlaze (softcore 32bit con mayores prestaciones) y el hard-core PowerPC incluido en sus productos de gama alta serie Virtex.

- Ofrece un servicio llamado Easypath en donde los productos desarrollados en su serie Virtex pueden ser migrados a ASIC para abaratar costos al realizar una producción masiva.

Altera:

- Ofrece en general dispositivos basados con tecnología SRAM.

- Ofrece sus productos clasificados básicamente en cuatro familias: una familia de CPLDs llamada MAX, familias de FPGAs de bajo costo llamadas Cyclone, una de gama media llamada Arria y otra de gama alta y mayores prestaciones llamada Stratix.
- Ofrece un conjunto de herramientas de diseño integradas en el Quartus II Web Edition y el MAX+PLUS II (con soporte para las plataformas Windows y Linux) de licencia libre, así mismo tiene herramientas dedicadas para procesamiento digital de señales como es el DSP Builder.
-Ofrece de igual manera IPs y diseños de referencia, algunos de estos, de diseño propio optimizados para sus productos como el microprocesador Nios II (32bit, softcore).
-Ofrece la posibilidad de una fácil transición a ASIC en diseños implementados en sus dispositivos de arquitectura Stratix, producto que el fabricante llama Hardcopy.

Lattice Semiconductor:

-Ofrece dispositivos con tecnología SRAM y dispositivos híbridos SRAM/FLASH.
-Ofrece varias familias entre ellas una de bajo costo llamada LatticeSC/M y otra de gama alta llamada LatticeECP (ejemplo LatticeECP3)
- Ofrece un conjunto de herramientas de diseño como el Lattice Diamond para diseño y análisis de sistemas en general y otro conjunto de herramientas para facilitar el diseño de DSPs llamado ispLEVER**.**
-Ofrece de igual manera IPs y diseños de referencia, ejemplo el Micos32 (32bit, softcore).

Actel:

- Ofrece dispositivos basados principalmente en tecnología anti-fusible y Flash.

-Ofrece una amplia gama de familias, algunas de estas son la familia IGLOO (la cual puede venir integrada con un microprocesador hardcore ARM Cortex-M1) y la familia ProASIC3, SmartFusion, Fusion, tiene una gama de FPGAs tolerantes a radiación como las series RTAX, RT ProASIC3, RTSX-SU, y tiene una serie basada en tecnología anti-fusible llamadas Axcelerator, SX-A, eX, MX.

-Ofrece un conjunto de herramientas de diseño como el Libero® IDE y el Synplify Pro® AE para diseño y análisis de sistemas en general y otro conjunto de herramientas para facilitar el diseño de SOCs, IP, procesadores llamado SmartDesign **y** SoftConsole entre otros.

- Ofrece de igual manera IPs y diseños de referencia, en particular hace uso intensivo de procesadores ARM en sus dispositivos (en hardcore y en softcore) y ofrece también un softcore exclusivo para sus dispositivos llamado Leon3 (32bit, arquitectura Sparc V8)

Atmel:

-Ofrece en general dispositivos basados con tecnología SRAM.

-Ofrece varias familias de uso general entre ellas la AT40K, AT40KAL.

-Ofrece un servicio llamado FPGA Conversion ULC, que permite la migración desde los distintos fabricantes de dispositivos FPGA en el mercado a un equivalente ASIC para su producción masiva abaratando costos.

-Ofrece algunas herramientas de diseño como el Integrated Development System [IDS], para diseño general, pero indica que su flujo de desarrollo es libre y es abierto al uso de herramientas EDA de uso estándar en la industria, ejemplo Mentor Graphics.

Achronix:

-Ofrece dispositivos FPGA de alta velocidad de alrededor de 1,5Ghz y densidades de hasta 2.5M LUTs, su principal familia es la Speedster22i.

-Ofrece algunas herramientas para Ruteo, analisis temporal y generación de bitstream llamada Achronix CAD Environment (ACE), para diseño general y simulación indica que su flujo de desarrollo es abierto al uso de herramientas EDA de uso estándar en la industria, como Mentor Graphics y Synopsys.

A continuación observamos en la tabla siguiente el Ranking mundial de vendedores de tecnología FPGA/PLD del año 2007/2008 que proporciona una idea de cuáles son los fabricantes más comúnmente utilizados por los diseñadores en estas tecnologías.

Tabla 3. Ranking Mundial 2007-2008 vendedores de FPGA/PLD (Gartner 2008)

Worldwide FPGA/PLD vendor revenues and rankings, 2007-2008

Rank 2007	Rank 2008	Company	Revenue ($M) 2007	Revenue ($M) 2008	Revenue Change 2007-2008	Market Share 2008
1	1	Xilinx	1,809	1.906	5.4%	51.2%
2	2	Altera	1,216	1,323	8.8%	35.5%
3	3	Lattice Semiconductor	229	222	–3.1%	6.0
4	4	Actel	196	218	11.2%	5.9%
6	5	QuickLogic	28	23	–17.9%	0.6%
5	6	Cypress Semiconductor	32	21	–34.4%	0.6%
7	7	Atmel	14	9	–35.7%	0.2%
8	8	Chengdu Sino Microelectronics System	4	3	–25.0%	0.1%
		Others	0	0	NM	0.0%
		Total Market	**3,528**	**3,725**	**5.6%**	**100.0%**

Source: Gartner

En la figura 22 se observa el ranking mundial para fabricantes de herramientas CAD- EDA los cuales como ya hemos hecho énfasis van de la mano en conjunto con el desarrollo de la tecnología FPGA, los mismos son los que hacen en gran medida

posible los cortos ciclos de desarrollo proporcionando herramientas de simulación, diseño, síntesis y uso óptimo del hardware disponible en la FPGA.

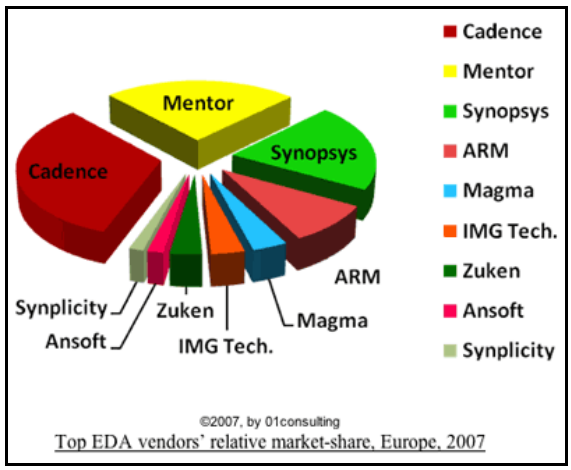


Figura 22. Principales vendedores EDA y su porcentaje en el mercado.

CAPÍTULO III

1. SELECCIÓN DE LA PLATAFORMA DE DESARROLLO Y LA FPGA

1.1. Características a tomar en cuenta para la escogencia de una FPGA.

Los fabricantes de tecnología FPGA en general no dan detalles en bajo nivel (a nivel de transistores, compuertas etc.) de cómo es el funcionamiento de los bloques constituyentes de una FPGA en particular por lo que existen algunas consideraciones a tomar en cuenta:

- Las características de performance entre un fabricante y otro son inciertas ya que existen muchas variables a considerar y que los fabricantes no dan a conocer abiertamente.
- Como se ha comentado, el performance de una FPGA va ligado a la eficiencia de la herramienta CAD-EDA que es la encargada de procesos como minimizaciones a nivel lógico, síntesis, inferencias del código HDL, análisis temporales, algoritmos de ruteo interno en la FPGA, etc. que varían de un fabricante a otro, lo cual conllevaría a el estudio de estas también, ya que gran parte de la eficiencia y performance radica en las mismas.
- Por otro lado, suponiendo que el software CAD-EDA no sea relevante o se estandarice de igual manera no es posible decir a ciencia cierta cuál fabricante o arquitectura es mejor que otro, si nos basamos por ejemplo en el rendimiento de un Core en particular éste puede estar o no beneficiado por la arquitectura interna que usa un determinado fabricante, por lo que se tendrá aleatoriedad en los resultados ya que para algunas arquitecturas un diseño en particular tendrá más performance que para otras.

Debido a que cada fabricante o en cada arquitectura en general la disposición de los bloques lógicos constituyentes y la manera como se interconectan varía no existe una medida clara de comparación entre un fabricante y otro en cuanto a recursos lógicos contenidos.

En los primeros días de esta tecnología los fabricantes daban un estimado en cuanto al número de compuertas internas que contenían (sobre todo en el caso de arquitecturas de granularidad fina que asemejaban un Gate Array), generalmente tomaban un patrón basado en el equivalente a compuertas NAND de 2 entradas, pero esto no fue puesto en practica por todos los fabricantes, además de no proporcionar una medida clara y ser impráctico. Hoy en día las FPGAs tienen granularidad de media a gruesa, y sus bloques constituyentes son multifacéticos en la mayoría de los casos, además de poseer recursos embebidos como se ha comentado anteriormente (desde bloques multiplicadores a microprocesadores) que complican aun más la tarea y deben ser evaluados por separado.

La escogencia de un dispositivo FPGA no es una tarea fácil de realizar en general, a nivel básico su escogencia viene dada por las características dadas a continuación:

- Experiencia en el área, lo cual supone conocimientos aproximados de los recursos y performance requeridos para el diseño a llevar a cabo.
- Aplicación para la cual van a ser utilizadas, dependiendo de la misma se puede facilitar la escogencia de la tecnología de las celdas lógicas (ejemplo para aplicaciones espaciales que tenga soporte para la radiación, “Radhard”).
- Número de celdas lógicas (LC caso Xilinx) ó de elementos lógicos (LE caso Altera) o del elemento de menor jerarquía posible para otros fabricantes, el cual nos proporciona una medida indirecta del número de LUTs, Flip/Flop o latch que tiene el dispositivo.
- Número y tamaño de bloques de RAM embebidas.

- Número y tamaño de los bloques multiplicadores embebidos.
- Número y tamaño de los bloques sumadores embebidos.
- Número y tamaño de los bloques MACs embebidos.
- Número y características de los DCM (PLL, DLL)
- Características específicas de los distintos bloques hardcore embebidos (ejemplo si posee un microprocesador o un transceiver).
- Características y cantidad de pines E/S, estándares soportados etc.

Para fines prácticos en el presente proyecto se toma como patrón referencia para poder estimar el tamaño en cuanto a recursos lógicos de la FPGA, un IP core comercial del microcontrolador 8051 de la casa especializada y reconocida en IP llamada Cast.inc (www.**cast**-inc.com) en la cual se puede acceder a una base de datos donde se puede visualizar los recursos ocupados por un determinado IP core para los principales fabricantes del mercado.

Tabla 4. Ocupación de recursos para el CAST T8051 Core, Xilinx

Xilinx Device	*Slices*	*BRAM*	*I/Os*	*Fmax (MHz)*	*ISE*
Spartan-3E 3S1200E-5	1631	4	121	60	12.1i
Spartan-6 6SLX150-3	667	6	121	90	12.1i
Virtex-4 4VLX40-12	1269	-	121	135	10.1.2i
Virtex-5 5VLX330-2	721	3	121	135	12.1i
Virtex-6 5VLX240T-3	557	3	121	120	12.1i

Tabla 5. Ocupación de recursos para el CAST T8051 Core, Altera

Altera Device	Area	Speed
Cyclone-II EP2C5F256C6	1874 LCs	68 MHz
Cyclone-III EP3C5F256C6	1899 LCs	78 MHz
Stratix-II EP2S15F484C3	1346 ALUTs	112 MHz
Stratix-III EP3SE50F484C2	1392 ALUTs	157 MHz

.

Tabla 6. Ocupación de recursos para el CAST R8051XC2 Core Xilinx (configurado con 2 timers, 1 interface serial, 2 DPTRs, 1 I2C, 1 SPI, OCDS y memoria)

Xilinx Devices	Slices	BRAM	I/Os	Fmax (MHz)	ISE
Spartan-3E XC31200E-5	2385	2	66	45	12.1i
Spartan-6 XC6SLX75-3	863	2	66	60	12.1i
Virtex-5 XC5VLX30-3	974	2	66	80	12.1i
Virtex-6 XC6VLX75T-3	820	2	66	81	12.1i

Tabla 7. Ocupación de recursos para el CAST R8051XC2 Core, Altera (configurado con 2 Timers, 1 Interface Serial y memoria)

Altera Devices	LEs/ALUTs	Memory	I/Os	Fmax (MHz)	Quartus
Cyclone-II EP2C5-6	2768	10 M4Ks	67	35	8.0
Cyclone-III EP3C5-6	2770	6 M9Ks	67	39	8.0
Stratix-II EP2S15-3	1816	10 M4Ks	67	57	8.0
Stratix-III EP3SL50-2	1813	6 M9Ks	67	72	8.0

En las tablas 4, 5, 6, 7 vemos dos de estos IP Core ofrecidos por el mismo, por ejemplo para tecnologías de LUT de 4 entradas, en las FPGAs Xilinx (Spartan 3E), un core de 8051 (el más completo R8051XC2) toma alrededor de 2385 Slices y 2 bloques de RAM, para Altera un Core similar toma alrededor de 2768 LEs y 10 Bloques de memoria 10MK4, por lo que al realizar la selección de la plataforma de desarrollo se tomara en cuenta dichos estimados.

1.2. Selección de la plataforma de desarrollo para uso en la EIE.

Luego de haber investigado y familiarizado a modo general con todo lo referente a la tecnología FPGA, nuestro siguiente paso fue investigar acerca de las plataformas de desarrollo disponibles en el mercado, y seleccionar una acorde con los requerimientos de la E.I.E.

En general, antes de tomar cualquier decisión respecto a la selección de las diversas plataformas de desarrollo existentes en el mercado, el conocimiento y la experiencia se deben reunir. El proceso de toma de decisiones, normalmente se apoya en la experiencia o en la semejanza a decisiones anteriormente tomadas que llevaron a buenos resultados, y raras veces se basa en un método sistemático o herramienta de apoyo a la resolución de tal disyuntiva.

Una buena decisión en general tiene las siguientes características:

- Es una decisión en la que se ha trazado el objetivo que se quiere conseguir.
- Se ha reunido toda la información relevante al caso.
- Se han tenido en cuenta las preferencias del decisor, según criterios preestablecidos.

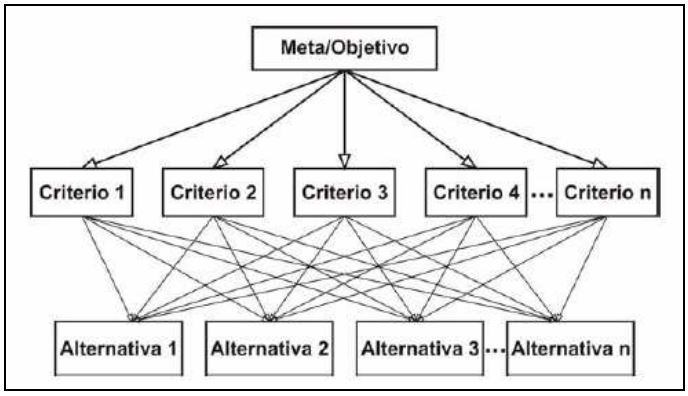


Figura 23. Esquema representativo de la selección multicriterio.

Para poder cumplir con el objetivo, en la selección de la plataforma de desarrollo, se tuvo que establecer una serie de criterios u requerimientos, los cuales nos permitirán evaluar las alternativas disponibles (ver figura 23), estos criterios los listamos a continuación:

Requerimientos Generales para la escogencia de la plataforma de desarrollo:

- Debe tener Soporte adecuado del fabricante, existir amplia documentación, manuales, libros, ejemplos de uso, todo lo necesario para que un estudiante o profesor pueda interactuar con la plataforma de desarrollo y sus periféricos fácilmente.
- Debe ser una plataforma intermedia, adecuada para cursos de pregrado, en particular adecuada para la realización de proyectos y practicas en las materias directamente involucradas (Sistemas digitales I, Microprocesadores I y II), pero no limitada a las mismas, debe tener suficientes recursos para la realización de un trabajo de grado en pregrado.
- La FPGA contenida en la plataforma de desarrollo debe tener al menos recursos suficientes para poder albergar un IP core de un microcontrolador 8051, para ser consecuentes con lo impartido en la materia Microprocesadores I.
- La posible tarjeta de desarrollo debe ser programada preferiblemente vía puerto USB, el cual esta integrado comúnmente en los PCs del laboratorio.
- Debe contar al menos con los siguientes periféricos básicos (no limitante):

    - Switches deslizantes.
    - Switches/ Pulsadores.
    - Led indicadores.

- Displays LCD / Displays 7 segmentos.
- Puertos RS-232
- Puerto USB para su programación.
- Conversores AD / DA
- Puerto Ethernet
- Diversos dispositivos de memoria DDR, SDRAM, FLASH, etc. para albergar un archivo de programa de un micro, o como RAM para este, o para uso en aplicaciones DSP.
- Puerto Ps/2 (para teclado o ratón).
- Puerto VGA.
- Puertos de expansión.
- Oscilador ó Reloj para uso con la lógica a implementar.
- Puerto SMA de E/S.
- Planos y diagramas esquemáticos.

Cabe destacar que en la E.I.E de la U.C.V, no hay experiencias previas en cuanto a escogencia de estas plataformas y sus usos, por lo que prácticamente un criterio fundamental es la experiencia encontrada en cuanto al uso de dichas plataformas en otras casas de estudio (nacional e internacional), y cuan soportada esta dicha plataforma en cuanto a documentación en el ámbito de pregrado.

Para simplificar la cantidad de alternativas bajo estudio, basado en la tabla de ranking del mercado de los FPGAs y PLD (tabla 3, Capítulo 2), se preseleccionaron los dos principales fabricantes de dicha tecnología Xilinx y Altera, la búsqueda se

concentró en los productos ofrecidos, recomendados por dichos fabricantes para aprendizaje en sus programas universitarios (Xilinx University, Altera University) para pregrado.

Cabe destacar que en Venezuela no existen distribuidores o representantes de ningún fabricante de la tecnología FPGA, tampoco de distribuidores de sus Kit de desarrollo, lo cual podría haber sido un criterio de peso respecto a la logística en la adquisición de dichos kits de desarrollo.

Como una referencia en la investigación realizada se encontró información de las plataformas de desarrollo utilizadas en la Universidad Simón Bolívar (USB) (FPGA Xilinx XC3S200, Tarjeta de desarrollo Spartan 3 Starter Board en su materia de Circuitos digitales) , y de la plataforma utilizada en la Universidad de los Andes (ULA) los cuales utilizan en sus practicas la (FPGA Xilinx XC4000XL, tarjeta de desarrollo XS40 v1.4 del fabricante XESS CORP, en su materia de Sistemas Digitales).

Existen de igual manera características funcionales a tomar en cuenta. Por ejemplo, muchas tarjetas traen micro-dip switch como periféricos, los cuales no son apropiados para uso continuo en un ambiente académico práctico y de prototipado (no se pueden siquiera manipular con los dedos, además de ser frágiles), algunas no traen led, otras no traen displays, etc. las casas fabricantes de estas tarjetas de desarrollo juegan con las características a suministrar en los periféricos, de igual manera juegan con los recursos disponibles en las FPGAs, puertos de expansión etc, por lo que se busca la mas completa en cuanto a recursos disponibles y costo.

A continuación se observa una tabla resumen (Tabla 8) con las alternativas consideradas y sus principales características, pros y contras:

Tabla 8. Alternativas consideradas en la selección de la plataforma de desarrollo.

Opción #	Kit de desarrollo	Recursos FPGA	Memorias	Periféricos	Pros y Contras
1	Nexys2 FPGA Board ***XC3S500E*** ($149.00) ***XC3S1200E*** ($189.00)	***XC3S500E:*** 10470 LCs 4656 Slices 20 Mult /73Kbit Ram Dist / 360Kbit B_Ram 4DCM´S ***XC3S1200E:*** 19512 LCs 8672 Slices 28 Mult /136Kbit Ram Dist / 504Kbit B_Ram 8DCM´S	-Xilinx 4 Mbit Platform Flash(FPGA) -16MB DDR SDRAM -16MB Intel StrataFlash	-8 switch -4 Pulsadores -8 Led -Rs-232 -4 displays 7segment -PS/2 -VGA	-No tiene Fuente (opcional). -No tiene soporte Xilinx -Se programa con software externo. -Riesgo de daño al puerto USB debido a que la alimentación del Kit se lleva a cabo a través del mismo y fácilmente se excede el Max. de 500mA.
2	Spartan-3E Starter Board ***XC3S500E*** ($189.00) ***XC3S1600E*** ($225.00) (MicroBlaze Development Kit Spartan-3E 1600E)	***XC3S500E:*** 10470 LCs 4656 Slices 20 Mult /73Kbit Ram Dist / 360Kbit B_Ram 4DCM´S ***XC3S1600E:*** 33192 LCs 14752 Slices 36 Mult /231Kbit Ram Dist / 648Kbit B_Ram 8DCM´S	-Xilinx 4 Mbit Platform Flash(FPGA) -64Mbyte DDR SDRAM -16Mbyte Intel StrataFlash (NOR FLASH) -16 Mbits SPI Flash	-LCD 2x16 caract. -8 leds -4 switch deslizantes -4 Pulsadores -1 switch rotativo (Pot. Digital) -2 RS232 (DCE-DTE) -1 Ethernet -1 Ps/2 -1 VGA -2 ADC 14bit, 1,5Msps, 4 DAC 12bit -1 SMA	-Soporte Xilinx -Con fuente de poder de 4A. - Soporte Jtag para programar las memorias desde la FPGA. -Programación via USB Xilinx

3	Spartan 3 Starter Board ***XC3S200*** ($109.00) ***XC3S1000*** $159.00)	***XC3S200:*** 4320 LCs 1920 Slices 12 Mult /30Kbit Ram Dist / 216Kbit B_Ram 4DCM´S ***XC3S1000:*** 17280 LCs 7680 Slices 24 Mult /120Kbit Ram Dist / 432Kbit B_Ram 4DCM´S	-Xilinx 2 Mbit Platform Flash(FPGA) -1 MB SRAM	-8 switch -4 Pulsadores -Rs-232 -8 leds -4 displays 7segment -PS/2 -VGA	-Soporte Xilinx -Con fuente de poder 4A. -Sin cable de programación USB: -Cable de prog. Xilinx costoso(129$) -Cable de Prog. Genérico Diligent con software externo. (47.95$) -Alto costo, pocos recursos.
4	Basys2 ***XC3S100E*** ($79.00) ***XC3S250E*** ($99.00)	***XC3S100E:*** 2160 LCs 960 Slices 4 Mult /15Kbit Ram Dist / 72Kbit B_Ram 2DCM´S ***XC3S250E:*** 5508 LCs 2448 Slices 12 Mult /38Kbit Ram Dist / 216Kbit B_Ram 4DCM´S	-Xilinx 2 Mbit Platform Flash(FPGA)	-8 leds -4 displays 7segment -8 switch -4 Pulsadores - Ps/2 -VGA	-Sin Soporte Xilinx -Sin conector de fuente de poder. -Fuente via USB -Oscilador de mala calidad -Se programa con Software Externo. -Riesgo de daño al puerto USB debido a que la alimentación del Kit se lleva a cabo a través del mismo y fácilmente se excede el Max. de 500mA
5	XSA-3S1000 Board V1.1 (199$) XESS CORP.	***Spartan 3*** ***XC3S1000:*** 17280 LCs 7680 Slices 24 Mult /120Kbit Ram Dist / 432Kbit B_Ram 4DCM´S	-32 MByte SDRAM -2 MByte Flash	-1 leds -1 displays 7segment -4 Micro-dip switch -2 Pulsadores - Ps/2 -VGA -100 MHz oscillator -1 Parallel port	-Sin fuente de poder. -Cable de programación paralelo (opcional USB) -Pocos periféricos de memoria. -No practica para prototipado.

6	Atlys Spartan 6 ($349.00)	**Xilinx Spartan-6 LX45 FPGA** 6,822 slices, LUT 6 2.1Mbits block RAM 8 DCMs 4 PLLs 58 DSP slices	128Mbyte DDR2 16Mbyte x4 SPI Flash	-USB-UART -USB-HID -HDMI VIDEO - Ethernet - 100MHz CMOS oscillator - 8 LEDs -6 buttons -8 slide switches -Audio DAC/ADC	-Con fuente de poder 4A. -Programación USB con software externo o software Xilinx con plugin -Menos resolución en ADC/DAC (AUDIO) con un codec modulación Sigma-delta (48Khz) -Sin ningún tipo de display, para que el usuario interactue. -Poca documentación
7	Altera® DE1 ($199.00)	***EP2C20:*** 18752 LEs 26 Mult / 239,6Kbit RAM / 4PLL´S	-Altera 4 Mbit Platform Flash(FPGA) -4 Mbyte NOR FLASH -8 Mbyte SDRAM -512Kbyte SRAM	-10 switch -4 Pulsadores -1-Rs-232 -18 leds -4 displays 7segment -PS/2 -VGA -SD_card socket -24 bit audio Codec -SMA	-Soporte Altera -Con fuente de poder. -No tiene Ethernet -Menos resolución en ADC/DAC (AUDIO) con un codec modulación Sigma-delta (8-96Khz)
8	Altera® DE2-70 ($599.00)	***EP2C70:*** 68416 Les 150 Mult. / 1152Kbit RAM / 4PLL´s	-Altera 64Mbit Platform Flash(FPGA) -64 MByte SDRAM -2 MByte SSRAM -8 MByte Flash	-18 switch -4 Pulsadores -1-Rs-232 -27 leds -4 displays 7segment -1LCD 2x16 caract. -PS/2 , VGA -SD_card socket -24 bit audio Codec -SMA -Ethernet -USB -SD_card socked -2-Video In -Puerto infrarojo	-Soporte Altera -Con puerto USB -Con fuente de poder. -Alto costo. -Menos resolución en ADC/DAC (AUDIO) con un codec modulación Sigma-delta (8-96Khz)

9	Altera® DE0 ($119.00)	***EP3C16:*** 15408 Les 56 Mult / 516,096Kbit RAM / 4PLL´***S***	-Altera 4 Mbit Platform Flash(FPGA) -4 Mbyte NOR FLASH -8 Mbyte SDRAM	-10 switch -3 Pulsadores -1-Rs-232 -10 leds -4 displays 7segment -PS/2 -VGA	-Soporte Altera -Con fuente de poder. -No tiene Ethernet -Sin ADC/DAC -Menos recursos en Memorias Externas

Luego de verificar, estudiar sus manuales, estudiar pros y contras, en las alternativas bajo estudio (Ver tabla 8), y basados en los criterios previamente establecidos se concluye lo siguiente:

-Se considera que la alternativa más conveniente para uso en nuestra escuela es alternativa numero 2, llamada "Spartan 3E Starter Board" la cual se encuentra disponible en dos versiones según la cantidad de recursos en la FPGA (tienen los mismos periféricos en tarjeta), se opto por la de mayor tamaño que contiene la FPGA XC3S1600E (esta versión es también conocida como "MicroBlaze Development Kit Spartan-3E 1600E"), con 14752 Slices suficientes para implementar mas de 4 Cores de 8051 (patrón de referencia tablas 4,5,6,7) y su lógica de pegamento.

- La plataforma de desarrollo "Spartan 3E Starter Board" presenta la mayor combinación de recursos y periféricos útiles para el desarrollo de experiencias y proyectos en nuestra escuela.
- Es ampliamente utilizada a nivel mundial en el ámbito universitario, es una tarjeta madura y se consigue documentación fácilmente.
- No requiere recursos adicionales, cables especiales u otros para la realización de prácticas y proyectos, es soportada oficialmente por Xilinx por lo que tiene ejemplos variados de uso.
- Es una plataforma de desarrollo de costo intermedio (225 $), lo que permite adquirir un número de estas tarjetas para formar un laboratorio.

CAPÍTULO IV

1.0. EXPERIENCIAS

Se realizó la descripción, diseño, simulación e implementación de dos experiencias prácticas que permiten poner a prueba las herramientas antes mencionadas, básicamente estas experiencias permiten familiarizarse y conocer cómo realizar una descripción de hardware, se debe recordar que los lenguajes HDL (Hardware Description Language) no son orientados a software propiamente hablando (no se ejecutan línea a línea como en los microcontroladores por ejemplo), si no que describen comportamiento de hardware físico, por lo que el modo de programación es distinto.

En general para la realización de las experiencias se debe tomar en cuenta lo siguiente:

- Para la realización de las experiencias practicas se utilizó la tarjeta Spartan 3E Starter Kit de Xilinx, seleccionada previamente.
- Se utilizó como software CAD-EDA el recomendado por el fabricante disponible para descarga en la página Web de Xilinx llamado ISE Webpack en su versión 12.1 el cual es de uso libre para estudiantes y profesores, este comprende de un DVD que contiene todas las herramientas necesarias. Para poder usarlo, requiere registrarse en su página y tras confirmación vía correo electrónico se recibe una licencia temporal que permite su uso.
- Como lenguaje de descripción de hardware (HDL) se optó por el uso de VHDL (ANSI/IEEE-1076), el cual es considerado uno de los más completos en el campo de los HDL en FPGA y ASIC. Para su aprendizaje se hizo lectura de diversos libros dedicados en específico a este tema y su aplicación en el campo de las FPGA.

- Para las dos experiencias desarrolladas se siguió el flujo de diseño ya descrito en el Capítulo II (ver figura 21).
- Los problemas planteados en las experiencias se descompusieron modularmente, hablando en términos de VHDL en entidades más pequeñas y simples, la unión de todas las entidades pequeñas resuelven el problema de diseño planteado. Cada una de estas entidades fueron simuladas por separado mediante un Testbench (terminología HDL, Banco de prueba).

Un testbench consiste en desarrollar vía HDL un hardware virtual que permite introducir los estímulos, secuencias necesarias en la unidad bajo prueba (denominada U.U.T, Unit Under Test) y así poder observar su respuesta (comportamiento) a través de un diagrama de tiempo. Ver figura 24.

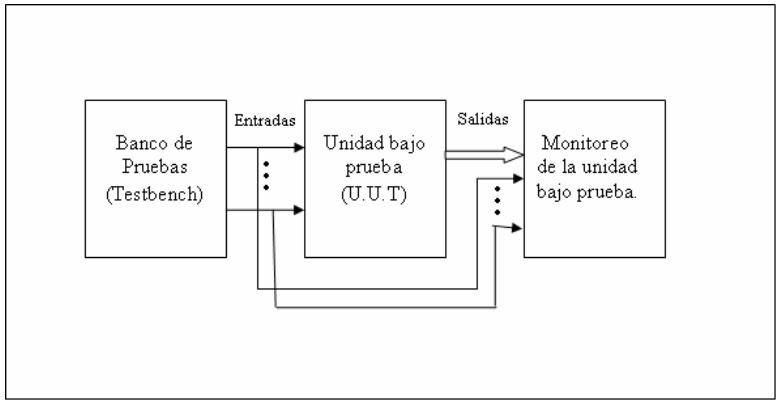


Figura 24. Simulación mediante testbench.

- Una vez probadas cada una de las entidades de forma independiente se procedió a la unión de las mismas por medio de un diagrama esquemático, para luego realizar la implementación física en la FPGA y comprobar su correcto comportamiento.

Experiencia #1

1.1. Unidad de control para ascensor de 4 pisos con memoria.

En esta experiencia se desea realizó una introducción en el mundo del diseño digital haciendo uso de una plataforma de desarrollo provista con una FPGA y con varios periféricos para tal fin, esto con fines de aplicar conocimientos de la asignatura Sistemas Digitales I, adquirir experiencia en el uso tanto de dicha plataforma, sus herramientas CAD-EDA asociadas, así como en el uso de el lenguaje de descripción de hardware VHDL. Esta práctica hace uso de estructuras básicas (combinacionales, secuenciales) a modo similar a como se haría en una práctica de dicha asígnatura, para poder comprender el modo de trabajo y poder luego extrapolarlo a sistemas de mayor complejidad.

Descripción:
En esta experiencia se desea diseñar un sistema digital que emule la dinámica de un ascensor de 4 pisos con memoria básico. En la figura 25 observamos el panel propuesto para el ascensor de 4 pisos, los mismos por simplicidad se corresponden a los periféricos presentes en la tarjeta de desarrollo Spartan 3E Starter Kit.

Panel de Ascensor consta de los siguientes periféricos:

- Cuatro (4) pulsadores para realizar llamada, uno por cada piso (P0 a P3), los cuales se corresponden a los 4 pulsadores presentes en la tarjeta Spartan 3E Starter Kit.
- Cuatro (4) LEDs indicadores de llamada, uno por cada piso (P0 a P3).
- Un Indicador de Siete Segmentos presentando el piso donde se encuentre la cabina del ascensor (Piso Actual) agregado vía puerto de expansión y proto-board a la tarjeta de desarrollo.

Panel de Sensores y Motores:

- Cuatro (4) switches para indicar la presencia de cabina, uno por cada piso (S0 a S3), los cuales se corresponden a los 4 switches de la tarjeta de desarrollo.
- Un indicador (Led) Puerta Abierta (PA).
- Un indicador (Led) Puerta Cerrada (PC).
- Un LED indicador para Motor de Ascensor Subiendo (MU).
- Un LED indicador para Motor de Ascensor Bajando (MD).

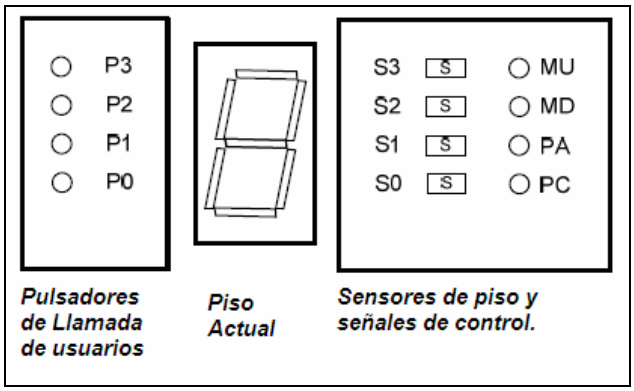


Figura 25. Panel de control ascensor, Sensores de piso y Señales de control

Básicamente, el sistema deberá reaccionar adecuadamente a las llamadas realizadas en el Panel de Ascensor una vez accionados uno o más pulsadores P0 a P3 por un usuario, moviendo la cabina por medio del Motor a través de las señales MU (Movimiento Ascendente) y MD (Movimiento Descendente) como lo haría cualquier ascensor convencional.

Para tal propósito, una vez realizada una llamada al ascensor, se deberá en secuencia cerrar las puertas de la cabina, movilizarla hasta llegar al piso de destino y abrir las puertas para permitir la salida de los usuarios, permaneciendo así hasta que haya un próximo llamado. El ascensor detectará que se ha alcanzado un determinado piso cuando se active uno de los sensores S0 a S3 por medio de un pulso, dependiendo del

piso a donde llegue. El indicador siete segmentos de Piso Actual mostrará en qué piso se encuentra el ascensor a medida que se desplace la cabina.

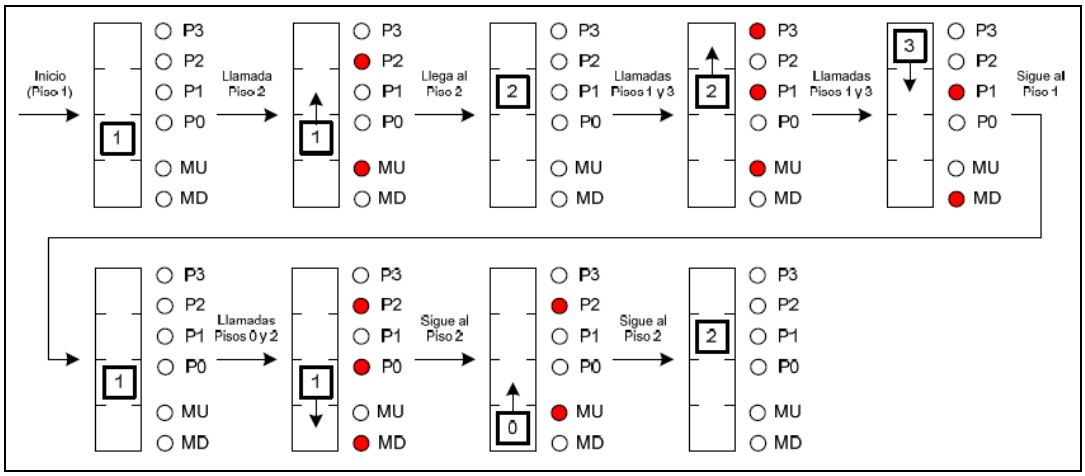


Figura 26. Ejemplo de secuencia memorización de pisos.

El ascensor deberá tomar en cuenta y almacenar todas las llamadas a todos los pisos. Para ello, el ascensor deberá decidir cuál será el piso más próximo al cual deberá dirigirse dependiendo de la dirección de movimiento que haya tomado el ascensor durante la última llamada, es decir, si por ejemplo el ascensor atendió una llamada determinada de un piso inferior a donde estaba, éste deberá mantenerse respondiendo prioritariamente a las llamadas de pisos inferiores hasta haber satisfecho todas las llamadas en esa dirección. Así mismo, si por el ascensor atendió una llamada determinada de un piso superior, deberá mantenerse respondiendo a todas las llamadas de pisos superiores (en la figura 26 observamos un ejemplo del comportamiento).

Para poder llevar a cabo el diseño propuesto se dividió el diseño en un conjunto de entidades más pequeñas, a continuación se explica cada una de las mismas:

1.1.1. Entidad validación estado de sensores de piso.

Esta entidad básicamente se encarga de eliminar los rebotes en los switches provistos por la tarjeta de desarrollo que simulan los sensores de posición en cada piso para evitar comportamientos erróneos en todas las demás entidades del sistema, así mismo se encarga de solo permitir a la salida combinaciones validas de posiciones de piso (tabla 9), no se tomaron en cuenta estados de alarma o falla de sensores. (Figura 27)

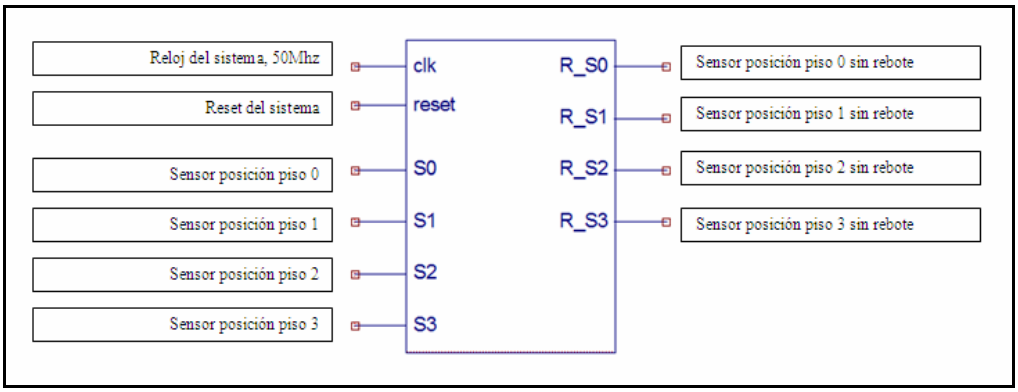


Figura 27. Entidad validación estado de sensores

Tabla 9. Salidas sin rebote permitidas en entidad validación de sensores.

Piso	Salidas sin rebote permitidas S_int=S0,S1,S2,S3
Piso 0	1000
Piso 1	0100
Piso 2	0010
Piso 3	0001

1.1.2. Entidad Memoria.

Esta entidad describe en VHDL básicamente un registro realizado con flip/flop tipo D para cada pulsador de piso, donde cada flip/flop D tiene su entrada de data colocada a 1 lógico, y su entrada de reloj a un pulsador, de manera que al ser accionado al primer flanco de subida retiene (memoriza) la data en la salida. Este esquema se repite para los 4 pulsadores presentes en la tarjeta de desarrollo que simulan los pulsadores de llamadas de los usuarios, cada uno de estos flip/flop tiene un reset independiente que permite borrar la memoria de algún piso solo cuando la señal ena_ok indica que hubo parada en el piso en específico. (Figura 28)

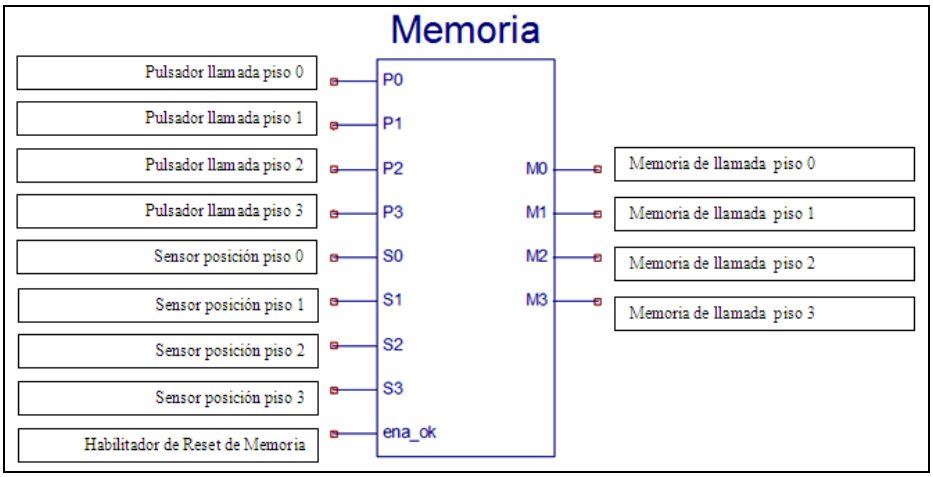


Figura 28. Entidad memoria, entradas y salidas.

1.1.3. Entidad prioridad sentido.

Como su nombre lo indica esta entidad se encarga de verificar e indicar en sus salidas si existe prioridad en el sentido de movimiento (si sube o baja) y en las paradas que realiza el ascensor en dicho sentido. El ascensor sólo atiende las llamadas que se encuentran en un sentido dado, sólo al terminar todas las llamadas de usuarios en un sentido específico, si existen llamadas adicionales en el sentido opuesto las atenderá repitiendo la secuencia, e ignorando las que ocurran en sentido opuesto. (Figura 29)

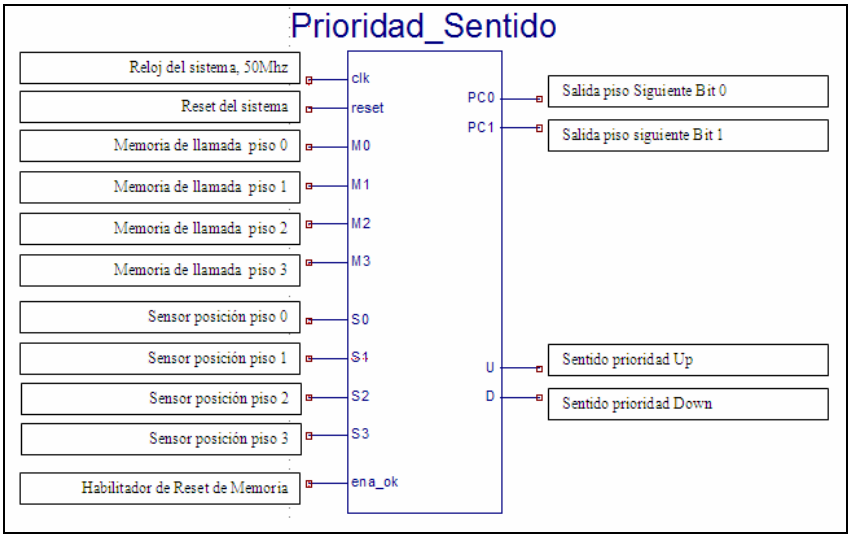


Figura 29. Entidad prioridad sentido, entradas y salidas.

Para llevar a cabo dicho comportamiento se realiza en VHDL la programación de una máquina de estado modelo Moore, la cual se especifica en la figura 30.

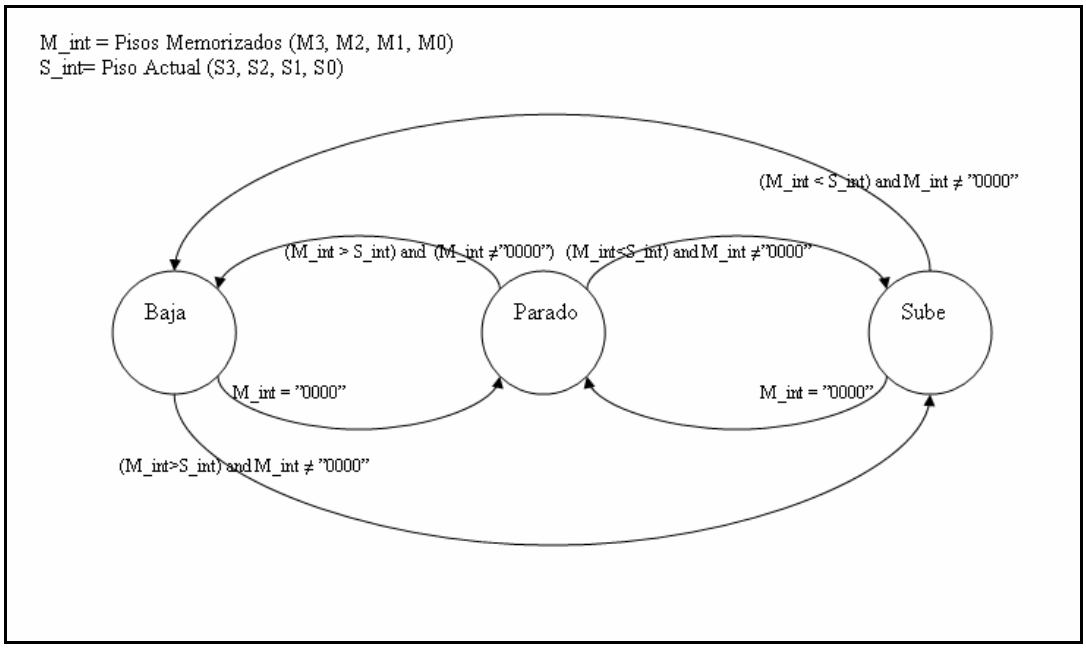


Figura 30. Maquina de estado que describe el comportamiento de la entidad Prioridad sentido.

1.1.4. Entidad secuencial principal.

Esta entidad es la que se encarga de llevar a cabo el rol del comportamiento del ascensor. Básicamente esta toma la información de las entradas del piso presente dado por los sensores de piso (S0, S1, S2, S3), piso siguiente (PC0, PC1) y el sentido de prioridad de movimiento sube, baja (U, D) o parado (Ver figura 31). La misma realiza una operación (secuencia) a la vez según la información en su entrada.

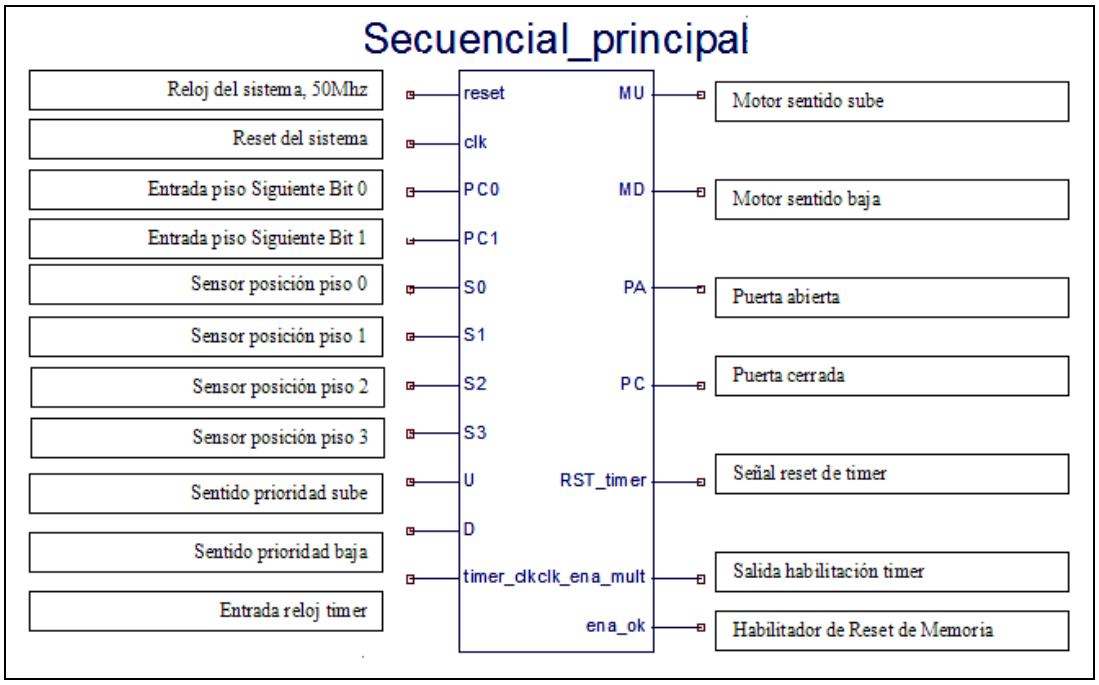


Figura 31. Entidad secuencial principal, entradas y salidas

La secuencia se puede expresar según la siguiente maquina de estados (Figura 32):

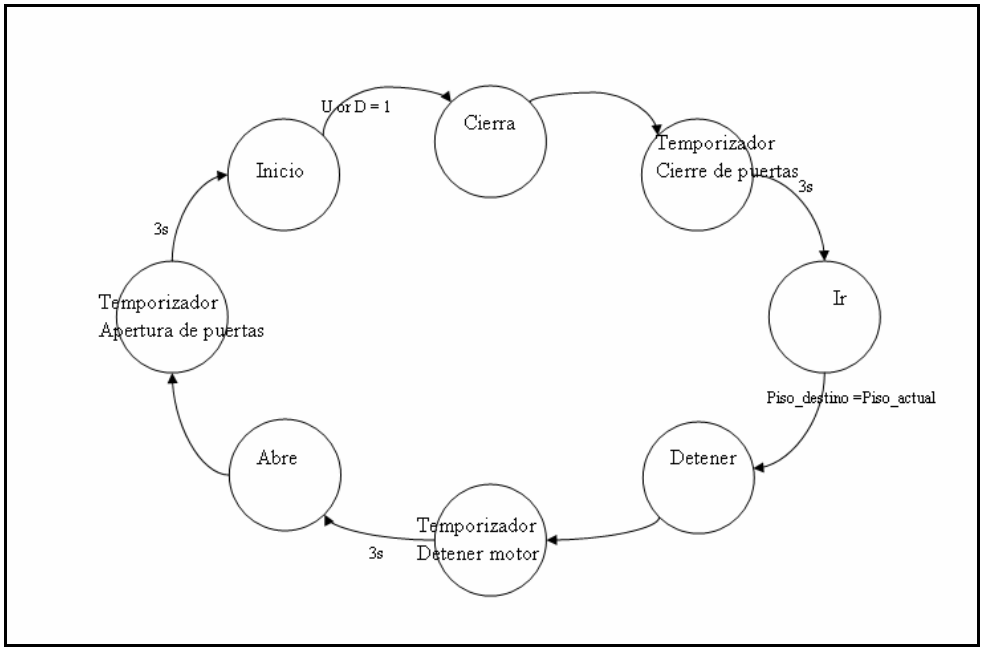


Figura 32. Diagrama de estado de la entidad Secuencial Principal.

1.1.5. Entidad Codificador-display.

Esta entidad realiza la implementación de un combinacional que se encarga de codificar los valores lógicos de los sensores de posición a su respectivo código a 7 segmentos. Cabe destacar que se utilizaron 2 puertos de expansión (J1 y J2) para agregar un display 7 segmentos a la tarjeta de desarrollo, sin hacer uso de el LCD provisto por la misma ya que no se necesitaban caracteres alfanuméricos (sólo se requiere un solo digito). (Figura 33)

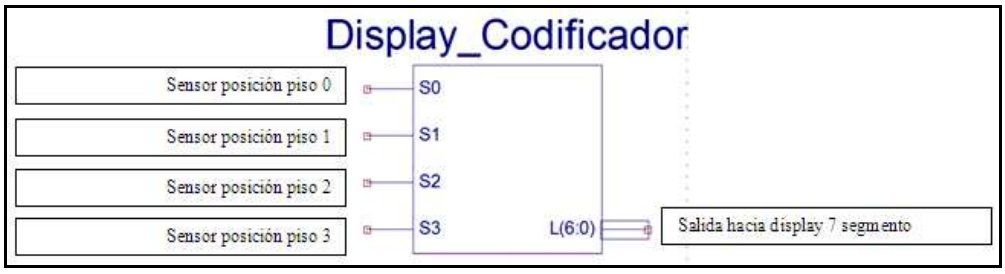


Figura 33. Entidad codificador display, entradas y salidas.

1.1.7. Entidad Temporizador.

Esta entidad se encarga de dividir la frecuencia del reloj del sistema de 50Mhz a aproximadamente un 1Hz, para esto se procedió a realizar un contador de décadas similar en comportamiento al ampliamente conocido CD4017 de la familia CMOS (en la figura 34 llamado counters_1), y por medio de una cascada de estos dispositivos dividir la frecuencia del reloj del sistema, al obtener un reloj de 1s de periodo, luego haciendo uso de otro contador es posible obtener diversas temporizaciones, en este diseño entre se utilizaron 3 bits que permiten desde 0s hasta 7s programables según se requiera en cada estado, por simplicidad se coloco en 3 segundos.

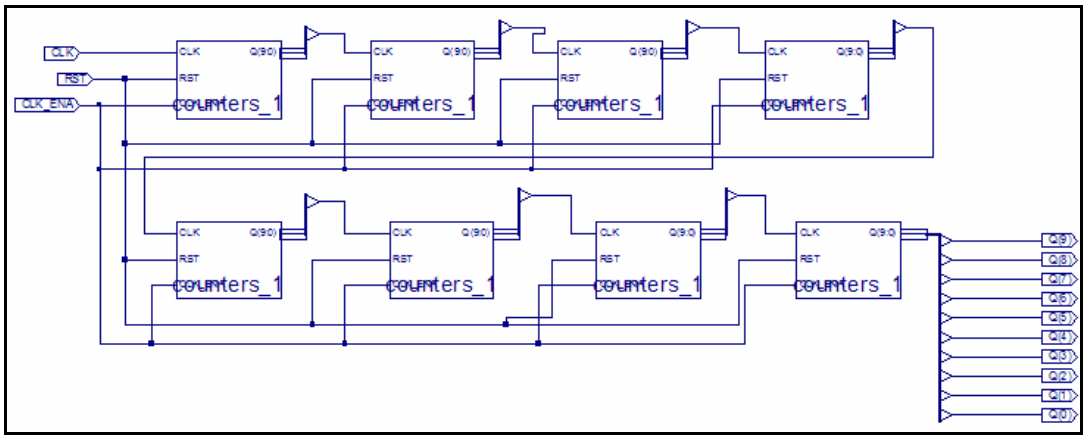


Figura 34. Cascada de contadores de décadas.

El esquema de la figura 34 se puede representar mediante una sola entidad la cual es la utilizada finalmente en el esquemático final (ver figura 35).

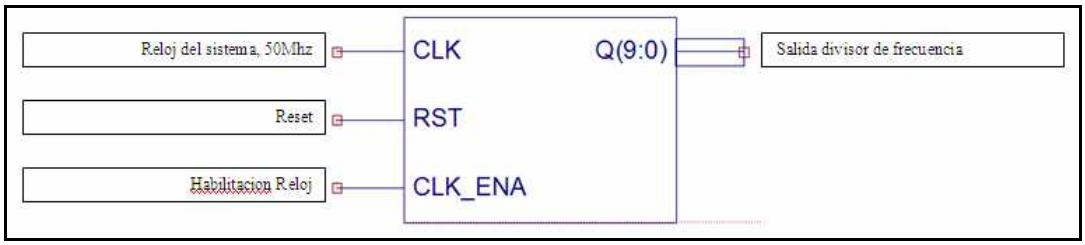


Figura 35. Entidad divisor de frecuencia.

En la figura 36 observamos el comportamiento de la entidad divisor de frecuencia.

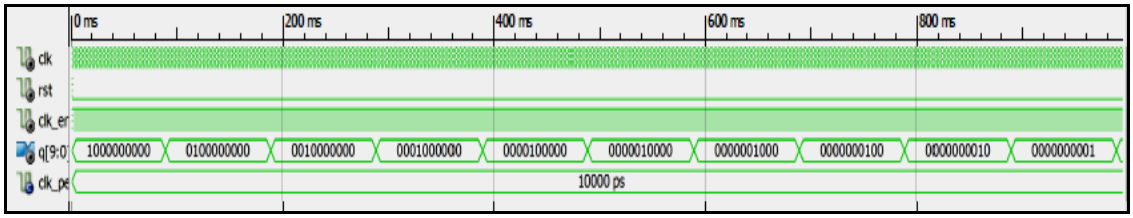


Figura 36. Simulación de la entidad divisor de frecuencia.

1.1.7. Implementación y puesta a prueba del sistema.

Luego de simular y probar todas las entidades por separado, se procedió a unirlas mediante un esquemático (ver figura 37). En la figura 38 se observa una simulación realizada con todas las entidades en funcionamiento. Cabe destacar que en dicha simulación se redujeron los tiempos de los temporizadores (Timer), ya que no se podría observar la evolución de las distintas máquinas de estado involucradas.

En dicha simulación se observa una secuencia completa donde inicialmente se realizan una serie de llamadas por parte de los usuarios, que están memorizadas (ver M_int en figura 38). A medida que las máquinas de estado evolucionan van limpiando cada memoria al cumplir con cada llamada, y su evolución cumple la condición de prioridad según el sentido.

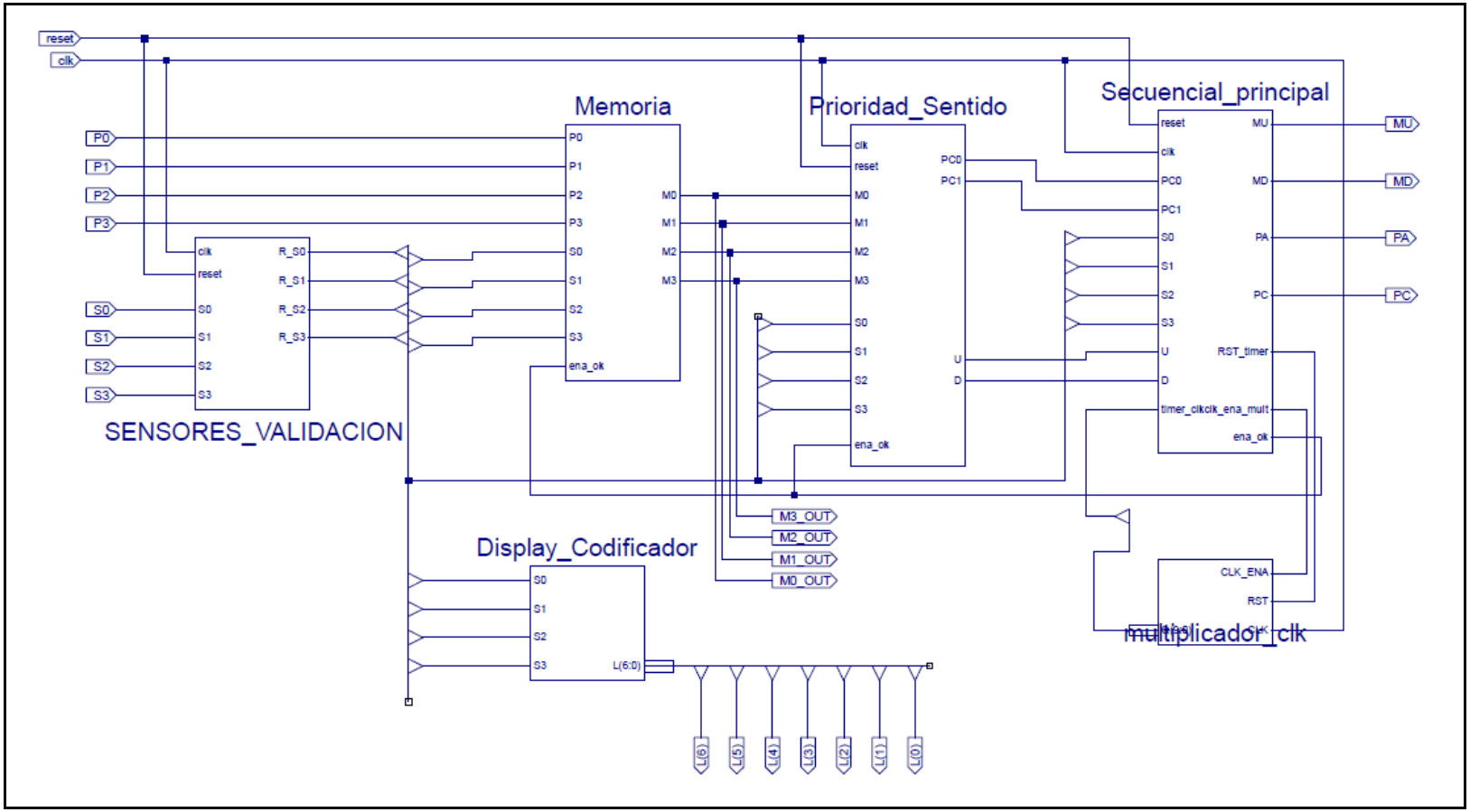


Figura 37. Diagrama esquemático interconexión de todas las entidades de la unidad de control del ascensor de 4 pisos

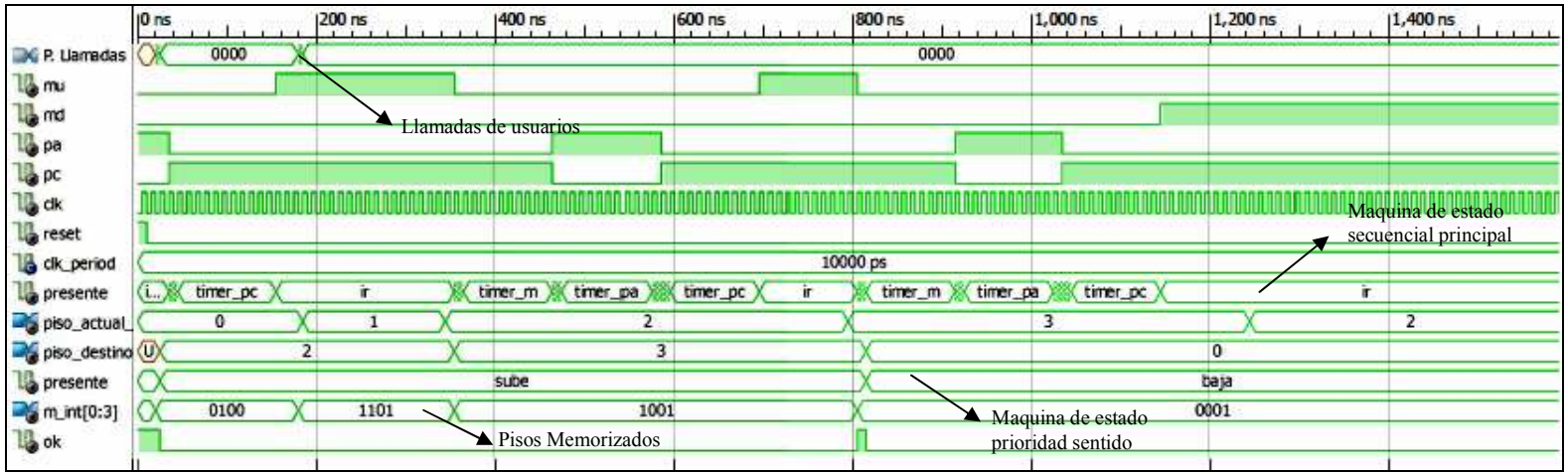
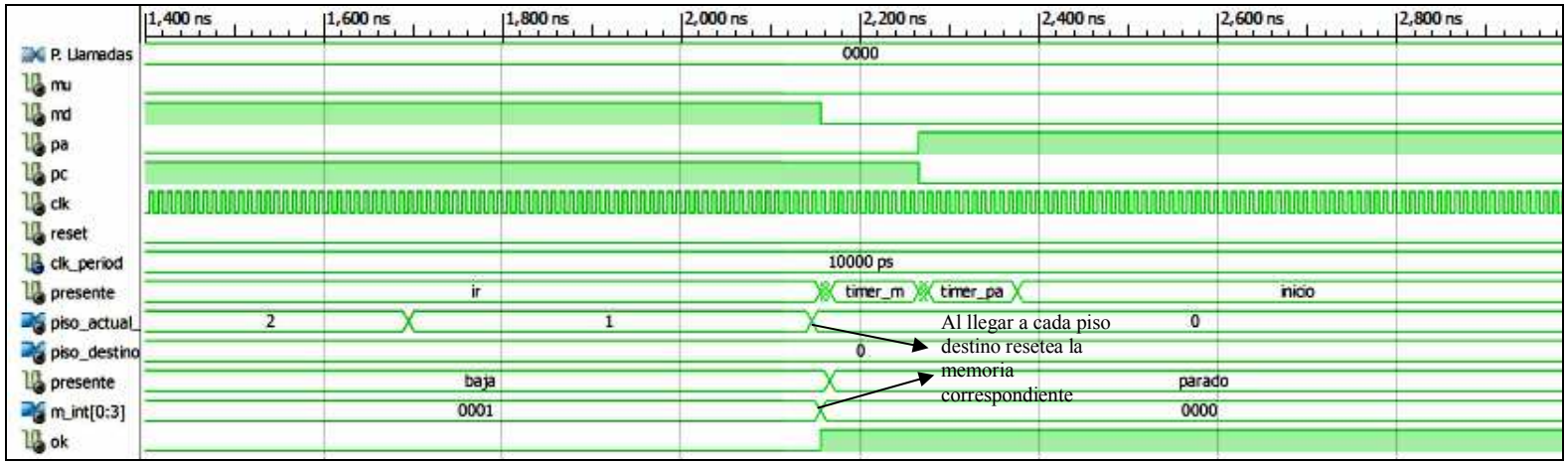


Figura 38. Simulación unidad de control ascensor, todas las entidades en funcionamiento



Continuación diagrama de tiempo

Experiencia #2

1.2. Filtro FIR de coeficientes constantes.

En esta experiencia se desea realizar una introducción en el mundo del procesamiento digital de señales haciendo uso de una plataforma de desarrollo provista con una FPGA y con varios periféricos para tal fin, esto con fines de aplicar conocimientos de materias afines, adquirir experiencia en el uso tanto de dicha plataforma, sus herramientas CAD-EDA asociadas, así como en el uso de el lenguaje de descripción de hardware VHDL.

Descripción:
Un filtro FIR (Finite Impulse Response) es descrito mediante la siguiente ecuación:

$$Y(n) = \sum_{0}^{K} C_k \cdot X(n-k) \qquad \text{Ec.1}$$

O su equivalente en términos de función de transferencia en Z :

$$Y(z) = \sum_{K=0}^{M-1} C_K \cdot Z^{-K} = C_0 + C_1 \cdot Z^{-1} + C_2 \cdot Z^{-2} + ... + C_{M-1} \cdot Z^{-(M-1)} \qquad \text{Ec.2}$$

Los filtros FIR son estructuras no recurrentes que se caracterizan por tener una respuesta a un impulso con un número finito de coeficientes. El algoritmo FIR consiste en una suma ponderada de un cierto número de muestras digitales previas multiplicadas por sus respectivos coeficientes de filtrado C_K.
Una manera de implementar un filtro FIR es a través de una estructura llamada forma directa, la cual se observa a continuación en la figura 39.

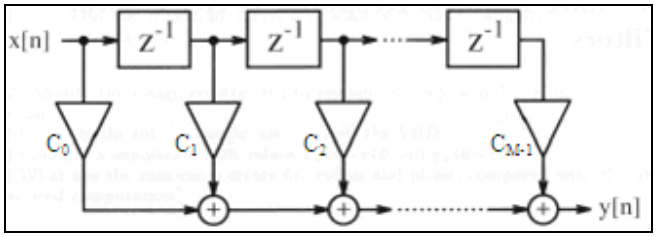


Figura 39. Estructura directa de un filtro FIR

En la figura 40 observamos un diagrama de bloques del filtro FIR a desarrollar, cada uno de estos bloques se corresponde con una entidad en el desarrollo del filtro en lenguaje VHDL.

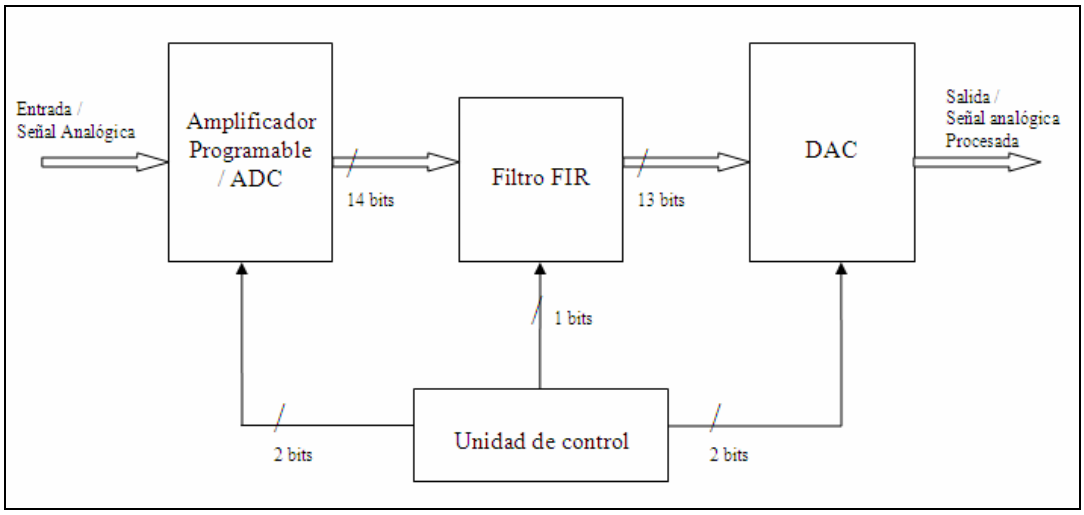


Figura 40. Diagrama de bloques filtro FIR.

Se toma como base en el desarrollo de la experiencia propuesta coeficientes para implementar un filtro Pasa-bajo con frecuencia de corte de 2,5Khz, luego si se desea con solo cambiar los coeficientes y manteniendo igual el mismo hardware se puede implementar filtros pasa-alto, pasa banda etc.

Se toma como frecuencia de muestreo Fs=50Khz, dicha frecuencia cumple a cabalidad el teorema de muestreo de Nyquist (al menos $Fs \geq 2 \cdot F$ max de la señal a muestrear) para el rango de frecuencias durante la experiencia.

En esta práctica se realizo el filtro con 8 etapas o Taps, lo que equivale a un filtro de estructura directa con 8 multiplicadores (ver figura 39).

Se establece 1V como amplitud máxima de entrada y salida, para no exceder los rangos en los convertidores analógico-digital (ADC), y los convertidores digital-analógico (DAC).

A continuación se irá detallando cada una de estas entidades:

1.2.1. Amplificador programable / Conversión analógica-digital:

Esta entidad comprende la interfaz de entrada al filtro propuesto, lleva a cabo tareas de adecuación y digitalización de la señal analógica de entrada. (Ver figura 41)

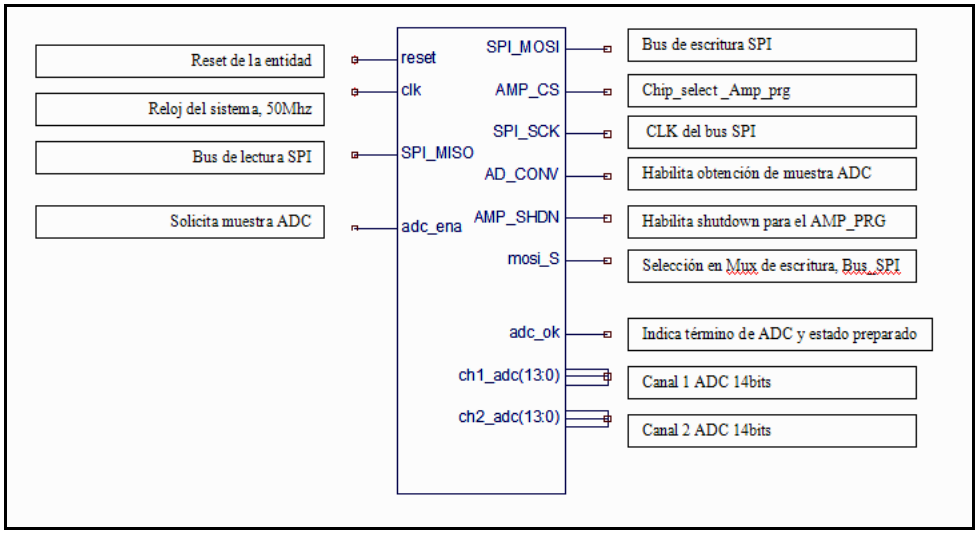


Figura 41. La entidad del amplificador programable/ADC, entradas y salidas.

Para llevar a cabo las tareas antes descritas se dispone de dos periféricos que se encuentran interconectados en conjunto con la FPGA en la tarjeta de desarrollo Spartan 3E starter kit, estos comprenden el pre-amplificador programable (Amp_prg) modelo LTC6912-1 y un conversor analógico-digital (ADC) modelo LTC1407A-1,

ambos del fabricante Linear Technology, en la figura 42. A continuación observamos un diagrama simplificado de su interconexión.

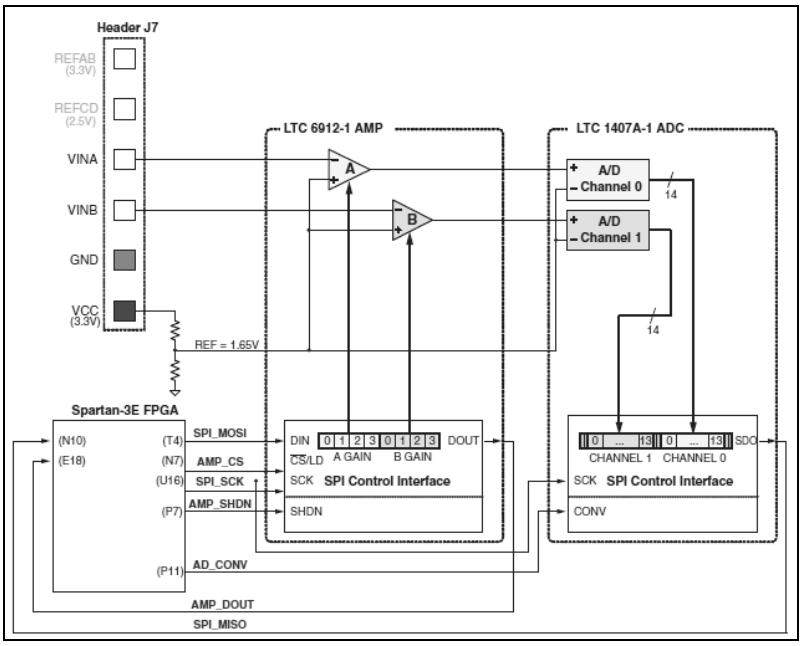


Figura 42. Estructura de interconexión del Amp_Prg y ADC del Spartan-3E (Extraído de Spartan-3E Starter Kit Board User Guide)

Especificaciones básicas del amplificador programable LTC6912-1-DUAL:

- Dos entradas (A y B), rango de voltaje +/- 1.25 V con Ref =1.65 V
- Inversor, ganancia con rango programable de -1 a -100
- Programación SPI (Interfaz periférica serial)
- Frecuencia máxima de programación bus SPI de 10 MHz

La siguiente tabla describe los valores a programar para el LTC6912-1 en el bus SPI y los niveles Min y Max en la entrada según la ganancia escogida, se implementó el diseño con ganancia de -1.

Tabla 10. Codificación binaria y niveles min y max soportados según la ganancia utilizada en el amplificador programable. (Extraído de Spartan-3E Starter Kit Board User Guide)

Gain	A3	A2	A1	A0	Input Voltage Range	
	B3	B2	B1	B0	Minimum	Maximum
0	0	0	0	0		
-1	0	0	0	1	0.4	2.9
-2	0	0	1	0	1.025	2.275
-5	0	0	1	1	1.4	1.9
-10	0	1	0	0	1.525	1.775
-20	0	1	0	1	1.5875	1.7125
-50	0	1	1	0	1.625	1.675
-100	0	1	1	1	1.6375	1.6625

Si se fija la ganancia del amplificador en -1 para tener un rango de excursión amplio tenemos la siguiente función de transferencia de la figura 43.

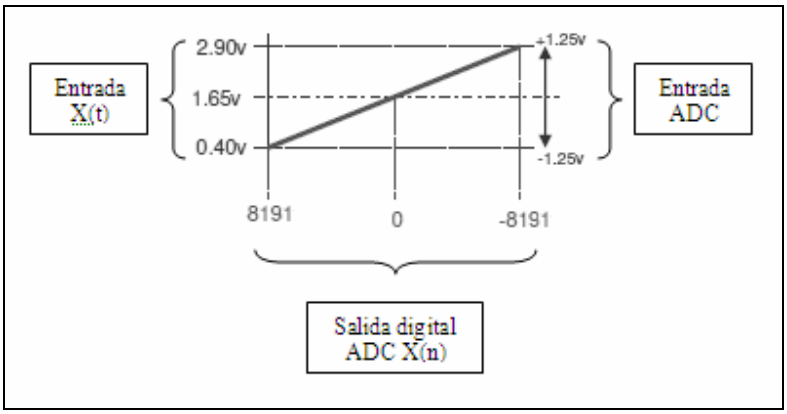


Figura 43. Función de transferencia para el conjunto amplificador programable/ ADC para una ganancia fijada en -1, ambos con nivel de referencia de 1,65 V.

Especificaciones básicas del ADC LTC1407A-1-DUAL:

- Dos entradas (A y B), rango de voltaje +/- 1.25 V con Ref=1.65 V
- 14 bits, con signo en complemento a 2

- Programación SPI (Interfaz periférica serial)
- Frecuencia máxima en bus SPI de 50 MHz
- Trama de programación de 34 ciclos de reloj, velocidad máxima de captura 1.5 Msps en cada canal.

La ecuación que describe su comportamiento es:

$$Muestra[13:0] = Ganancia \cdot \frac{(V_{in} - 1{,}65V)}{1{,}25} \cdot 8192 \qquad \text{Ec. 3}$$

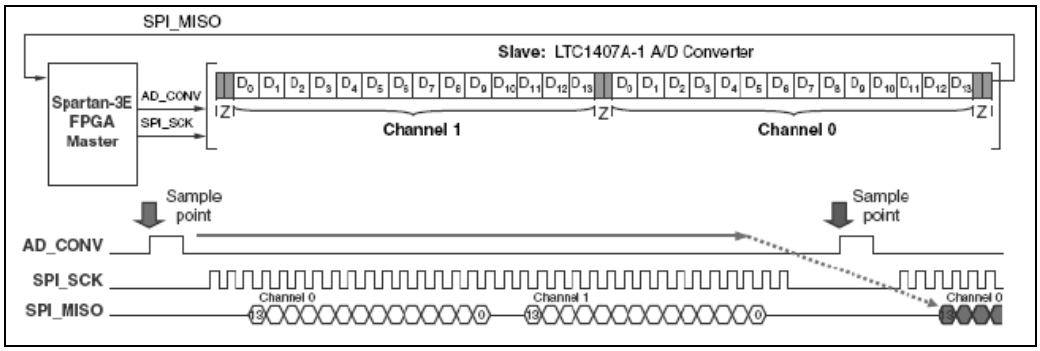


Figura 44. Estructura de la trama y diagrama de tiempo simplificado para adquisición de datos del ADC LTC1407A (Extraído de Spartan-3E Starter Kit Board User Guide)

A continuación se muestraen la figura 45 un diagrama de flujo que describe el comportamiento en la entidad Amp_Prg/ADC.

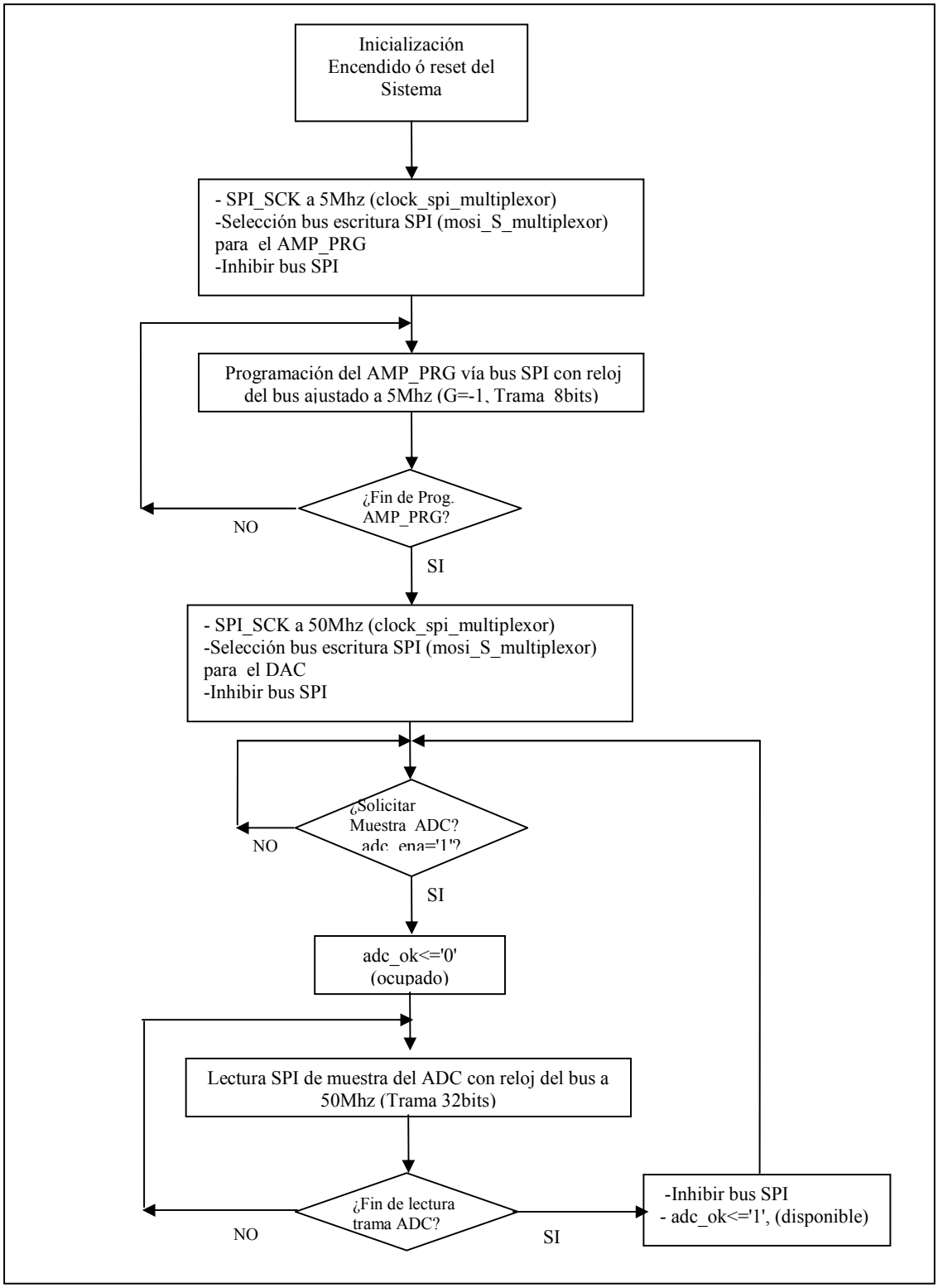


Figura 45. Diagrama de flujo implementado en la entidad Amp_Prg/ADC

1.2.2. Filtro FIR

Esta entidad comprende el filtro propiamente dicho. El diagrama equivalente del sistema se muestra en la figura 46.

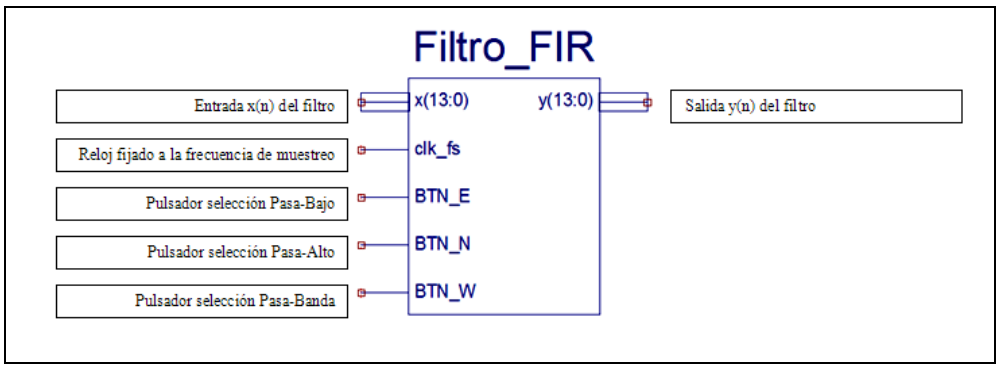


Figura 46. Entidad Filtro FIR, entradas y salidas

En la misma se lleva a cabo en VHDL explícitamente la ecuación 2 que describe un filtro FIR de coeficientes constantes. Para ello se utiliza básicamente un registro de desplazamiento con 8 etapas que conforman los taps del filtro, con 14 bit de tamaño de palabra (bus x(n), y(n)), el registro de desplazamiento implementa un FIFO (First In, First Out) con la muestra actual y las anteriores. A cada flanco positivo de la señal clk_fs se realizan las multiplicaciones entre el valor contenido en cada tap del filtro, con su correspondiente coeficiente constante, todos estos valores son acumulados y contemplan para un instante dado la salida y(n) del filtro, luego se realiza el desplazamiento y actualización de las muestras en el FIFO.

Estas tareas se describen en VHDL ver anexo 3, en particular al hacer la síntesis se hacen uso de los bloques multiplicadores embebidos de la FPGA Spartan 3E.

Para poner a prueba la entidad se realizó una simulación mediante un banco de prueba (Testbench) en la herramienta Isim de Xilinx introduciendo como señal de

entrada un escalón, y de igual manera se realizó una simulación en Matlab haciendo uso de la herramienta Fdatool para el mismo filtro. En las figuras 47 y 48 se puede apreciar la correspondencia entre ambas simulaciones, se obtuvo la misma secuencia y amplitud en los impulsos de la respuesta escalón {0, 185, 278, 390, 513, 636, 748, 841, 1026}.

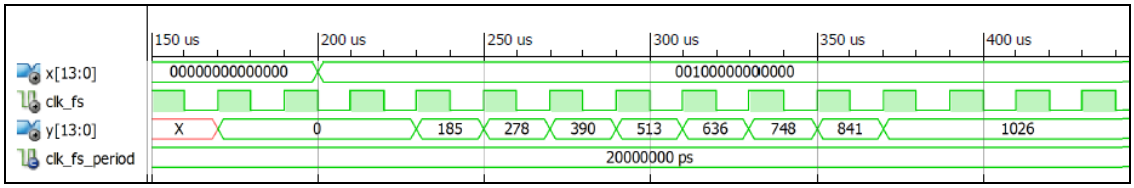


Figura 47. Simulación en la herramienta Isim del filtro pasa-bajo ante una señal escalón.

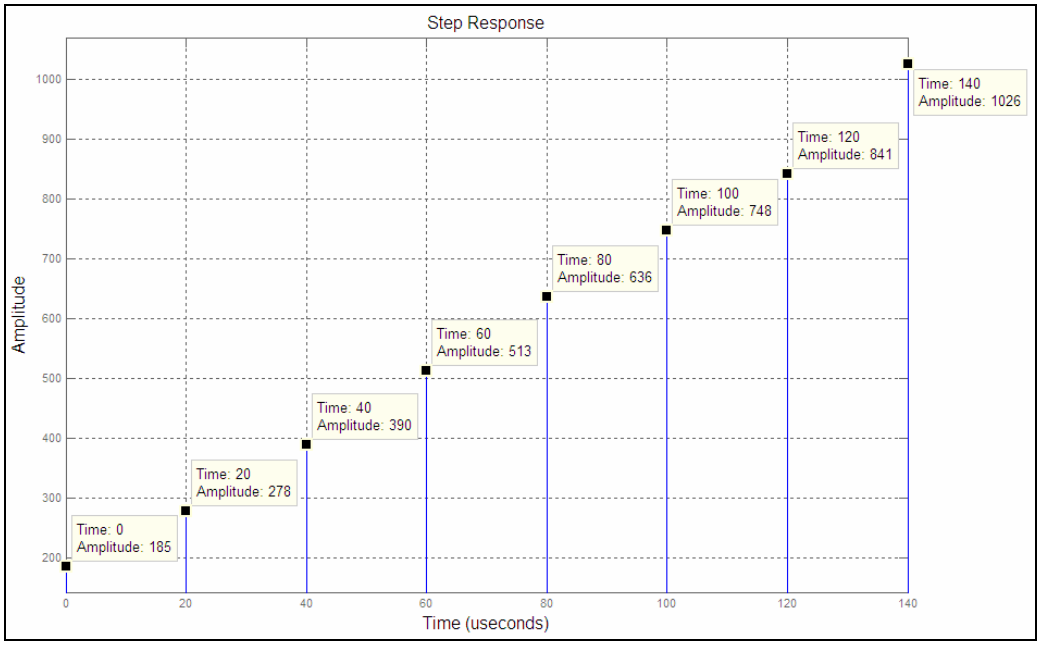


Figura 48. Simulación en Matlab, repuesta ante un escalón filtro pasa-bajo.

Cabe destacar que las multiplicaciones y divisiones se están realizando tomando en cuenta solo la parte entera del resultado, para otros valores se va a tener un error de truncamiento, ya que no se implementó aritmética de punto flotante.

1.2.2.1. Obtención de los coeficientes.

Para la obtención de los coeficientes del filtro se utilizó la herramienta de Matlab, Fdatool. Ésta permite diseñar, poner a prueba y obtener la respuesta para entradas básicas como Impulso, escalón, obtener la repuesta en frecuencia entre otros.

La herramienta antes comentada entrega los coeficientes en formato punto flotante y normalizado, por lo que para implementar el filtro se llevó un proceso de redondeo y escalamiento para poder adaptar los valores entre las distintas etapas del filtro y obtener mayor precisión en una aritmética binaria entera.

En la tabla 11 se muestra los coeficientes obtenidos para un filtro FIR para una frecuencia de muestreo de Fs=50Khz y una frecuencia de corte Fc= 2,5Khz con su error porcentual cometido al redondear dichos valores.

Tabla 11. Coeficientes del filtro FIR pasa-bajo y error porcentual debido a el redondeo, Fs=50Khz , Fc= 2,5Khz

Coeficientes normalizados	Coeficientes*1024	Redondeo	Error %
0,18024019	184,565955	185	0,235170667
0,090500281	92,67228816	93	0,353624418
0,109345538	111,9698308	112	0,026944028
0,11991399	122,791926	123	0,169452503
0,11991399	122,791926	123	0,169452503
0,109345538	111,9698308	112	0,026944028
0,090500281	92,67228816	93	0,353624418
0,18024019	184,565955	185	0,235170667

El escalamiento de los coeficientes se lleva a cabo para poder expresarlos como números enteros y ganar precisión, luego de efectuar todas las operaciones en el algoritmo FIR se realiza una única división binaria, cuyo resultado corresponde al valor a la salida y(n).

En la figura 49 se aprecia la respuesta en frecuencia (Magnitud) para los coeficientes entregados por Matlab en punto flotante y para los coeficientes utilizados en el filtro pasa-bajo propuesto (tabla 11) en donde se observa que presentan prácticamente el mismo comportamiento (se solapan), también se observa la frecuencia de corte aproximada (-3dB).

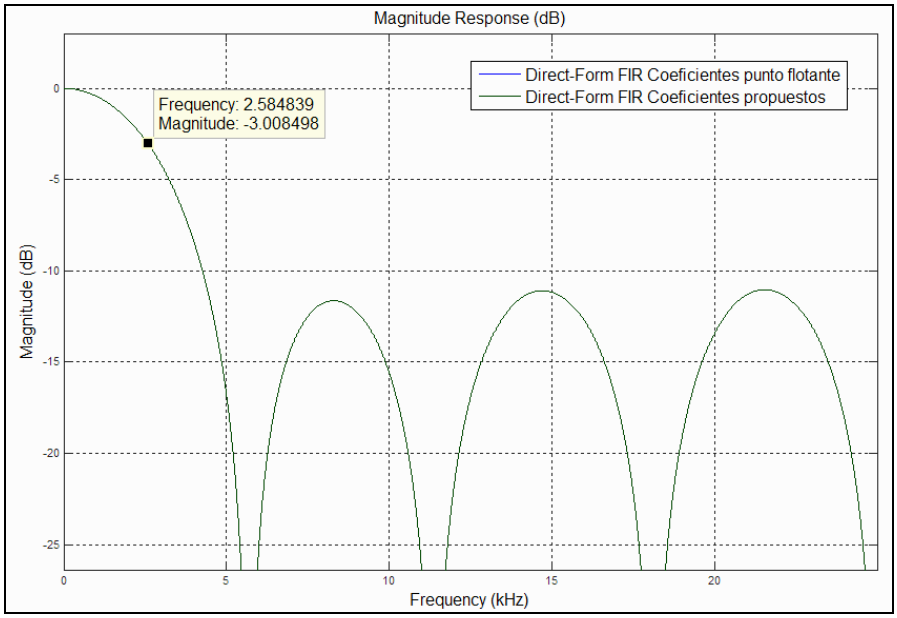


Figura 49. Respuesta en frecuencia (Magnitud) para el filtro FIR pasa-bajo, comparación de la respuesta entre coeficientes punto flotante y los propuestos para la implementación.

1.2.3. Conversión digital-analógica.

En esta entidad se lleva a cabo el proceso trasformar las señales digitales procesadas por el filtro y(n) a nuevamente una señal analógica, en la figura 50 observamos dicha entidad con sus respectivas entradas y salidas.

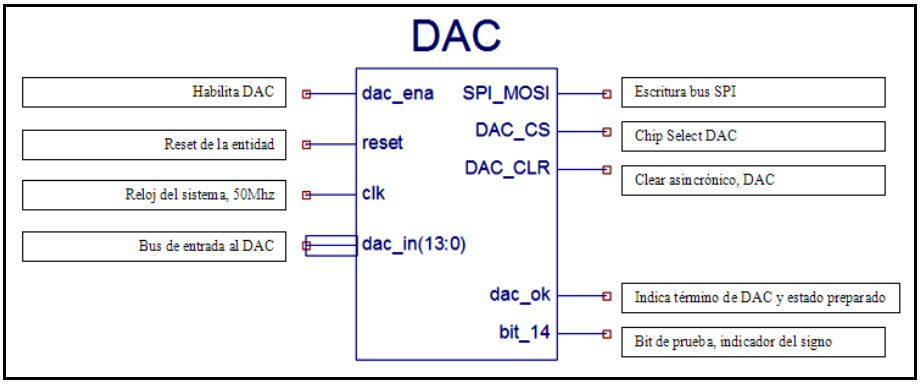


Figura 50. Entidad DAC, entradas y salidas

Para llevar a cabo esta tarea se emplea el conversor digital-analógico (DAC) provisto por la plataforma de desarrollo usada, el cual es un LTC2624 del fabricante Linear Technology.

En la figura 51, observamos un esquema del conexionado entre la FPGA Spartan 3e y el LTC2624 con sus principales señales de interconexión.

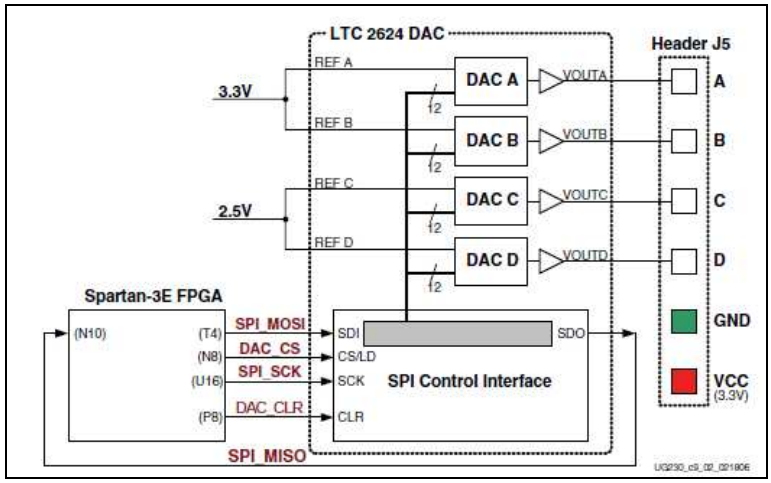


Figura 51. Diagrama de interconexión entre la FPGA Spartan 3E y el LTC2624 (Extraído de Spartan-3E Starter Kit Board User Guide).

Como los DAC utilizados son sin signo (sin complemento a 2) y como no se dispone de fuente simétrica en la tarjeta de desarrollo, se hizo uso de 2 conversores DAC en modo diferencial, uno para la parte positiva de la señal y otro para la parte negativa,

esto se llevó a cabo en VHDL mutiplexando entre los dos DAC según el bit más significativo (bit 14 correspondiente al signo) de la señal y(n).

En esta práctica se hizo uso de los DACs "C","D" del diagrama figura 52, los cuales tienen un nivel de referencia de +2,5V. La ecuación que rige su comportamiento es:

$$Vout = \frac{D[11:0]}{4096} \cdot 2{,}5V \qquad \text{Ec. 4}$$

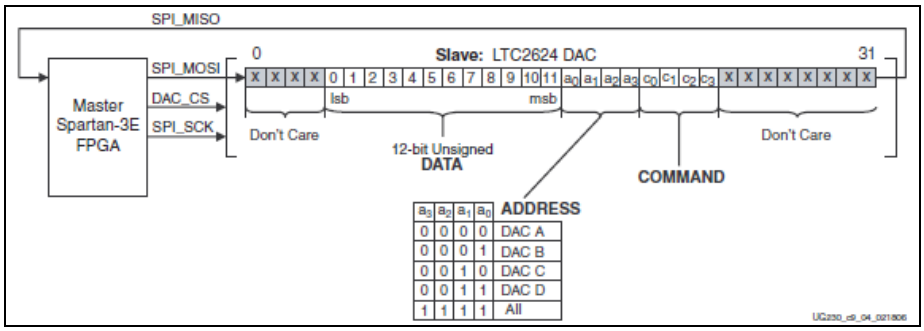


Figura 52. Estructura de la trama del LTC2624 y las diferentes señales de interconexión con la Spartan 3E (Extraído de Spartan-3E Starter Kit Board User Guide).

Luego para proceder a unir la parte negativa y la parte positiva de la señal de salida se hace uso de un amplificador diferencial común con ganancia de 1, y una fuente simétrica de 5V externa (figura 53).

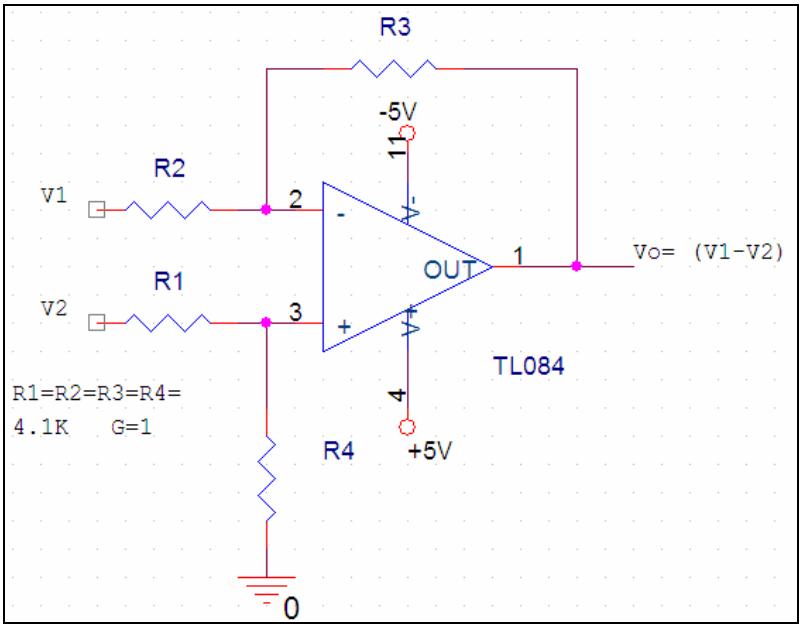


Figura 53. Amplificador diferencial utilizado para unir la salida de los DAC C y D.

En la figura 54 observamos un diagrama de flujo de las operaciones llevadas a cabo en la entidad DAC.

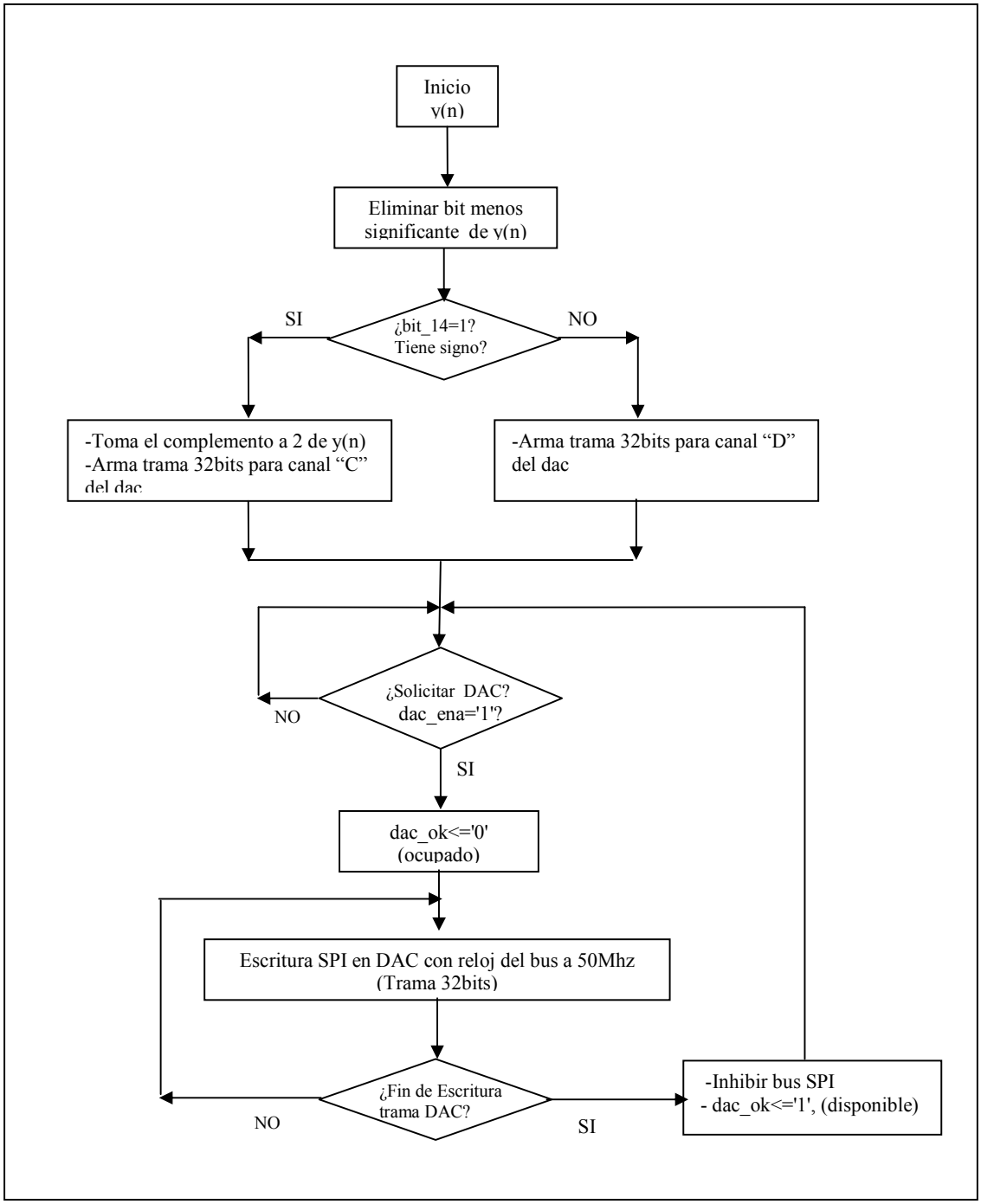


Figura 54. Diagrama de flujo operaciones realizadas en la entidad DAC.

1.2.4. Unidad de control.

Esta entidad se encarga de establecer la secuencia y control adecuado para poder poner a funcionar todas las demás entidades de forma ordenada, además de fijar la frecuencia de muestreo y desactivar otros dispositivos en la tarjeta de desarrollo que están conectados en el bus SPI con el fin de evitar una posible contención del mismo.(Ver figura 55)

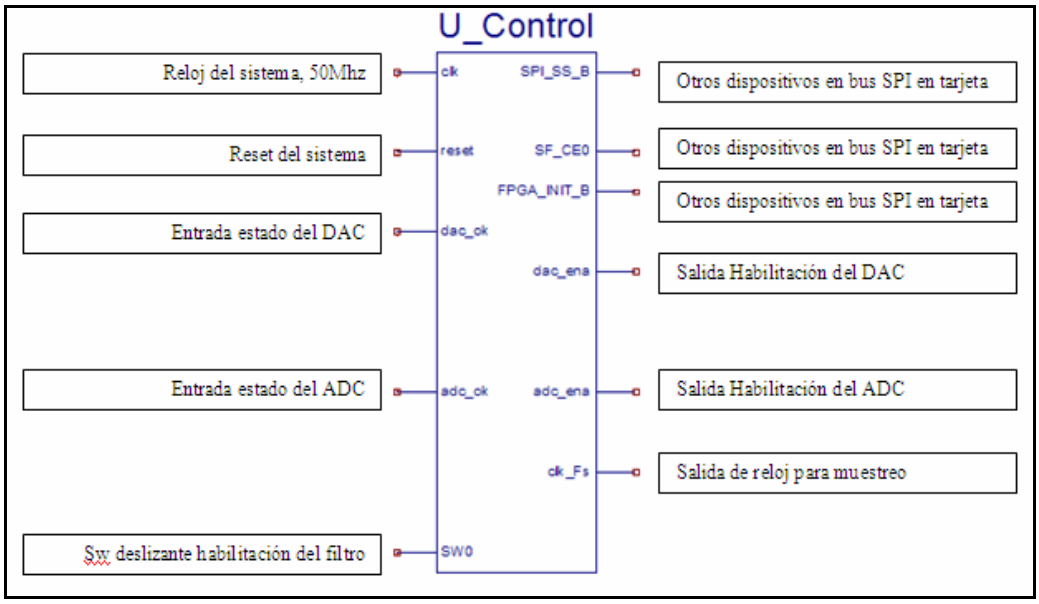


Figura 55. Entidad unidad de control, entradas y salidas.

La unidad de control ejecuta el siguiente flujo (figura 56):

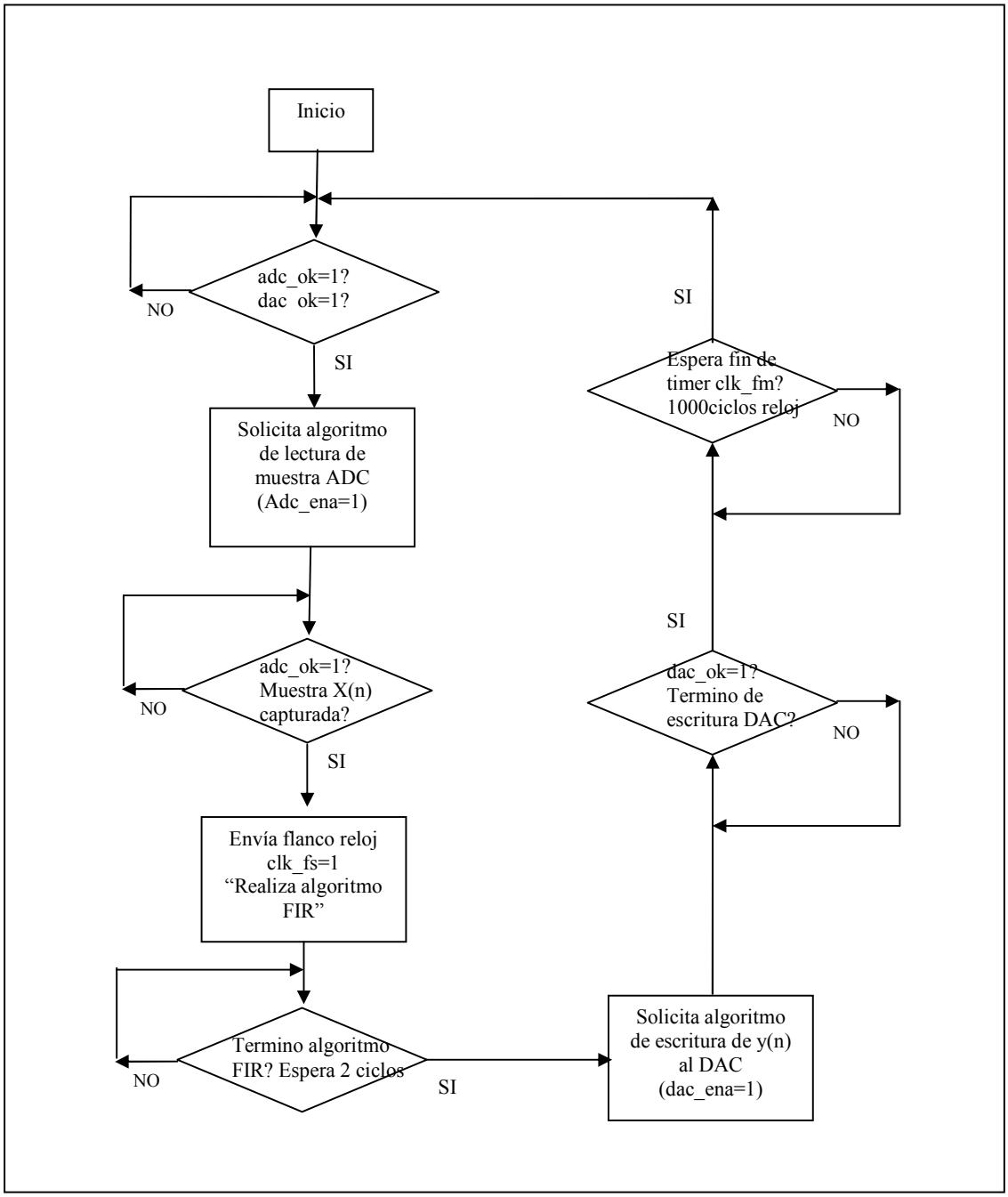


Figura 56. Diagrama de flujo comportamiento de la entidad unidad de control.

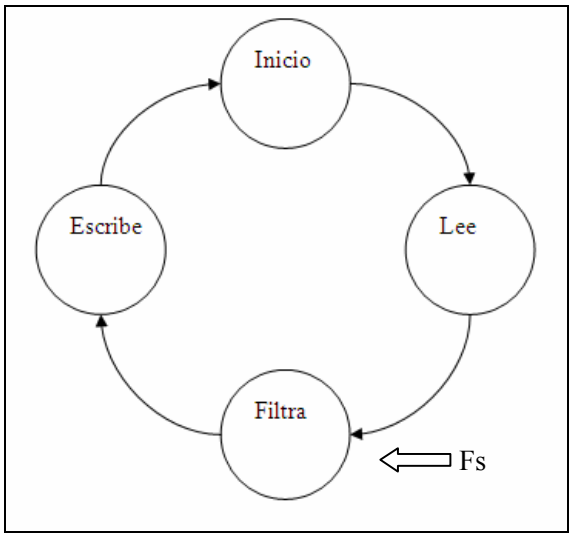


Figura 57. Secuencia de operaciones de la entidad unidad de control

En la figura 57 se observa la secuencia de operaciones que se implementa en VHDL usando una estructura de máquina de estados. Se implementa en VHDL un divisor de frecuencia, haciendo uso de un contador, este se encarga a partir de los 50Mhz del reloj del sistema, de obtener los 50Khz necesarios como referencia de la frecuencia de muestreo.

1.2.5. Implementación y puesta a prueba del sistema.

Luego de probar cada entidad de forma independiente mediante la realización de testbench con la herramienta ISE de Xilinx, se procedió a unir todas las entidades para que trabajen en conjunto, en la figuras 58 y 59. A continuación observamos el diagrama esquemático que une todas las entidades, y luego observamos una simulación del filtro con todas las entidades trabajando.

Se puede observar en la simulación los instantes en los cuales se toman las muestras para una frecuencia de muestreo de 50Khz, esta frecuencia se puede variar de requerirse, ver en el diagrama de tiempo que las entidades Amp_Prg/ADC y DAC trabajan a velocidad elevada, por lo que se observa en el ADC y DAC un estado de

disponible para la mayoría del tiempo, el tiempo entre muestra y muestra se pudiese reducir drásticamente para otros diseños de filtros, se pudiese tener una frecuencia de muestreo máxima de aproximadamente 680Khz realizando lectura y escritura.

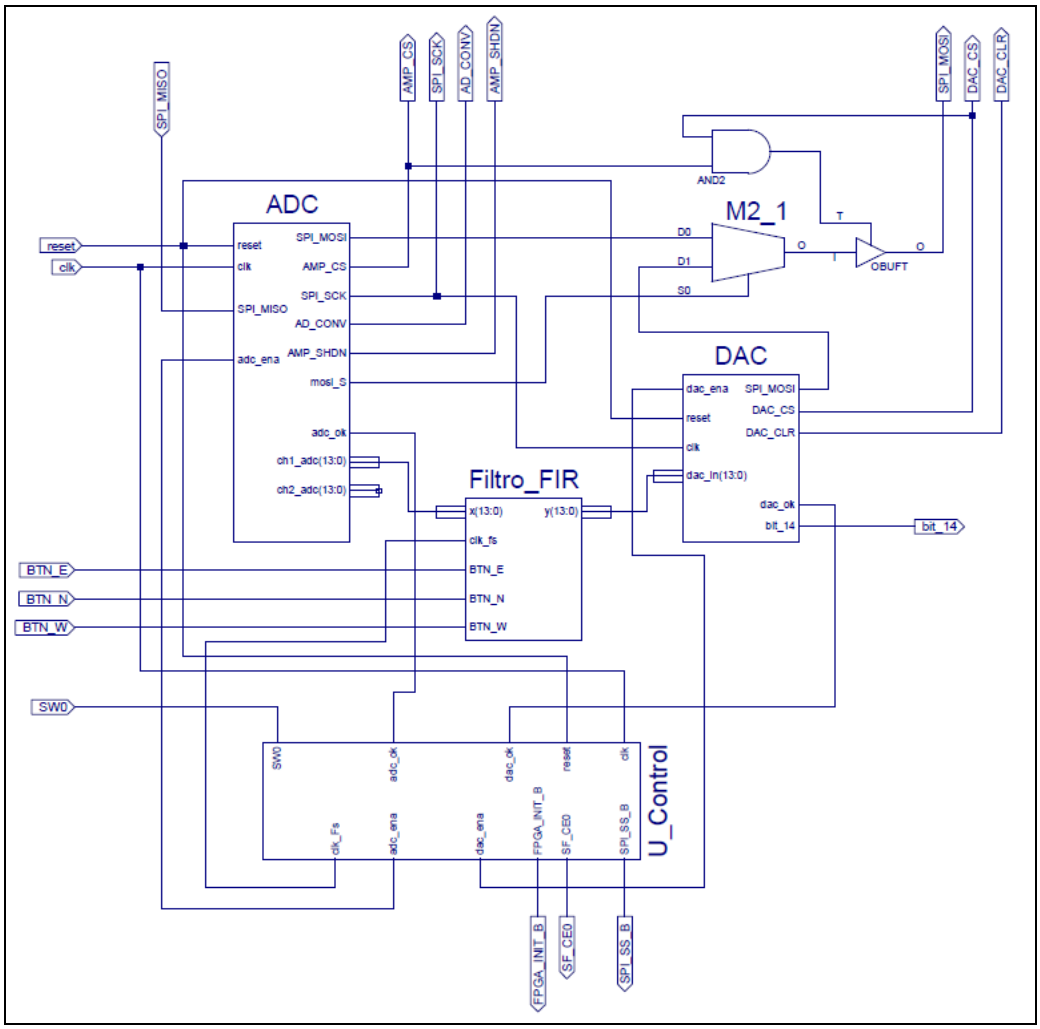


Figura 58. Diagrama esquemático, interconexión de todas las entidades que componen el filtro FIR.

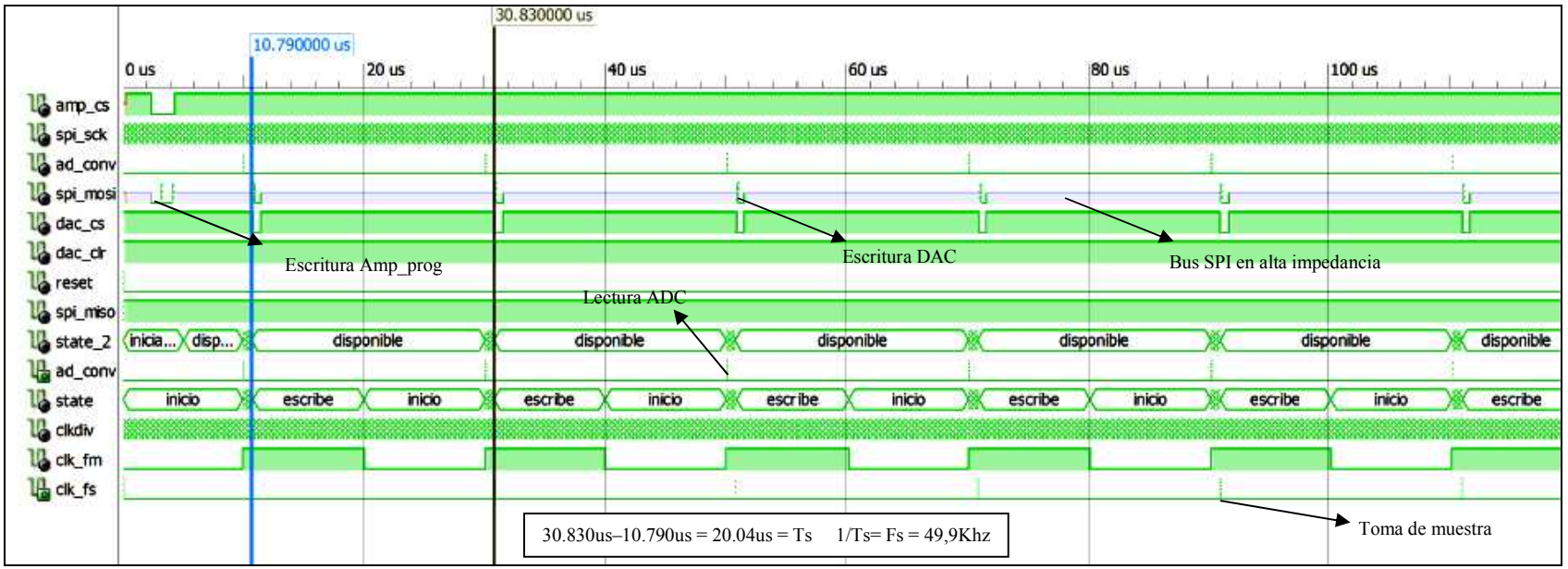


Figura 59. Simulación del filtro FIR con la herramienta Isim de Xilinx, todas las entidades en funcionamiento.

Se realizó de igual manera una simulación en Matlab utilizando la herramienta Simulink del filtro pasa bajo a fin de observar su comportamiento y tener un patrón de comparación con el filtro en físico implementado en la FPGA, para esto se simulo el comportamiento con una señal senosoidal pura de 1V de amplitud, de frecuencia variable. (Ver figuras 60,61,62,63)

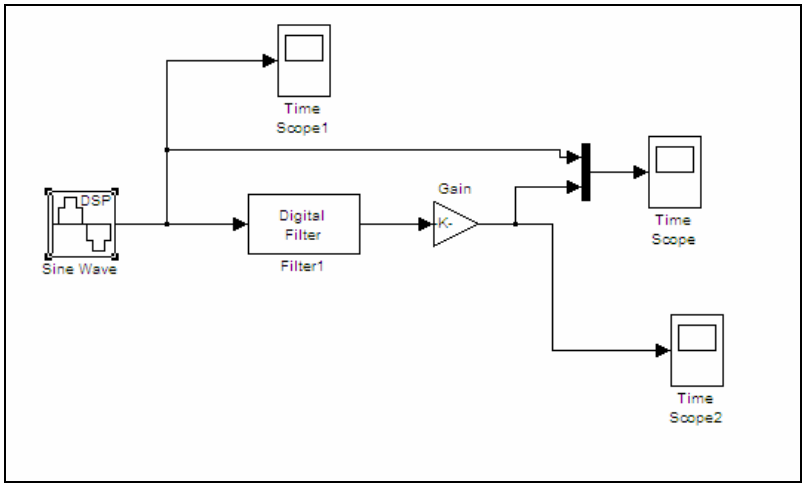


Figura 60. Esquema en Simulink (Matlab) para la simulación del filtro con una entrada senosoidal.

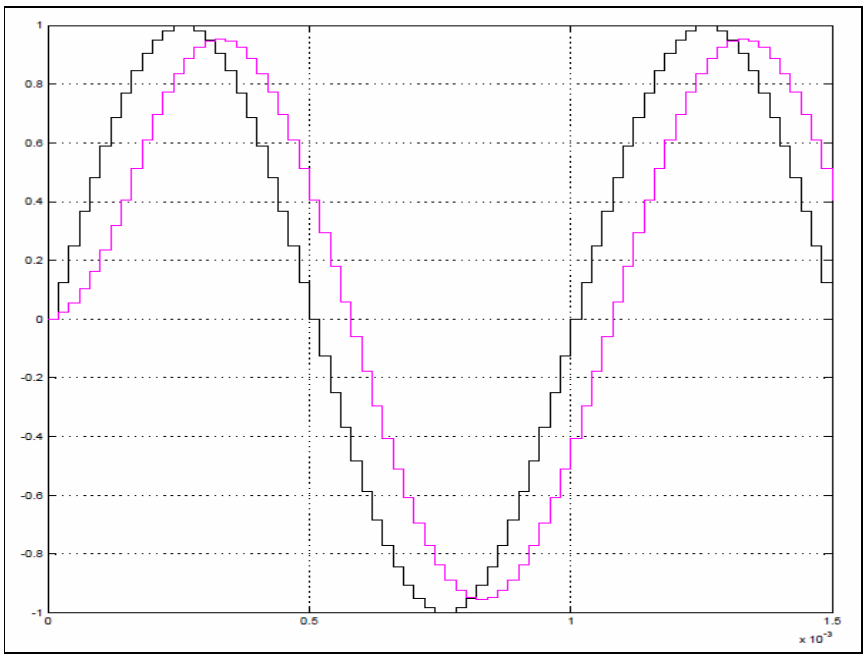


Figura 61. Simulación en Simulink (Matlab), 1V de entrada, frecuencia 1Khz < Fc

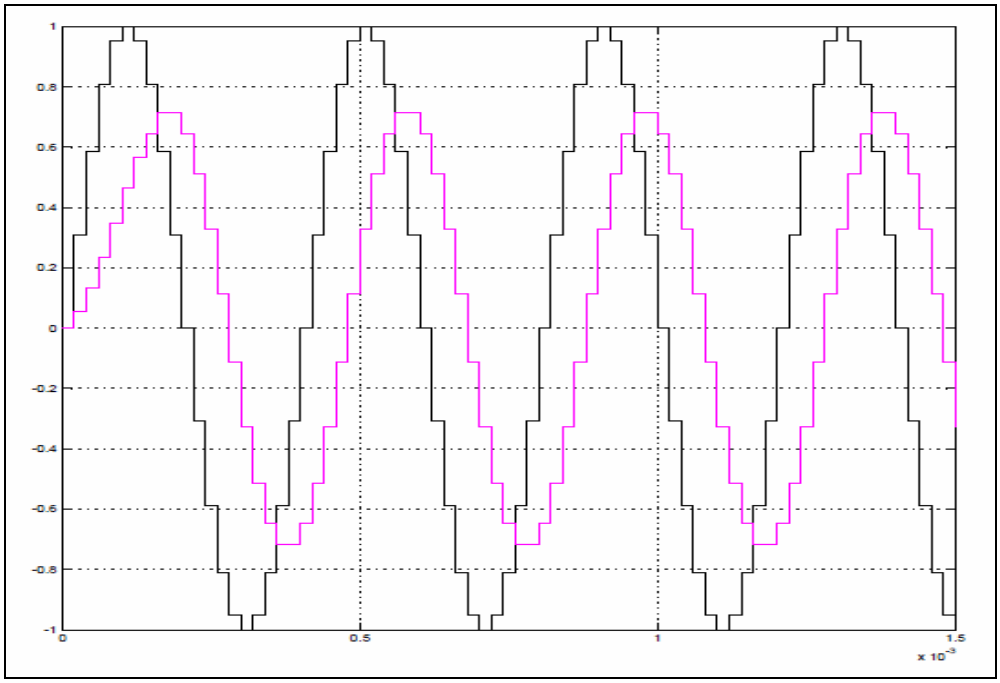


Figura 62. Simulación en Simulink (Matlab), 1V de entrada, frecuencia 2,5Khz=Fc

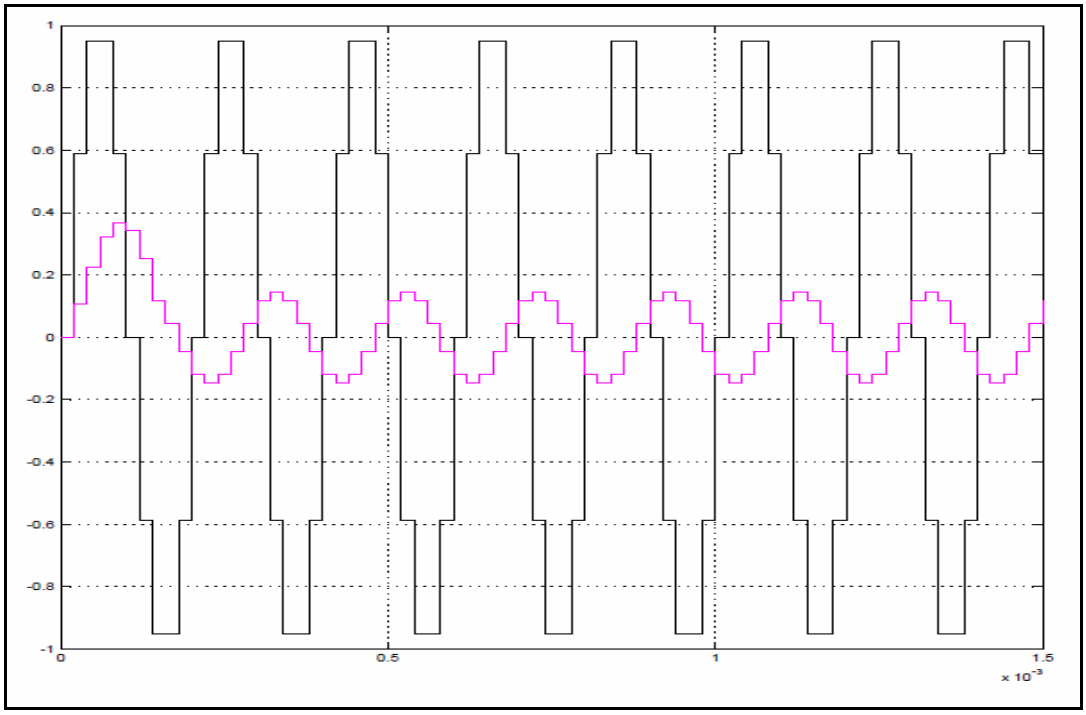


Figura 63. Simulación en Simulink (Matlab), 1V de entrada, frecuencia 5Khz > Fc

Luego se procedió a conectar la tarjeta de desarrollo a un generador de frecuencia y un osciloscopio de manera de verificar su comportamiento de forma similar a la realizada en Matlab. Para esto de igual manera se le coloco a la entrada una señal senosoidal de 1V constante desacoplada a través de un capacitor electrolítico, y se varió la frecuencia.

Para efectuar las mediciones se utilizo un osciloscopio Velleman para PC modelo PCSU1000, el canal 1 (CH1) se colocó en la entrada y el canal 2(CH2) a la salida del operacional diferenciador (figura 53) antes comentado.

A continuación se observan las siguientes capturas en el osciloscopio para valores antes y después de la frecuencia de corte (2,5Khz):

-Filtro Pasa-Bajo:

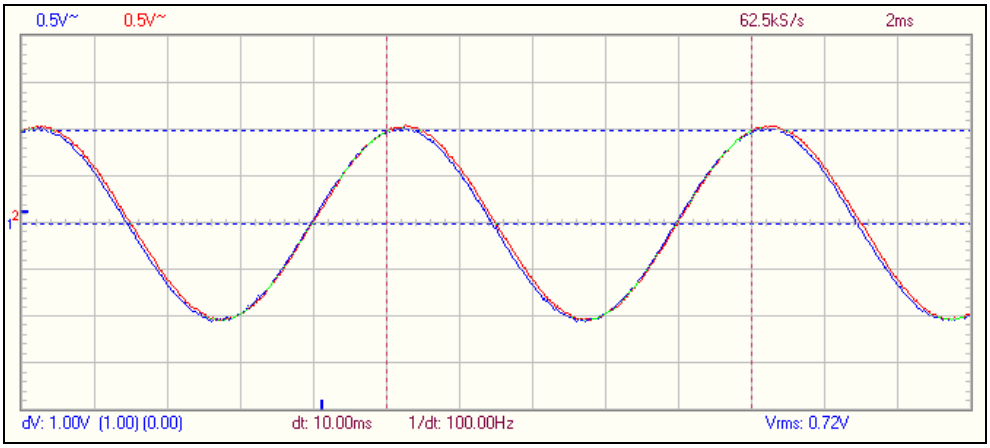


Figura 64. Filtro Pasa-Bajo, 1V de entrada, frecuencia 100Hz < FC

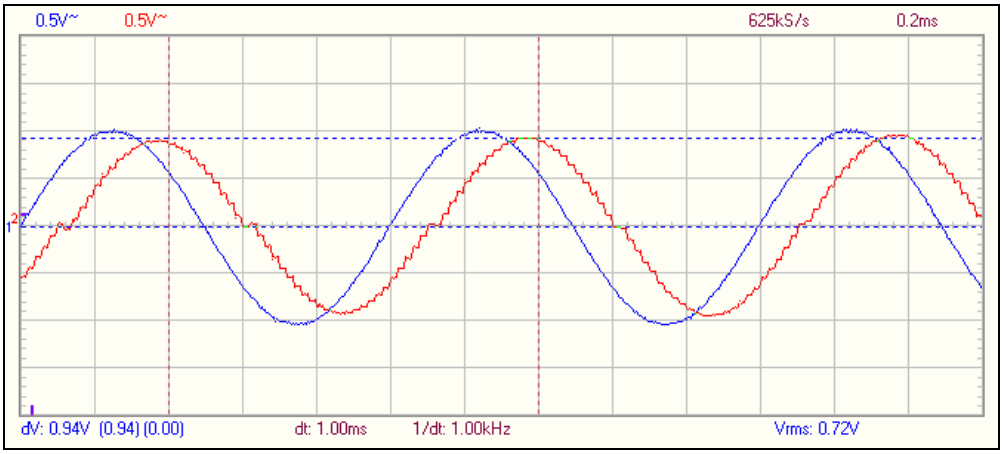


Figura 65. Filtro Pasa-Bajo, 1V de entrada, frecuencia 1KHz < FC

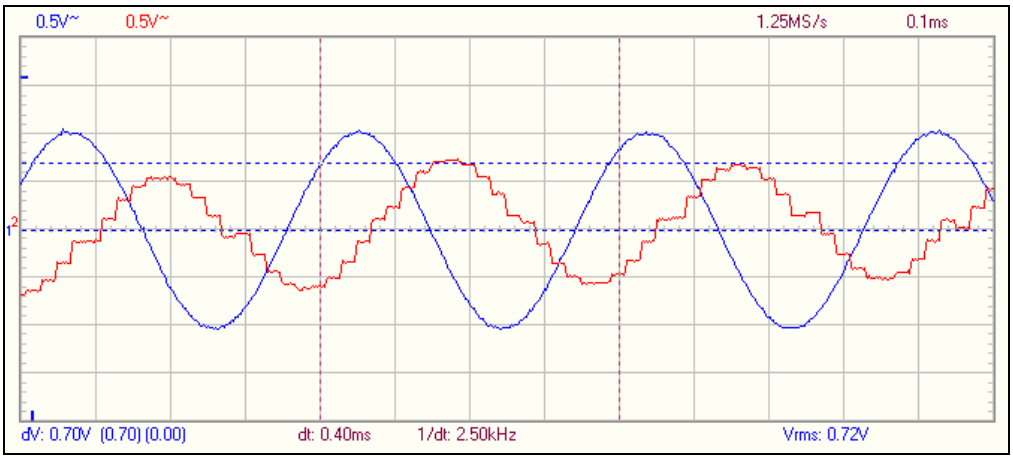


Figura 66. Filtro Pasa-Bajo, 1V de entrada, frecuencia 2,5KHz = FC

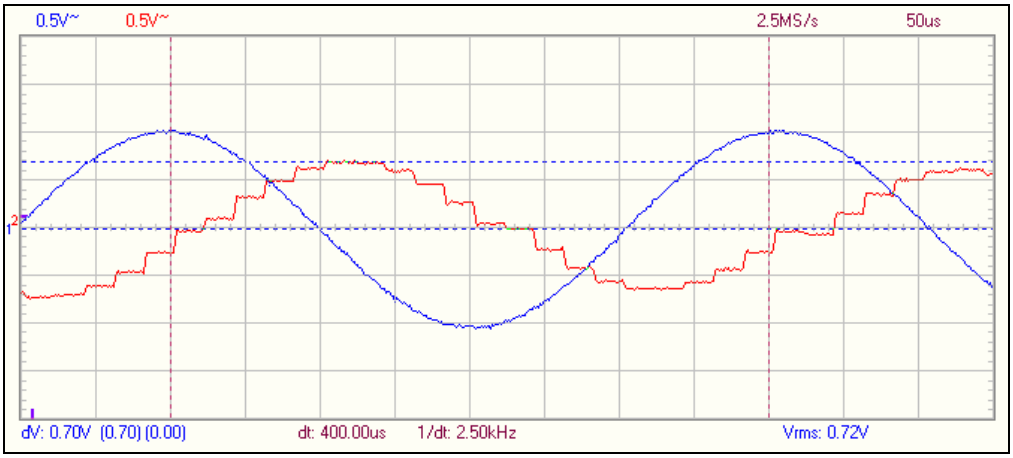


Figura 67. Filtro Pasa-Bajo 1V de entrada, frecuencia 2,5KHz = FC

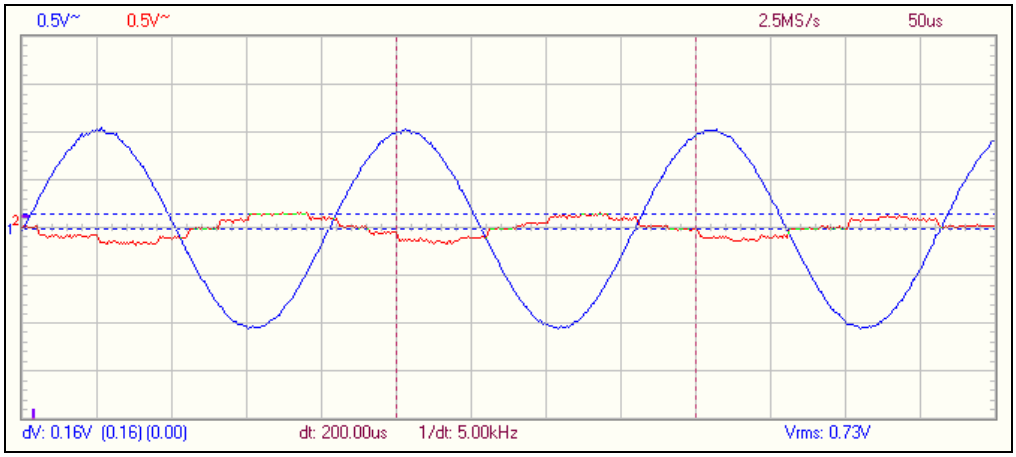


Figura 68. Filtro Pasa-Bajo, 1V de entrada, frecuencia 5KHz > FC

-Filtro Pasa-alto:

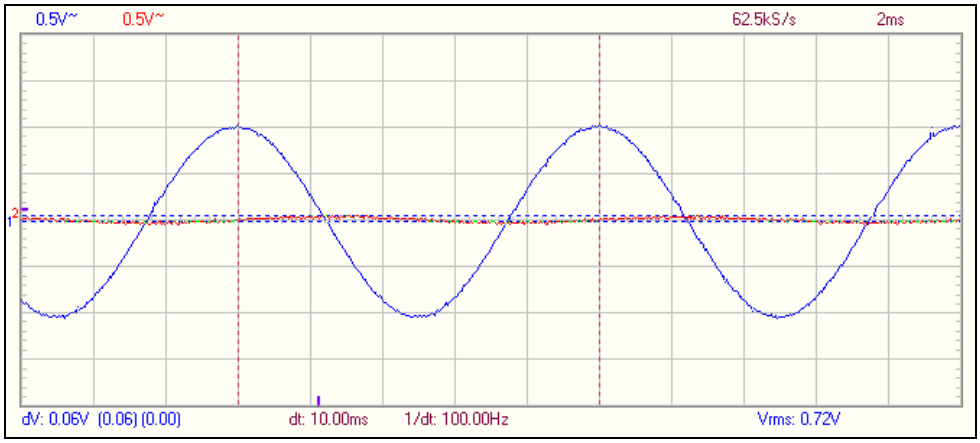


Figura 69. Filtro Pasa-Alto, 1V de entrada, frecuencia 100Hz < FC

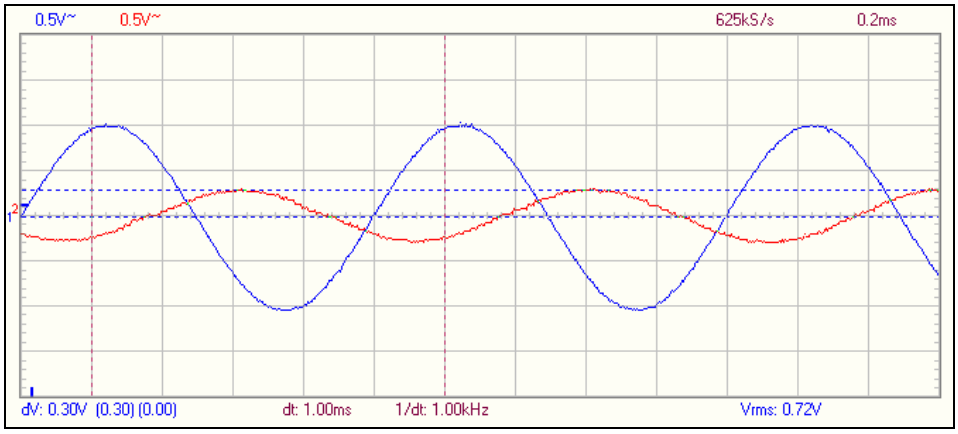


Figura 70. Filtro Pasa-Alto, 1V de entrada, frecuencia 1KHz < FC

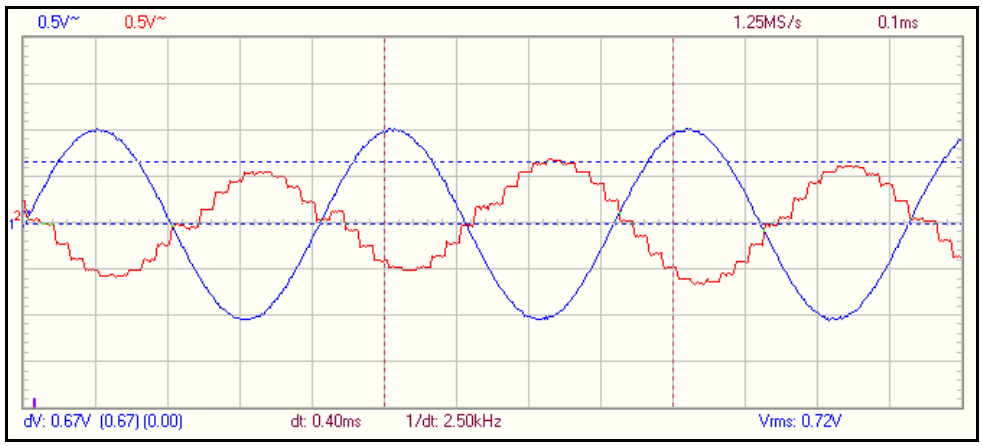


Figura 71. Filtro Pasa-Alto, 1V de entrada, frecuencia 2,5KHz = FC

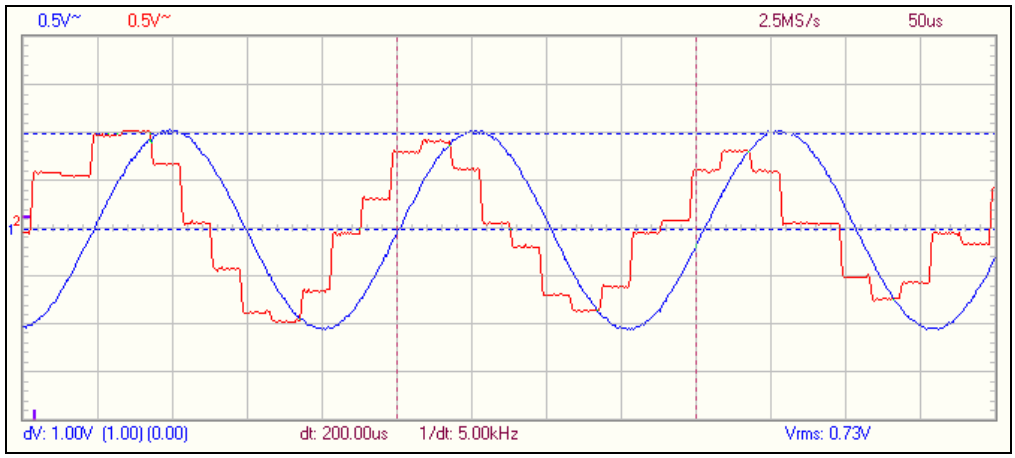


Figura 72. Filtro Pasa-Alto, 1V de entrada, frecuencia 5KHz > FC

En las figuras 64 a la 72 se observan las capturas del comportamiento para diversos valores de frecuencia a la entrada del filtro FIR (para coeficientes Pasa- bajo y Pasa-Alto).

Se destaca entre las mismas los valores de frecuencia iguales a la frecuencia de corte (FC=2,5Khz) en donde se comprobó la existencia de un polo (-3dB, 0,7 V), igualmente se observan los niveles (escalones) debido a el muestreo de la señal (Fs=50Khz, Fmax= 5Khz Fs=10.Fmax), este efecto puede reducirse aumentando aun mas la frecuencia de muestreo (Fs), se realizaron varias pruebas pero al realizar esto, entonces varían las características de respuesta en frecuencia del filtro, y la banda atenuada tiende a volverse muy amplia a medida que aumentamos Fs, (aumentando el ancho de banda y perdiendo el efecto abrupto de atenuación, en el lobulo 1, figura 49), consideramos que para efectos prácticos en esta experiencia se observo de igual manera el comportamiento deseado del filtro.

En la tabla 12 observamos la cantidad de recursos utilizados en la implementación (alrededor del 7% de ocupación). En futuros ensayos con filtros digitales se puede reducir la cantidad de recursos usando menos multiplicadores, multiplexando y haciendo uso solo de una función MAC (multiplica y acumula), ya que se dispone de tiempo entre muestra y muestra para dichas operaciones y el diseño seria mas eficiente.

Tabla 12. Resumen Post-implementación de recursos utilizados en la FPGA Spartan 3E

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	182	9,312	1%	
Number of 4 input LUTs	499	9,312	5%	
Number of occupied Slices	342	4,656	7%	
Number of Slices containing only related logic	342	342	100%	
Number of Slices containing unrelated logic	0	342	0%	
Total Number of 4 input LUTs	512	9,312	5%	
Number used as logic	499			
Number used as a route-thru	13			
Number of bonded IOBs	18	232	7%	
Number of BUFGMUXs	2	24	8%	
Number of MULT18X18SIOs	8	20	40%	
Average Fanout of Non-Clock Nets	2.58			

```
Design statistics:
   Minimum period:  16.266ns{1}   (Maximum frequency:  61.478MHz)
```

CONCLUSIONES

-Se realizó la selección de una plataforma de desarrollo y su FPGA acorde a los requerimientos de pre-grado en la E.I.E, dicha plataforma se considero adecuada para la realización de prácticas y proyectos en nuestra Escuela.

La tarjeta de desarrollo propuesta fue la Spartan 3E Starter Board 1600E, (también llamada MicroBlaze Development Kit Spartan-3E 1600E) la cual cumple a cabalidad los objetivos propuestos para el aprendizaje de dicha tecnología y su uso en materias afines (Sistemas Digitales I, Microprocesadores I y II, Laboratorio de proyectos entre otras) así como para su uso general en proyectos, si bien se podría disponer de mayor complejidad en los periféricos observamos que a nivel didáctico los propuestos son los de uso común en las asignaturas, la experiencia del día a día en el uso de esta tecnología ira sugiriendo el uso de nuevos periféricos y de mayores prestaciones según las aplicaciones a desarrollar, pero de requerirse una nueva selección, ya se contaría con una base sólida de conocimientos y experiencias previas en el uso de esta tecnología.

-Así mismo se realizaron exitosamente dos experiencias prácticas en donde se pusieron a prueba las nuevas herramientas de diseño.

-Cabe destacar que luego de haber realizado dicha investigación y sus experiencias prácticas se observo la alta importancia a nivel mundial que tiene dicha tecnología en el campo del prototipado y desarrollo de proyectos, en particular en nuestra escuela nos ofrece la oportunidad de poder realizar proyectos con más libertad, y sin limitantes de hardware, en un futuro próximo ya no se dispondrán de elementos discretos para la realización de circuitos (hoy en día ya son escasos) y será de gran ayuda esta tecnología para poder continuar impartiendo cursos y desarrollando proyectos en nuestra Escuela.

RECOMENDACIONES

A continuación se sugieren las siguientes recomendaciones:

-Se recomienda a la comunidad de la EIE de la UCV la creación de una materia electiva en donde se cubran los tópicos referentes al uso de los FPGA y DSP, en particular haciendo énfasis en lenguajes de descripción de hardware, su uso y estructuras, su aprendizaje es de vital importancia a futuro ya que cada día son descontinuados y menos accesibles estructuras discretas de hardware (transistores, integrados VLSI, etc.), y por otro lado para hacer uso de las claras ventajas que nos proporciona la tecnología de las FPGAs respecto al desarrollo de proyectos y prototipos de hardware de alta densidad que no serian posibles de realizar en nuestra escuela sin el uso de la misma.

-Se recomienda la incorporación de practicas en la en la unidad docente Sistemas Digitales I, como una introducción a el uso de la tecnología FPGA, herramientas CAD-EDA y lenguajes HDL.

-Se recomienda la creación de un espacio en un servidor TCP/IP en la EIE donde se aloje una biblioteca de librarías HDL ó IP core con documentación detallada para uso exclusivo de estudiantes y profesores de la escuela, esto para dar un impulso a la realización de nuevos, mejores y mas complejos proyectos cada día.

REFERENCIAS BIBLIOGRÁFICAS

[1] Meyer-Baese, U. Digital signal processing with field programmable gate arrays, 1ra. Ed. Germany: Editorial Springer-Verlag Berlin Heidelberg, 2001. P. 1-139

[2] Woods, Roger. FPGA-based Implementation of Signal Processing Systems 1ra. Ed. United Kingdom: Editorial John Wiley & Sons, Ltd, 2008. P. 1-23

[3] Maxfield, Clive. The design warrior´s guide to fpga´s, 1ra. Ed. USA: Editorial Newnes Elsevier, 2004. P. 25-351

[4] Parnell, Karen. Programmable logic design quick start hand book, 2da. Ed: Xilinx, 2002. P. 11-34

[5] Tocci, Ronald. Sistema digitales, principios y aplicaciones, 8va, Ed. México: Editorial Prentice Hall, Inc, 2003. P. 751-792

BIBLIOGRAFÍA

Libros

- Bernadas, Jorge A. Circuitos secuenciales: Diseño y análisis. 1ra, Ed. Caracas, Venezuela: Editorial Innovación Tecnológica, 2000.
- Brown, Stephen. Fundamentals of digital logic with VHDL, 2da, Ed, USA: Editorial Mc Graw Hill, Inc, 2005.
- Proakis, John G. Tratamiento digital de señales. 3ra, Ed, España: Editorial Prentice Hall, Inc, 1998.
- Pardo, Fernando. VHDL, Lenguaje para síntesis y modelado de circuitos, 1ra, Ed, Madrid, España : Editorial RAMA, 1999.
- Olloz, Serafín. VHDL, Lenguaje estándar de diseño electrónico, 1ra, Ed, Madrid, España : Editorial Mc Graw Hill, Inc, 1998.
- Chu, Pong P. Prototyping by VHDL examples, Xilinx Spartan 3 Version, 1ra Ed, New Jersey, USA: Editorial John Wiley & Sons, Ltd, 2008.
- Chu, Pong P. RTL Hardware Design Using VHDL, 1ra Ed, New Jersey, USA: Editorial John Wiley & Sons, Ltd, 2006.
- Maxinez , David. El arte de programar sistemas digitales, 1ra, Ed, Mexico : Editorial Continental, 2002.

Manuales

- Manual de Referencia: Versión 2004-10 / IEEE 1076 International Standard, VHDL Language Reference Manual, 1ra, Ed, New York, USA : IEEE, 2004. 300p.
- Manual de Referencia: Versión 1.6 2009-10 / Spartan-3 Generation configuration User Guide UG332, Xilinx ,2009. 532p.

- Manual de Referencia: Versión 1.6 2010-08 / Spartan-6 configuration User Guide UG380, Xilinx, 2010. 156p.
- Manual de Referencia: Versión 1.7 2011-03 / Spartan-6 Family Overview Ds160, Xilinx, 2011. 11p.
- Manual de Referencia: Versión 3.1 2010-08 / Virtex-4 Family Overview Ds112, Xilinx, 2010. 9p.
- Manual de Referencia: Versión 2.4 2009-12 / Virtex-4 FPGA User Guide UG070, Xilinx, 2009. 406p.
- Manual de Referencia: Versión 5 2009-02 / Virtex-5 Family Overview Ds100, Xilinx, 2009. 13p.
- Manual de Referencia: Versión 1.8 2010-02 / Embedded Processor Block in Virtex-5 FPGAs UG200, Xilinx, 2009. 347p.
- Manual de Referencia: Versión 2.2 2010-01 / Virtex-6 Family Overview Ds150, Xilinx, 2010. 11p.
- Manual de Referencia: Versión V1-3.3 2008-02 / Cyclone II Device Handbook, Volume 1, Altera, 2008. 470p.
- Manual de Referencia: Versión V1-3.3 2010-01 / Cyclone III Device Handbook, Volume 1, Altera, 2010. 350p.
- Manual de Referencia: Versión V1-1.5 2010-12 / Cyclone VI Device Handbook, Volume 1, Altera, 2010. 478p.
- Manual de Referencia: Versión 2.2 2011-03 / Stratix III Device Handbook, Volume 1, Altera, 2011. 797p.
- Manual de Referencia: Versión 4.3 2011-01 / Stratix V Device Handbook, Volume 1, Altera, 2011. 516p.
- Manual de Referencia: Versión V1-1.2 2011-04 / Stratix VI Device Handbook, Volume 1, Altera, 2011. 516p.
- Manual de Referencia: Versión 2010-10 / Atlys Board Reference Manual, Diligent, 2010. 19p.
- Manual de Referencia: Versión 2010-11 / Basys2 Board Reference Manual, Diligent, 2010. 12p.

- Manual de Referencia: Versión 2008-06 / Nexys2 Board Reference Manual, Diligent, 2010. 17p.
- Manual de Referencia: Versión 2005-05 / Spartan-3 Starter Kit Board User Guide, Xilinx, 2005. 64p.
- Manual de Referencia: Versión 2006-03 / Spartan-3E Starter Kit Board User Guide, UG230, Xilinx, 2006. 164p.
- Manual de Referencia: Versión 2007-12 / Spartan MicroBlaze Development Kit Spartan-3E 1600E Edition User Guide ,UG257, Xilinx, 2007. 168p.
- Manual de Referencia: Versión 2008-06 / Spartan-3A/3AN FPGA Starter Kit Board User Guide ,UG334, Xilinx, 2008. 140p.
- Manual de Referencia: Versión 1.0 2010-12 / XuLA Board V1.0 User Manual, XESS CORP, 2010. 30p.
- Manual de Referencia: Versión 1.1 2007-09 / XSA-3S1000 Board V1.1 User Manual, XESS CORP, 2007. 48p.
- Manual de Referencia: Versión 1.5 2011 / DE0 Board User Manual, Altera, 2011. 56p.
- Manual de Referencia: Versión 1.1 2006 / DE1 Board User Manual, Altera, 2006. 56p.
- Manual de Referencia: Versión 1.03 2010 / DE115 Board User Manual, Altera, 2010. 116p.
- Manual de Referencia: Versión 1.4 2006 / DE2 Board User Manual, Altera, 2006. 72p.

Tesis

- Murillo M., Denic D. Filtro digital implementado en un PLD. / Murillo Murillo Denic Danay (Tesis). San Jose, Costa Rica: Universidad de Costa Rica. 2004.
- Castro J. Gerardo. Procedimiento de diseño e implementación de circuitos digitales utilizando herramientas EDA de código abierto. / Castro Jiménez Gerardo (Tesis) . San Jose, Costa Rica: Universidad de Costa Rica. 2008.

- Vera L. Mario. Diseño de funciones DSP usando VHDL y CPLDs –FPGAs. / Vera Lizcano Mario (Articulo). Cali, Colombia: Universidad del Valle. 2003.
- Buj G., Robert A. Procedimiento de diseño de circuitos digitales mediante FPGAs./ Buj Gelonch Robert Antoni (Tesis). Lleida, España: Universidad de Lleida, 2007.

Internet

- USB [en línea] <http://prof.usb.ve/jregidor/cursos/ec1723/info_lab.html> [Consulta : 2010]
- ULA [en línea] <http://www.ing.ula.ve/%7Earaujol/lsd/p1/index.html#Pasos> [Consulta : 2010]
- FPGAs en Español [en línea]. <http://fpga.com.ar/> ,[Consulta : 2010]
- FPGA Central [en línea]. <http://www.fpgacentral.com/docs/fpga-tutorial>[Consulta : 2010]
- FPGA Libre [en línea].<http://fpgalibre.sourceforge.net > [Consulta : 2011]
- OpenCores [en línea] <http://opencores.org/> , [Consulta : 2011]
- Xilinx [en línea] <http://xilinx.com/> , [Consulta : 2011]
- Altera [en línea] <http://altera.com/> , [Consulta : 2011]
- Achronix [en línea] <http://www.achronix.com/>, [Consulta : 2011]
- Actel [en línea] <http://www.actel.com/>[Consulta : 2011]
- Atmel. [en línea] <http://www.atmel.com/>[Consulta : 2011]
- Latticesemi [en línea] <http://www.latticesemi.com/>[Consulta : 2011]
- Terasic [en línea] <http://www.terasic.com/>[Consulta : 2011]

ANEXO 1

Código VHDL de la experiencia # 1 (Unidad de control para ascensor de 4 pisos con Memoria)

```
1 – MEMORIA.VHD                                          ANTONIO BERTSCH / UCV
/EIE
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity Memoria is
6 Port ( P0 : in STD_LOGIC;
7          P1 : in STD_LOGIC;
8          P2 : in STD_LOGIC;
9          P3 : in STD_LOGIC;
10         S0 : in STD_LOGIC;
11         S1 : in STD_LOGIC;
12         S2 : in STD_LOGIC;
13         S3 : in STD_LOGIC;
14         ena_ok : in STD_LOGIC;
15         M0 : out STD_LOGIC;
16         M1 : out STD_LOGIC;
17         M2 : out STD_LOGIC;
18         M3 : out STD_LOGIC);
19 end Memoria;
20
21 architecture Funcional of Memoria is
22 signal M_0: STD_LOGIC:='0'; --señales temporales de inicialización de Memorias
23 signal M_1: STD_LOGIC:='0';
24 signal M_2: STD_LOGIC:='0';
25 signal M_3: STD_LOGIC:='0';
26 begin
27 process (P0,P1,P2,P3,S0,S1,S2,S3,ena_ok,M_0,M_1,M_2,M_3) begin –
se declara signal como var temporal de entrada/salida
28     if (S0='1' and ena_ok='1') then – reset memoria en piso 0
29             M_0 <= '0';
30     else
31             if (P0'event and P0='1') then –Memoriza llamada piso 0
32                     M_0 <= '1';
33             end if;
34     end if;
35     if (S1='1' and ena_ok='1') then – reset memoria en piso 1
36             M_1 <= '0';
37     else
38             if (P1'event and P1='1') then –Memoriza llamada piso 1
39                     M_1 <= '1';
40             end if;
41     end if;
42     if (S2='1' and ena_ok='1') then – reset memoria en piso 2
43             M_2 <= '0';
44     else
```

```
45            if (P2'event and P2='1') then –Memoriza llamada piso 2
46                  M_2 <= '1';
47            end if;
48     end if;
49     if (S3='1' and ena_ok='1') then – reset memoria en piso 3
50            M_3 <= '0';
51     else
52            if (P3'event and P3='1') then –Memoriza llamada piso 3
53                  M_3 <= '1';
54            end if;
55     end if;
56 M0<=M_0; -- Mapeo señales de señales temporales
57 M1<=M_1;
58 M2<=M_2;
59 M3<=M_3;
60 end process;
61
62
63
64
65
66 end Funcional;
```

```
--ADC.VHD                                ANTONIO BERTSCH / UCV /EIE
1 –
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5
6 entity Prioridad_Sentido is
7 Port ( clk : in STD_LOGIC;
8         reset : in STD_LOGIC;
9         M0 : in STD_LOGIC;
10        M1 : in STD_LOGIC;
11        M2 : in STD_LOGIC;
12       M3 : in STD_LOGIC;
13        S0 : in STD_LOGIC;
14        S1 : in STD_LOGIC;
15        S2 : in STD_LOGIC;
16        S3 : in STD_LOGIC;
17        ena_ok : in STD_LOGIC;
18        U : out STD_LOGIC;
19        D : out STD_LOGIC;
20        PC1 : out STD_LOGIC;
21        PC0 : out STD_LOGIC);
22 end Prioridad_Sentido;
23
24 architecture Funcional of Prioridad_Sentido is
25 type estado is (baja,parado,sube);
26 signal presente: estado:=parado;
27
28
29 –signal U_int:STD_LOGIC:='0';
30 –signal D_int:STD_LOGIC:='0';
31 –signal PC0_int:STD_LOGIC:='0';
32 –signal PC1_int:STD_LOGIC:='0';
33 signal M_int:STD_LOGIC_VECTOR(3 DOWNTO 0):="0000";
34 signal S_int:STD_LOGIC_VECTOR(3 DOWNTO 0):="1000";
35 signal ok:STD_LOGIC:='1';
36
```

```
begin
combinacional:
process (S0,S1,S2,S3,M0,M1,M2,M3) begin
    S_int<=S0&S1&S2&S3;
    M_int<=M0&M1&M2&M3;
end process combinacional;

maq_estado: --Maquina de estado Moore, --cambio de estado según entradas
process (reset, clk, ena_ok)
begin
    if reset='1' then presente <= parado;
    elsif (clk='1' and clk'event and ena_ok='1' and ok='1') then
           case presente is
                  when parado =>
                         if (M_int > S_int and M_int/="0000" )
then
                                presente <= baja;
                         end if;
                         if (M_int < S_int and M_int/="0000")
then
                                presente <= sube;
                         end if;
                  when baja =>
                         if (M_int ="0000") then
                                presente <= parado;
                         end if;
                         if (M_int < S_int and M_int/="0000")
then
                                presente <= sube;
                         end if;
                  when sube =>
                         if (M_int ="0000") then
                                presente <= parado;
                         end if;
                         if (M_int > S_int and M_int/="0000")
then
                                presente <= baja;
                         end if;
           end case;
    end if;
end process maq_estado;

salida: --Salidas en cada estado, Maquina Modelo Moore
process (presente,M0,M1,M2,M3)
begin
case presente is
    when parado => --en espera de llamada
           U<='0';
           D<='0';
           ok<='1';
    when baja =>
           ok<='0';
           if (M2='1') then –baja piso 2
                  PC0<='1';
                  PC1<='0';
                  U<='0';
                  D<='1';
           elsif (M1='1') then –baja piso 1
                  PC0<='0';
                  PC1<='1';
                  U<='0';
                  D<='1';
```

```
95            elsif (M0='1') then –baja piso 0
96                   PC0<='0';
97                   PC1<='0';
98                   U<='0';
99                   D<='1';
100           else
101                  ok<='1'; --no hay mas paradas en este sentido
102           end if;
103    when sube =>
104           ok<='0';
105           if (M1='1') then –sube piso 1
106                  PC0<='0';
107                  PC1<='1';
108                  U<='1';
109                  D<='0';
110           elsif (M2='1') then –sube piso 2
111                  PC0<='1';
112                  PC1<='0';
113                  U<='1';
114                  D<='0';
115           elsif (M3='1') then –sube piso 3
116                  PC0<='1';
117                  PC1<='1';
118                  U<='1';
119                  D<='0';
120           else
121                  ok<='1'; --no hay mas paradas en este sentido
122           end if;
123 end case;
124
125 end process salida;
126 –U<=U_int;
127 –D<=D_int;
128 –PC0<=PC0_int;
129 –PC1<=PC1_int;
130
131
132 end Funcional;
133
134
```

```
1 –SECUENCIAL_PRINCIPAL.VHD                ANTONIO BERTSCH / UCV /EIE
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.numeric_std.all;
5 use IEEE.STD_LOGIC_ARITH.all;
6 use ieee.STD_LOGIC_UNSIGNED.all;
7
8 entity Secuencial_principal is
9 Port ( reset : in STD_LOGIC; -- reset Maquina Secuencial
10        clk : in STD_LOGIC; -- Clk Moore Sincronico
11        PC0 : in STD_LOGIC;
```

```
12        PC1 : in STD_LOGIC;
13         S0 : in STD_LOGIC;
14         S1 : in STD_LOGIC;
15         S2 : in STD_LOGIC;
16        S3 : in STD_LOGIC;
17         U : in STD_LOGIC;
18         D : in STD_LOGIC;
19        MU : out STD_LOGIC;
20        MD : out STD_LOGIC;
21        PA : out STD_LOGIC;
22        PC : out STD_LOGIC;
23    ena_ok : out STD_LOGIC; -- habilita combinacional(prioridad
sentido) y habilita el reset en P-Memorizados
24 timer_clk : in STD_LOGIC; -- proveniente de bloque Multiplicador
de CLK
25 RST_timer : out STD_LOGIC; -- Rst bloque Multiplicador de CLK
26 clk_ena_mult : out STD_LOGIC); -- Habilitador de Cuenta en bloque
Multiplicador de CLK
27 end Secuencial_principal;
28
29
30 architecture Funcional of Secuencial_principal is
31 type estado is
(inicio,cierra,timer_PC,timer_PA,timer_M,ir,detener,abre);
32 signal presente: estado:=inicio;
33
34 –SALIDAS TEMP
35
36 –signal ena_ok_int: STD_LOGIC:=’1’;
37 signal clk_ena_mult_int: STD_LOGIC:=’0’;
38 –signal RST_timer_int:STD_LOGIC:=’1’;
39 signal timer_vector_int:STD_LOGIC_VECTOR(2 downto 0):=”000”;
40 signal piso_actual_int: STD_LOGIC_VECTOR(1 downto 0):= “00”;
41 signal piso_destino_int: STD_LOGIC_VECTOR(1 downto 0):= “00”;
42 –signal S_int:STD_LOGIC_VECTOR(3 downto 0):=”0000”;
43
44
45 begin
46
47 combinacional:
48 process (S0,S1,S2,S3,PC0,PC1)
49 variable S_int: STD_LOGIC_VECTOR(3 downto 0); --variable solo
valida en el proceso actual,
50 begin
51     S_int:=S0&S1&S2&S3;
52     case S_int is –Combinacional que codifica “piso_actual”
proveniente de los sensores Sx de cada piso
53            when “1000” => piso_actual_int <= “00”;
54            when “0100” => piso_actual_int <= “01”;
55            when “0010” => piso_actual_int <= “10”;
56            when “0001” => piso_actual_int <= “11”;
57            when others => piso_actual_int <= “00”;
58     end case;
59     piso_destino_int<= PC0&PC1;
60 end process combinacional;
61
62 maq_estado: --Maquina de estado Moore , --cambio de estado según
entradas
63 process (reset, clk)
64 begin
65     if reset=’1’ then presente <= inicio;
```

```
66     elsif clk='1' and clk'event then
67             case presente is
68                     when inicio =>
69                             if ((U or D)='1') then
70                                     presente <= cierra;
71                             end if;
72                     when cierra =>
73                             presente <= timer_PC;
74                     when timer_PC =>
75                             if (timer_vector_int="001") then –espera
cuenta clk_timer 011=3s
76                                     presente <= ir; --para cerrar
puertas
77                             end if;
78                     when ir =>
79                             if (piso_destino_int=piso_actual_int)
then –va al piso solicitado indicado por PC1,PC2
80                                     presente <= detener;
81                             end if;
82                     when detener =>
83                             presente <= timer_M;
84                     when timer_M =>
85                             if (timer_vector_int= "001") then –
espera cuenta clk_timer 011=3s
86                                     presente <= abre;
87                             end if;
88                     when abre =>
89                             presente <= timer_PA;
90                     when timer_PA =>
91                             if (timer_vector_int= "001") then –
espera cuenta clk_timer 101=5s
92                                     presente <= inicio; --para abrir
Puertas
93                             end if;
94             end case;
95     end if;
96 end process maq_estado;
97
98 salida: --Salidas en cada estado, Maquina Modelo Moore
99 process (presente)
100 begin
101    case presente is
102            when inicio => --en espera de llamada
103                    MU<='0'; --motor parado
104                    MD<='0';
105                    PA<='1'; --puerta abierta
106                    PC<='0';
107                    clk_ena_mult_int<='0'; --desactiva reloj timer
108                    ena_ok<='1'; --habilita modulos "prioridad
sentido" y "Memoria"
109                    RST_timer<='0'; --reset cd4017
110            when cierra =>
111                    MU<='0'; --motor parado
112                    MD<='0';
113                    PA<='0'; --puerta cerrada
114                    PC<='1';
115                    clk_ena_mult_int<='0';
116                    ena_ok<='0'; --inhabilita modulos "prioridad
sentido" y "Memoria"
117                    RST_timer<='1'; --reset Mult del clk timer
118            when timer_PC =>
119                    MU<='0'; --motor parado
```

```
120                   MD<=’0’;
121                   PA<=’0’; --puerta cerrrada
122                   PC<=’1’;
123                   clk_ena_mult_int<=’1’; --activa reloj timer
124                   ena_ok<=’0’; --inhabilita modulos “prioridad
sentido” y “Memoria”
125                   RST_timer<=’0’;
126            when ir =>
127                   MU<=U; --enciende motor
128                   MD<=D;
129                   PA<=’0’; --puerta cerrada
130                   PC<=’1’;
131                   clk_ena_mult_int<=’0’; --desactiva reloj timer
132                   ena_ok<=’0’; --inhabilita modulos “prioridad
sentido” y “Memoria”
133                   RST_timer<=’0’;
134            when detener =>
135                   MU<=’0’; --detiene motor
136                   MD<=’0’;
137                   PA<=’0’; --puerta cerrada
138                   PC<=’1’;
139                   clk_ena_mult_int<=’0’; --desactiva reloj timer
140                   ena_ok<=’1’; --habilita modulos “prioridad
sentido” y “Memoria”
141                   RST_timer<=’1’;
142            when timer_m =>
143                   MU<=’0’; --detiene motor
144                   MD<=’0’;
145                   PA<=’0’; --puerta cerrada
146                   PC<=’1’;
147                   clk_ena_mult_int<=’1’; --activa reloj timer
148                   ena_ok<=’1’; --habilita modulos “prioridad
sentido” y “Memoria”
149                   RST_timer<=’0’;
150            when abre =>
151                   MU<=’0’; --detiene motor
152                   MD<=’0’;
153                   PA<=’1’; --puerta abierta
154                   PC<=’0’;
155                   clk_ena_mult_int<=’0’; --desactiva reloj timer
156                   ena_ok<=’1’; --habilita modulos “prioridad
sentido” y “Memoria”
157                   RST_timer<=’1’; --reset Mult del clk timer
158            when timer_PA =>
159                   MU<=’0’; --detiene motor
160                   MD<=’0’;
161                   PA<=’1’; --puerta abierta
162                   PC<=’0’;
163                   clk_ena_mult_int<=’1’; --activa reloj timer
164                   ena_ok<=’1’; --habilita modulos “prioridad
sentido” y “Memoria”
165                   RST_timer<=’0’;
166    end case;
167
168 end process salida;
169
170 timer:
171    process (timer_clk,clk_ena_mult_int)
172    begin
173            if (timer_clk’event and timer_clk=’1’ and
clk_ena_mult_int=’1’) then
```

```
174           timer_vector_int<=timer_vector_int + 1; --incrementa
cuenta cada vez que ocurre en evento timer_clk
175           end if; --proveniente del bloque mult_clk
176           if (clk_ena_mult_int='0') then
177                 timer_vector_int<="000";
178           end if;
179    end process timer;
180
181 –mapeo señales temporales
182 clk_ena_mult<=clk_ena_mult_int;
183 –RST_timer<=RST_timer_int;
184 –ena_ok<=ena_ok_int;
185 –MU<=MU_int;
186 –MD<=MD_int;
187 –PA<=PA_int;
188 –PC<=PC_int;
189
190 end Funcional;
191
```

```
1
2 –SENSORES_VALIDACION.VHD                ANTONIO BERTSCH / UCV /EIE
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6
7
8 entity SENSORES_VALIDACION is
9 Port ( S0 : in STD_LOGIC;
10        S1 : in STD_LOGIC;
11       S2 : in STD_LOGIC;
12       S3 : in STD_LOGIC;
13      clk : in STD_LOGIC;
14     reset: in STD_LOGIC;
15     R_S0 : out STD_LOGIC;
16     R_S1 : out STD_LOGIC;
17     R_S2 : out STD_LOGIC;
18     R_S3 : out STD_LOGIC);
19 end SENSORES_VALIDACION;
20
21 architecture funcional of SENSORES_VALIDACION is
22 signal S_int: STD_LOGIC_VECTOR(3 downto 0):="1000";
23 signal R_int: STD_LOGIC_VECTOR(3 downto 0):="1000";
24 begin
25 combinacional:
26 process (S0,S1,S2,S3) begin
27     S_int<=S0&S1&S2&S3;
28 end process combinacional;
29 registro:
30     process (clk, reset) begin
31           if reset='1' then R_int <= "1000";
32           elsif clk='1' and clk'event then
33                 if (S_int="1000") then
34                       R_int <= "1000";
35                 elsif (S_int="0100") then
36                       R_int <= "0100";
37                 elsif (S_int="0010") then
38                       R_int <= "0010";
39                 elsif (S_int="0001") then
40                       R_int <= "0001";
```

```
41                   else
42                          R_int <= R_int;
43                   end if;
44             end if;
45     end process registro;
46 R_S0<=R_int(3);
47 R_S1<=R_int(2);
48 R_S2<=R_int(1);
49 R_S3<=R_int(0);
50 end funcional;
51
52
```

```
1 – Display_Codificador.vhd                ANTONIO BERTSCH / UCV /EIE
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity Display_Codificador is
6 Port ( S0 : in STD_LOGIC;
7         S1 : in STD_LOGIC;
8        S2 : in STD_LOGIC;
9        S3 : in STD_LOGIC;
10         L: Out STD_LOGIC_VECTOR(6 downto 0));
11
12 end Display_Codificador;
13
14 architecture Behavioral of Display_Codificador is
15
16 signal S_int: STD_LOGIC_VECTOR(3 downto 0):= “0000”;
17
18 begin
19
20 S_int<=S0&S1&S2&S3;
21
22
23 Codificador_7segmentos:
24     with S_int select
25             L <= “1”&”0”&”1”&”1”&”1”&”1”&”0” when “0001”, --piso 3
26                  “1”&”0”&”1”&”1”&”0”&”1”&”1” when “0010”, --piso 2
27                  “0”&”0”&”0”&”1”&”1”&”0”&”0” when “0100”, --piso 1
28                  “0”&”1”&”1”&”1”&”1”&”1”&”1” when “1000”, --piso 0
29                  “0”&”0”&”0”&”0”&”0”&”0”&”0” when others; --C. no
valida
30
31
32 end Behavioral;
33
34
```

```
-- Counters_1.vhd                                ANTONIO BERTSCH / UCV
/EIE
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_unsigned.all;
4 use ieee.std_logic_arith.all;
5 use IEEE.numeric_std.all;
6
7
8 entity counters_1 is
9 port(CLK : in std_logic;
```

```
10      RST : in std_logic;
11 CLK_ENA : in std_logic;
12        Q : out std_logic_vector(9 downto 0));
13end counters_1;
14
15 architecture archi of counters_1 is
16 signal tmp: std_logic_vector(9 downto 0):=”1000000000”;
17 begin
18 process (CLK, RST, CLK_ENA)
19 begin
20     if (CLK’event and CLK=’1’ and CLK_ENA=’1’) then
21             if (RST=’1’) then
22                     tmp <= “1000000000”;
23             else
24                     tmp <= tmp(0 downto 0) & tmp(9 downto 1) ;
25             end if;
26 – else
27
28     end if;
29 end process;
30
31 Q <= tmp;
32
33 end archi;
34
35
```

Union_Modulos_ucf.ucf

```
1
2 #clock
3 NET “clk” LOC = “C9” | IOSTANDARD = LVCMOS33 ;
4 NET “clk” PERIOD = 20.0ns HIGH 40%;
5 ##
6 NET “reset” LOC = “V16” | IOSTANDARD = LVTTL | PULLDOWN ;
#ROT_CENTER → reset
7
8 ##led SALIDAS
9 NET “MU” LOC = “F9” | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8
;
10 NET “MD” LOC = “E9” | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE =
8 ;
11 NET “PA” LOC = “D11” | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE =
8 ;
12 NET “PC” LOC = “C11” | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE =
8 ;
13
14 ##led piso MEMORIZADOS
15 NET “M3_OUT” LOC = “F11” | IOSTANDARD = LVTTL | SLEW = SLOW |
DRIVE = 8 ;
16 NET “M2_OUT” LOC = “E11” | IOSTANDARD = LVTTL | SLEW = SLOW |
DRIVE = 8 ;
17 NET “M1_OUT” LOC = “E12” | IOSTANDARD = LVTTL | SLEW = SLOW |
DRIVE = 8 ;
18 NET “M0_OUT” LOC = “F12” | IOSTANDARD = LVTTL | SLEW = SLOW |
DRIVE = 8 ;
19
20 ##SW de Piso Actual
21 NET “S0” LOC = “L13” | IOSTANDARD = LVTTL | PULLUP ; ## SW0
22 NET “S1” LOC = “L14” | IOSTANDARD = LVTTL | PULLUP ; ## SW1
23 NET “S2” LOC = “H18” | IOSTANDARD = LVTTL | PULLUP ; ## SW2
24 NET “S3” LOC = “N17” | IOSTANDARD = LVTTL | PULLUP ; ## SW3
```

```
25
26 ##Pulsadores de llamada
27 NET “P0” LOC = “D18” | IOSTANDARD = LVTTL | PULLDOWN ; ##BTN_WEST
28 NET “P1” LOC = “V4” | IOSTANDARD = LVTTL | PULLDOWN ; ##BTN_NORTH
29 NET “P2” LOC = “H13” | IOSTANDARD = LVTTL | PULLDOWN ; ##BTN_EAST
30 NET “P3” LOC = “K17” | IOSTANDARD = LVTTL | PULLDOWN ;
##BTN_SOUTH
31
32 NET “P0” CLOCK_DEDICATED_ROUTE = FALSE;
33 NET “P1” CLOCK_DEDICATED_ROUTE = FALSE;
34 NET “P2” CLOCK_DEDICATED_ROUTE = FALSE;
35 NET “P3” CLOCK_DEDICATED_ROUTE = FALSE;
36
37 ##Salidas a Display a traves de puertos de expansión J1 y J2
38
39 NET “L(0)” LOC = “B4” | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE
= 8 ;
40 NET “L(1)” LOC = “A4” | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE
= 8 ;
41 NET “L(2)” LOC = “D5” | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE
= 8 ;
42 NET “L(3)” LOC = “C5” | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE
= 8 ;
43 NET “L(4)” LOC = “A6” | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE
= 8 ;
44 NET “L(5)” LOC = “B6” | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE
= 8 ;
45 NET “L(6)” LOC = “E7” | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE
= 8 ;
46
47
```

ANEXO 2

Codigo VHDL de la experiencia #2 (Filtro FIR de coeficientes constantes)

```
1 --ADC.VHD                                                    ANTONIO
BERTSCH / UCV /EIE
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;
5
6
7
8 entity ADC is
9 Port ( SPI_MOSI : out STD_LOGIC;     --escritura G_amp
10      AMP_CS : out STD_LOGIC;                --chip_select G_amp
11      SPI_SCK : out STD_LOGIC;               --clk_div para uso spi
12      AMP_SHDN : out STD_LOGIC;              --habilita shutdown para el
AMP_PRG
13      AD_CONV : out STD_LOGIC; --habilita obtención de muestra hacia el
ic_adc
14
15      -- SPI_SS_B : out STD_LOGIC;           --otros dispositivos bus SPI
16      -- DAC_CS : out STD_LOGIC;             --otros dispositivos bus SPI
17      -- SF_CE0 : out STD_LOGIC;             --otros dispositivos bus SPI
18      -- FPGA_INIT_B : out STD_LOGIC;               --otros dispositivos
bus SPI
19
20      mosi_S : out STD_LOGIC;          --Mux escritura en Bus_SPI
21      ch1_adc : out STD_LOGIC_VECTOR(13 DOWNTO 0);
22      ch2_adc : out STD_LOGIC_VECTOR(13 DOWNTO 0);
23      adc_ok: out STD_LOGIC;          --indica cuando termino y esta
preparado
24      --AMP_DOUT : in STD_LOGIC;
25       adc_ena : in STD_LOGIC;               --solicita muestra adc
26       SPI_MISO : in STD_LOGIC;              --salida spi canales adc
27      reset : in STD_LOGIC;
28      clk : in STD_LOGIC);
29 end ADC;
30
31 architecture funcional of ADC is
32
33 type state_type is (inicio,escribe,finalizando,listo);
34 signal state : state_type := inicio;
35
36 type state_type_2 is (inicializando_amp,disponible,preparando,lee);
37 signal state_2 : state_type_2 := inicializando_amp;
38
39 signal bit_act : integer range 0 to 16 := 0;
40 signal bit_act_2 : integer range 0 to 64 := 0;
41 signal clkdiv : integer range 0 to 8;       --división de clk
42 signal clk_spi : std_logic := '0';          --clk_spi para adc
43 signal clk_spi_amp : std_logic := '0';      --clk_spi para amp
44 signal flanco_up_amp : std_logic := '1';    --flanco de subida
45
46 constant config_AMP : std_logic_vector(7 downto 0):="00010001"; --
ganancia fijada en -1 (0,4--2.9V)
47
48 signal dato_adc_1 : std_logic_vector(13 downto 0):="11111111111111";
49 signal dato_adc_2 : std_logic_vector(13 downto 0):="11111111111111";
50
```

```
51
52 begin
53
54 ch1_adc<=dato_adc_1;
55 ch2_adc<=dato_adc_2;
56 clk_spi <= clk;                        --clk SPI para ADC a 50Mhz
57 AMP_SHDN<= '0';                        --deshabilita shutdown amp en
power_up
58
59
60 clock_spi_multiplexor:
61      with state select
62              SPI_SCK <= clk_spi_amp when inicio,
63                         clk_spi_amp when escribe,
64                         clk_spi_amp when finalizando,
65                        clk_spi when listo;
66
67 mosi_S_multiplexor:          --multiplexa BUS de escritura SPI_MOSI
68      with state select
69              mosi_S <= '0' when inicio,
70                         '0' when escribe,
71                         '0' when finalizando,
72                         '1' when listo;
73
74
75 clock_div:
76      process(clk, reset) begin
77              if(reset = '1') then
78              elsif(clk'event and clk='1') then
79                      if(clkdiv = 5) then --clk_spi para el amp/
5x2Tx20ns=200ns=>5Mhz
80                              flanco_up_amp <= not flanco_up_amp;
81                              clk_spi_amp <= not clk_spi_amp;
82                              clkdiv <= 0;
83                      else
84                              clkdiv <= clkdiv + 1;
85                              end if;
86                              end if;
87      end process clock_div;
88
89 secuencia_inicio_amp:
90      process(clk, reset)
91      begin
92              if(reset = '1') then state<=inicio;
93              elsif(clk'event and clk='1') then
94              if( clkdiv = 5 and flanco_up_amp = '0') then
95                      case state is
96                              when inicio =>
97                                      AMP_CS <= '1';
98                                      SPI_MOSI<= '0';
99                                      if(bit_act = 8) then
100                                             bit_act <= 0;
101                                             state <= escribe;
102                                             AMP_CS <= '1';
103                                     else
104                                             bit_act <= bit_act + 1;
105                                             state <= inicio;
106                                     end if;
107                             when escribe =>
108                                     AMP_CS <= '0';
109                                     if (bit_act >= 0 and bit_act < 8)
then
110                                             SPI_MOSI <= config_AMP(7 -
bit_act);
111                                             bit_act <= bit_act + 1;
112                                             state <= escribe;
113                                     elsif(bit_act = 8) then
114                                             bit_act <= 0;
115                                             SPI_MOSI<= '0';
```

```
116                                             AMP_CS <= '1';
117                                             state <= finalizando;
118                                     end if;
119                             when finalizando =>
120                                     AMP_CS <= '1';
121                                     SPI_MOSI<= '0';
122                                     bit_act <= 0;
123                                     if(bit_act = 2) then
124                                             bit_act <= 0;
125                                             AMP_CS <= '1';
126                                             state <= listo;
127                                     else
128                                             bit_act <= bit_act + 1;
129                                             state <= finalizando;
130                                     end if;
131                             when listo =>
132                                     AMP_CS <= '1';
133                                     SPI_MOSI<= '0';
134                                     bit_act <= 0;
135                                     state <= listo;
136                             end case;
137                     end if;
138             end if;
139     end process secuencia_inicio_amp;
140
141 secuencia_lectura_adc:
142     process(clk, reset)
143     begin
144             if(reset = '1') then state_2<=inicializando_amp;
145             elsif(clk'event and clk='0') then
146                     case state_2 is
147                             when inicializando_amp =>
148                                     AD_CONV <= '0';
149                                     adc_ok<='0';
150                                     if(state=listo) then
151                                             state_2 <= disponible;
152                                     else
153                                             state_2 <=inicializando_amp;
154                                     end if;
155                             when disponible =>
156                                     AD_CONV <= '0';
157                                     adc_ok<='1';
158                                     if(adc_ena='1' and state=listo) then
159                                             AD_CONV <= '1';
160                                             adc_ok<='0';
161                                             state_2 <= preparando;
162                                     else
163                                             state_2 <=disponible;
164                                     end if;
165                             when preparando =>
166                                     adc_ok<='0';
167                                     if(bit_act_2 = 0 or bit_act_2 = 1)
then
168                                             AD_CONV <= '0';
169                                             bit_act_2 <= bit_act_2 + 1;
170                                     elsif (bit_act_2 = 2) then
171                                             AD_CONV <= '0';
172                                             bit_act_2 <= 0;
173                                             state_2 <= lee;
174                                     end if;
175                             when lee =>
176                                     AD_CONV <= '0';
177                                     adc_ok<='0';
178                                     if (bit_act_2 >= 0 and bit_act_2 <
14) then
179                                             dato_adc_1(13 - bit_act_2) <=
SPI_MISO;
180                                             bit_act_2 <= bit_act_2 + 1;
181                                             state_2 <= lee;
```

```
182                                     elsif(bit_act_2 = 14 or bit_act_2 =
15) then
183                                             state_2 <= lee;
184                                             bit_act_2 <= bit_act_2 + 1;
185                                     elsif (bit_act_2 >= 16 and bit_act_2
< 30) then
186                                             dato_adc_2(29 - bit_act_2) <=
SPI_MISO;
187                                             bit_act_2 <= bit_act_2 + 1;
188                                             state_2 <= lee;
189                                     elsif(bit_act_2 = 30 or bit_act_2 =
31) then
190                                             state_2 <= lee;
191                                             bit_act_2 <= bit_act_2 + 1;
192                                             elsif(bit_act_2 = 32) then
193                                             state_2 <= disponible;
194                                             bit_act_2 <= 0;
195                                             adc_ok<='1';
196                                     end if;
197                             end case;
198                     end if;
199     end process secuencia_lectura_adc;
200
201
202 --desactivando otros dispositivos en el bus spi
203 --SPI_SS_B<='1';
204 --DAC_CS<='1';
205 --SF_CE0<='1';
206 --FPGA_INIT_B<='1';
207
208
209 end funcional;
210
211
```

```
1--Filtro_FIR.vhd                              ANTONIO BERTSCH  UCV/EIE
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;
5 --use IEEE.STD_LOGIC_ARITH.all;
6 --use IEEE.STD_LOGIC_UNSIGNED.all;
7
8 entity Filtro_FIR is
9      Port ( BTN_E : in STD_LOGIC;
10            BTN_N : in STD_LOGIC;
11            BTN_W : in STD_LOGIC;
12            x : in STD_LOGIC_VECTOR(13 downto 0); --
13bit_dato+1_signo
13            y : out STD_LOGIC_VECTOR(13 downto 0);
14            clk_fs : in STD_LOGIC);
15 end Filtro_FIR;
16
17 architecture Funcional of Filtro_FIR is
18
19 subtype muestra is signed(13 downto 0);
20 type Arreglo_Tap is array (7 downto 0) of muestra; --declara
arreglo con tap del filtro
21 signal Tap : Arreglo_Tap;
22
23 subtype coeficientes is signed(10 downto 0);
24 type Arreglo_Coef is array (7 downto 0) of coeficientes;
25 signal coef : Arreglo_Coef;
26
27 type state_type is (pasa_bajo,pasa_alto,pasa_banda);
28 signal state : state_type := pasa_bajo;
29
-- declara coeficientes del filtro
30 --pasa bajo
31 constant coef0 : signed(10 downto 0):="00010111001";
32 constant coef1 : signed(10 downto 0):="00001011101";
33 constant coef2 : signed(10 downto 0):="00001110000";
34 constant coef3 : signed(10 downto 0):="00001111011";
35 constant coef4 : signed(10 downto 0):="00001111011";
36 constant coef5 : signed(10 downto 0):="00001110000";
37 constant coef6 : signed(10 downto 0):="00001011101";
38 constant coef7 : signed(10 downto 0):="00010111001";
39
40
41 --pasa alto
42 constant coef8 : signed(10 downto 0):="11110101001";
43 constant coef9 : signed(10 downto 0):="11110101010";
44 constant coef10 : signed(10 downto 0):="11100110101";
45 constant coef11 : signed(10 downto 0):="10100110100";
46 constant coef12 : signed(10 downto 0):="01011001100";
47 constant coef13 : signed(10 downto 0):="00011001011";
48 constant coef14 : signed(10 downto 0):="00001010110";
49 constant coef15 : signed(10 downto 0):="00001010111";
50
51
52 --pasa banda
53 constant coef16 : signed(10 downto 0):="10100000101";
54 constant coef17 : signed(10 downto 0):="00000000000";
55 constant coef18 : signed(10 downto 0):="00000011101";
56 constant coef19 : signed(10 downto 0):="00000110001";
57 constant coef20 : signed(10 downto 0):="00000110001";
58 constant coef21 : signed(10 downto 0):="00000011101";
59 constant coef22 : signed(10 downto 0):="00000000000";
```

```
60 constant coef23 : signed(10 downto 0):="10100000101";
61
62 signal y_int: signed(24 downto 0):= (others => '0');
63 signal s: std_logic_vector(1 downto 0):= "00";
64
65 begin
66
67 Filtro:
68     process(clk_fs) begin
69             if(clk_fs'event and clk_fs='1') then
70y_int<=(coef(0)*Tap(0)+coef(1)*Tap(1)+coef(2)*Tap(2)+coef(3)*Tap(3
)+coef(4)*Tap(4)+coef(5)*Tap(5)+coef(6)*Tap(6)+coef(7)*Tap(7))/2048;
71                   for I in 7 downto 1 loop
72                          Tap(I)<=Tap(I-1);
73                   end loop;
74                   Tap(0)<= signed(x);
75             end if;
76     end process Filtro;
77
78
79 y <= std_logic_vector(y_int(13 downto 0));
80
81
82 MUX_Seleccion_coef0:
83     with s select
84             coef(0) <= coef0 when "00",
85                        coef8 when "01",
86                       coef16 when "10",
87                        coef0 when others;
88
89 MUX_Seleccion_coef1:
90     with s select
91             coef(1) <= coef1 when "00",
92                        coef9 when "01",
93                  coef17 when "10",
94                        coef1 when others;
95
96 MUX_Seleccion_coef2:
97     with s select
98             coef(2) <= coef2 when "00",
99                        coef10 when "01",
100                    coef18 when "10",
101                       coef2 when others;
102
103 MUX_Seleccion_coef3:
104    with s select
105            coef(3) <= coef3 when "00",
106                       coef11 when "01",
107                       coef19 when "10",
108                       coef3 when others;
109
110 MUX_Seleccion_coef4:
111    with s select
112            coef(4) <= coef4 when "00",
113                       coef12 when "01",
114                       coef20 when "10",
115                       coef4 when others;
116
117 MUX_Seleccion_coef5:
118    with s select
119          coef(5) <= coef5 when "00",
120                       coef13 when "01",
```

```
121                       coef21 when "10",
122                       coef5 when others;
123
124 MUX_Seleccion_coef6:
125    with s select
126           coef(6) <= coef6 when "00",
127                      coef14 when "01",
128                      coef22 when "10",
129                 coef6 when others;
130
131 MUX_Seleccion_coef7:
132    with s select
133           coef(7) <= coef7 when "00",
134                      coef15 when "01",
135                      coef23 when "10",
136                      coef7 when others;
137
138 Seleccion_coef:
139    process(clk_fs) begin
140           case state is
141                  when pasa_bajo =>
142                         s<="00";
143                         if BTN_N='1' then
144                                state<=pasa_alto;
145                                s<="01";
146                         elsif BTN_E='1' then
147                                state<=pasa_banda;
148                                s<="10";
149                         else
150                                state<=pasa_bajo;
151                         end if;
152                  when pasa_alto =>
153                         s<="01";
154                         if BTN_W='1' then
155                                state<=pasa_bajo;
156                                s<="00";
157                                elsif BTN_E='1' then
158                                state<=pasa_banda;
159                                s<="10";
160                         else
161                                state<=pasa_alto;
162                         end if;
163                  when pasa_banda =>
164                         s<="10";
165                         if BTN_W='1' then
166                                state<=pasa_bajo;
167                                s<="00";
168                         elsif BTN_N='1' then
169                                state<=pasa_alto;
170                                s<="01";
171                         else
172                                state<=pasa_banda;
173                         end if;
174           end case;
175    end process Seleccion_coef;
176
177
178
179 end funcional;
180
181
```

```
1 --U_Control.vhd                                    ANTONIO BERTSCH
       UCV /EIE
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 --use IEEE.NUMERIC_STD.ALL;
5
6
7 entity U_Control is
8      Port ( clk : in STD_LOGIC;
9             reset : in STD_LOGIC;
10            dac_ok : in STD_LOGIC;
11            adc_ok : in STD_LOGIC;
12
13            SW0 : in STD_LOGIC;
14
15            SPI_SS_B : out STD_LOGIC;   --otros dispositivos bus
SPI
16            SF_CE0 : out STD_LOGIC;     --otros dispositivos bus
SPI
17            FPGA_INIT_B : out STD_LOGIC;       --otros
dispositivos bus SPI
18
19            clk_Fs : out STD_LOGIC;
20            adc_ena : out STD_LOGIC;
21            dac_ena : out STD_LOGIC);
22 end U_Control;
23
24 architecture Funcional of U_Control is
25
26 type state_type is (inicio,lee,Filtra,escribe);
27 signal state : state_type := inicio;
28
29 signal clkdiv : integer range 0 to 1024;       --contador división de clk
30 signal cuenta : integer range 0 to 4;
31 signal clk_Fm : std_logic := '0';       --Frecuencia de Muestreo
Fm
32
33 begin
34
35 clock_Fm:
36     process(clk, reset) begin
37            if(reset = '1') then
38            elsif(clk'event and clk='1') then
39                   if(clkdiv = 500) then --Frecuencia de Muestreo
Fm ,,
1/2*(500x20ns)=50Khz
40                          clk_Fm <= not clk_Fm;
41                          clkdiv <= 0;
42                   else
43                          clkdiv <= clkdiv + 1;
44                   end if;
45            end if;
46     end process clock_Fm;
47
48
49 maquina_de_estado:
50     process(clk, reset)
51     begin
52            if(reset = '1') then state<=inicio;
53            elsif(clk'event and clk='1') then
54                   case state is
55                          when inicio =>
```

```
56                                      adc_ena<='0';
57                                      dac_ena<='0';
58                                      clk_Fs<='0';
59                                      if (SW0='1' and adc_ok='1' and
dac_ok='1' and clk_Fm='1') then
60                                              state<=lee;
61                                              adc_ena<='1';
62                                      else
63                                              state<=inicio;
64                                      end if;
65                              when lee =>
66                                      adc_ena<='0';
67                                      dac_ena<='0';
68                                      clk_Fs<='0';
69                                      if (adc_ok='1' and dac_ok='1')
then
70                                              state<=Filtra;
71                                              clk_Fs<='1';
72                                      else
73                                              state<=lee;
74                                      end if;
75                              when Filtra =>
76                                      adc_ena<='0';
77                                      dac_ena<='0';
78                                      clk_Fs<='0';
79                                      if (adc_ok='1' and dac_ok='1' and
cuenta=2) then
80                                              state<=escribe;
81                                              dac_ena<='1';
82                                              cuenta <= 0;
83                                      else
84                                              state<=Filtra;
85                                              cuenta<=cuenta+1;
86                                      end if;
87                              when escribe =>
88                                      adc_ena<='0';
89                                      dac_ena<='0';
90                                      clk_Fs<='0';
91                                      if dac_ok='1' and adc_ok='1' and
clk_Fm='0' then
92                                              state<=inicio;
93                                      else
94                                              state<=escribe;
95                                      end if;
96                      end case;
97              end if;
98      end process maquina_de_estado;
99
100
101 --otros dip´s SPI a desactivar
102 SPI_SS_B<='1';
103 SF_CE0<='1';
104 FPGA_INIT_B<='1';
105
106 end Funcional;
107
108
```

```
1--DAC.vhd                                                    ANTONIO BERTSCH
UCV/EIE
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;
5 use IEEE.STD_LOGIC_UNSIGNED.all;
6
7 entity DAC is
8      Port ( SPI_MOSI : out STD_LOGIC; --escritura G_amp y DAC
9             DAC_CS : out STD_LOGIC; --Chip select DAC
10            DAC_CLR : out STD_LOGIC; -- clear asincronico, activo
en bajo
11
12            bit_14 : out STD_LOGIC; --bit de signo
13            dac_in : in STD_LOGIC_VECTOR(13 DOWNTO 0); --Bus de
entrada DAC
14            dac_ok: out STD_LOGIC; --indica cuando termino y esta
preparado
15            dac_ena : in STD_LOGIC; --procesa muestra dac
16            reset : in STD_LOGIC;
17            clk : in STD_LOGIC);
18 end DAC;
19
20 architecture Funcional of DAC is
21
22 type state_type is (inicio,escribe,finalizando);
23 signal state : state_type := inicio;
24
25 signal bit_act : integer range 0 to 64 := 0;
26 signal trama_DAC : std_logic_vector(31 downto 0):=(others =>
'0');
27
28 begin
29
30 DAC_CLR<='1';
31 bit_14<= dac_in(13);
32
33 Seleccion_DAC_segun_Signo:
34     with dac_in(13) select
35            trama_DAC <= "00000000"&"0011"&"0000"&dac_in(11 downto
0)&"0000" when '0',--dac señal +, canal a
36                         "00000000"&"0011"&"0001"&(not
(dac_in(11 downto 0)) +1 )&"0000" when '1', --dac señal -, canal b, toma el
complemento a 2 del numero
37
"00000000"&"0011"&"0000"&"000000000000"&"0000" when others; --si
ocurre una condición no valida std_ulogic
38
39
40 escritura_DAC:
41     process(clk, reset)
42     begin
43            if(reset = '1') then state<=inicio;
44            elsif(clk'event and clk='0') then
45                   case state is
46                          when inicio =>
47                                 DAC_CS <= '1';
48                                 SPI_MOSI<= '0';
49                                 dac_ok <= '1';
50                                 bit_act <= 0;
51                                 if dac_ena='1' then
```

```
52                                          state <= escribe;
53                                          dac_ok <= '0';
54                                    else
55                                          state <= inicio;
56                                    end if;
57                              when escribe =>
58                                    DAC_CS <= '0';
59                                    dac_ok <= '0';
60                                    if (bit_act >= 0 and bit_act <
32) then
61                                          SPI_MOSI <= trama_DAC(31 -
bit_act);
62                                          bit_act <= bit_act + 1;
63                                          state <= escribe;
64                                    elsif(bit_act = 32) then
65                                          bit_act <= 0;
66                                          SPI_MOSI<= '0';
67                                          DAC_CS <= '1'; --
deshabilita dac BUS SPI
68                                          state <= finalizando;
69                                    end if;
70                              when finalizando =>
71                                    DAC_CS <= '1';
72                                    SPI_MOSI<= '0';
73                                    dac_ok <= '0';
74                                    if(bit_act = 1) then
75                                          bit_act <= 0;
76                                          dac_ok <= '1';
77                                          state <= inicio;
78                                    else
79                                          bit_act <= bit_act + 1;
80                                          state <= finalizando;
81                                    end if;
82                        end case;
83              end if;
84     end process escritura_DAC;
85
86
87 end Funcional;
88
89
```

Sch_principal_UCF.ucf

```
1
2
3 # ==== Clock inputs (CLK) ====
4 NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
5 NET "clk" PERIOD = 20.0ns HIGH 50%;
6
7
8 # ==== Analog-to-Digital Converter (ADC) ====
9 NET "AD_CONV" LOC = "P11" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 6 ;
10
11 # ==== Digital-to-Analog Converter (DAC) ====
12 NET "DAC_CS" LOC = "N8" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 8 ;
13 NET "DAC_CLR" LOC = "P8" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 8 ; ##-- clear
asincronico, activo en bajo
14
15 # ==== Programmable Gain Amplifier (AMP) ====
```

```
16 NET "AMP_CS" LOC = "N7" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 6 ;
17 ##NET "AMP_DOUT" LOC = "E18" | IOSTANDARD = LVCMOS33 ;
18 NET "AMP_SHDN" LOC = "P7" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 6 ;
19
20 # ==== Señales bus SPI compartidas con otros dip´s ====
21 NET "SPI_MOSI" LOC = "T4" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 6 ;
22 NET "SPI_SCK" LOC = "U16" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 8 ;
23 NET "SPI_MISO" LOC = "N10" | IOSTANDARD = LVCMOS33 ;
24
25
26 # otros dispositivos bus SPI a desactivar
27 NET "SPI_SS_B" LOC = "U3" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 6 ; ##STMicro
SPI serial Flash (SPI)
28 NET "SF_CE0" LOC = "D16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 |
SLEW = SLOW ; ##Intel
StrataFlash Parallel NOR Flash (SF)
29 NET "FPGA_INIT_B" LOC = "T3" | IOSTANDARD = LVCMOS33 | SLEW =
SLOW | DRIVE = 4 ;##FPGA
Configuration Mode, INIT_B Pins (FPGA)
30
31
32 ## CONECTOR DTE
33 ##NET "rx" LOC = "U8" | IOSTANDARD = LVTTL ;
34 ##NET "tx" LOC = "M13" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW =
SLOW ;
35
36 ##
37 NET "reset" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
#BTN_SOUTH --> reset
38 NET "SW0" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
39 NET "BTN_E" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN ;
40 NET "BTN_N" LOC = "V4" | IOSTANDARD = LVTTL | PULLDOWN ;
41 NET "BTN_W" LOC = "D18" | IOSTANDARD = LVTTL | PULLDOWN ;
42
43 ##led de prueba de signo DAC
44 NET "bit_14" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW |
DRIVE = 8 ;
45
46
```