UNIVERSIDAD JOSÉ ANTONIO PÁEZ

DISEÑO DE EXPERIENCIAS PRÁCTICAS CON EL SISTEMA ATLYS DE XILINX PARA EL LABORATORIO DE MICROPROCESADORES DE LA UJAP

Autor:
Gerardo Alberto Villalonga Soto

Urb. Yuma II, calle Nº 3. Municipio San Diego
Teléfono: (0241) 8714240 (máster) – Fax: (0241) 8712394

REPÚBLICA BOLIVARIANA DE VENEZUELA
UNIVERSIDAD JOSÉ ANTONIO PÁEZ
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELECTRÓNICA

DISEÑO DE EXPERIENCIAS PRÁCTICAS CON EL SISTEMA ATLYS DE XILINX PARA EL LABORATORIO DE MICROPROCESADORES DE LA UJAP

Trabajo de Grado para optar al título de
INGENIERO ELECTRÓNICO

Autor:
Gerardo Villalonga
C.I. 27.443.979
Tutor:
Wilmer Sanz
C.I. 7.130.496

San Diego, Junio 2023

UNIVERSIDAD JOSÉ ANTONIO PÁEZ
COORDINACIÓN DE PASANTÍA Y TRABAJO DE GRADO

ACTA DE APROBACIÓN

INFORME FINAL DE PASANTÍA ☐

TRABAJO DE GRADO ☑

El jurado designado por la Facultad de Ingeniería para la evaluación del **Informe Final de Pasantía o Trabajo de Grado** titulado:

Diseño de Experiencias prácticas con el Sistema ATlys de XILINX para el laboratorio de microprocesadores de la UJAP.

Realizado por el (la) Br. Gerald Villalonga

C.I. Nº 27443979 cursante de la carrera de Ing. Electrónica

hace constar después de analizar su contenido y oída la exposición oral, considera que el Informe Final o Trabajo de Grado ha obtenido la calificación de:

APROBADO ☑ **NO APROBADO** ☐

El Jurado

Tutor Académico (Coordinador)
Nombre: Wilmer Sanz
C.I.: 7130476

Jurado
Nombre: CÉSAR SEIJAS
C.I.: 4.567.093

Jurado
Nombre: Wilmer Espinoza
C.I.: 9885895

Fecha: 03 / 07 / 2023

UJAP

ANEXO N

REPÚBLICA BOLIVARIANA DE VENEZUELA
UNIVERSIDAD JOSÉ ANTONIO PÁEZ
FACULTAD DE INGENIERÍA
ESCUELA DE ELECTRÓNICA

CONSTANCIA DE APROBACIÓN PARA LA PRESENTACIÓN PÚBLICA DEL TRABAJO DE GRADO

Quien suscribe, Wilmer Sanz, portador de la cédula de identidad N° 7.130.469, en mi carácter de tutor del trabajo de grado presentado por el ciudadano Gerardo Alberto Villalonga Soto, portador de la cédula de identidad N° 27.443.979, titulado: **DISEÑO DE EXPERIENCIAS PRÁCTICAS CON EL SISTEMA ATLYS DE XILINX PARA EL LABORATORIO DE MICROPROCESADORES DE LA UJAP,** presentado como requisito parcial para optar al título de INGENIERO ELECTRÓNICO, considero que dicho trabajo reúne los requisitos y méritos suficientes para ser sometido a la presentación pública y evaluación por parte del jurado examinador que se designe.

En San Diego, a los trece días del mes de junio del año dos mil dos mil veintitrés.

Wilmer E. Sanz F.
7.130.496

REPÚBLICA BOLIVARIANA DE VENEZUELA
UNIVERSIDAD JOSÉ ANTONIO PÁEZ
FACULTAD DE INGENIERÍA

FI E 008 2022-3CR TG

Valencia, 14 de abril de 2023

Ciudadano:
VILLALONGA SOTO, GERARDO ALBERTO
27.443.979
Presente -

Cumplo con informarle que la comisión de Trabajo de Grado y Pasantías de la Facultad de Ingeniería en su reunión **N° 05-2023** de fecha **10/02/2023** aprobó el proyecto de grado titulado:

Diseño de experiencias prácticas con el sistema ALTYS de XILINX para el laboratorio de microprocesadores de la UJAP

Presentado por usted como requisito para optar al título de Ingeniero en Electrónica.

Se ratifica la designación del Tutor Académico que lo asesorará en el desarrollo de este proyecto a: Ing. Wilmer Eduardo Sanz Fernández, titular de la cédula de identidad V- 7.130.496

Atentamente

Dra. Laura Aurora Sáenz Palencia
Decana de la Facultad de Ingeniería

Facultad de Ingeniería UJAP DECANATO

c.c. Coordinación de Pasantías y Trabajo de Grado de la Facultad de Ingeniería

AGRADECIMIENTOS

Le agradezco primeramente a Dios, por ser mí guía, por darme sabiduría para lograr alcanzar mis metas y proporcionarme la determinación necesaria para vencer cada adversidad.

A mis padres Zaret Soto y Richard Villalonga, por darme el apoyo constante y creer en que podría cumplir mis sueños, por cada palabra de aliento en los momentos difíciles, son mi ejemplo de perseverancia y demostración que todo es posible.

A mi hermana, Diannellys Villalonga, que constantemente ha sido como una madre y ejemplo a seguir, admiro su determinación y fuerte, que gracias a ella este logro ha sido posible.

A mi novia, Rosangel Jiménez, que ha sido un apoyo incondicional en la carrera y en la vida, que ha estado en buenos y malos momentos, y por cada uno de las risas y felicidad.

A mis amigos, Moisés Trasolini, Juan Oliva, Gabriela Figueredo que son como mis hermanos, he podido disfrutar con ellos incontables momentos, alegrías, lágrimas y sé que desde el cielo Moisés estará cuidando y observando que si se logró dicha meta.

A mis tíos, Yaritza Villalonga, Yelluz Villalonga, York Villalonga, Rohiman Villalonga, Lucy y todos, que constantemente estuvieron en cada momento observando y apoyando el trayecto que tuve en la universidad y son parte de este logro.

A mi abuela, Juana Barrios que estuvo atenta y siempre enviaba su comida que subía los ánimos, apoyando y pendiente de como transcurría todo.

A mi Tutor, Wilmer Sanz, que ha sido un excelente guía e inspiración para seguir en esta hermosa carrera, agradezco que haya confiado en que podría llevar a cabo este proyecto, mi total admiración ha sido un excelente mentor en este trayecto.

A mis profesores, Antonio Rodríguez, Gerson Sánchez han sido excelente profesores que me han ayudado a crecer tanto profesionalmente, como personalmente

A mis primos, Miguel Macia, Sandra Macia, Ray Villalonga, Wilyer Villalonga y todos los que faltaron, agradecido por todo el apoyo, por ser mis hermanitos y siempre estar en todo momento, este también es su logro.

A la familia de mi novia, Nelsy Paredes, José Jiménez, Mariangel Jimenez y Angel Jimenez, por ser mi segunda familia y motivarme a seguir adelante, Le agradezco

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

REPÚBLICA BOLIVARIANA DE VENEZUELA
UNIVERSIDAD JOSÉ ANTONIO PÁEZ
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELECTRÓNICA

DISEÑO DE EXPERIENCIAS PRÁCTICAS CON EL SISTEMA ATLYS DE XILINX PARA EL LABORATORIO DE MICROPROCESADORES DE LA UJAP

Autor. Gerardo Villalonga
Tutor: Wilmer Sanz
Fecha: Junio 2023

RESUMEN

El presente trabajo de grado tiene como objetivo el desarrollo de experiencias prácticas con el diseño de hardware mediante el sistema ATLYS de XILINX para el Laboratorio de Microprocesadores de la UJAP. Esta propuesta se enfoca directamente en un solo objetivo el cual es favorecer el desarrollo de las competencias que los estudiantes obtienen en el Laboratorio de Microprocesadores de la UJAP. El estudio se adapta a los lineamientos metodológicos de un proyecto factible, bajo el diseño de investigación de campo, en un nivel descriptivo; y en el marco de la línea de investigación "Avances tecnológicos en tecnologías de información y comunicación" de la facultad de Ingeniería de la Universidad José Antonio Páez. La población y muestra son las tarjetas de desarrollo basadas en FPGA y la tarjeta de desarrollo ATLYS de XILINX. La recolección de datos se realizó por medio de revisión documental y entrevistas estructurada. Los resultados presentados consisten en simulaciones y descripción de interfaces y sistemas digitales implementados con el apoyo de tablas de proyectos sobre temáticas tales como el procesamiento digital de señales, monitoreo de variables, comunicación de datos y control digital, mediante la descripción de hardware con VHDL.

Descriptores: ATLYS, FPGA, VHDL.

INTRODUCCIÓN

Al pasar de los años, ha incrementado el uso de las FPGA estas toman cada vez más fuerza en la industria al ser una opción muy versátil y precisa, su crecimiento ha avanzado en áreas como el procesamiento digital de señales, telecomunicaciones, comunicaciones de datos, procesamiento de video e imágenes y como alternativa para la implementación de redes neuronales a partir del diseño del hardware. Cabe destacar que hubo un aumento en el uso de estas tarjetas en autos eléctricos por la rápida respuesta que se obtiene en el procesamiento digital de las señales que son enviadas por los sensores del auto. Es común ver está tecnología en los autos eléctricos debido al procesamiento en tiempo real de las señales para así obtener un mejor tiempo de reacción que son necesarios en el piloto automático que poseen los autos eléctricos.

Con este aumento de demanda que obtienen las FPGA y el como cada año aumentan las apuestas de las empresas por esta tecnología basada en matrices reprogramables, al ser una tecnología que empieza a tener su auge en procesos industriales complejos y las adaptaciones en la industria han sido un total éxito. Se observa el camino hacia donde se dirige este nuevo sector y como podrá tener un uso cotidiano en un futuro. Es por ello que recurrir al uso de nuevas tecnologías y a la ingeniera detrás de las FPGA es una forma de brindarles una mejor experiencia a los estudiantes del Laboratorio de Microprocesadores. Por medio de esta investigación se logran diseñar hardware que demostrara la flexibilidad que se ofrece en el hardware basado en semiconductores a base de matrices de bloques lógicos configurables o CLB.

Es importante destacar, que el proyecto reúne las características de una investigación de campo, y se encuentra estructurado en cuatro (04) capítulos, los cuales se especifican a continuación:

Capítulo I: Este presenta el problema que conduce al desarrollo de la investigación, así como su formulación que se tiene del problema, el objetivo general como sus objetivos específicos; por último, la justificación junto al alcance y limitaciones.

Capítulo II: En este capítulo se aborda el desarrollo del marco teórico, se presentan los antecedentes de la investigación, junto a las bases teóricas que se encuentran referidas al lenguaje VHDL, las características de la tarjeta de desarrollo ATLYS de XILINX, como el funcionamiento de los componentes utilizados en el presente trabajo.

Capítulo III: Se desarrolla el marco metodológico, con sus fases metodológicas en donde se exponen los instrumentos y técnicas para el análisis y recolección de datos para dar respuesta a los objetivos del estudio.

Capítulo IV: Se exponen los resultados obtenidos en cada uno de los objetivos que se proponen en el trabajo de investigación.

Capítulo V: Conclusiones y recomendaciones finales.

CAPÍTULO I

EL PROBLEMA

1.1 Planteamiento del Problema

Los diseñadores de circuitos digitales están en la constante búsqueda de formas alternativas de diseñar sistemas para proporcionar una solución óptima, que logre abarcar todos los problemas que se necesite en una aplicación o sistema electrónico. Luego de la introducción que tuvieron los sistemas lógicos programables (Field Programmable Gate Array o FPGA) en la década de los ochenta como sencillos sistemas digitales de lógica de acoplamiento (glue logic), los cuales tenían limitado número de recursos lógicos y su función principal era interconectar grandes bloques lógicos o dispositivos. Actualmente, se ha visto que estos dispositivos han llegado a un nivel muy alto de sofisticación, logrando tener FPGA de muy alto rendimiento, con sistemas de millones de compuertas y bloques embebidos de microprocesadores veloces, DSP (Digital Signal Processing) e interfaces de entrada/salida de muy alta velocidad.

Las FPGA destacan en el procesamiento digital de señales o DSP (Digital Signal Processing). Debido a que es una de las técnicas de la industria de la electrónica con un alto crecimiento. Es utilizada en una amplia gama de campos de aplicación, como comunicaciones de datos, telecomunicaciones, en procesamiento de videos y en mejoras para el procesamiento de imágenes, dentro de estas categorías entra también el reconocimiento de voz. Las FPGA tienden a ofrecer una excelente solución en las necesidades de los sistemas de procesamientos digitales de señales de alto rendimiento, bloques específicos, conectividad y capacidad de procesamiento.

Las FPGA son reprogramadas para realizar tareas en pre procesamientos de imágenes tales como: filtración de imagen, convolución y conversión de formato de color. Dado al avance tecnológico que se han tenido en los diseños, se puede acceder con facilidad a los datos que están almacenados en caché mediante una FPGA. Los enfoques que están basados en una FPGA son utilizados con frecuencia en la segmentación del color con el fin de implementar una red neuronal artificial para el seguimiento manual, segmentaciones de imágenes y reconocimiento de gestos con sistemas de realidad aumentada. El uso del FPGA en el procesamiento de imágenes es evaluar imágenes individuales o cuadros múltiples de una señal de video para la búsqueda o seguimiento de objetos o una extracción de información en profundidad y movimiento. Otro campo que se encuentra beneficiado es el de la robótica por los resultados que arroja en el procesamiento de

imagen en FPGA (Ver figura 1). La principal razón del aumento en la implementación de algoritmos basados en FPGA es el paralelismo.

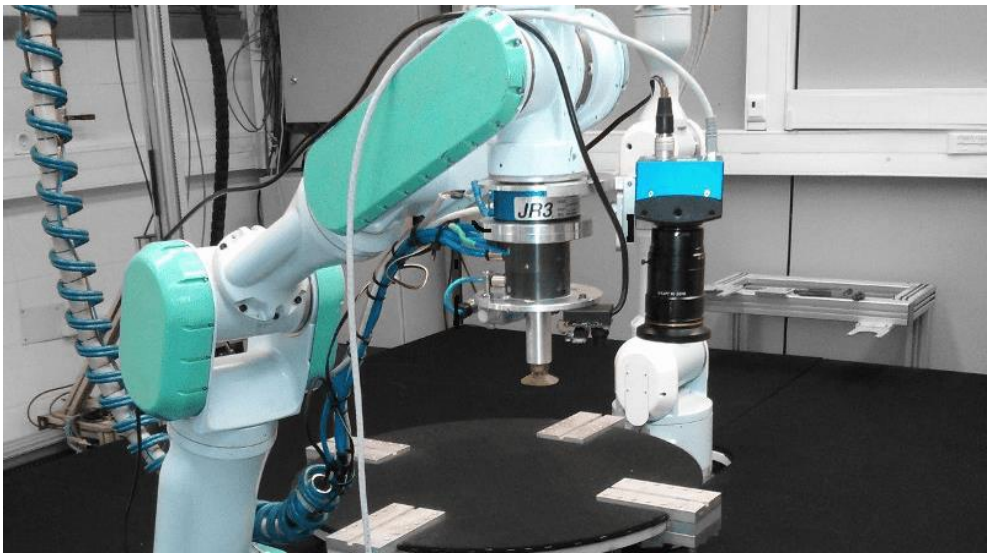


Figura 1. Control visual con cámara FPGA basado en VISP
Fuente: https://teralco.com/magazine/control-visual-con-camara-fpga-basado-en-visp/

Otro campo que logra encontrar beneficios en las FPGA es la inteligencia artificial la cual representa la próxima revolución tecnológica, cambiando así las formas en cómo operan las industrias y empresas. Se debe tener en cuenta que la inteligencia artificial es un concepto realmente amplio de máquinas que pueden realizar tareas de una manera óptima e inteligente, simulando las capacidades de análisis humano. Las redes neuronales son un conjunto de algoritmos que son altamente precisos, como es nombrado con anterioridad para el procesamiento de imágenes, sonidos y sistemas de recomendación.

Una red neuronal profunda, requiere un alto nivel de cómputo, mayor capacidad de potencia y menos tiempo en procesado de la información, sus diferentes sistemas digitales están beneficiados con el paralelismo de las FPGA. Gracias a los recursos de memorias que se encuentran disponibles en la FPGA donde logran implementarse redes neuronales independientes, que luego pueden ser conectadas de distintas maneras. La gran aceleración general sobre las implementaciones de la FPGA en el mercado por la visión artificial, la conducción autónoma y el centro de datos, estas aplicaciones están siendo mejoradas por las capacidades de la rápida implementación que presenta una FPGA con redes neuronales.

La FPGA se ha establecido como candidato preferido al momento de la implementación digital en una enorme cantidad de procesos industriales en la actualidad. Hasta hace poco tiempo, las FPGA se encontraban en dos enfoques distintos a nivel industrial, uno para diseñar sistemas digitales para control de procesos industriales a nivel secuencial (Software) basado en microprocesadores o procesadores de señal digital (DSP) y otro enfoque paralelo (Hardware), generalmente limitado para resolver partes específicas de problemas que requieran alto rendimiento. El desarrollo industrial de este segundo enfoque estuvo condicionado por el conocimiento limitado de la tecnología, las herramientas que existían de diseño, la falta de avances en dichas herramientas, el costo elevado que presentaban y la falta de funciones especializadas de hardware.

A medida que la FPGA fue evolucionando, esta aprovecho la reducción de la tecnología de fabricación, los proveedores observaron el potencial y comenzaron a desarrollar núcleos de procesadores flexibles que puedan ser implementados a partir de los recursos que se tenían de las FPGA estándar. Este nuevo enfoque obtuvo nuevos diseños que dieron como resultado un cambio de paradigma que constituye el principal activo actual de las FPGA, tanto así que pueden verse como plataformas muy potentes de System-on-Chip (SoC). La combinación en un solo dispositivo de procesadores integrados con periféricos adicionales de hardware optimizados y de alto rendimiento, ha abierto las puertas en todas las áreas de diseño digital para procesos industriales. Al igual como logra el control difuso, la principal contribución que se tiene de las FPGA es su capacidad para implementar un controlador como un sistema robusto en tiempo real.

Dentro de los sistemas robustos que requieren una precisión y cálculos de alta velocidad de procesamiento digital de señales DSP, el control de distintos motores exigen multifunciones y multiejes, se considera un robot en la fabricación industrial automatizada en donde se requiere realizar una actividad donde necesite un tiempo de respuesta rápido o incluso logra ser implementada en sistemas como el novedoso robot que es utilizado dentro de las cirugías medicas asistidas que necesita un control preciso en los ángulos de movimientos de los motores implementados en el sistema. La industria automovilística no se queda atrás, los nuevos autos eléctricos requieren una gran cantidad de sensores como cámaras, con el paralelismo que ofrecen las FPGA los sistemas resultan más eficiente y con un menor consumo eléctrico, siendo perfectos para el modo de piloto automático donde el procesamiento digital de señales en tiempo real es necesaria para evitar accidente o un retraso con las respuestas que se obtienen de los sensores, aumentando así la competencia de las FPGA sobre los sistemas ASIC.

Al analizar el crecimiento de la FPGA en la industria y la necesidad de diseñadores de FPGA, la compañía Lattice Semiconductor ha creado un centro de educación dentro del cual se encarga de capacitar a los ingenieros en el diseño de FPGA (Ver figura 2). Siendo así la fuerza impulsora de esta demanda, la electrificación del automóvil y la necesidad de las FPGA en la industria del control.

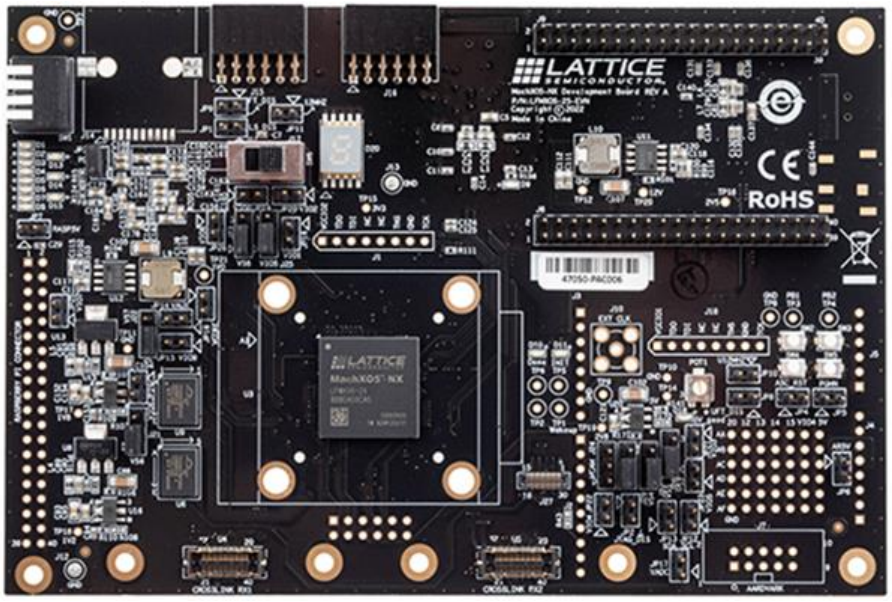


Figura 2. Tarjeta programable Mach05-NX de Lattice Semiconductor
Fuente: Página Oficial de Lattice Semiconductor

XILINX, que está especializada en el diseño de las FPGA automotrices, ha aumentado la producción para los OEM en la medida que las ADAS se convierten en un elemento indispensable en los nuevos vehículos que salen al mercado. Se tiene que Gowen Semiconductor se introduce en esta nueva tecnología para satisfacer la demanda que están teniendo las FPGA de grado automotriz y especialmente los diseños para la telemática, infoentretenimiento y los trenes con la capacidad de potencia en vehículos. Esta compañía señala en múltiples entrevistas que dichos dispositivos cuentan con la certificación AEC-Q100, dicha certificación verifica que cumplen con la resistencia y la durabilidad para soportar las condiciones de un vehículo.

Por su crecimiento mundial se tiene en cuenta que industrias reconocidas apuestan por las capacidades tecnológicas que alcanza esta tecnología, XILINX fue comprada por AMD en el 2020 por 35.000 millones de dólares, donde se da entender la apuesta que tiene a futuro AMD por las posibilidades únicas que brindan estas mismas.

Se resalta que esta tecnología no queda hasta ahí, su implementación está siendo fuertemente introducida en sistemas embebidos dando así que la combinación de procesadores y FPGA en el diseño de sistemas digitales está creciendo en muchos sectores. Como ejemplo se tiene a ePower Technology, una empresa de Dinamarca que diseña sistemas de control embebido en equipos para probar y entrenar músculos. El sistema de entrenamiento energético SYGNUM, el cual desarrollaron, utiliza en sus mecanismos un motor de cinco fases que está patentado para controlar de manera equilibrada la aplicación de resistencia de fuerza durante movimientos de ejercicios. ePower Technology usa FPGA para controlar los bucles de control de alta velocidad, estos son los que mantienen los puntos de ajuste de la velocidad y posición mientras que un procesador externo es el que se encarga de ejecutar un sistema operativo en tiempo real para controlar los bucles de control de baja frecuencia como se muestra en la (Ver figura 3).

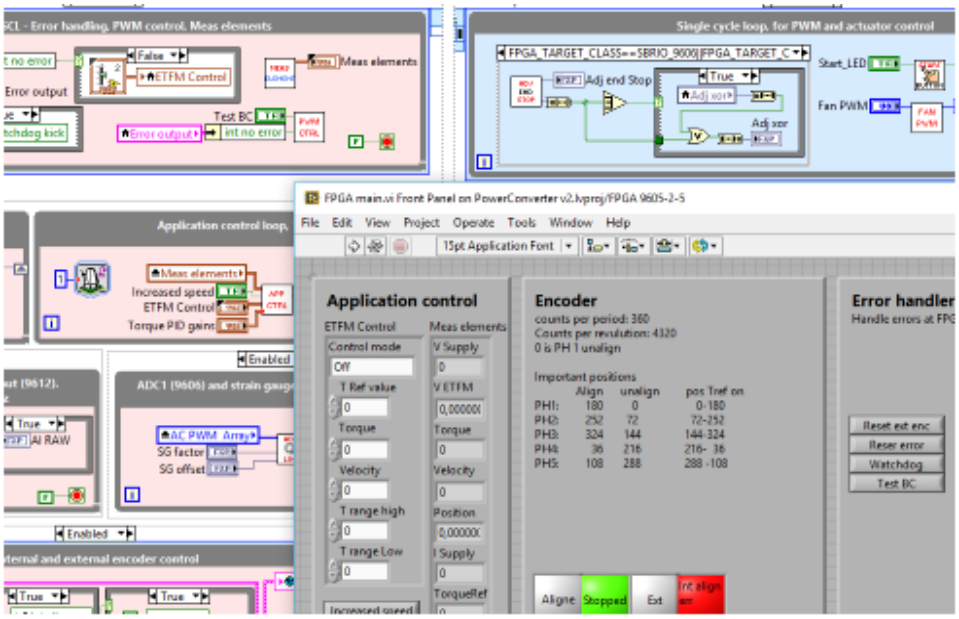


Figura 3. Plataforma de desarrollo LabView SRM basada en real-time y FPGA.
Fuente: Página Oficial epower technology

Otro ejemplo de arquitectura estándar que logra encontrarse en el sector energético es el de Xtreme Power, una empresa en EE.UU. la cual tiene un diseño de un sistema de almacenamiento de energía distribuida con control embebido y basado en varios procesadores y FPGA. Las FPGA se utilizan para tomar medidas precisas de alta velocidad de potencia trifásica y ejecutar algoritmos avanzados para determinar cómo responder mejor a la inestabilidad de la red eléctrica.

Mientras tanto, los procesadores permiten la comunicación Ethernet a otros nodos distribuidos y facilitan el acceso remoto a datos, la gestión de sistemas y diagnósticos.

Debido a este crecimiento se encuentran instituciones universitarias como la UNAM (Universidad Nacional Autónoma de México) en su pensum para cursar la carrera de Ingeniero Electrónico ven como materia obligatoria **"dispositivos electrónicos programables**" donde tienen como objetivo en el curso que el alumno diseñe sistemas digitales basados en dispositivos electrónicos programables; preparando así a sus egresados en el manejo de nuevas tecnologías y capacitándolos para cubrir la demanda de profesionales que requiere la industria actualmente.

1.2 Formulación del problema

¿Cómo se pueden ampliar las experiencias prácticas de los estudiantes de la asignatura Microprocesadores de la UJAP, de manera que obtengan capacitación en el diseño de hardware reconfigurable?

1.3 Objetivos de la Investigación

1.3.1 Objetivo General

Diseñar experiencias prácticas con el sistema ATLYS de XILINX para el laboratorio de microprocesadores de la UJAP.

1.3.2 Objetivos Específicos

- Describir las características del sistema ATLYS de XILINX y las herramientas de diseño asociadas al mismo.
- Elaborar una guía de usuario sobre las herramientas de software de XILINX para la implementación de circuitos basados en FPGA.
- Diseñar sistemas digitales con FPGA para experiencias prácticas reproducibles en el laboratorio de microprocesadores de la UJAP (manejo de entradas / salidas digitales, comunicación de datos y aplicaciones de control).
- Elaborar una guía de prácticas factibles basadas en el uso de FPGA para los estudiantes de la asignatura Microprocesadores de la UJAP.

1.4 Justificación de la Investigación.

Los avances tecnológicos que se están observando en la industria pueden decidir el futuro de los microprocesadores y como se implementaran en sistemas de control y en sistemas de comunicación. Las características que ofrece un FPGA al ser reconfigurable abren un sinfín de posibilidades al momento de resolver una situación o dificultad, logrando así que sea versátil al

momento de implementar un sistema y si se requiere una actualización de hardware lograrlo mediante su reconfiguración predeterminada y no tener que cambiar dicha tarjeta hasta que sea necesaria más capacidad de la que soporta la misma.

Se puede ver como las instituciones conocen la capacidad que traerán las FPGA al mercado, tanto así que preparan a sus egresados para los nuevos cambios que pueda haber en la industria. Se tiene en cuenta que la UNAM (Universidad Nacional Autónoma de México) no es la única universidad que tiene en su pensum esta asignatura, también universidades como Universitat de València preparan a sus egresados para el futuro que se avecina, logrando así cubrir la demanda que se tiene en esta área de la ingeniera. Estas reflexiones muestran la importancia de que la Universidad José Antonio Páez avance en la misma senda de sus pares más destacadas.

Cabe destacar que el consumo energético de la FPGA con respecto a otros tipos de tecnología, resulta más eficiente por el bajo consumo energético que ofrecen estás, demostrando una ventaja sobre alternativas como los microcontroladores. Siendo así amigables con el medio ambiente, sin dejar atrás el efecto positivo que trae hacia las industrias y laboratorios por la flexibilidad al momento de diseñar y actualizar un sistema o proceso industrial, aumentando el tiempo de respuesta para el mantenimiento y actualización de hardware, con esta ventaja los costes de producción y manteniendo disminuyen. Logrando economizar el producto final hacia el mercado.

Es así una herramienta que va encaminada hacia el futuro y a la mejora de experiencia dentro de los Laboratorios de Microcontroladores.

1.5 Alcance.

La investigación consiste en diseñar experiencias prácticas con el sistema ATLYS de XILINX para el laboratorio de microprocesadores de la UJAP, a manera de evaluar los aportes entregados por esta tecnología y la evolución que tendrían los estudiantes.

1.6 Limitaciones.

Durante el desarrollo de la investigación pueden surgir limitantes como la dificultad de la obtención del Software de XILINX Ise Design Suite 14.7. Debido a las restricciones que tiene dicha empresa sobre Venezuela para la descarga del Software, el cual es el encargado de la compilación y la sintaxis para la placa ATLYS de XILINX. Para el diseño se busca cubrir sistemas digitales basados en VHDL (Hardware Description Language), sin embargo, no cuenta con la adaptación de los sistemas digitales en el lenguaje Verilog, es decir que no se desarrollan sistemas en este lenguaje por el campo tan amplio que existe.

CAPÍTULO II
MARCO TEÓRICO

En toda investigación, se hace indispensable la consulta de distintas fuentes bibliográficas, las proporcionan las bases teóricas sobre las cuales se ha de fundamentar la investigación, Sabino (2010) señala que las teorías de estudio, "deben ser planteadas al igual que el problema desde el contexto macro al particular", (p.102). En el actual trabajo se extrae la información de distintas bibliografías para el sustento de todas las bases teóricas descritas en el presente trabajo, partiendo de teorías centrales, tales como el diseño lógico digital, el procesamiento digital de señales y Array Lógico Programable (PLA), para el diseño de los sistemas tanto el entorno de desarrollo XILINX ISE 14.7 y su lenguaje de descripción de hardware (VHDL) para la simulación y compilación de los diseños digitales.

2.1. Antecedentes de la Investigación

Los antecedentes representan una indagación bibliográfica de trabajos anteriores, que tiene pertinencia con las variables en estudio. Para Arias (2012) los antecedentes son "investigaciones realizadas anteriormente que guardan alguna vinculación con problema de estudio" (p.39). Se refiere a los estudios previos que tienen relación con el tema y que sirven de soporte teórico y metodológico para la investigación.

2.1.1 Antecedentes Internacionales

Trabajo de Grado realizado por Martin Caballero en la Universidad de Valladolid en el 2019 cuyo título fue "**Desarrollo en FPGA de un enlace de comunicaciones serie para un convertidor de potencia distribuido**" para optar por el título de Ingeniero Electrónico cuyo objetivo general de investigación fue realizar una implementación de un convertidor de potencia distribuido en una planta solar fotovoltaica para la producción de energía eléctrica donde es necesaria e imprescindible un enlace de comunicaciones que permita transmitir la información de forma fiable y a la suficiente velocidad para que se puedan tomar a tiempo las decisiones correspondientes de los switches de conmutación en función de los requerimientos del convertidor.

Los resultados obtenidos por dicho autor tras el testeo de las distintas combinaciones de mensajes que pueden ser enviados a través de los switches de la plataforma de trabajo de ATLYS, llega a la conclusión que el enlace de comunicaciones funciona a una velocidad de transmisión de 50 Mbps.

Así también el Trabajo de Grado realizado por Alejandro Aguilocho Domínguez en la Universidad De Sevilla (Escuela Politécnica Superior de Sevilla) en el año 2018 cuyo título fue "**Procesado de una señal de video para la detección de bordes en tiempo real sobre una FPGA**" para optar por el título de Ingeniero Electrónico Industrial cuyo objetivo general de investigación es realizar un procesamiento para la detección de bordes de una señal de video que es obtenida mediante una cámara y como resultado final mostrada a través de cualquier pantalla que disponga de tecnología HDMI o transmisión instantánea a un ordenador, para ello se planteó el análisis del diseño de un sistema base que es proporcionado por el fabricante de la placa de desarrollo para la comprensión de su funcionamiento, donde es necesario comprender la estructura que posee una imagen y el procesamiento digital que estas tienen, también interfaces avanzadas para el uso de memorias DDR, el control y la adquisición de datos de una cámara y la generación de señales HDMI.

Finalmente el investigador concluye que la realización de un sistema basado en FPGA es una tarea que requiere más tiempo y estudio comparado con el uso de microprocesadores, ya que los últimos disponen con una cantidad amplia de software especializados haciendo la programación más sencilla, pero la potencia en tiempo real de la FPGA da una compensación a el tiempo requerido.

Siguiendo el orden, En el desarrollo del Trabajo de Grado realizado Manuel Pérez Aguilar en la Universidad Nacional Autónoma de México en el año 2017 cuyo título fue "**Diseño de un microcontrolador de 32 bits con base en una FPGA**" para optar por el título de Ingeniero Eléctrico – Electrónico cuya investigación tuvo como objetivo general es el desarrollo de un microcontrolador de propósito general con base en una FPGA para la aplicación en la industria petrolera en el manejo de sensores y actuadores por medios de protocolos de comunicación que son útiles en las aplicaciones industriales donde se espera la presentación de un dispositivo físico y la evaluación del mismo, se tiene en cuenta que se hizo el uso de un FPGA comercial que posee las características más avanzadas para el desarrollo que se tuvo.

El cual estuvo diseñado para hacer parte de un sistema en una Petrolera, se tiene en cuenta que el diseño de sistemas embebidos a medida que aumenta la complejidad de estos, partiendo desde cero de cada componente de hardware, llega a ser costoso y poco práctico para el diseñador en cuestión. Por lo tanto, la idea del uso de propiedad intelectual (IP) pre-diseñados y pre-probados se convirtió en una gran alternativa. El microcontrolador fue diseñado como una alternativa para el protocolo de comunicaciones de los sensores y actuadores donde la FPGA

facilita el procesado de las señales y por su paralelismo aporta la rapidez requerida junto con la precisión, sin dejar atrás la flexibilidad que ofrece para realizar cambios en su funcionalidad cuando se requiera donde las conclusiones obtenidas por el investigador en cuestión es que el diseño del microcontrolador que se basó en una FPGA pone en un marco competitivo con respecto a los microcontroladores basados en hardcores gracias a la escalabilidad que posee una FPGA.

Por último, se observa una investigación de una revista relacionada al Trabajo de Grado fue realizada por Cecilia Sandoval Ruiz en la Universidad de Carabobo del año 2018, con el nombre de **"Modelado VHDL de control neuronal sobre tecnología FPGA orientado a Aplicaciones Sostenibles**", Esta investigación tuvo como objetivo general esquemas de control neuronal y el diseño personalizado de sus componentes en lenguaje descriptor de hardware (VHDL), el cual tuvo como propósito la construcción de un modelo matemático para el soporte de un control reprogramable y la optimización de estos esquemas para implementación con tecnología FPGA donde el autor concluye que el modelado de control neuronal sobre un hardware reprogramable, tiende a simplificar la etapa de modelado de sistemas digitales, de potencia y diseños del esquema de control avanzado, logrando así facilitar el entrenamiento en circuitos y la adaptación optima en tiempo real, de este modo amplia las aplicaciones para la configuración de arreglos de micro-convertidores.

2.2 Bases Teóricas

Para que se logre una mayor comprensión de la presente investigación, se presentara a continuación una serie de conceptos pertinentes al propósito de la investigación, al respecto Arias, 2012 expone que:

> Las bases teóricas se refieren al desarrollo de los aspectos generales del tema, los cuales comprenden un conjunto de conceptos y proposiciones que constituyen un punto de vista o enfoque determinado, dirigido a explicar el fenómeno o problema planteado por el investigador. (p.110).

Es por ello que definir las bases teóricas requiere de la capacidad de identificación, descripción, distinción y evaluación a fin de establecer la categoría requerida en la investigación.

2.2.1 Tarjeta ATLYS Spartan-6

La tarjeta de desarrollo ATLYS es una plataforma de desarrollo que cuenta con una serie de dispositivos y elementos ya instalados que le permite al diseñador realizar un sinfín de proyectos y ponerlos a prueba fácilmente. La ATLYS cuenta con la Spartan-6 LX45 FPGA de

XILINX; Las principales características son: comunicación por medio Ethernet a Velocidades de Gbits, HDMI, memoria DDR2 de 128 MB y de 16 bits, puerto USB y salida de audio (Ver Figura 4).

El Spartan-6 LX45 está optimizado para lógica de alto rendimiento y ofrece:

- 6822 slices, cada uno con cuatro LUT de 6 entradas y ocho flip-flops.
- Un bloque de RAM de alta velocidad de 2.1 Mbit.
- Cuatro controladores de reloj (8 DCM y 4 PPLS).
- Seis lazos de seguidores de fase (PPLS).
- Velocidad de reloj a más de 500 MHz.

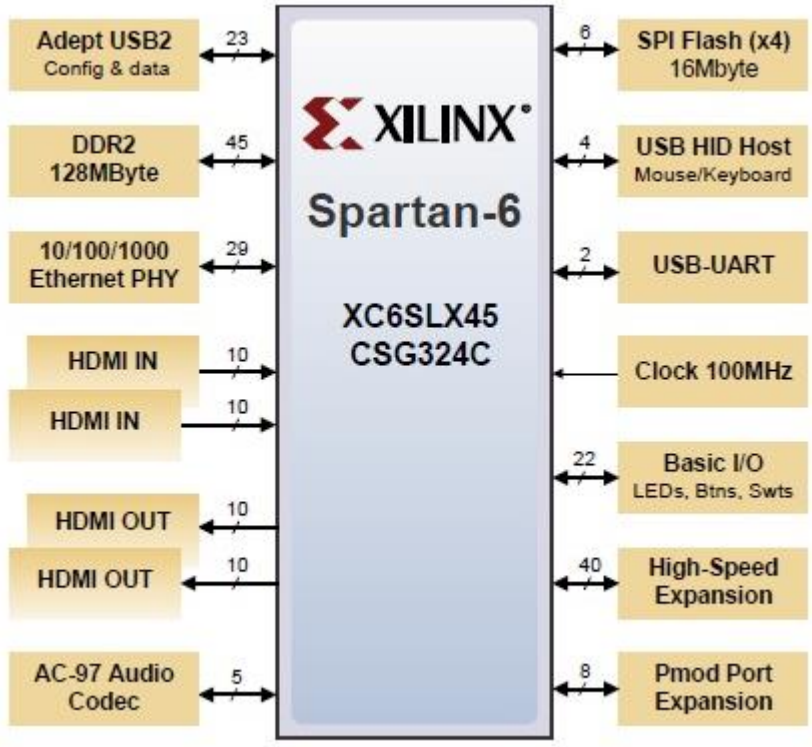


Figura 4. Esquema de Spartan-6
Fuente: Manual Esquemático de conexiones de la tarjeta ATLYS

2.2.2 Características Spartan-6 LX45

- FPGA Spartan-6 LX45 de XILINX, con encapsulado BGA de 324 pines.
- Memoria DDR2 de 16 bits y 128 MB.
- Ethernet PHY 10/100/1000.
- Puerto USB2 incorporado para la transferencia de datos y programación.

- Puerto USB-UART para comunicación serial con PC.
- Puerto USB HID (para ratón / teclado por medio de microcontrolador PIC24).
- Dos puertos de entrada de vídeo HDMI y dos puertos de salida HDMI.
- Codec de audio AC-97 con: línea de entrada, línea de salida, micrófono y auriculares.
- Monitor de energía para todas las fuentes de alimentación.
- Memoria FLASH SPI de 16 Mbyte x4 para alimentación de datos y de configuración.
- Oscilador CMOS de 100 MHz.
- 48 pines de entradas o salidas conectados a los conectores de expansión.
- Entradas y salidas generales como: 8 LED, 6 botones y 8 interruptores.
- Adaptador de voltaje de 20W y cable USB.

2.2.3 Osciladores y Reloj.

La ATLYS contiene un oscilador de 100 MHz conectado al pin L15 (la cual es una entrada de reloj del banco 1). Esta entrada de reloj puede ser conectada a cualquier controlador de reloj de SPARTAN-6. Cada controlador cuenta con 2 administradores digitales de reloj (DCM) y 4 lazos seguidores de fase (PLL).

Los administradores digitales de reloj proveen señales con cuatro posibles fases, las cuales son: 0º, 90º, 180º y 270º. También dispone de divisores de reloj por valor entre 2 y 16, y también por 1.5, 2.5, 3.5 hasta 7.5 y dos salidas de reloj desfasadas que pueden ser multiplicadas por un valor entero entre 2 y 32, y dividirse de forma simultánea por valores enteros de 1 a 32.

Los Lazos seguidores de fases (PLL) y los osciladores controlados de voltaje (VCO) se pueden programar para generar frecuencias en el rango de 400 MHz a 1080 MHz mediante la configuración de tres conjuntos de divisores programables en el momento de la configuración del FPGA. Las salidas del oscilador controlado por voltaje (VCO) tiene ocho salidas desfasadas entre sí con valores de: 0º, 45º, 90º, 135º, 180º, 225º, 270º y 315º, y se puede dividir por valores enteros entre 1 y 128.

2.2.4 Teclado

Los teclados usan controladores de colector abierto para ser conectado a un dispositivo que tenga funciones de anfitrión (HOST) con capacidad de establecer comunicaciones a dos hilos; cuando el dispositivo anfitrión no está enviando datos al teclado, los pines deben estar configurados como entrada.

En el protocolo PS/2 el teclado escanea constantemente las teclas para determinar cuál es la que se ha presionado, cada tecla en el teclado tiene asignado un código de exploración que es el que se envía al anfitrión cuando se presiona; en caso de que la tecla se mantenga presionada el código de tecla será enviado cada 100ms. Cuando la tecla es soltada se envía un código que indica que la tecla se ha soltado (F0) seguido se envía el código de la tecla soltada Cuando se desea producir un carácter en mayúscula es necesario presionar dos teclas al mismo tiempo; por ejemplo se tiene que presionada primero la tecla mayúscula para la cual el teclado enviara el código de tecla correspondiente, luego se debe presionar la tecla de la letra deseada y entonces el teclado enviara el código de tecla, cuando las teclas se sueltan se enviara el código de tecla soltada seguido del código de la tecla. Por cada tecla que se suelte se envía el código de tecla soltada. Hay teclas especiales a las cuales les llaman teclas del código ASCII extendido y que cuando se sueltan envían un código de tecla suelta diferente a las demás; este código especial de tecla suelta es E0 F0 y luego se envía el código de la tecla que se ha soltado.

Se verán algunos códigos que el dispositivo anfitrión puede enviar al teclado.

- **ED:** Controlar los LED de: NUM LOCK, CAPS LOCK, SCROLL LOCK. Y el teclado responde al anfitrión con el código FA si ha recibido el código ED, después de que el teclado ha respondido con el código FA, el anfitrión debe enviar un dato con el estado de los Led que desea encender, la ubicación de los bits de control de los Led es: el bit 0 es para SCROLL LOCK, el bit 1 es para NUM LOCK y el bit 2 es para CAPS LOCK; los demás bits son ignorados.
- **EE:** Respuesta (Test). Cuando el anfitrión envía este código el teclado responde con el código EE.
- **FE:** Con este código el anfitrión le pide al teclado que envié el último código de tecla oprimida.

El teclado solo inicia una comunicación cuando las líneas de datos y de reloj se encuentran inactivas y están en un valor lógico de 1. Antes de enviar cualquier dato el teclado verifica si el anfitrión está utilizando las líneas. Para facilitar la identificación del uso de las líneas de comunicación, la señal de reloj se utiliza como indicador de: "Libre para enviar" ya que el reloj debe estar en un valor de 1 lógico; si el dispositivo anfitrión desea que el teclado no inicie ninguna comunicación solo debe colocar el valor lógico de la señal de reloj a 0.

El formato de datos que envía el teclado al dispositivo anfitrión es una trama de 11 bit, los cuales se detallan a continuación: el primer bit es el bit de arranque (START), los 8 bit siguiente es el dato correspondiente a código de teclas o códigos de control, el bit 10 es un bit de paridad y el bit 11 es el bit de parada (STOP). En total son 11 transiciones de reloj a una frecuencia de 20 a 30 kHz y los datos son válidos en los flancos de bajada del reloj.

En la (Ver figura 5), en forma de teclado se muestran los códigos de teclas, para las principales teclas de un teclado estándar.

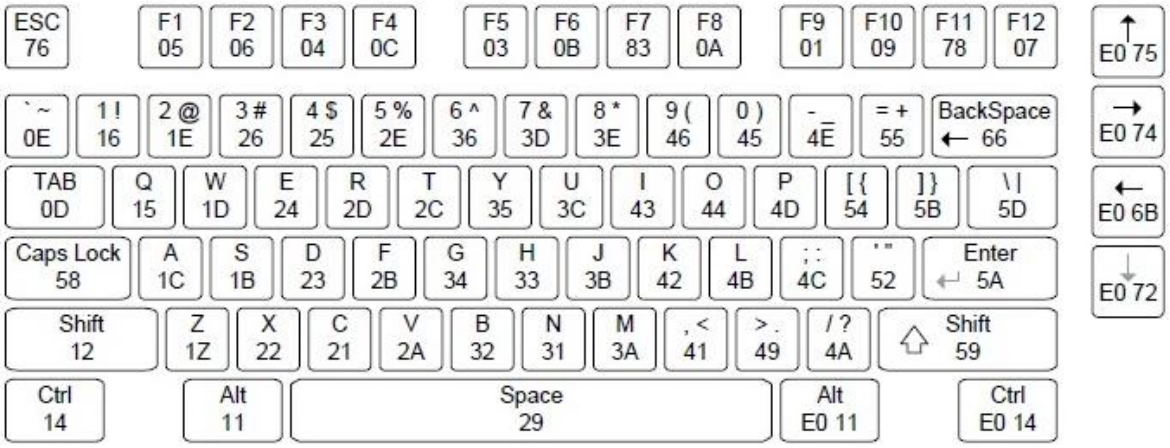


Figura 5, teclado estándar.
Fuente: Manual Esquemático de conexiones de la tarjeta ATLYS

2.2.5 Lenguaje VHDL

Actualmente, cuando se habla sobre el lenguaje más utilizado para la descripción de hardware a un nivel industrial y de diseño de sistemas digitales, se tiene a VHDL (Hardware Description Language). Donde tuvo su primera aparición en la década de los ochenta como un lenguaje estándar para el diseño digital industrial, demostrando capacidades para soportar el proceso de diseños electrónicos complejos, capaz de disminuir considerablemente el tiempo de diseño y los recursos tecnológicos requeridos.

Este utiliza distintos niveles de abstracción. VHDL viene de las siglas VHSCI (Very High Speed Integrated Circuits), observando así que VHDL permite la aceleración de procesos, Téngase en consideración que VHDL no es un lenguaje de programación, con esto se entiende que el conocimiento de la sintaxis no implica que se tiene la capacidad de diseñar en esta lenguajes sistemas digitales complejos, VHDL como es nombrado anteriormente es un lenguaje de

descripción de hardware, que permite la descripción de circuitos eléctricos digitales asíncronos o síncronos

VHDL permite una descripción estructural del circuito (descripción a partir de subcircuitos más sencillos), especificando la funcionalidad del circuito utilizando formas similares a los lenguajes de programación. La meta más importante que se tiene con VHDL es que sea capaz de simular perfectamente el comportamiento lógico de un circuito sin que el programador necesite imponer restricciones.

2.2.6 Elementos Básicos VHDL

Un sistema digital está descrito por sus salidas y entradas y la relación que existe entre ellas, en el caso de VHDL por un lado se busca describir la apariencia externa del circuito: entradas y salidas; y por otro lado la manera de relacionar las entradas con las salidas. La apariencia externa se conocerá como entity, y la descripción interna del circuito donde se encuentra el comportamiento del mismo architecture, toda arquitectura tiene que estar asociada a una entity.

Entity

Una entidad en un circuito, se entiende como la abstracción que se requiere ya sea de un sistema electrónico complejo a una simple compuerta lógica. La entidad es la manera que se describe la forma exterior del circuito, en ella se inicializan las entradas y las salidas que tendrá el diseño. Una entidad es semejante a un símbolo esquemático en los diagramás electrónicos, el cual tiene la descripción de las conexiones externas del dispositivo hacia el resto de elementos. Se resume con las siguientes características:

- Describe el exterior del circuito electrónico.
- Nombra y enumera las entradas y salidas, como sus tipos de datos.
- Tiene la información necesaria para conectarla a circuitos externos al diseño.

Ejemplo de la descripción en lenguaje VHDL en la (Ver figura 6):

```
entity contador is
port(
   clk,reset: in std_logic;
   Q: out std_logic_vector(3 downto 0)

);
end contador;
```

Figura 6, Descripción de la entidad en VHDL.
Fuente: Entorno de desarrollo ISE Design Suite 14.7

Los puertos descritos en la entidad pueden ser como en la figura 6, **in** para las entradas clk y reset del circuito y **out** para la salida Q del contador del ejemplo, se tienen también las entradas-salidas **inout** o **buffer**. Se debe tomar en cuenta que los puertos en la entidad no pueden modificar su valor posteriormente debido que esta son las conexiones externas que tendrá el circuito.

Architecture

Las partes de entidad y arquitectura son utilizadas para describir el diseño digital completo. Una arquitectura es la encargada de otorgar funcionabilidad a la entidad, de la cual hace referencia, es decir, dentro de la architecture tendremos que describir el comportamiento de la entidad a la que está siendo asociada donde se utilizaran sentencias y expresiones propias de VHDL. Sus características son:

- Describe internamente el circuito
- Tiene señales internas, funciones, procedimientos, constantes
- La descripción de la arquitectura tiene dos niveles, estructural o por comportamiento.

El código VHDL propiamente dicho se encuentra dentro de la architecture, cada architecture está asociada a una entidad, como en la figura 7. Se indica en la primera instancia. A continuación, y antes del begin, se podrán definir todas las variables (señales) internas que sean necesarias para describir el funcionamiento necesario. Observemos la estructura que tiene una arquitectura en la (Ver figura 7).

```
architecture Behavioral of contador is

signal contar: std_logic_vector(3 downto 0) := "0000";
signal cuenta: integer range 0 to 49999999 := 0;
begin

process(clk,reset) is
begin
   if reset = '1' then
      contar <= "0000";
      cuenta <= 0;
   elsif clk'event and clk = '1' then
      if cuenta = 49999999 then
         cuenta <= 0;
         contar <= contar + 1;
      else
         cuenta <= cuenta + 1;
      end if;
   end if;
end process;

Q <= contar;

end Behavioral;
```

Figura 7. Estructura de una arquitectura en VHDL.
Fuente: Entorno de desarrollo ISE Design Suite 14.7

Identificadores

En VHDL existen tres clases de objetos por defecto:

- Constant. Son el tipo de objeto en esta clase que tienen un valor inicial permante, el cual es asignado de una forma previa a la simulación y que no permite que sea modificado durante está.
- Variable. Son el tipo de objeto en esta clase que continen un único valor donde este a diferencia del constant puede ser modificado durante la simulación con una sentencia de asignación. Estan son utilizadas generalmente en índices, principalmente son vistas en instrucciones de bucles o tambien para tomar valores que permitan modelar componentes, cabe destacar que las variables no representan conexiones dentro de la arquitectura o estados de memoria.
- Singnal. Este ultimo objeto nos representan estados de memorias o las mismas conexiones en si, estas logran ser sintetizadas para el diseño, visto de otra manera, a cada a objeto en VHDL que es declarado como un signal le corresponde ya sea un cable o un elemento de memoria (biestables, registros) en el diseño del circuito.

Tipos de puertos

- BIT. Este tipo de puerto solo admite valores 0 y 1, es la representación de los valores binarios.
- BIT_VECTOR(RANGO). Siempre se definira el rango entre paréntesis, indicando e número de bits de dicho vector, éstos sólo pueden estar formados por ceros y unos.
- BOOLEAN. Este tipo de puerto solo admite valores true o false
- CHARACTER. Cualquier carácter con codigo ASCII
- INTEGER. Admite cualquier número entero dentro del rango establecido
- STD_LOGIC. Este tipo de puerta esta predifinido por el estándar IEEE 1164. Este logra ser la representación de un puerto real, donde tiene una logica multievalueda de 9 valores. Además del “0” lógico y el “1” lógico, posee alta impedancia “Z”, un valor desconocido “X”, valores sin inicializar “U” entre otros.
- STD_LOGIC_VECTOR(RANGO). Esta es la representación de los elementos std_logic, posee las mismas reglas para la asignacion y la misma definición para su rango que bit_vector.

2.2.7 Liquid Cristal Display (LCD1602A de 16x2)

Los LCD alfanumericos son una pantalla plana y delgada que está formada por un numero determinado de pixeles en color o monocromaticos, que se encuentran delante de una fuente de luz reflectora. Estos cuentan con una memoria interna donde son almacenados todos los caracteres que se requieren para mostrar en la pantalla, los cuales se pueden extender a ocho caracteres especiales adicionales. Los caracteres soportados por la mayoria de modelos de LCD son los siguientes en la (Ver figura 8):

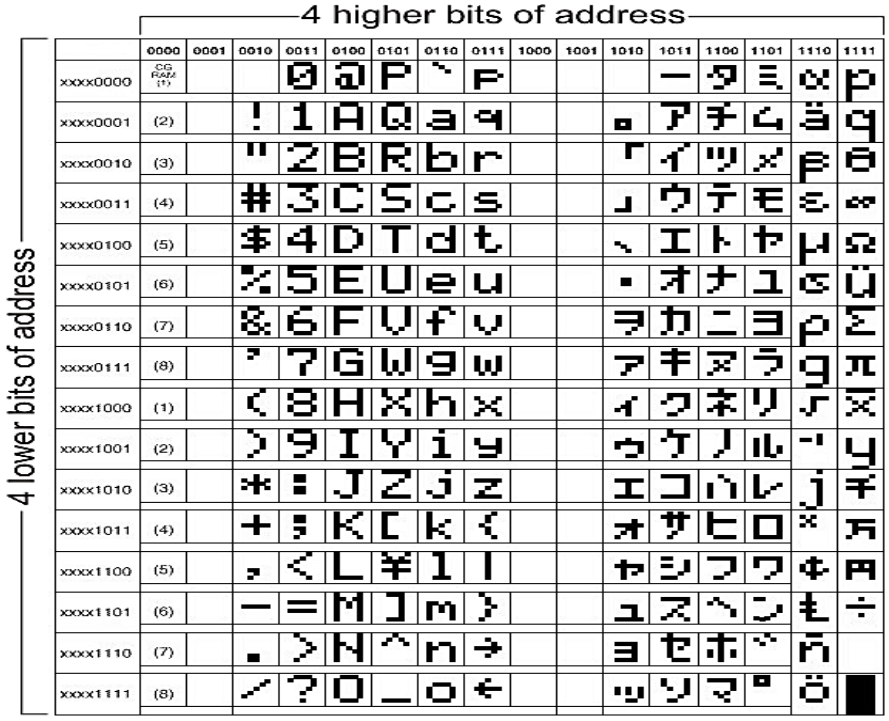


Figura 8. Caracteres soportados por una LCD.
Fuente: Hoja de datasheet de LCD 1602A

Secuencia de inicialización de la LCD para 4 bits

Se necesita una secuencia de iniciaización para la LCD en el caso de la investigación se trabajara en 4 bits para el ahorro de pines utilizados en la FPGA, esta inicialización se realizara de manera automatica con la FPGA si se cumplen los requisitos de alimentación expuestos en su manual la inicialización estara preparada para recibir los caracteres entregados por el teclado.

Dichos requisitos estan constituidos para que el tiempo debera mantenerse estable la tension de 0.2 V a 4.5 V minimo necesario sea entre 0.1 ms a 10 ms, teniendo en cuenta tambien que el tiempo de desconexión debe ser como minimo de 1 ms antes de volver a conectar. La secuencia de inicio ejecutada es la siguiente en la (Ver figura 9):

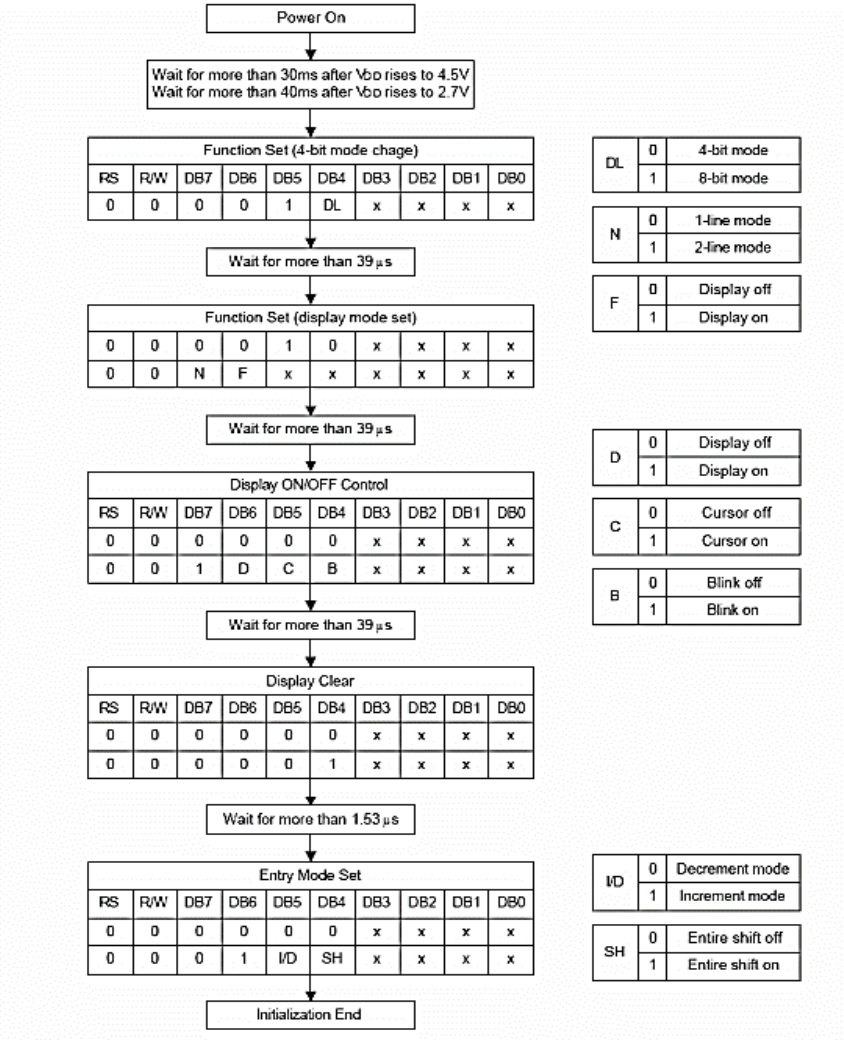


Figura 9. Esquema para inicializar LCD
Fuente: Hoja de datasheet de LCD 1602[a]

2.2.8 Sensor Ultrasonico HC-SR04.

El sensor HC-SR04 es un sensor de distancia el cual es de bajo costo que utiliza en su arquitectura ultrasonido para conocer la distancia que se encuentra de un objeto en un rango que tiene de 2 cm a 450 cm. Es destacado entre los sensores de distancia por su pequeño tamaño, el bajo consumo energetico y excelente precisión.

El sensor HC-SR04 esta diseñado con dos transductores: un receptor y un emisor piezoelectricos, el funcionamiento consta del emisor piezoelectrico emitira 8 pulsos de ultrasonido de 40 kHz, luego de que este reciba una señal en el pin TRIG, las ondas del sonido se transportan por el aire y rebotan al encontrarse con el objeto, luego este rebote se transportaa de vuelta para ser detectado por el receptor piezoelectrico del sensor, el tiempo del ECO sera medido por un

microcontrolador o FPGA como seria en el caso de la investigación y asi calcular la distancia que tiene dicho objeto.

Con la capacidad de paralelismo que se cuenta en las FPGA, se lograra diseñar una arquitectura capaz de medir la distancia que se encuentra dicho objeto.

Caracteristicas

- Voltaje de operación: 5 V.
- Corriente de reposo: < 2 mA.
- Corriente de trabajo: 15 mA.
- Rango de medición: 2 cm a 450 cm.
- Precisión: 3 mm.
- Ángulo de apertura: 15 º.
- Frecuencia de ultrasonido: 40 kHz.
- Duración minima del pulso de disparo TRIG (Nivel TTL): 10 uS
- Duración del pulso ECO de salida (Nivel TTL): 100 - 25000 uS
- Dimensiones: 45*20*15 mm
- Tiempo minimo de espera entre una medida y el inicio de otra 20 ms (recomendado 50ms).

2.2.9 Servomotor

El servomotor es un tipo de motor con caracteristicas especiales que permite controlar la posicion del eje en un momento dado. Este diseño le permite moverse en una determinada cantidad de grados y mantener la posicion en la que se encuentra. Al hablar de un servomotor se hace referencia a un sistema compuesto por componente electromecánicos y electrónicos.

El servomotor en su interior esta compuesto por un motor DC común. El eje del motor se acopla a una caja de engranajes que tiene similitud con una transmisión. Esto se hace para potenciar el torque del motor y poder mantener una posición fija cuando se requiera. De forma similar a un automóvil, a mayor velocidad, menor torque. E circuito electrónico es el encargado de manejar el movimiento y la posición del motor.

La presencia del sistema de engranajes hace que el movimiento del eje del motor sienta una incer muy superior a la de un motor DC convencional, Se observa que un servomotor es un conjunto de partes que forman un sistema.

Existen servomotores para cualquier tipo de uso. En la industria, la robótica, en el interior de las impresoras, máquinas CNC. Los servomotores de modelismo que son comunes en

aplicaciones der robótica operan a bajos voltajes, tipicamente entre los 4 y 6 voltios, éstos pueden clasificarse según sus características de rotación.

- **Servomotores de rango de giro limitado:** Es el tipo de servomotor más utilizado. Permite una rotación de giro de 180 grados, por lo cual son incapaces de completar una vuelta completa.
- **Servomotores de rotación continua:** Se caracterizan por ser capaces de girar 360 grados, es decir, una rotación completa. Su funcionamiento principal es similar al de un motor DC, pero con características propias de un servomotor. Esto quiere decir que es posible su control tanto en posición, como velocidad de giro.

2.3 Definición de términos.

- XILINX ISE: Es la herramienta de software de XILINX para la sintesis y el análisis de los diseños HDL, donde el objetivo encargado para dicho entorno es el desarrollo de firmware incorporados para la familia de circuitos integrados en FPGA y CPLD de XILINX
- TTL: El TTL (Tiempo de vida) esta definido como un mecanismo que es usado para dar un limite a la información que circula por una red.
- Ultrasonido: Es una serie de ondas mecanicas, donde la frecuencia se encuentra por encima de la capacidad de percepción el sonido el oido humano.
- Piezoelectrico: Es la capacidad de la carga eléctrica para acumular en ciertos materiales sólidos, como cristales, algunos tipos de cerámicas y matería biológica. En respuesta a la tensión mecanica aplicada.
- ALU: Es un circuito lógico digital que calcula operaciones aritméticas como suma, resta, multiplicación y división, también operaciones lógicas entre dos números.
- Slices: Es una matriz unidimensional de una secuencia de elementos consecutivos de otra matriz unidimensional.

CAPÍTULO III
MARCO METODOLÓGICO

3.1 Enfoque de la investigación

El enfoque de la investigación, corresponde con el enfoque cuantitativo. Damiani (2005) expone: "la cuantificación consiste en atribuir algunas dimensiones, a algunas propiedades o cualidades de los conceptos, un orden de naturaleza clasificatoria, que permita establecer una correspondencia entre las dimensiones de cada concepto y el mundo numérico" (p. 121). Así, pues en el presente estudio se analizan variables que se expresan en forma numérica, y luego se realiza el análisis matemático que da base a conclusiones y recomendaciones.

3.2 Tipo de Investigación

Según el Manual de Trabajos de Grado de Especialización y Maestría y Tesis del UPEL (2016), el proyecto factible hace referencia a "la investigación, elaboración y desarrollo del diseño de un modelo operativo viable para solucionar problemas requerimientos o necesidades de organizaciones o grupos sociales; puede referirse a la formulación de políticas, programas, tecnologías, métodos o procesos" (p.21). La presente investigación, se apoya en el tipo de investigación conocido como proyecto factible, en vista de que busca desarrollar un procedimiento que comprende, a su vez, tecnología con métodos para la resolución de la problemática expuesta. En este caso, se requieren realizar unas experiencias prácticas para el laboratorio de microprocesadores con el sistema ATLYS de XILINX, donde son expuestas las capacidades de la tarjeta antes mencionada.

3.3 Diseño de la investigación

El presente Trabajo de Grado se realizará bajo los lineamientos metodológicos de un diseño de investigación de campo, la cual según el Manual de Trabajo de Grado de Maestría y Tesis Doctórales de la UPEL (2016). Donde define que:

"Se entiende por investigación de Campo, el análisis sistemático de problemas en la realidad, con el propósito bien sea de describirlo, interpretarlos, entender su naturaleza y factores constituyentes, explicar sus causas y efectos, o predecir su ocurrencia, haciendo uso de métodos característicos de cualquier de los paradigmas o enfoques de investigación conocidos o en desarrollo" (p. 22).

La presente investigación se basó en una investigación de campo, ya que es posible recolectar información relacionada con el problema mediante registros originales sin datos agregados mediante procedimientos estadísticos y modelos matemáticos.

3.4 Nivel de la Investigación

El Trabajo de Grado se enmarcó dentro del nivel de investigación descriptiva, sobre el cual Arias (2012). Define que: "consiste en la caracterización de un hecho, fenómeno o grupo con el fin de establecer su estructura o comportamiento" (p. 22). En el presente caso se investigara y diseñara sistemas digitales los cuales serán reproducibles en el laboratorio de microprocesadores de la UJAP por sus estudiantes y ayudarlos mediante guías con las respectivas problemáticas que se le presente, aun cuando no se formulen hipótesis, tales variables aparecen enunciadas en los objetivos de la investigación.

3.5 Población y muestra

Arias, F. (2012), Da una definición sobre la población y la muestra en una investigación como: "conjunto para el cual serán validadas las conclusiones que se obtengan: a los elementos o unidades (personas, instituciones o cosas) involucradas en la investigación" (p.22).

3.5.1 Población

Se establecen los elementos sobre los cuales se aplicarán los postulados y se platearan las alternativas propuestas en esta investigación, en este sentido Arias, F. (2012) define a "la población, o en términos más precisos población objetivo, es un conjunto finito o infinito de elementos con características comunes para los cuales serán extensivas las conclusiones de la investigación. Esta queda delimitada por el problema y por los objetivos del estudio." (p.96).

3.5.2 Muestra

Para efecto de la investigación se hace necesario la selección de muestra, para tal caso Arias (2012), señala que: "Un subconjunto representativo y finito que se extrae de la población accesible." (p.83). Tomando esto como referencia tenemos que la muestra es un grupo o zona seleccionada en donde se aplicará la investigación con el fin de mejorar dicha muestra.

Para el proyecto actual se toma como referencia la población a los sistemas de tarjetas reprogramables como o son las FPGA y como muestra a las tarjetas reprogramables de la familia XILINX como ATLYS.

3.6 Técnicas de Recolección de Datos

Bernal. (2006) tiene como definición que dichas técnicas son "el proceso de obtención de datos e información útil para el desarrollo del sistema y procedimientos a proponer" (p.114). En el presente proyecto para obtener la información concerniente a la misma se aplicaron técnicas como: Revisión Documental y Entrevista Estructurada.

3.6.1 Revisión documental:

Según el Manual para la elaboración de Trabajo de Grado de la Universidad Pedagógica Experimental Libertador (UPEL, 2016), "consiste en la etapa del modelo científico a través de la cual, el investigador reúne los antecedentes teóricos y las investigaciones anteriores existentes sobre el tema dado" (p.123). Para efectos de esta investigación, la revisión documental es una técnica fundamental para poder conocer antecedentes e información relevante sobre el proceso descrito, brindándole soporte y nuevos conocimientos a la investigación que permitan exponer nuevas perspectivas para el cumplimiento de los objetivos.

3.6.2 Entrevista estructurada

Arias (2012) describe la entrevista estructurada como "una serie de preguntas preestablecidas", es una técnica basada en la formulación de preguntas guiadas, "cara a cara" entre el entrevistador y el entrevistado acerca de un tema previamente determinado, de tal manera que el entrevistador pueda obtener la información requerida.

3.7 Instrumentos de Recolección de Datos

Balestrini (2008), Para dicho autor los instrumentos de recolección de información "Son los medios que permiten observar y registrar características, conductas, etc., y en general cualquier otro dato que se desea obtener en una situación educativa a investigar, evaluar o supervisar". Con esta definición, se podrán emplear los siguientes elementos de recolección de información:

- **Guías prácticas:** Para llevar a cabo la recolección de información mediante guías para profundizar los temas a tratar en dicha investigación y así tener los aspectos más relevantes para el desarrollo de la presenta investigación.
- **Análisis de Contenido**: Bernal, C. (2006). afirma que este análisis amplía la descripción del problema e integra la teoría con la investigación y sus relaciones mutuas; en una palabra, es la teoría del problema y tiene como fin ayudar a precisar y a organizar los elementos contenidos en la descripción del problema, este instrumento será empleado para dar cumplimiento a los objetivos específicos planteados.

3.8 Validez

Para la calidez de un instrumento de recolección de datos, como lo son los análisis de contenido y guías prácticas, deben poseer dos puntos esenciales, validez y confiabilidad. Estas características garantizan y permiten la coherencia, persistencia y consistencia de los datos. Según

la definición de Palella y Martíns (2012) afirman: “la validez de un instrumento representa la relación entre lo que se mide y aquello que realmente se quiere medir”. Por otro lado, Rusque (2003) señala que: “La validez representa la posibilidad de que un método de investigación sea capaz de responder a las interrogantes formuladas”.

3.9 Técnicas de análisis de información.

En el presente Trabajo de Grado los datos recolectados deben tener un significado dentro de la investigación, se hace necesario la introducción de métodos para el análisis de datos, que tengan el propósito de dar respuestas a las interrogantes y objetivos que son planteados en la presente investigación, como organizarlos y dar evidencia de los principales hallazgos encontrados, donde serán relacionados de una manera directa con las bases teóricas que sustentan los conceptos y teorías manejadas, así como, con los conocimientos que se disponen en relación al problema que se propone estudiar. Según Arias (2012, p. 114), expresa que “en este punto se describen las distintas operaciones a las que estarán sometidos los datos que se obtengan: clasificación, registro, tabulación y codificación si fuere el caso”.

En esta investigación se empleará la técnica de análisis de contenido de textos, con ellos es posible el estudio del lenguaje VHDL como el manejo de la Tarjeta de ATLYS de XILINX.

3.10 Fases Metodológicas

Las fases metodológicas que se desprenden de este Trabajo de Grado son las siguientes:

- **Fase I. Descripción de las características del sistema ATLYS de XILINX y las herramientas de diseño asociadas al mismo**.

En esta primera fase busca describir las conexiones y componente que nos ofrece la tarjeta de desarrollo ATLYS de XILINX para sus usos en la presente investigación como las herramientas de diseño la cual es ISE Design Suite 14.7 con VHDL.

- ✓ Descripción de las características del sistema ATLYS por el manual
- ✓ Descripción del entorno de desarrollo ISE Design Suite 14.7 con las características que presenta

- **Fase II. Elaboración de una guía de usuario sobre las herramientas de software de XILINX para la implementación de circuitos basados en FPGA.**

En la segunda fase se elabora una guía de usuario para el correcto uso de las herramientas que ofrece XILINX al diseñador de circuitos digitales basados en FPGA.

- ✓ Instalación del entorno de desarrollo ISE Design Suite 14.7 de XILINX, el cual es utilizados para los diseños que serán visto en la presente investigación
- ✓ Manejo de la sintaxis de VHDL con la herramienta ISE Design Suite 14.7 de XILINX
- ✓ Uso del banco de pruebas para simulaciones de circuitos
- ✓ Ejemplos de Circuitos combinacionales y secuenciales
- ✓ Ejemplos para el manejo de Máquinas de Estado en VHDL basados en FPGA.

- **Fase III. Diseño de sistemas digitales con FPGA para experiencias prácticas reproducibles en el laboratorio de microprocesadores de la UJAP (manejo de entradas / salidas digitales, comunicación de datos y aplicaciones de control).**

En la fase III luego de la guía de usuarios para la introducción en el mundo de las FPGA, se diseñan distintos circuitos digitales con la idea de crear experiencias prácticas reproducibles en el laboratorio de microprocesadores de la UJAP como lo serán: el manejo de entradas / salidas digitales, comunicación de datos y aplicaciones de control.

- ✓ Desarrollar el diseño en VHDL sobre el manejo de entradas / salidas digitales mediante el uso de un teclado con comunicación PS/2 donde será mostrada la información en una pantalla LCD de 16x2 para observar las capacidades que da la tarjeta de desarrollo ATLYS de XILINX, en este diseño estará compuesto por una unidad de procesamiento para la comunicación PS/2, un decodificador encargado de mostrar la tecla presionada en el LCD de 16x2, una unidad lógica para la inicialización que es necesaria para el LCD y mostrar lo versátil al momento de optimizar un proceso y aprovechar al máximo el paralelismo que ofrece esta misma para la lectura de datos en tiempo real.
- ✓ Luego seguirá el desarrollo del diseño para la comunicación de datos, donde se podrá observar cómo pueden reproducirse distintos tipos de hardware que para el diseño de comunicación de datos será utilizado un sensor ultrasónico, donde a nivel de hardware tendrá una unidad encargada del procesamiento de la señal entregada por el sensor ultrasónico y mostrar la distancia por medio de los leds presentes en la tarjeta.
- ✓ En esta Fase III se procede al desarrollo de hardware con VHDL basado en la ATLYS que será el encargado del control de un servomotor. Diseñando un componente capaz de procesar esa lectura enviada por el sensor y controlar dicho servomotor.
- ✓ Por último, se diseñara un componente capaz de trasmitir y recibir datos por medio del protocolo de comunicación serial I2C.

- **Fase IV. Elaboración de una guía de prácticas factibles basadas en el uso de FPGA para los estudiantes de la asignatura Microprocesadores de la UJAP**.

Al completar con las fases anteriores, se procede a elaborar una guía de prácticas factibles basadas en el uso de FPGA para los estudiantes de la asignatura de Microprocesadores de la UJAP, donde esta los hará capaces de reproducir los diseños nombrados anteriormente y tendrán propuestas de ejercicios prácticos para seguir en constante preparación con el futuro que traerán las FPGA.

- ✓ Se elabora una guía con ejercicios prácticos factibles, con diseños capaces de mostrar el paralelismo y la versatilidad de la FPGA, donde se tendrán ejercicio sobre el manejo de entradas / salidas digitales, comunicación de datos y control digital.
- ✓ El uso del lenguaje VHDL con el entorno de desarrollo ISE Design Suite 14.7 de XILINX

CAPÍTULO IV
RESULTADOS

En este capítulo se presenta el resultado de cada una de las fases metodológicas a fin de cumplir con los objetivos para diseñar un sistema de experiencias prácticas para los estudiantes del laboratorio de microcontroladores de la UJAP. En donde se presentan el desarrollo de los objetivos específicos planteados a través cuatro objetivos como la descripción del sistema ATLYS de XILINX, Elaboración de una guía práctica de usuario, Diseño de sistemas digitales con FPGA para experiencias prácticas reproducibles y Elaboración de una guía de prácticas factibles con el uso del FPGA.

4.1 Descripción de las caracteriscas del sistema ATLYS de XILINX y las herramientas de diseño asociadas al mismo.

En lo que se refiere al objetivo 1, se llevó acabo la descripción de las caracteristicas del sistema ATLYS de XILINX en conjunto a las herramientas de diseño asociadas al mismo, La tarjeta de desarrollo ATLYS es una plataforma de desarrollo que cuenta con una serie de dispositivos y elementos ya instalados que le permite al diseñador realizar una gran variedad de proyectos y ponerlos a prueba fácilmente. La ATLYS cuenta con el Spartan-6 LX45 FPGA de XILINX; las principales características son: Comunicación por medio Ethernet a velocidades de Gbits, HDMI, memoria DDR2 de 128MB y de 16bits, puerto USB y salida de audio; todas estas características y más hacen de la tarjeta ATLYS una de las mejores tarjetas de desarrollo basadas en FPGA. También cuenta con un procesador embebido para diseño basados en Microblaze de XILINX.

El Spartan-6 LX45 está optimizado para lógica de alto rendimiento y ofrece:

- 6822 slices, cada uno con cuatro LUTs de 6 de entrada y ocho flip-flops.
- Un bloque de RAM de alta velocidad de 2.1Mbits.
- Cuatro controladores de reloj (8 DCMs y 4 PLLs).
- Seis lazos de seguidores de fase (PLLs).
- 58 slices DSP.
- Velocidad de reloj a más de 500 MHz.

La tarjeta ATLYS incluye el sistema más reciente de ADEPT USB2 que ofrece: la programación del dispositivo, el monitoreo de las fuentes de alimentación en tiempo real, pruebas

automáticas de la tarjeta, entradas y salidas virtuales además se han simplificado la función de transferencia de datos del usuario.

Caracteristicas:

- FPGA Spartan-6 LX45 de XILINX, con encapsulado BGA de 324 pines.
- Memoria DDR2 de 16 bits y 128 MBytes.
- Ethernet PHY 10/100/1000
- Puerto USB2 incorporado para la transferencia de datos y programación.
- Puerto USB-UART para comunicación serial con PC.
- Puerto USB HID (para ratón / teclado por medio de microcontrolador PIC24).
- Dos puertos de entrada de vídeo HDMI y dos puertos de salida HDMI.
- Codec de audio AC-97 con: línea de entrada, línea de salida, micrófono y auriculares.
- Monitor de energía para todas la fuentes de alimentación.
- Memoria FLASH SPI de 16 MByte x4 para almacenamiento de datos y de configuración.
- Oscilador CMOS de 100 MHz.
- 48 pines de entradas o salidas conectados a los conectores de expansión.
- Entradas y salidas generales como: 8 LEDs, 6 botones y 8 interruptores.
- Adaptador de voltaje de 20 W y cable USB.

La FPGA ATLYS cuenta con un modulo de configuración de encendido la cual debe configurarse antes de realizar cualquier función dentro de la misma. LA FPGA puede configurarse de tres maneras distintas: Una PC conectada mediante un USB puede configurar la placa usando el puerto JTAG cada vez que se enciende, un archivo de configuración almacenado en el SPI Flash ROM o puede transferirse un archivo de programación desde una memoria USB conectada al puerto USB HID (ver figura 10) lo cual mediante el puente de modo incorporado (JP11) se logra elegir entre las tres configuraciones distintas que dispone.

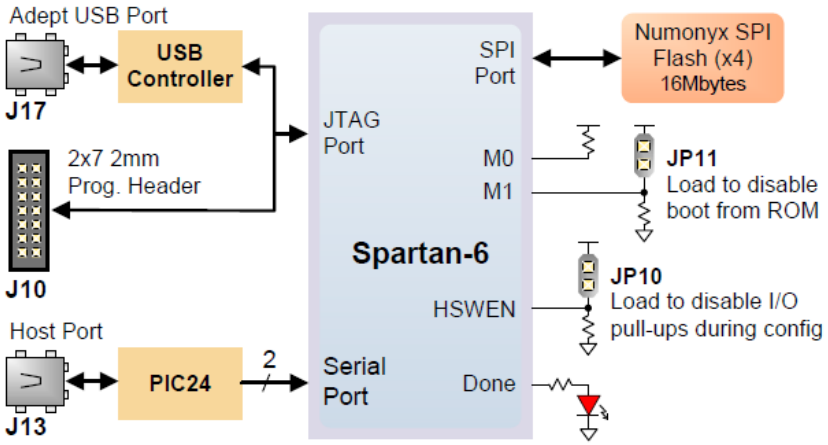


Figura 10. Diagrama esquematico para la configuración de inicio

Fuente: ATLYS de XILINX.

En la parte de alimentación la tarjeta cuenta con un jack de alimentación y un switch capaz de interrumpir y permitir el paso de la corriente, para verificar que dicha tarjeta se encuentra energizada tendra un Led indicador de alimentación (ver figura 11).

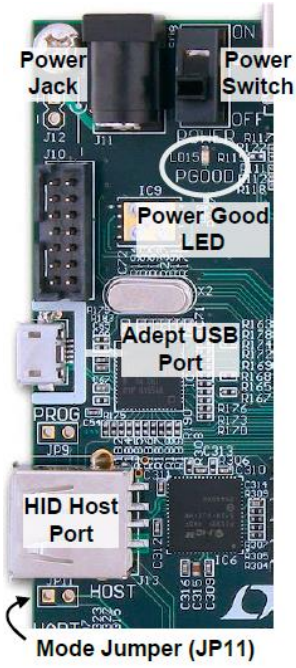


Figura 11. Modulo principal de alimentación
Fuente: ATLYS de XILINX.

Los cuatro rieles de voltaje principales en la placa ATLYS utilizan ADC de 16 bits Delta Sigma LTC2481 de tecnología lineal para medir continuamente la corriente de suministro. Con

una precisión del 1%, estos valores medidos se pueden ver en una PC usando el medidor de potencia que forma parte del software Adept.

Las fuentes de alimentación ATLYS se habilitan mediante un interruptor de nivel lógico (SW8). Un LED de buena alimentación (LD15), impulsado por el OR cableado en todas las salidas de buena alimentación en los suministros, indica que todos los suministros están funcionando dentro del 10% del valor nominal (ver figura 12).

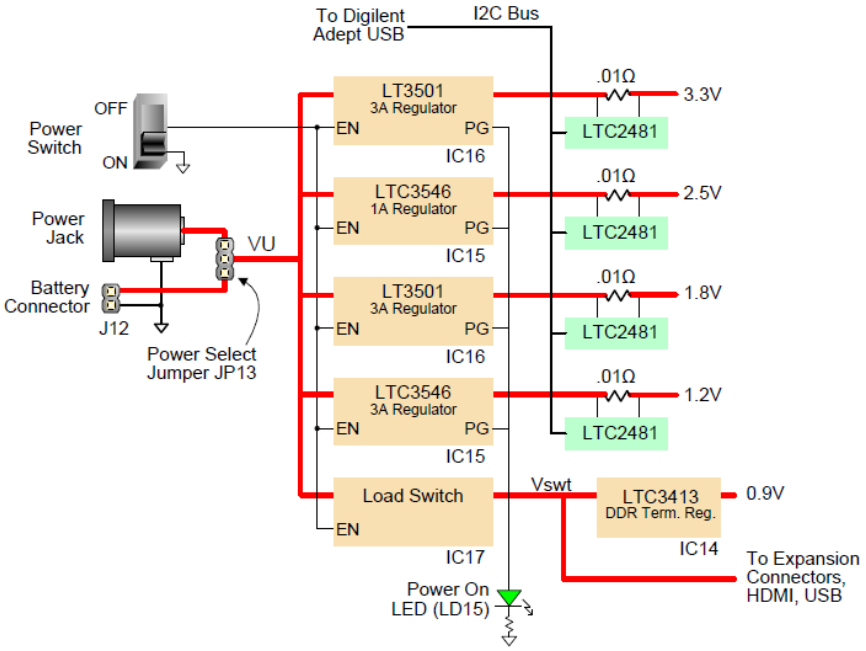


Figura 12. Diagrama esquematico interno de alimentación
Fuente: ATLYS de XILINX.

La ATLYS cuenta con un chip de memoria DDR2 de 1 Gbit que es controlado desde el bloque del controlador de memoria en el Spartan6. El cual proporcionan un bus de datos de 16 bits y 64 millones de ubicaciones y han sido probados para el funcionamiento de DDR2 a una velocidad de datos de hasta 800 MHz. Al igual que cuenta con una memoria Flash la cual es suministrada para el almacenamiento no volatil de archivos de la configuración del FPGA. El SPI Flash puede ser programado con un archivo .bit, .bin o .mcs (ver figura 13)

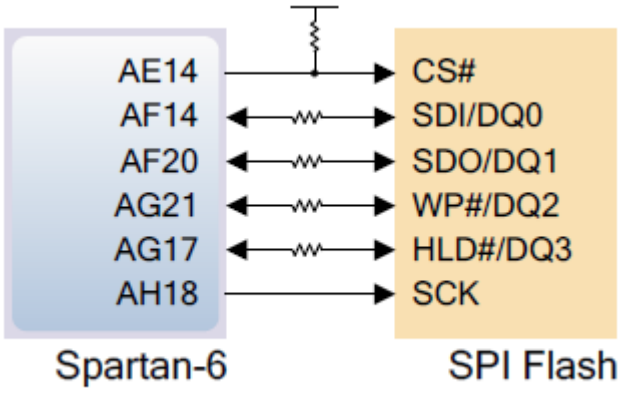


Figura 13. Diagrama esquematico interno de SPI Flash
Fuente: ATLYS de XILINX.

La tarjeta ATLYS incluye un PHY de Ethernet que va emparejado con un conector Halo HFJ11 en donde los modos de interfaz MII/GMII son compatibles a 10/100/1000Mb/s (ver figura 14).

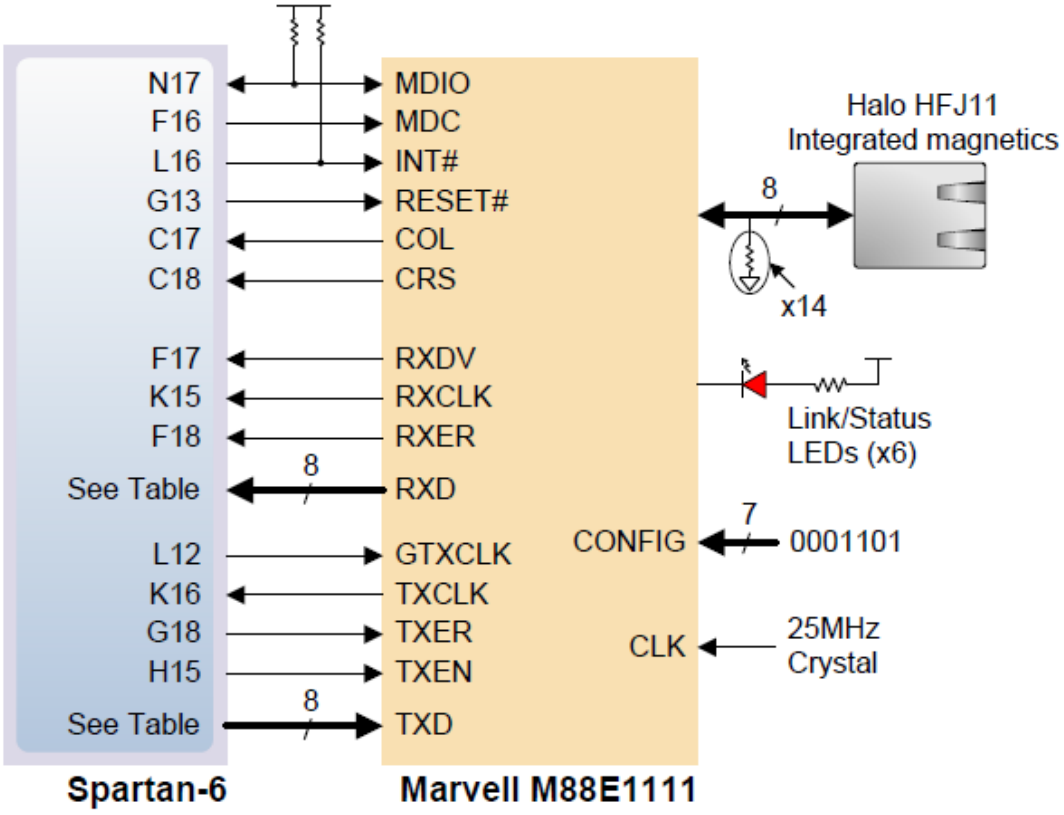


Figura 14. Diagrama esquematico interno Puerto ETHERNET
Fuente: ATLYS de XILINX.

La placa ATLYS contiene cuatro puertos HDMI, incluidos dos puertos de entrada/salida HDMI con búfer, un puerto de salida HDMI con búfer y un puerto sin búfer que puede ser de entrada o salida (generalmente se usa como puerto de salida). Los tres puertos con búfer usan HDMI tipo A conectores, y el puerto sin búfer utiliza un conector tipo D cargado en la parte inferior de la PCB inmediatamente debajo del puerto Pmod (el conector tipo D es mucho más pequeño que el tipo A).

Las señales de datos en el puerto sin búfer se comparten con un puerto Pmod. Esto limita un poco el ancho de banda de la señal: es posible que el conector compartido no pueda producir o recibir las señales de video de mayor frecuencia, especialmente con cables HDMI más largos.

Dado que los sistemas HDMI y DVI usan el mismo estándar de señalización TMDS, se puede usar un adaptador simple para conectar un conector DVI desde cualquiera de los puertos de salida HDMI. El conector HDMI no incluye señales VGA, por lo que no se pueden controlar pantallas analógicas (ver figura 15).

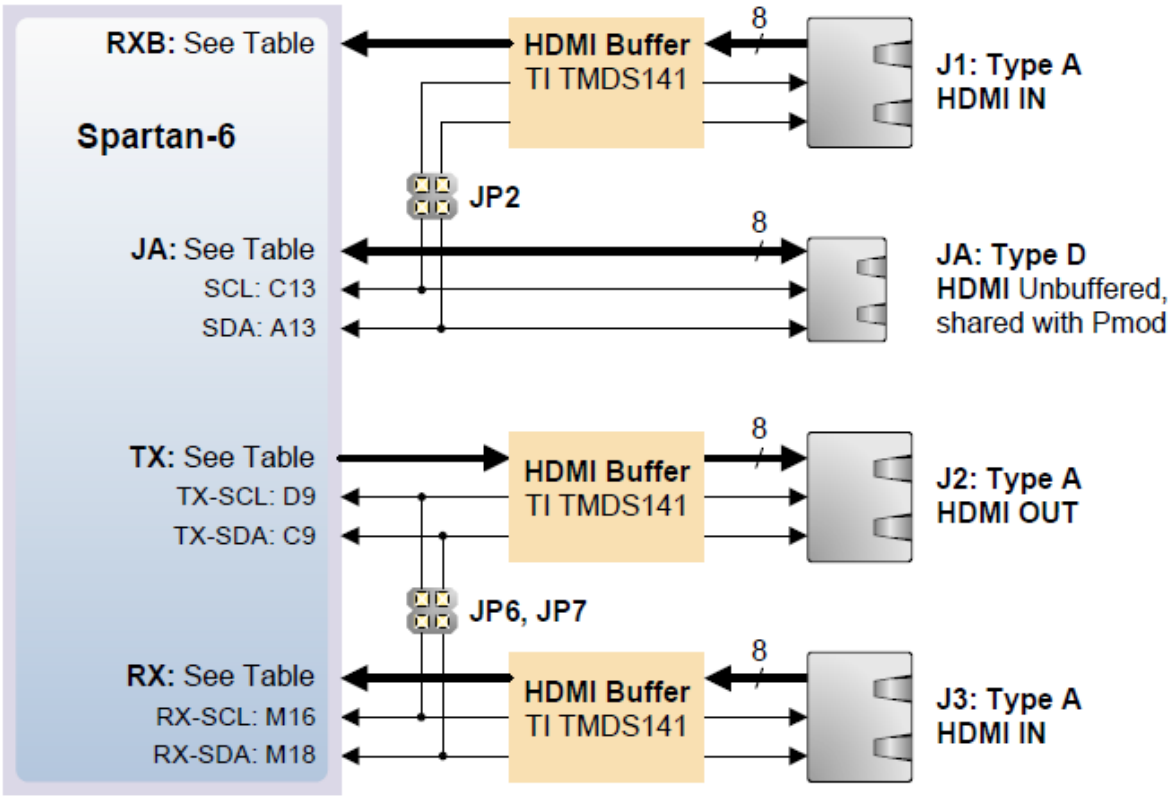


Figura 15. Diagrama esquematico interno para los puertos HDMI.
Fuente: ATLYS de XILINX.

La tabla de ATLYS incluye un National Audio Semiconductor LM4450 AC*97 Codek (IC3) con cuatro conectores de audio de 1/8" para salida de línea (J5), salida de auriculares (J7), entrada de línea (J4) y entrada de micrófono (J6). Datos de audio de hasta 18 bits y 48 KHz El muestreo es compatible, y las frecuencias de muestreo de entrada (grabación) y salida de audio (reproducción) pueden ser diferentes. El conector del micrófono es mono, todos los demás conectores son estéreos (ver figura 16).

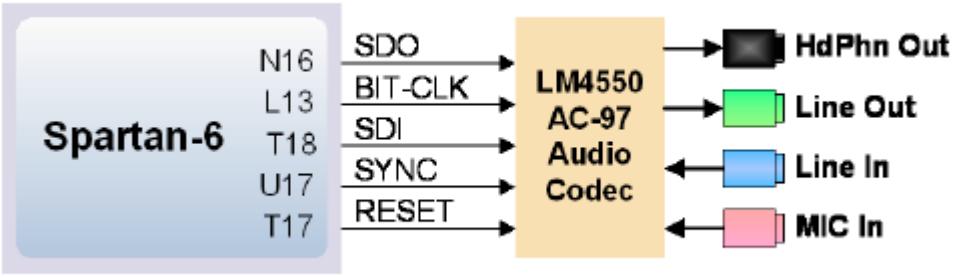


Figura 16. Diagrama esquematico interno para los puertos de audio.
Fuente: ATLYS de XILINX.

Tambien la placa de desarrollo ATLYS incluye un puente EX2R USB UART para permitir que las aplicaciones de la PC se comuniquen con la placa mediante un puerto COM. (es decir, puerto serie) en la PC se transfiera sin problemas a la placa ATLYS usando el puerto USB en J17 marcado como UART. La parte EX2R entrega los datos al Spartan-6 utilizando un puerto serie de dos hilos con control de flujo de software (XON/XOFF) (ver figura 17).

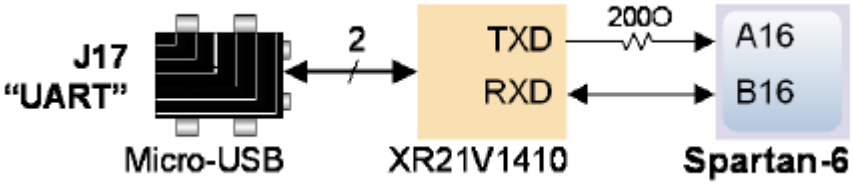


Figura 17. Diagrama esquematico interno del conector UART.
Fuente: ATLYS de XILINX.

Un microcontrolador Microchip PIC24FJ192 proporciona a la placa ATLYS capacidad de host USB HID. El firmware en el microcontrolador MCU puede controlar un mouse o un teclado conectado al conector USB tipo A en J13 con la etiqueta "Host". El PIC24 maneja cuatro señales en el FPGA: dos se utilizan como puerto de teclado siguiendo el protocolo PS/2 de teclado, y dos se utilizan como puerto de ratón siguiendo el protocolo PS/2 de ratón (ver figura 18).

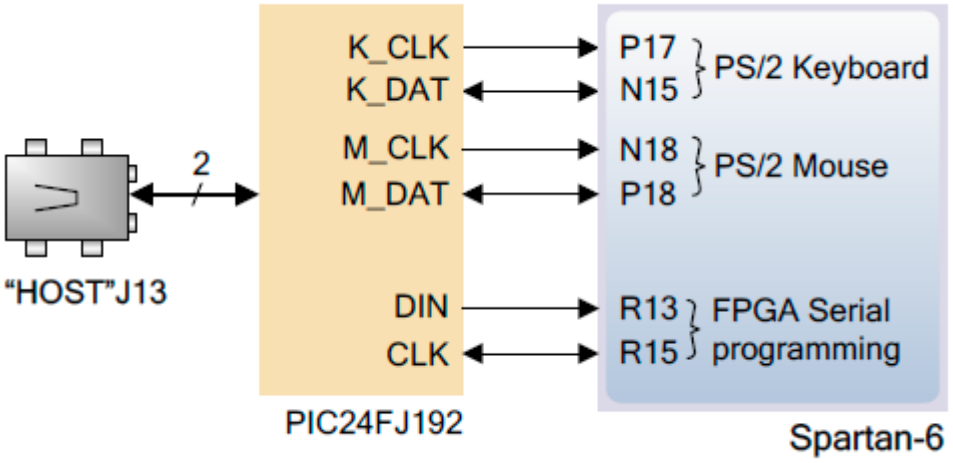


Figura 18. Diagrama esquematico interno del conector HOSTJ13.
Fuente: ATLYS de XILINX.

La placa ATLYS incluye seis botones, ocho interruptores deslizantes y ocho LED para entrada y salida digital básica. Un botón tiene un émbolo rojo y está etiquetado como "restablecer" en la serigrafía de la placa de circuito impreso; este botón no es diferente de los otros cinco, pero se puede usar como una entrada de restablecimiento para los sistemas de procesador. Los botones y los interruptores deslizantes están conectados a la FPGA a través de resistencias en serie para evitar daños por cortocircuitos involuntarios. Los ánodos LED de alta eficiencia están conectados al FPGA a través de resistencias de 390 ohmios, y se iluminarán intensamente con aproximadamente 1 mA de corriente cuando se aplique un alto voltaje lógico a su respectivo pin de E/S (ver figura 19).

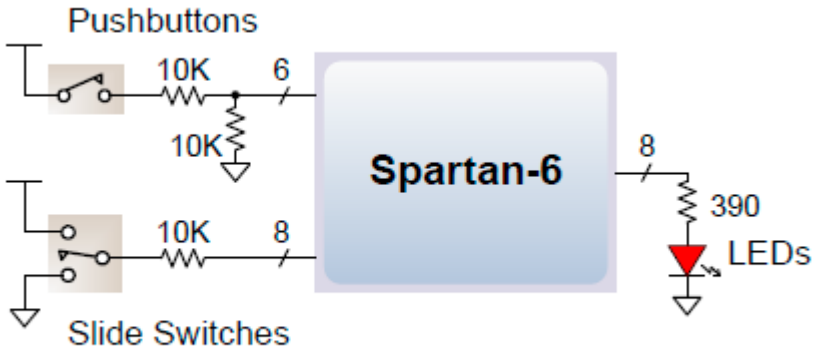


Figura 19. Diagrama esquematico interno de E/S
Fuente: ATLYS de XILINX.

La placa ATLYS tiene un conector VHDC de 68 pines para E/S paralelas/de alta velocidad y un puerto Pmod de 8 pines para E/S de menor velocidad y número de pines (ver figura 20).

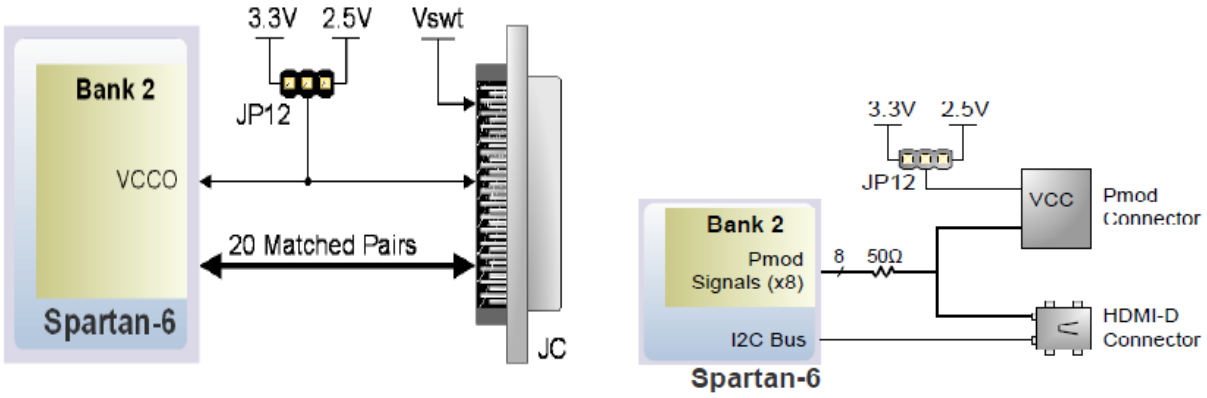


Figura 20. Diagrama esquematico de los conectores de expasión
Fuente: ATLYS de XILINX.

4.1.1 Adept System.

Adept tiene una interfaz de programación simplificada que ofrece una serie de herramientas que para facilitar los test y las conexiones que presenta la placa de desarrollo ATLYS de XILINX, de las cuales se cuenta como seran descritas a continuación .

4.1.2 Adept iMPACT USB PORT

El puerto Adept es compatible con el software de programación iMPACT de XILINX si el complemento Digilent para XILINX Tools está instalado en la PC. El complemento es capaz de

traducir automáticamente los comandos JTAG generados por iMPACT a formatos compatibles con el puerto USB de Digilent, lo que brinda una experiencia de programación perfecta sin salir del entorno de la herramienta XILINX. Una vez que se instala el complemento, se puede seleccionar la opción de programación de "terceros" desde el menú de herramientas de iMPACT, e iMPACT funcionará como si se estuviera usando un cable de programación XILINX. Todas las herramientas de XILINX (iMPACT, ChipScope, EDK, etc.) pueden funcionar con el complemento y se pueden usar junto con las herramientas de Adept (como el monitor de fuente de alimentación).

El sistema USB2 de alta velocidad de Adept se puede usar para programar la FPGA y la ROM, ejecutar pruebas de placa automatizadas, monitorear las cuatro fuentes de alimentación de la placa principal, agregar dispositivos de E/S virtuales basados en PC (como botones, interruptores y LED) a los diseños de FPGA e intercambiar datos basados en registros y archivos con la FPGA. Adept reconoce automáticamente la placa ATLYS y presenta una interfaz gráfica con pestañas para cada una de estas aplicaciones. Adept también incluye API/DLL para que los diseñadores puedan describir aplicaciones para intercambiar datos con la placa ATLYS a una velocidad de hasta 38 Mbytes/seg.

4.1.3 Interfaz de programación.

El sistema Adept cuenta con una interfaz de programación que se encarga de asegurar de que cualquier archivo seleccionado contenga el codigo de identificación de FPGA correcto, con esto se evita que se envien archivos .bit incorrectos al FPGA.

Además de la barra de navegación y los botones de exploración y programación, la interfaz de configuración proporciona un botón Inicializar cadena, una ventana de consola y una barra de estado. El botón Inicializar cadena es útil si se han interrumpido las comunicaciones USB con la placa. La ventana de la consola muestra el estado actual y la barra de estado muestra el progreso en tiempo real al descargar un archivo de configuración (ver figura 20).

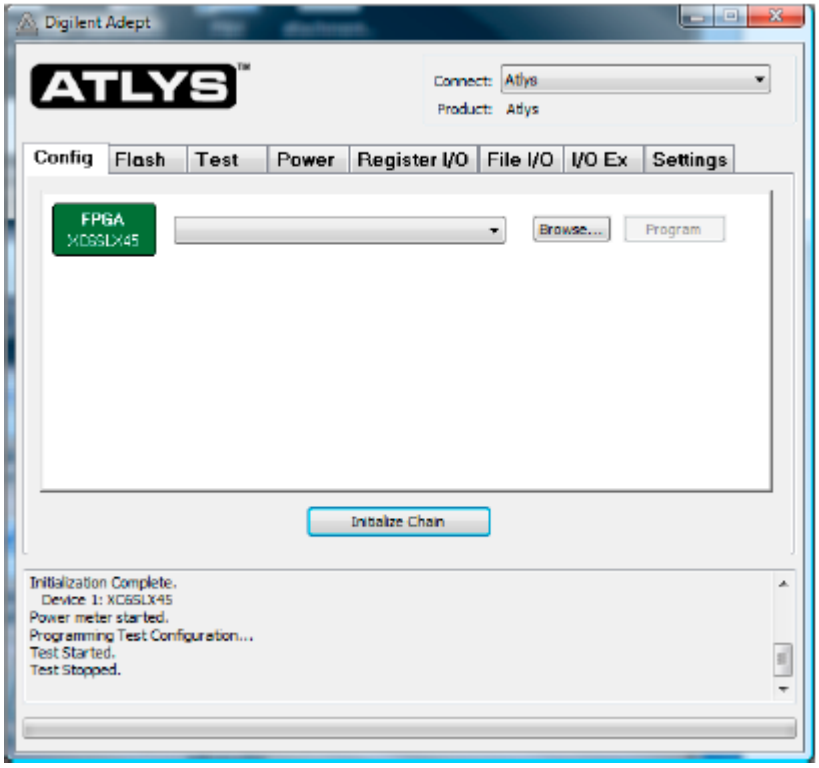


Figura 20. Interfaz de configuración Adept

Fuente: Digilent Adept

4.1.4 Interfaz Flash

La aplicación de programación Flash permite que los archivos de configuración .bin, .bit y .mcs se transfieran a la Flash ROM SPI integrada para la programación de FPGA, y permite que los archivos de datos del usuario se transfieran a/desde Flash en las direcciones especificadas por el usuario (ver figura 21).

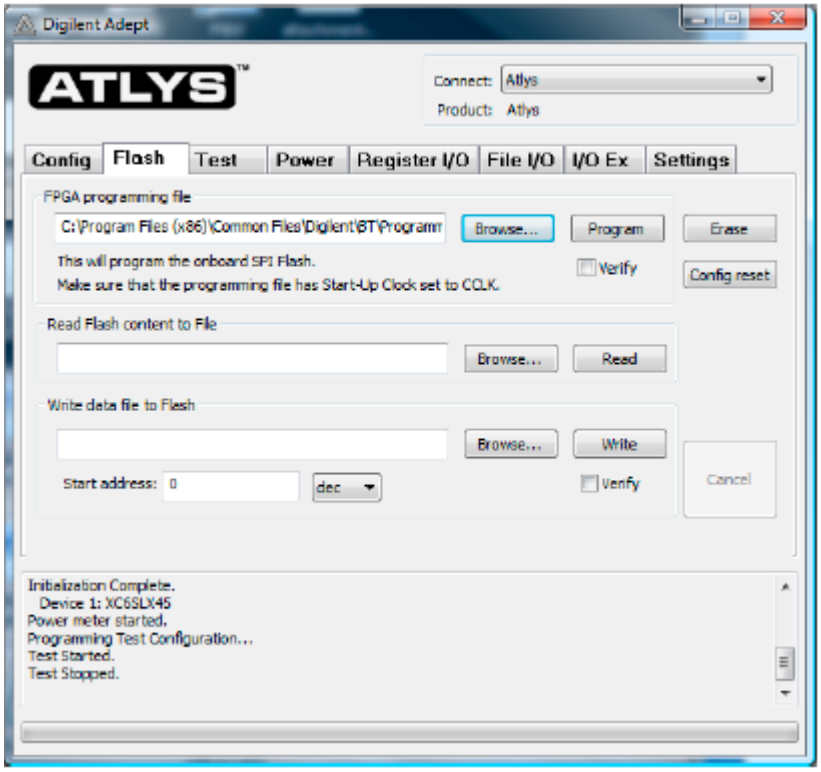


Figura 21. Interfaz de Flash.

Fuente: Digilent Adept.

4.1.5 Interfaz de prueba.

La interfaz de prueba proporciona una manera fácil de verificar muchos de los circuitos e interfaces de hardware de la placa. Estos se dividen en dos categorías principales: memoria integrada (DDR2 y Flash) y periféricos. En ambos casos, la FPGA se configura con circuitos de prueba y comunicación con PC, sobrescribiendo cualquier configuración de FPGA que pueda haber estado presente.

La función Test Shorts verifica todas las E/S discretas en busca de cortocircuitos a Vdd, GND y pines de E/S vecinos. Los gráficos de interruptores y botones muestran los estados actuales de esos dispositivos en la placa ATLYS. Cada pulsación de botón generará un tono de los conectores de audio LINE OUT o HP OUT (ver figura 22).

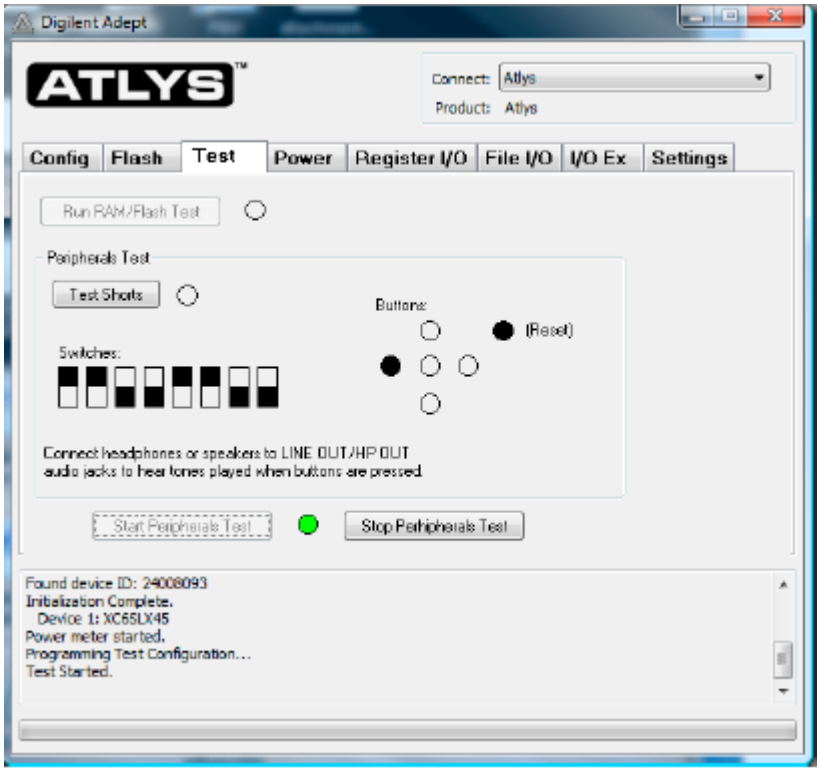


Figura 22. Interfaz de Prueba.

Fuente: Digilent Adept.

4.1.6 Power

La aplicación de energía proporciona lecturas de energía y corriente en tiempo real de alta precisión (mejor que 1 %) de cuatro monitores de fuente de alimentación integrados. Los monitores están basados en Convertidores analógico a digital LTC2481C sigma delta de Technology que devuelven muestras de 16 bits para cada canal.

Los datos de corriente y potencia en tiempo real se muestran en forma tabular y se actualizan continuamente cuando el medidor de potencia está activo (o iniciado) (ver figura 23).

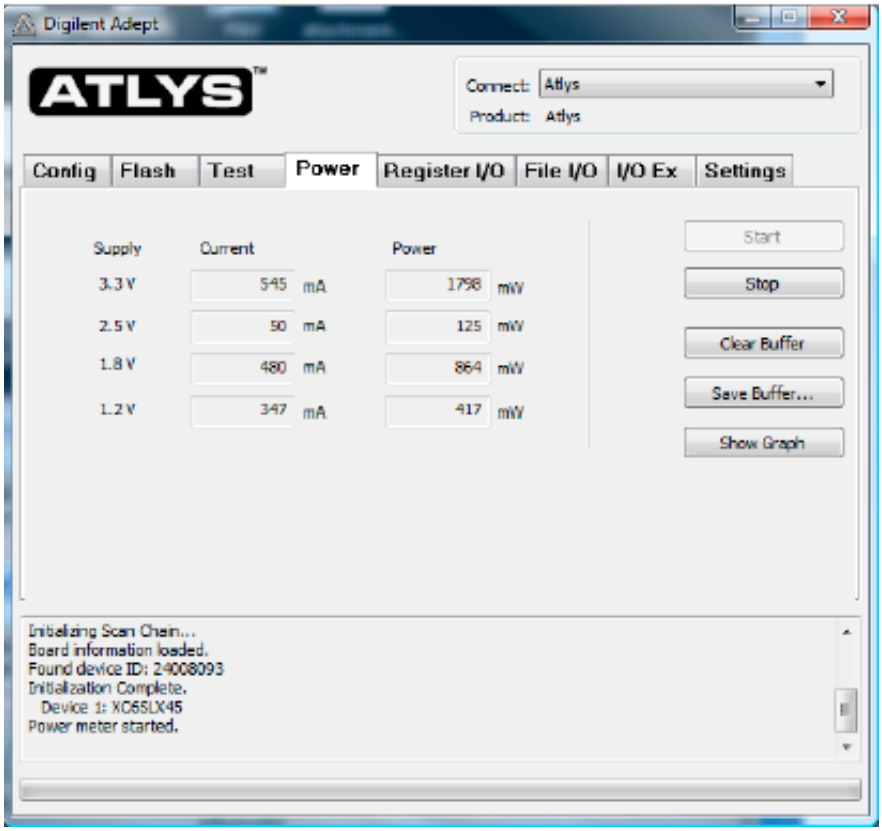


Figura 23. Interfaz Power.
Fuente: Digilent Adept.

4.1.7 Registro de E/S

Register I/O proporciona una manera fácil de mover pequeñas cantidades de datos dentro y fuera de registros específicos en un diseño determinado. Esta función simplifica enormemente la transferencia de parámetros de control a un diseño o la lectura de información de estado de baja frecuencia de un diseño (ver figura 24).

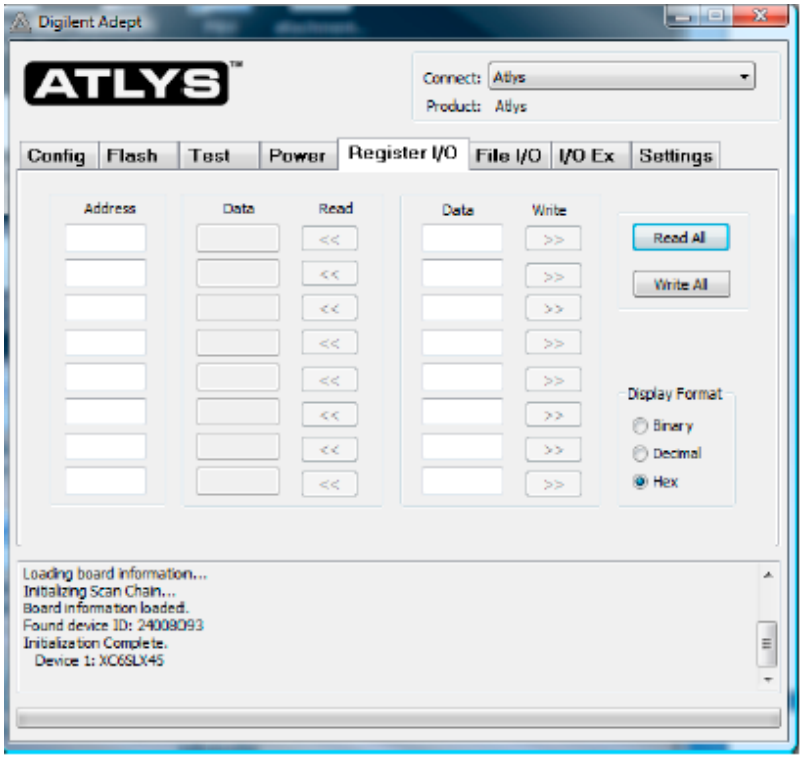


Figura 24. Interfaz E/S.

Fuente: Digilent Adept.

4.1.8 E/S de archivos

La pestaña File I/O puede transferir archivos entre la PC y el ATLYS FPGA. Se puede transmitir una cantidad de bytes (especificados por el valor de longitud) a una dirección de registro específica desde un archivo o desde una dirección de registro específica a un archivo. Durante la carga y descarga, la ubicación de inicio del archivo se puede especificar en términos de bytes (ver figura 15).

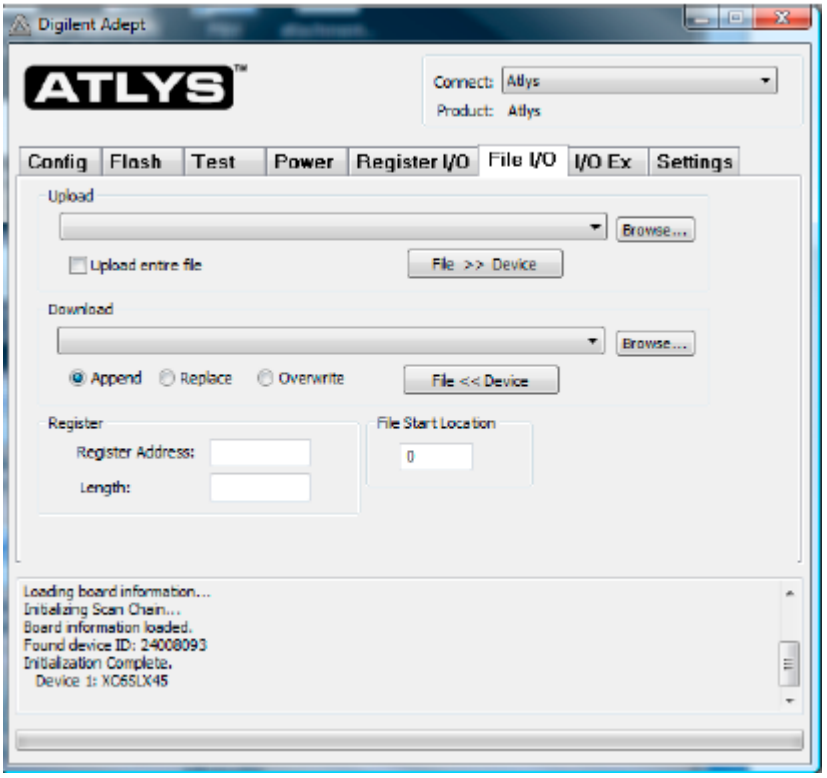


Figura 25. Interfaz E/S de archivos.

Fuente: Digilent Adept.

4.1.9 Expasión de las E/S

La pestaña I/O Expand funciona con un bloque de IP en el FPGA para proporcionar E/S simple adicional más allá de los dispositivos físicos que se encuentran en la placa ATLYS. Los dispositivos de E/S virtuales incluyen una barra de luz de 24 LED, 16 interruptores deslizantes, 16 botones pulsadores, 8 LED discretos, un registro de 32 bits que se puede enviar a la FPGA y un registro de 32 bits que se puede leer desde el FPGA (ver figura 26).

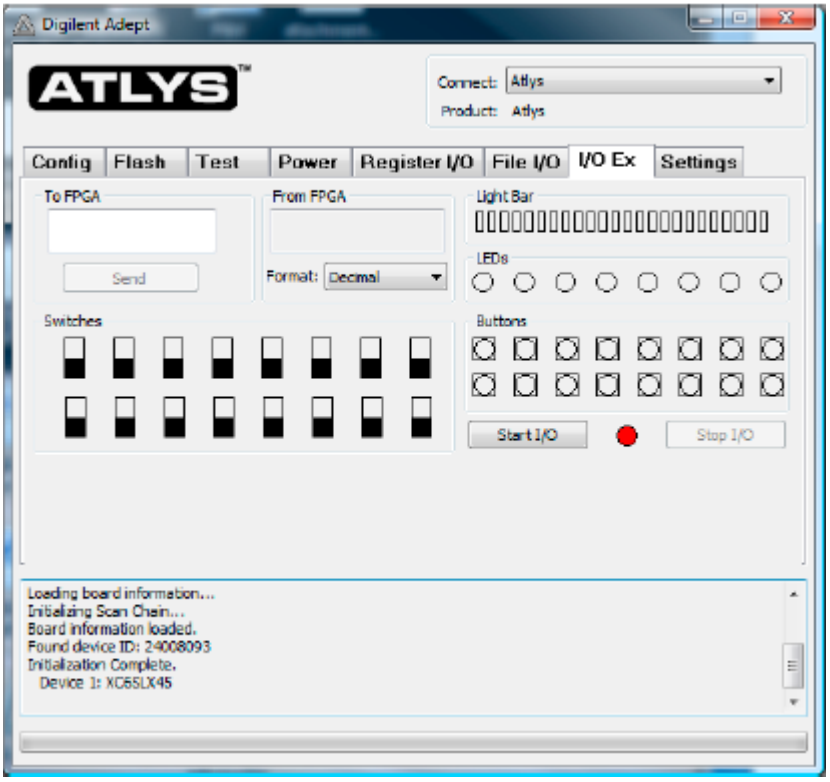


Figura 26. Interfaz E/S de expasión.

Fuente: Digilent Adept.

4.2 Elaboración de una guía de usuario sobre las herramientas de software de XILINX para la implementación de circuitos basados en FPGA.

4.2.1 Introducción

Debido a los requerimientos de funcionamiento, la complejidad que están alcanzando los diseños digitales aumenta día a día. Estos diseños implementan multitud de funciones, las cuales deben realizarse de forma rápida y precisa, por lo que resulta necesario el empleo de circuitos integrados digitales programables con mayor complejidad que los ampliamente difundidos PLD (Programmable Logic Device). Los dispositivos PLD ofrecen una respuesta muy rápida, pero están fuertemente restringidos por su capacidad, y principalmente por su estructura, ya que está completamente fijada y ha de ser nuestro diseño el que se adapte al PLD. Otra posibilidad de implementación la podemos encontrar en los circuitos semi-custom. Estos circuitos permiten la implementación de cualquier sistema digital por complejo que sea, pero tienen algunas desventajas: En primer lugar se tiene un coste elevado, únicamente rentabilizado si el volumen de producción es alto, en segundo lugar, el tiempo de desarrollo es largo (un diseño puede tardar varios meses en comercializarse), esto hace que no resulte interesante debido a la pérdida de competitividad en un mercado que innova constantemente.

Con los dispositivos FPGA (Field Programmable Gate Array) se ha alcanzado un punto intermedio entre los dispositivos PLD y los circuitos semi-custom. Un FPGA puede alcanzar una alta densidad de integración en un solo circuito integrado (hasta 1.000.000 de puertas lógicas equivalentes aproximadamente, y cada año aumenta su complejidad), y con velocidades de tratamiento de la información de entrada muy altas; cada día se continúa mejorando la eficiencia de estos dispositivos tanto en velocidad como en complejidad. Aunque su estructura está completamente fijada a nivel de silicio, la flexibilidad en su programación es grande ya que un FPGA está formado por células independientes que se pueden programar para realizar funciones sencillas, pero debido a los amplios recursos de interconexión de que disponen, estas células se pueden conectar entre ellas para generar unas funciones lógicas de salida complejas. La programación de este tipo de dispositivos se realiza en cuestión de minutos, con lo que se hacen altamente recomendables para prototipos o bajas producciones. Los problemas de un FPGA frente a la tecnología semi-custom son principalmente la menor velocidad y optimización del chip, pues a menudo, en un FPGA quedan recursos sin utilizar debido a la falta de canales de conexión; además, los canales introducen retardos en la señal, y a veces es necesario un canal de gran longitud (mayor retardo) para interconectar las células requeridas. Todo esto hace que el

dispositivo pierda eficiencia, pero estos factores se ven subsanados por la rapidez con la que pueden ser reprogramados, adaptándose rápidamente a las necesidades del mercado. Actualmente, los dispositivos FPGA son empleados en todo tipo de aplicaciones tanto científicas como de consumo; por ejemplo, actualmente se emplean en reproductores de CD, tarjetas de expansión para PC's, dispositivos para telecomunicaciones, tareas de control de dispositivos.

Actualmente, un nuevo sistema de diseño tiende a implantarse. Una vez hechas las especificaciones, el sistema es dividido en grandes bloques como pueden ser memorias, microprocesador, PLD y FPGA, se realiza una descripción de alto nivel a través de un esquema o lenguaje de descripción lógica para posteriormente simular el diseño, cuando la simulación es correcta se realiza el diseño de la placa que contendrá los diferentes elementos, mientras la placa está siendo fabricada, se depura el diseño, pero en la mayoría de los casos esto no afecta a la placa sino a los circuitos programables incluidos en ella.

El entorno de desarrollo de XILINX ISE consiste en una herramienta que permite realizar un diseño completo basado en lógica programable (tanto CPLD como FPGA), es decir, incluye todas las etapas necesarias como son:

- La entrada de diseño, bien a través de captura esquemática, lenguajes de descripción hardware como ABEL, VHDL o Verilog, o representación gráfica de diagramas de estado.
- Luego se obtiene del código HDL un esquemático RTL basado en compuertas lógicas del comportamiento.
- Se parte a verificar el comportamiento del hardware mediante los bancos de pruebas o simulaciones que proporciona el entorno de desarrollo ISE Design 14.7.
- A continuación, se verifica la síntesis del código HDL, al no presentar ningún error en la descripción del mismo, se procede con la etapa de implementación.
- En la implementación se realiza la traducción de la descripción y verificar la funcionalidad que tiene dicho hardware, en el mapeo de los pines asignados y verificar que no se presenten malas prácticas de diseño en la descripción HDL, junto al análisis de tiempos estáticos junto a Place & Route.
- Como etapa final se encuentra el Bitstream que se encarga que convertir todo el código a un archivo .bit, el cual es necesario para configurar el hardware interno en la FPGA y cargar el diseño creado.

A continuación, se podrá observar el flujo de diseño para FPGA en el ISE Design Suite 14.7 (ver figura 27).

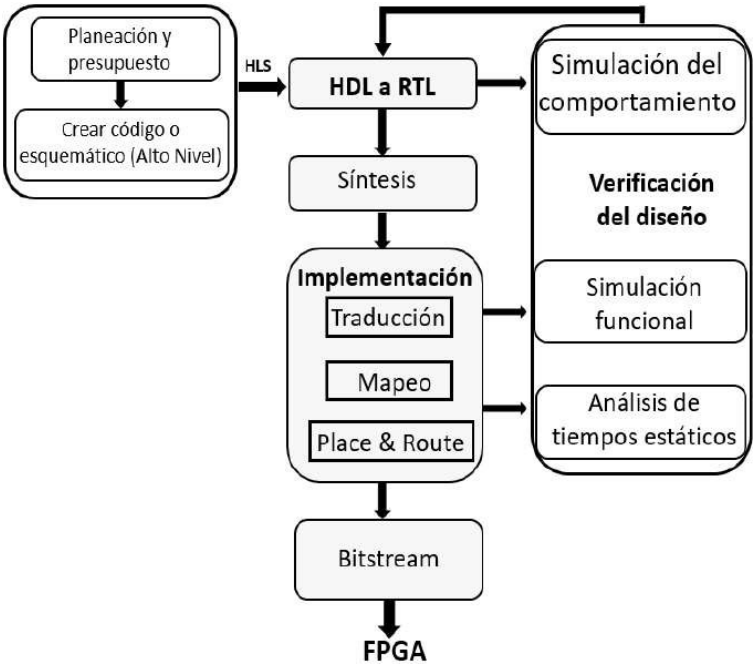


Figura 27. Flujo de diseño para el ISE Design Suite 14.7

Fuente: XILINX ISE Design Suite 14.7.

4.2.2 Descripción del entorno de desarrollo ISE Design Suite 14.7

Esta sección se va encargar de describir el entorno de desarrollo ISE del fabricante XILINX. Este entorno, tiene entre otras características, un simulador para el lenguaje VHDL que será usado en varias actividades para la asignatura del laboratorio de Microcontroladores de la UJAP.

Creación de un proyecto en XILINX ISE Design Suite 14.7

El primer paso va a ser arrancar el entorno de desarrollo de XILINX, luego de iniciado el programa nos dirigimos a la opción en la esquina superior izquierda en el menú **File** de la cual se desplegará una pestaña y escogemos la opción New Project obteniéndose la ventada mostrada en la figura 28 donde debe escribirse el nombre del diseño a realizar para el proyecto, por ejemplo ActividadLMU1. La herramienta creara una carpeta con ese nombre y guardara en su interior todos los archivos que vayamos generando de la descripción del hardware.

Tras a asignación del nombre oprimimos el botón Next y se mostrara la siguiente ventana mostrada en la figura 29, donde deben establecerse todas las opciones indicadas a continuación.

- **Family**: Spartan 6.

- **Device**: XC6SLX45.
- **Package**: CSG324.
- **Speed**: -2.
- **Preferred Language**: VHDL.

El resto de opciones deberán estar por defecto a los mismos valores que se observan en la figura 29. Tras establecer las opciones correctas, utilizamos el botón Next aparece una ventana de información y un botón Finish que es pulsado para la creación del proyecto. La figura 30 muestra el proyecto recién creado, solo se muestra el nombre del proyecto y el tipo de FPGA "XC6SLX45".

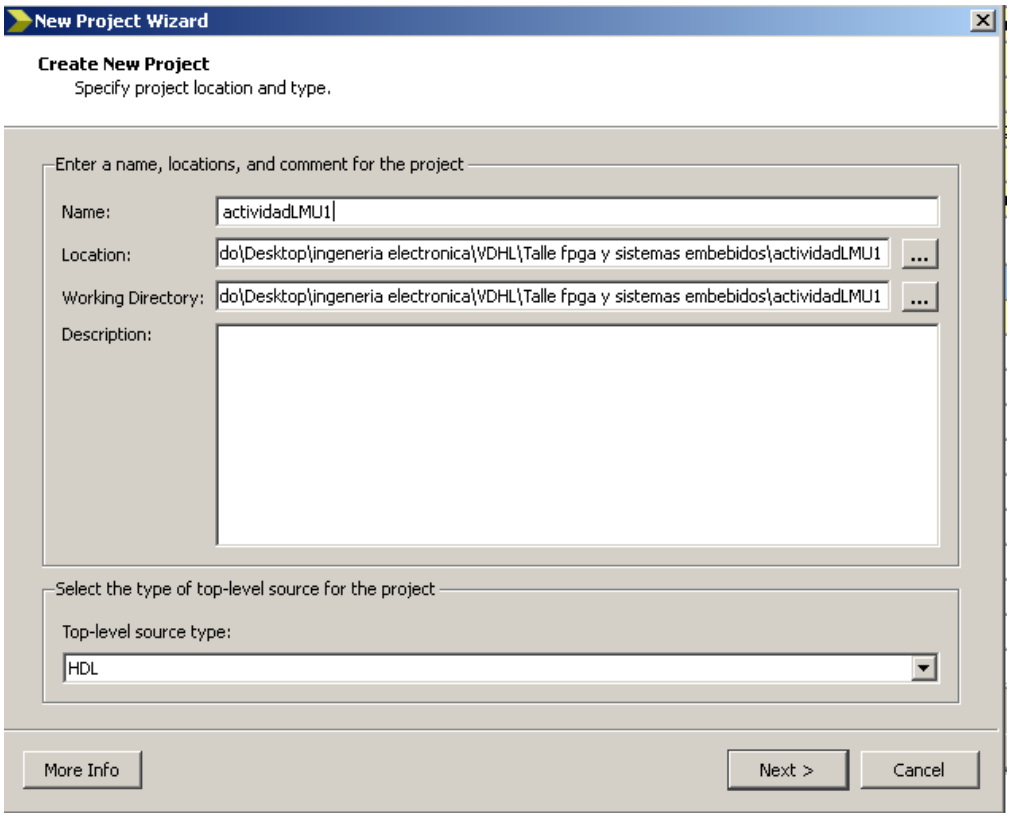


Figura 28. New Project

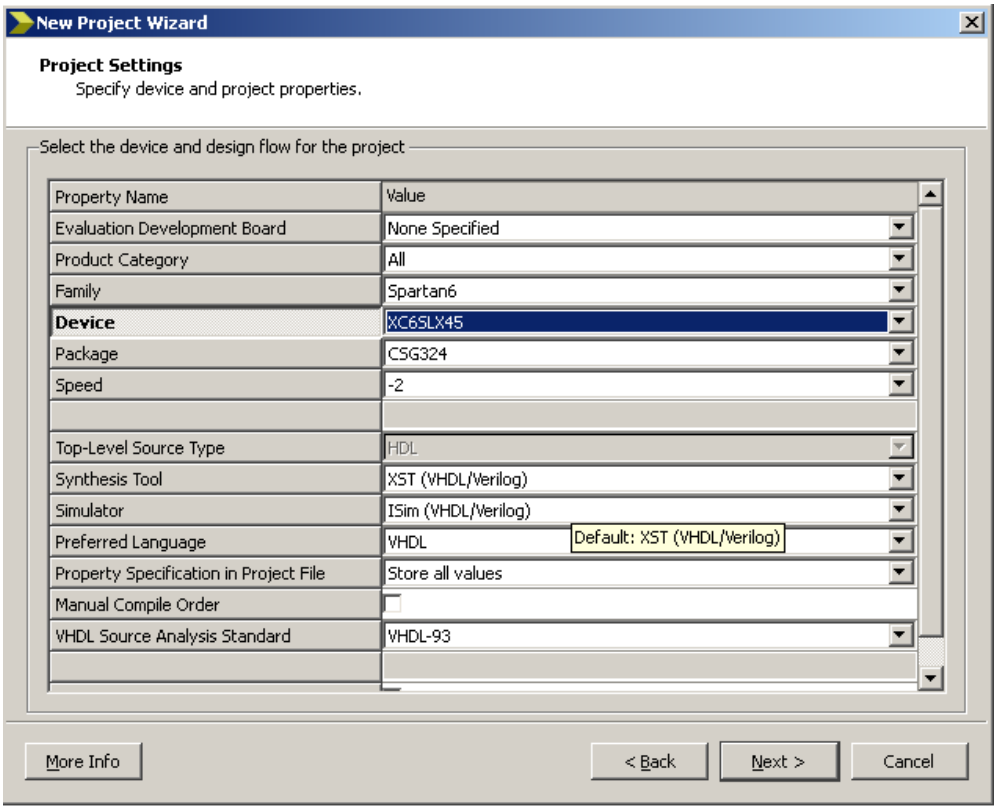


Figura 29. Especificaciones de la FPGA

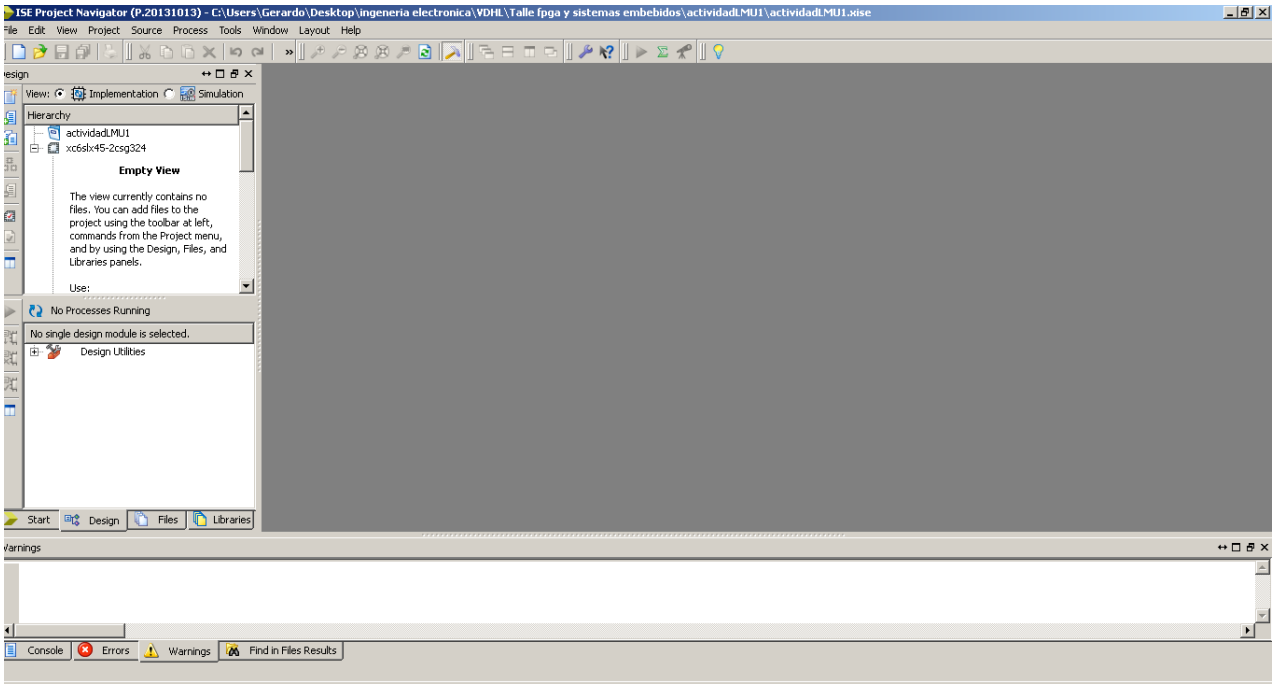


Figura 30. Proyecto creado

Nótese que encima del nombre del proyecto aparecen dos iconos **Implementation** y **Simulation**, (ver figura 30) cada uno con distintas vistas del mismo proyecto. Debemos procurar estar siempre en modo de implementación para evitar problemas con el entorno ISE. También se recomienda utilizar la entrada de menú Layout → Restore Default Layout en caso de no ver correctamente las ventanas o los controles de ISE Project Navigator (o del entorno de Simulación Isim que se usará más adelante).

Añadir ficheros al proyecto.

El primer paso tras la creación del proyecto es añadir los ficheros VHDL (diseños y testbenchs) al proyecto. Para añadir ficheros al proyecto se puede utilizar la opción de menú **Project** o, pulsar el botón derecho del ratón en (**xc6slx45-2csg324**) de la vista del proyecto (ver figura 31), eligiendo entre:

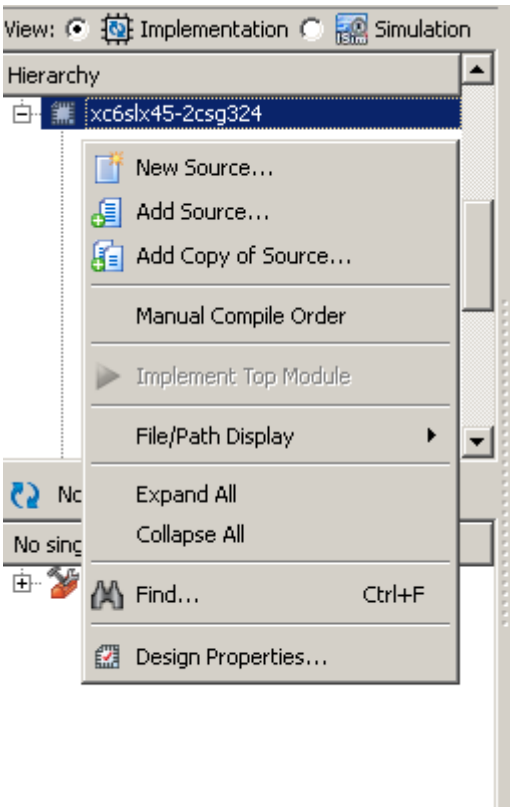


Figura 31. Añadir un fichero en el ISE Design Suite 14.7

- **New Source**: crea un nuevo fichero vacío en el que habrá que escribir el código del diseño.
- **Add Source**: no crea un nuevo fichero dentro del proyecto, utiliza el propio fichero fuente seleccionado sin crear ninguna copia. Así las modificaciones afectarán al fichero fuente en su ubicación original, es decir, el fichero residirá en la misma carpeta dónde está almacenado previamente.

- **Add Copy of Source**: crea un nuevo fichero dentro del proyecto que inicialmente será una copia del fichero fuente seleccionada, el cual se encuentra previamente creado. Así las modificaciones serán propias a este proyecto y el fichero fuente original no se modificará. La copia residirá dentro de la carpeta del proyecto.

Si estamos partiendo de un proyecto en blanco, seleccionaremos **New Source**, y elegiremos el tipo de fichero que sea necesario crear para la actividad. Normalmente elegiremos VHDL Module (módulo VHDL donde se especifica la descripción del hardware) o VHDL Test (fichero TestBench de simulación de un módulo) ver figura 33.

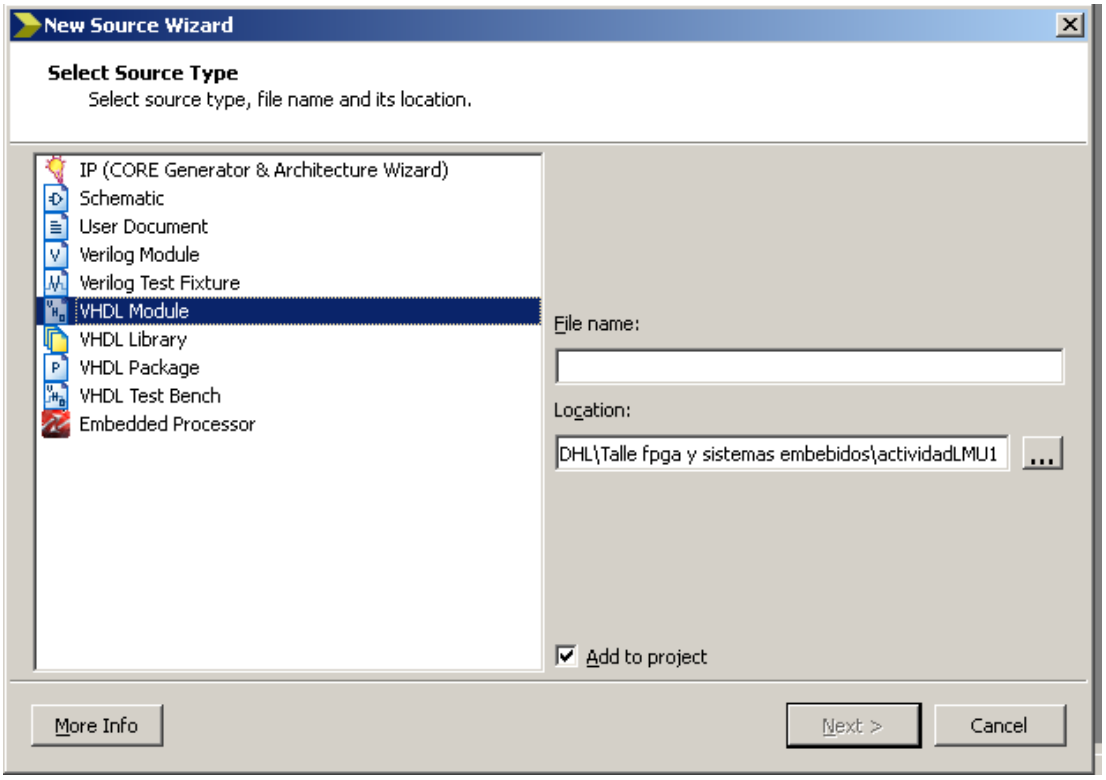


Figura 33.Selección del nuevo fichero a crear.

Si hemos pedido la creación de un módulo VDHL, la herramienta nos solicita los terminales (ports) del nuevo módulo, debiendo indicar si son entradas o salidas, y si son líneas individuales o buses. Podemos incorporar también ficheros previamente creados, en este caso, elegimos la opción Add Source seleccionando uno o más ficheros a añadir al proyecto. Hay que confirmar los ficheros que son para implementación y cuales son exclusivamente para simulación (cómo los ficheros de testbench suministrados).

Una vez añadidos los ficheros, éstos son mostrados en la vista del proyecto de forma jerárquica y, componiendo el árbol de proyecto de ficheros en función de que un fichero necesite

de otro fichero, puede ser visualizado en la jerarquía de ficheros. El fichero que incluye a los demás será el primer fichero del árbol de proyecto (ver figura 34).

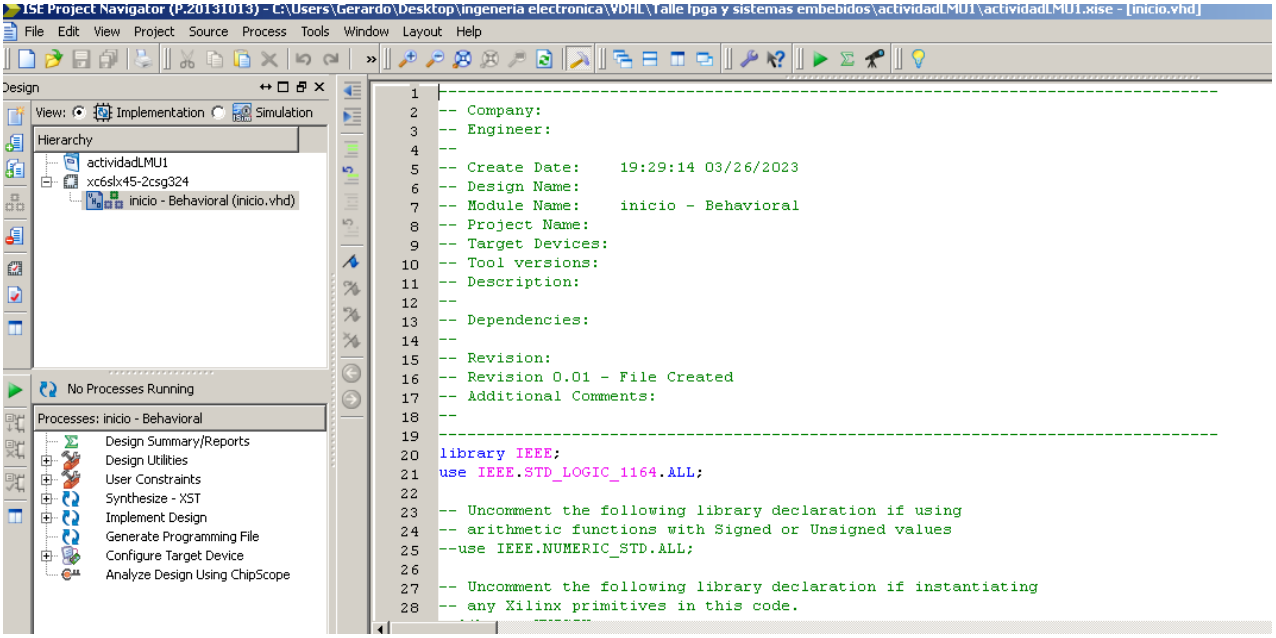


Figura 33. Fichero principal del proyecto.

Simulación y verificación de diseño.

La simulación nos permite verificar el correcto funcionamiento de una unidad/módulo diseñado, para los casos que se plantean en el fichero de testbench. Éstos podrán ser más o menos completos según el caso y si encontramos algún problema, nos permite indagar la causa del mismo antes de pasar a modificar el código. Efectuando correcciones y simulaciones se consigue solucionar todos los problemas que pueda tener el diseño.

Tras añadir un fichero de testbench, éste no se muestra en la vista de implementación, únicamente aparece en la vista simulación. Esto es debido a que dichos ficheros contienen información para realizar una simulación/verificación del diseño lógico, pero no se usa para realizar una implementación de la unidad (la última etapa del proceso de diseño). También puede comprobar como en la vista de simulación aparecen las unidades en un orden jerárquico distinto al de implementación.

En la siguiente figura 34, se muestra un ejemplo de varios módulos de un proyecto, con sus módulos de test asociados.

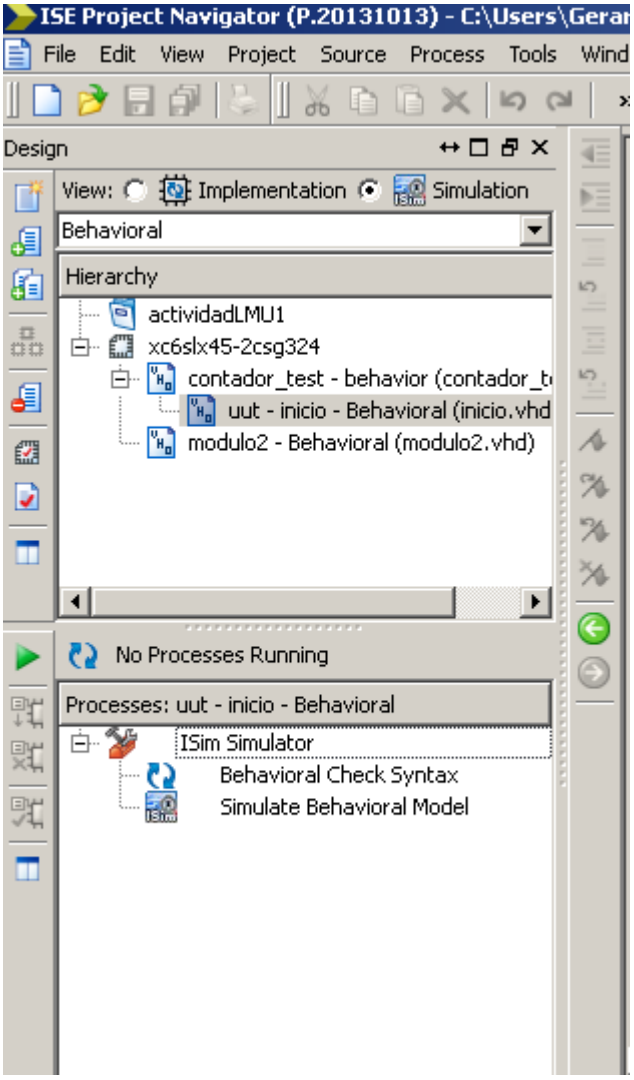


Figura 34. Vista jerárquica de un proyecto.

Así, para simular una unidad debemos resaltar el nombre de la unidad/fichero de TestBench a simular y abajo en la caja titulada **Processes** desplegar la entrada ISim Simulator para ejecutar la orden **Simulate Behavioral Model** pulsando el ratón dos veces, como muestra la figura 35.

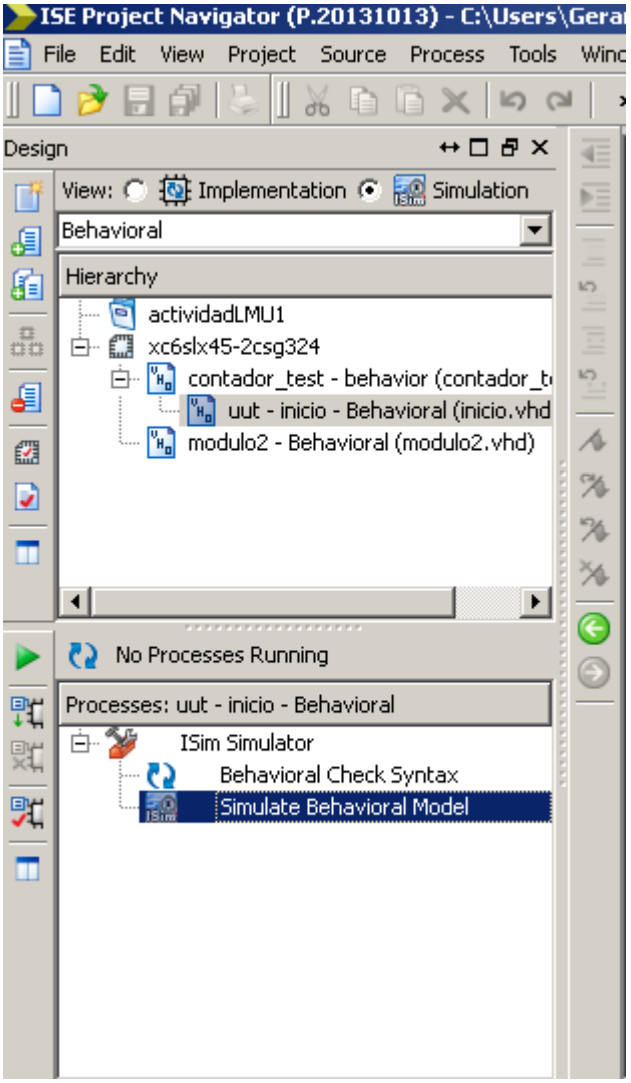


Figura 35. Selección de simulación para el diseño requerido.

Si no hay errores en el diseño, se abrirá una nueva ventana con el simulador ISim y ejecutará una simulación durante un corto periodo de tiempo (habitualmente 1s), deteniéndose la simulación en ese punto, salvo que el testbench detenga la simulación con antelación (con la orden $finish). Al no encontrar errores en el código se abrirá el simulador ISim donde, para ver las formas de onda, hay que utilizar la pestaña DEFAULT.WCFG. Es aconsejable utilizar en este momento el icono (ZOOM TO FULL VIEW) para tener una vista completa de toda la simulación.

En esta vista, mostrada en (ver figura 36), se dispone de una ventana con las formas de onda a la derecha en fondo negro y varias señales representadas con sus valores simulados, que deben ser las entradas y salidas de nuestra unidad (al menos aquellas que aparecen en el testbench). Si hacemos pulsamos el ratón sobre el gráfico de formas de ondas, se sitúa un cursor amarillo indicando datos exactos en ese instante de tiempo (nótese como cambian los valores de

las señales al situarse en distintos instantes). Para las señales de varios bits podemos cambiar la codificación pulsando el botón derecho del ratón sobre el nombre de la señal y, accediendo en el menú flotante a la opción RADIX (binario, hexadecimal, decimal, etc.).

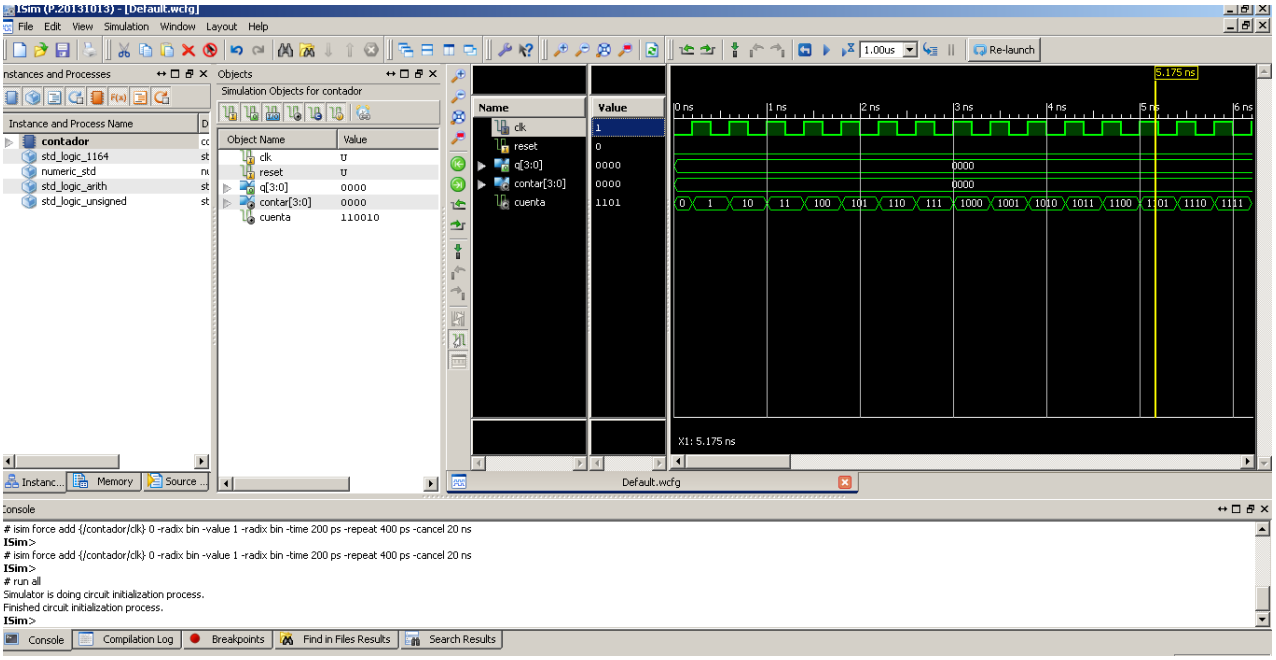


Figura 36. Simulación de un contador ascendente con Isim.

Otros controles importantes de ISim se encuentran en dos bloques situados a la izquierda con los que se puede navegar por todas las unidades que componen el diseño. Utilizando estos dos bloques se pueden buscar señales y componentes internos de cualquier módulo para poder mostrar sus formas de ondas, pero, para ello habría que reiniciar la simulación tras añadirlas a la vista de simulación.

Por último, en la barra de iconos superior hay varios iconos que permiten hacer ampliaciones y reducciones de escala en la imagen. Además de estos iconos, varios iconos verdes que hay a continuación sirven para navegar por la forma de onda y, los iconos azules, controlan la simulación utilizándose para continuar o detener el proceso.

Los iconos verdes se deben usar para buscar o centrarse en una parte concreta de las formas de onda, por ejemplo, **Previous Transition** y **Next Transition** nos permiten ir al anterior/siguiente flanco de una señal seleccionada previamente en la ventana de formas de onda. Los iconos azules controlan el flujo de ejecución de la simulación permitiendo: borrar la simulación actual volviendo al instante cero (Restart), continuar la simulación indefinidamente (Run All), continuar un tiempo de simulación determinado deteniéndose automáticamente (Run),

ejecutar la simulación línea a línea de VHDL (Step) y, por último, detener una simulación en ejecución (Break).

Implementación del sistema digital en la placa FPGA ATLYS de XILINX

Repasemos brevemente el proceso que hemos seguido hasta ahora:

- hemos especificado nuestro diseño en VHDL y hemos depurado errores.
- lo hemos simulado con un fichero testbench y hemos comprobando que funciona según nuestra especificación.

Finalmente se implementará el sistema digital realizado en un dispositivo programable tipo FPGA incluida en la placa de desarrollo ATLYS. Esta placa tiene como elemento fundamental un dispositivo programable FPGA (sobre el que vamos a volcar nuestro diseño lógico), pero también incluye dispositivos físicos que podemos conectar a la FPGA, como periféricos de entrada o salida (pulsadores, interruptores, LED, displays, entre otros).

Para ello es necesario incluir en nuestro proyecto otro fichero donde se indican las conexiones entre las entradas y salidas del módulo top y los pads del chip FPGA que van conectados a los componentes de la placa. Este fichero se llama ATLYSGeneral.ucf y ya contiene la asignación de las conexiones del sistema a los componentes de la placa.

Para lograr una correcta implementación del sistema digitales que hemos diseñado, se deben seguir los siguientes pasos:

- Cambiar el modo de simulación a implementación en el Proyecto ISE.
- Añadir el fichero de asociaciones ATLYSGeneral.ucf y modificarlo para especificar adecuadamente los terminales del circuito especificado a los componentes de la placa de la FPGA (ver figura 36)
- Generar el fichero de programación del proyecto.
- Programar la placa FPGA con el programa Adept.

La implementación se realiza seleccionando el módulo principal de nuestro diseño en el árbol de proyecto. Debe seleccionar la opción Generate Programming File (ver figura 37) y seguir los siguientes pasos:

- Pulsando el botón derecho del ratón sobre la opción Generate Programming File aparecerá un menú flotante como el mostrado en la figura 37, seleccionando la opción de menú de Process Poperties aparecerá el diálogo mostrado en (ver figura 38).

- En el diálogo hay que elegir la categoría Startup Options y cambiar el valor del primer campo FPGA Start-Up Clock de CCLK a Jtag-Clock. Tras aceptar los cambios con el botón OK se volverá a la ventana principal de ISE.
- Pulsando dos veces botón izquierdo del ratón sobre Generate Programming File se ejecuta el proceso completo y se genera el fichero de programación. Tal y como se muestra en la figura 36, si el proceso ha terminado con éxito aparecerá un indicador verde, en caso contrario una marca de verificación roja, indicando la existencia de errores. En caso de existir errores debe corregirlos y repetir el proceso.

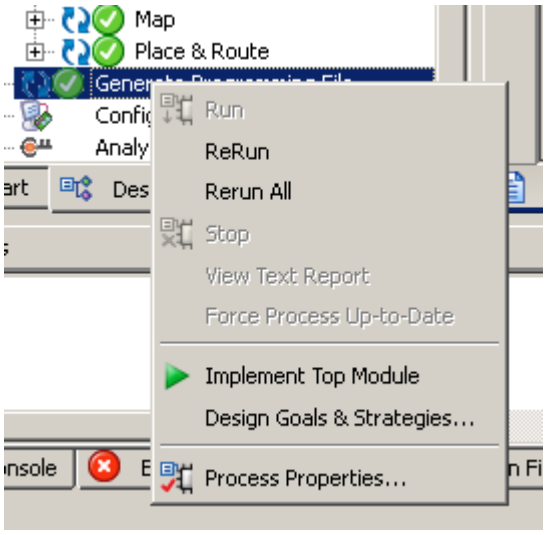


Figura 37. Opciones de generación del fichero.

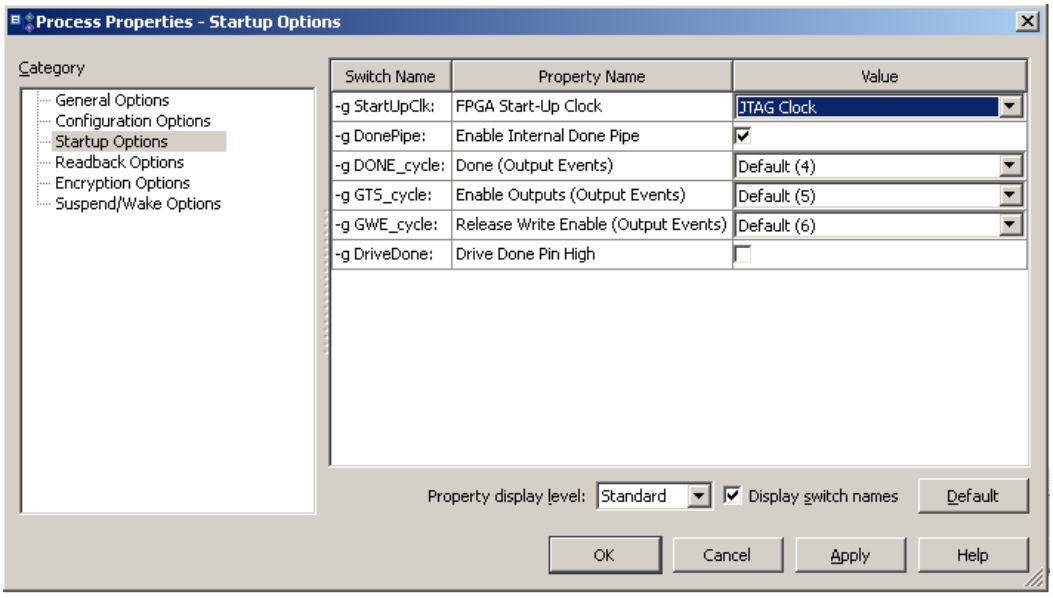


Figura 38. Dialogo de opciones para la generación del fichero

El último paso consiste en programar la placa de desarrollo con un fichero que se ha generado tras el proceso del paso anterior. Concretamente, en este proceso debe haberse generado un fichero de programación de nuestro proyecto con extensión .bit, que debe ser transferirlo por la conexión USB a la FPGA. Para ello haremos lo siguiente:

1. Compruebe en la placa el modo de programación establecido, para ello donde está situado el conmutador Modo de Programación. Asegúrese que la placa está configurada en modo PC (jumper azul en la parte superior derecha de la placa).
2. Conecte el puerto USB de la placa de desarrollo y conmute la alimentación de la placa.
3. Inicie el programa Adept desde el menú de inicio (menú Digilent → Adept, icono) y aparecerá el programa mostrado en (ver figura 39). Con este programa se puede transferir el fichero de programación (bitstream) a la FPGA. El programa Adept permite programar los componentes de la placa ATLYS, estos son una FPGA. Se programará la FPGA, por tanto, se debe utilizar el botón Browse indicado en la figura y seleccionar el fichero .bit. (Hay buscar la carpeta del proyecto ISE en el que está trabajando, allí encontrará el resultado de la síntesis en un fichero con extensión ".bit"). Una vez seleccionado este fichero se activará el botón Program y bastará con pulsarlo para la placa se programe y tengamos nuestro sistema funcionando.

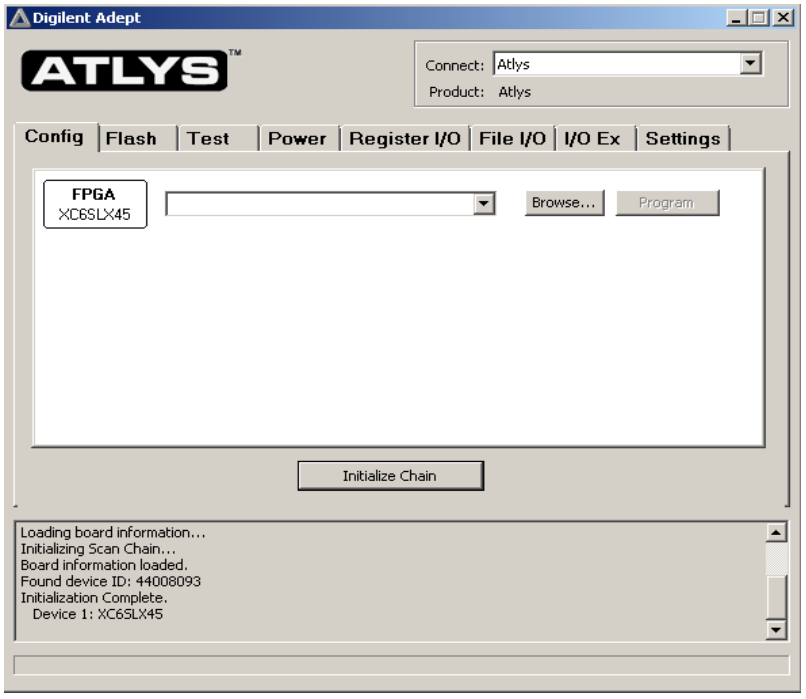


Figura 39. Pogramación de placas Digilent con Adept.

Debe tener en cuenta La FPGA es un dispositivo lógico programable volátil, por lo que se pierde la programación de la misma (y por tanto, nuestra implementación), cuando se retira la alimentación de la placa. Si quiere comprobarlo, apague y encienda la placa y podrá observar cómo nuestro circuito deja de funcionar.

No obstante, tenemos la posibilidad de guardar el diseño en una PROM de la placa (memoria ROM programable no volátil) para que, en caso de pérdida de alimentación, pueda restablecerse el diseño de la programación de la FPGA.

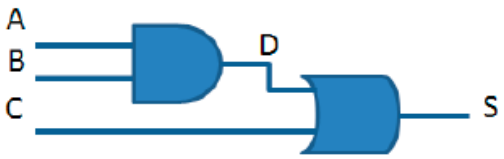
Introducción a VHDL

VHDL es un lenguaje de descripción de circuitos electrónicos digitales que utiliza distintos niveles de abstracción. El significado de las siglas VHDL es VHSIC (Very High Speed Integrated Circuits) Hardware Description Language. Esto significa que VHDL permite acelerar el proceso de diseño.

VHDL no es un lenguaje de programación, por ello conocer su sintaxis no implica necesariamente saber diseñar con él. VHDL es un lenguaje de descripción de hardware, que permite describir circuitos síncronos y asíncronos. Para realizar esto debemos de tener una visión de estos sistemas como:

- Compuertas lógicas (AND, OR, NOT) y biestables, no son variables ni funciones.
- Evitar bucles combinacionales y relojes condicionados.
- Saber qué parte del circuito es combinacional y cuál secuencial.

La misión más importante de un lenguaje de descripción HW es que sea capaz de simular perfectamente el comportamiento lógico de un circuito sin que el programador necesite imponer restricciones (ver ejemplo 1). En el ejemplo, una ejecución del código utilizando las reglas básicas de cualquier lenguaje de programación al uso daría dos resultados diferentes sobre la misma descripción del circuito. Esto es debido a que en HW todos los circuitos trabajan a la vez para obtener el resultado (todo se ejecuta en paralelo) mientras que en software el orden de las instrucciones delimita la actualización de las variables (ejecución secuencial de las instrucciones). Un lenguaje de descripción HW, VHDL o cualquier otro de los existentes en el mercado, nos debe dar el mismo resultado en simulación para los dos programas del ejemplo 1.



Prog 1	Prog 2
D = A and B; S = D or C;	S = D or C; D = A and B

Simulación Hardware		
t = 0ns	t = 5ns	t = 10ns
A = 0	A = 0	A = 1
B = 0	B = 1	B = 1
C = 0	C = 0	C = 0

	Simulación Software	
	Prog 1	Prog 2
S(5 ns)	0	0
S(10 ns)	1	0

Ejemplo 1. Simulación incorrecta de un circuito.

Los circuitos descritos en VHDL pueden ser simulados utilizando herramientas de simulación que reproducen el funcionamiento del circuito descrito. Para la realización de la simulación existe un estándar aprobado por el IEEE, en el cual se explican todas las expresiones propias de VHDL y cómo se simulan. Además, existen herramientas que transforman una descripción VHDL en un circuito real (a este proceso se le denomina síntesis). La sintaxis para síntesis y su implementación final, aunque sigue unas normas generales, depende en gran medida de la herramienta de síntesis seleccionada.

Elementos básicos en VHDL.

Se conoce que un sistema digital está descrito por sus entradas y sus salidas y la relación que existe entre ellas. En el caso de VHDL se describirá el aspecto exterior del circuito: entradas y salidas; y por otro la forma de relacionar las entradas con las salidas. El aspecto exterior, cuántos puertos de entrada y salida tenemos, es lo que denominaremos *entity* (Entidad). Y la descripción del comportamiento del circuito *architecture* (Arquitectura interna), toda *architecture* tiene que estar asociada a una *entity*. Además, aunque no es estrictamente necesario, podemos definir también las bibliotecas y paquetes que vamos a utilizar, lo que nos indicará que tipos de puertos y operadores podemos utilizar. Siempre ha de aparecer la definición de las bibliotecas y paquetes antes de la definición de la *entity* todo esto para evitar errores de sintetizado y el llamado de componentes que no se encuentren definidos.

La biblioteca IEEE y estos tres paquetes asociados los cuales nos permiten el uso adecuado de las compuertas, estos paquetes aparecen por defecto al momento de la creación de un proyecto (ver ejemplo 2).

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use ieee.std_logic_arith.all;
23  use ieee.std_logic_unsigned.all;
```

Ejemplo 2. Bibliotecas de la IEEE.

Entity

Una entidad es la abstracción de un circuito, ya sea desde un complejo sistema electrónico o una simple puerta lógica. La entidad únicamente describe la forma externa del circuito, en ella se declaran las entradas y las salidas del diseño. Una entidad es idea análoga a un símbolo esquemático en los diagramas electrónicos, el cual describe las conexiones del dispositivo hacia el resto del diseño, con esto se puede decir que una entidad es la declaración externa de nuestro diseño, donde se definen las entradas y salidas del mismo.

Los puertos pueden ser de entrada in, salida *out*, entrada-salida *inout* o buffer. Los puertos de entrada sólo se pueden leer y no se puede modificar su valor internamente en la descripción del comportamiento del circuito *(architecture)*, sobre los puertos de salida sólo se puede escribir, pero nunca tomar decisiones dependiendo de su valor (esto implica una lectura). Si es estrictamente necesario escribir sobre un puerto a la vez que se tiene que tener en cuenta su valor el tipo sería *inout* o *buffer*.

Además, en la *entity* se pueden definir unos valores genéricos *(generic*) que se utilizarán para declarar propiedades y constantes del circuito, independientemente de cuál sea la arquitectura. A nivel de simulación utilizaremos *generic* para definir retardos de señales y ciclos de reloj, estas definiciones no serán tenidas en cuenta a nivel de síntesis. También se puede utilizar *generic* para introducir una constante que será utilizada posteriormente en la *architecture*, utilizaremos esa constante para hacer nuestro circuito más general. Por ejemplo, podemos definir el comportamiento de un banco de registros teniendo en cuenta que puede tener cualquier número de registros, fijando el número de registros particular que queremos simular e implementar a través de una constante del *generic*. Esto implica que en toda la parte de nuestro código (el que vamos a escribir dentro de *architecture*) donde haga falta el número de registros utilizaremos el nombre de la constante definida en *generic*, de manera análoga a como se haría en cualquier lenguaje de

programación convencional. La sentencia *generic* no es necesaria, en caso de que no vayamos a utilizarla puede desaparecer de la *entity*.

A continuación, se presenta un ejemplo (ver ejemplo 3) de descripción externa del circuito (*entity*). Para el ejemplo sabemos que el circuito presentará dos entradas de tamaño N bits y una salida de tamaño un bit, particularizamos la entidad para N igual a 8. Como hemos advertido anteriormente, aunque la función de *generic* es permitirnos generar un código más general, una vez que definimos el circuito, tenemos que particularizarlo, por lo que siempre debe darse un valor a las constantes del campo *generic*.

```
34  entity intro is
35     generic (N: natural :=8);
36     port (
37        A, B: in bit_vector(N-1 downto 0);
38        Y: out bit);
39  end intro;
```

Ejemplo 3. Declaración de una entidad en VHDL.

Architecture

Los pares de entidades y arquitecturas se utilizan para representar la descripción completa de un diseño. Una arquitectura describe el funcionamiento de la entidad a la que hace referencia, es decir, dentro de *architecture* tendremos que describir el funcionamiento de la entidad a la que está asociada utilizando las sentencias y expresiones propias de VHDL.

El código VHDL propiamente dicho se escribe dentro de *architecture.* Cada *architecture* va asociada a una *entity* y se indica en la primera sentencia. A continuación, y antes de begin se definen todas las variables (señales) internas que vas a necesitar para describir el comportamiento de nuestro circuito, se definen los tipos particulares que necesitamos utilizar y los componentes, otros circuitos ya definidos y compilados de los cuales conocemos su interfaz en VHDL (su *entity*).

Desde *begin* hasta *end* escribiremos todas las sentencias propias de VHDL, pero no todas pueden utilizarse en cualquier parte del código. Así pues, aquellas sentencias de VHDL que tengan definido un valor para cualquier valor de la entrada (y que nosotros denominamos sentencias concurrentes) podrán ir en cualquier parte del código, pero fuera de la estructura *process*. Aunque no es el fin de este manual, puede afirmarse que todas las sentencias concurrentes se traducirán en subcircuitos combinacionales. También fuera de la estructura *process*, se instanciarán los componentes, subcircuitos ya definido sutilizados por el circuito actual, indicando cuáles son sus entradas y sus salidas de entre las señales del circuito del que forman parte.

El *process* es una estructura particular de VHDL (que se describe con mucho más detalle más adelante) que se reserva principalmente para contener sentencias que no tengan obligatoriamente que tener definido su valor para todas las entradas (el ejemplo más común es una estructura *if-else* incompleta). Esto obliga a que la estructura *process* almacene los valores de sus señales y pueda dar lugar (no siempre) a subcircuitos secuenciales. Además, en simulación sólo se ejecutan las sentencias internas a esta estructura cuando alguna de las señales de su lista de sensibilidad cambia de valor (ver ejemplo 4).

```
42
43  architecture Behavioral of intro is
44  -- declaraciones de la arquitectura:
45  -- tipos
46  -- señales
47  -- componentes
48  begin
49  -- código de descripción
50  -- instrucciones concurrentes
51  -- ecuaciones booleanes
52  -- componentes
53  process (lista de sensibilidad)
54  begin
55  -- código de descripción
56  end process;
57  end Behavioral;
```

Ejemplo 4. Declaración de la *architecture* en VHDL.

Identificadores

En VHDL existen tres clases de objetos por defecto:

- **Constant**. Los objetos de esta clase tienen un valor inicial que es asignado de forma previa a la simulación y que no puede ser modificado durante ésta.
- **Variable**. Los objetos de esta clase contienen un único valor que puede ser cambiado durante la simulación con una sentencia de asignación. Las variables generalmente se utilizan como índices, principalmente en instrucciones de bucle, o para tomar valores que permitan modelar componentes. Las variables NO representan conexiones o estados de memoria. Pueden ser declaradas antes del *begin* de la *architecture* y/o antes del *begin* del *process*, en su declaración se les puede asignar un valor por defecto.
- **Signal**. Las señales representan elementos de memoria o conexiones y sí pueden ser sintetizados, dicho de otra manera, a cada objeto de nuestro código VHDL que sea declarado como *signal* le corresponde un cable o un elemento de memoria

(biestable, registro) en nuestro circuito. Por lo tanto, su comportamiento en simulación será el esperado de ese elemento físico, aunque no lo describamos en el código explícitamente. Tienen que ser declaradas antes del *begin* de la *architecture*. Los puertos de una entidad son implícitamente declarados como señales en el momento de la declaración, ya que estos representan conexiones.

A continuación, se ven las declaraciones de cada uno de los tipos de objetos, (ver ejemplo 5).

```
constant identificador: tipo:= valor;

variable identificador: tipo [:= valor];
nombre variable := valor o expresión;
i := 10;

signal identificador: tipo;
nombre señal <= valor o expresión;
A <= 10;
```

Ejemplo 5. Declaración y asignaciones de los valores.

Estructura básica de un archivo fuente en VHDL.

Como hemos visto los modelos VHDL están formados por dos partes: la entidad *(entity*) y la arquitectura *(architecture);* es en esta última donde se escriben las sentencias que describen el comportamiento del circuito, a este modelo de programación en VHDL se le suele denominar *behavioral.*

Sentencias Concurrentes.

When - Else

Las sentencias concurrentes son sentencias condicionales que tienen al menos un valor por defecto para cuando no se cumplen ninguna de las condiciones. Aunque podría utilizarse una sentencia común como un if con obligación de else, los desarrolladores de VHDL han preferido utilizar dos sentencias particulares:

```
WHEN - ELSE
señal_a_modificar <= valor_1 when condición_1 else
valor_2 when condición_2 else
...
valor_n when condición_n else
valor_por defecto;

-----------------------------------------------
-- Ejemplos when-else
-----------------------------------------------
C <= "00" when A = B else
"01" when A < B else
"10";
```

Ejemplo 6. Sentencia concurrente WHEN-ELSE.

En esta sentencia (ver ejemplo 6) siempre modificamos el valor de una misma señal, pero las condiciones pueden ser independientes (actuar sobre distintas señales cada una), dónde la colocación de las condiciones indica la preferencia de unas sobre otras, es decir, la condición 1 tiene preferencia sobre el resto, la condición 2 sobre todas menos la 1 y así sucesivamente.

With – Select - When

```
WITH - SELECT - WHEN
with señal_condición select
señal_a_modificar <= valor_1 when valor_1_señal_condición,
valor_2 when valor_2_señal_condición, ...
valor_n when valor_n_señal_condición, valor_por_defecto when others;


-------------------------------------------
-- Ejemplo with-select
-------------------------------------------
with entrada select
salida <= "00" when "0001",
"01" when "0010",
"10" when "0100",
"11" when others;
-------------------------------------------
```

Ejemplo 7. Declaración de la sentencia WITH – SELECT – WHEN

Esta sentencia (ver ejemplo 7) es menos general que la anterior. En este caso se modificará el valor de una señal dependiendo de los valores de una señal condición, aparecerán como máximo tantas líneas como valores posibles pueda tener la señal condición.

Desde un punto de vista de HW estas dos sentencias dan como resultado HW combinacional puro, es decir, puertas lógicas, multiplexores, decodificadores.

Sentencias condicionales.

VHDL permite utilizar otro tipo de sentencias condicionales más parecidas a los lenguajes de programación usados. Todas estas sentencias tienen que ir obligatoriamente dentro de un *process*. Las sentencias condicionales más comunes en VHDL son las siguientes:

If – Then – Else.

```
process (lista de sensibilidad)
begin
   if condición then
   -- asignaciones
   elsif otra_condición then
      -- asignaciones
   else
      -- asignaciones
   end if;
end process;
-----------------------------------------------
-- Ejemplo
-----------------------------------------------
process (control, A, B)
begin
   if control = "00" then
      resultado <= A + B;
   elsif control = "11" then
      resultado <= A - B;
   else
      resultado <= A;
   end if;
end process;
-----------------------------------------------
```

Ejemplo 8. Declaración de la sentencia *IF-THEN_ELSE*

La sentencia *if-else* (ver ejemplo 8) permite cualquier tipo de combinación y encadenamiento, exactamente igual que ocurre en C o PASCAL o cualquier otro lenguaje de programación de alto nivel.

Case-When

```
begin
case señal_condición is
   when valor_condición_1 =>
      -- asignaciones
      …
   when valor_condición_n =>
      -- asignaciones
   when others =>
      -- asignaciones
end case;
end process;
-----------------------------------------------
-- Ejemplo
-----------------------------------------------
process (control, A, B)
begin
case control is
   when "00" =>
      resultado <= A+B;
   when "11" =>
      resultado <= A-B;
   when others =>
      resultado <= A;
end case;
end process;
-----------------------------------------------
```

Ejemplo 9. Declaración de sentencia *Case - When*

Dentro de las asignaciones (ver ejemplo 9) pueden parecer también sentencias *if-else*. Es necesario que aparezca en la estructura *when others*, pero no es necesario que tenga asignaciones, se puede dejar en blanco.

For –Loop

```
process (lista de sensibilidad)
begin
   for loop_var in range loop
      -- asignaciones
   end loop;
end process;

-------------------------------------------
-- Ejemplo
-------------------------------------------
process (A)
begin
   for i in 0 to 7 loop
      B(i+1) <= A(i);
   end loop;
end process;
-------------------------------------------
```

Ejemplo 10. Declaración de la sentencia *For – Loop*

Igual que en los lenguajes software, existen distintos tipos de bucles (ver ejemplo 10), en el diseño de hardware este tipo de sentencia no son comúnmente usadas, pero siempre se tienen como herramientas para diseños específicos.

Sentencias Process

VHDL presenta una estructura particular denominada *process* que define los límites de un dominio que se ejecutará (simulará) si y sólo si alguna de las señales de su lista de sensibilidad se ha modificado en el anterior paso de simulación, el *process* tiene la siguiente estructura (ver ejemplo 11).

```
process (lista_de_sensibilidad)
-- asignacion de variables
-- opcional no recomendable
begin
-- Sentenicas condicionales
-- Asignaciones
end process;
```

Ejemplo 11. Estructura del *Process*

La sentencia *process* es una de las más utilizadas en programación con VHDL ya que tanto las sentencias condicionales como la descripción de HW secuencial se realiza dentro de él. Pero a

la vez es, para aquellos que se acercan por primera vez a la simulación y síntesis con VHDL, el principal problema para un correcto diseño. Por eso a continuación se van a enumerar una serie de normas relacionadas directamente con las propiedades de la sentencia *process*, que serán de obligado cumplimiento para que el código generado simule y sintetice de manera correcta.

- **Propiedad I:** En una estructura *process* sólo se ejecutan las instrucciones internas en el instante 0 de simulación y cuando varía alguna de las señales de su lista de sensibilidad.
- **Problema:** El resultado de la simulación del circuito puede ser inesperada debido al efecto maligno de la lista de sensibilidad.
- **Solución**: En la lista de sensibilidad han de incluirse al menos todas las señales que se lean dentro del process (señal_escritura <= señal_lectura), esto para asegurarse que a señal_escritura se le sea asignado su valor señal_lectura apenas cambie de estado.
- **Propiedad II:** Las asignaciones a señales que se realizan dentro de un *process* tienen memoria.
- **Problema:** Si en un paso de simulación se entra dentro del *process* y debido a las sentencias internas se modifica el valor de la señal C, y en otro paso de simulación posterior se entra dentro del *process* pero no se modifica C, la señal C conservará el valor asignado con anterioridad. El resultado de la simulación del circuito puede ser inesperado debido al efecto maligno de la memoria del *process*, esto terminara creando un latch.
- **Solución:** Siempre que se escriba una sentencia condicional es obligatorio asegurar el valor que deben tener todas las señales en cada rama del árbol condicional. Además, excepto si la definición del diseño nos lo prohíbe, toda condición debe tener su rama *else* (ver ejemplo 12).

```
--------------------------
-- Condicional completo
--------------------------
process (a, b)
begin
   if a = b then
      c <= a or b;
   elsif a<b then
      c <= b;
   else
      c <= "00";
   end if;
end process;
--------------------------
--------------------------
-- Condicional incompleto
--------------------------
process (a, b)
begin
   if a = b then
      c <= a or b;
   elsif a<b then
      c <= b;
   end if;
end process;
--------------------------
```

Ejemplo 12. Propiedad II de un *process*.

- **Propiedad III:** Dentro de un *process* todas las instrucciones se ejecutan en paralelo, igual que ocurre con las instrucciones que se encuentran fuera de los *process*. Sin embargo, si dentro del *process* se asigna valor a una señal en dos sitios diferentes el resultado será aquel de la última asignación, exactamente igual que en los lenguajes de programación comunes.
- **Solución**: Siempre comprobar que no estamos asignando el valor a una señal en dos sitios diferentes del *process* (si puede hacerse en dos ramas diferentes del mismo *if*).
- **Propiedad IV:** Los *process* se ejecutan en paralelo entre sí.
- **Problema:** Existirá con bastante probabilidad un código con dos o más *process*, en esos casos éstos se ejecutan en paralelo como ocurre con el resto de las sentencias. Si dos *process* se están ejecutando en paralelo no se puede modificar la misma señal en ambos *process*, porque en ese caso no se podría saber cuál es el valor real de la señal (el obtenido en el *process* 1 ó el obtenido en el *process* 2).

- **Solución:** Siempre hay que comprobar que no se modifica la misma señal en dos *process* diferentes, en caso de que esto ocurra habrá que intentar fusionar los dos *process*.
- **Propiedad V:** Los valores de las señales que se modifican internamente en los process no se actualizan hasta que no se ha ejecutado el *process* completo.
- **Problema:** Como resultado de esta regla, si no se ha escrito correctamente la lista de sensibilidad podría ocurrir lo que se observa en el ejemplo "actualización 1", C va retardada con respecto a B, problema que se soluciona escribiendo correctamente la lista de sensibilidad (ver ejemplo 13).
- **Solución:** Si la asignación problemática se realiza dentro de una estructura *process* + sentencias que dan como lugar HW secuencial, la solución pasa por sacar alguna asignación del *process* y llevarla a otro *process* nuevo independiente.

```
-------------------------
-- Actualizacion 1
-------------------------
process (A)
begin
C <= A;
B <= C;
end process;
-------------------------
-- Sensibilidad arreglada
-------------------------
process (A, C)
begin
C <= A;
B <= C;
end process;
```

Ejemplo 13. Propiedad V de los process.

Descripción de Lógica secuencial

Una de las propiedades más importantes de un *process* es la capacidad de la estructura para almacenar los valores de las señales que se asignan en su interior si durante el paso de simulación no se entra dentro del *process* o no se realiza ninguna asignación a esa señal. Debido a esta característica se utilizarán los *process* para generar HW secuencial.

Es importante conocer que a pesar del comportamiento que tenga el *process* genere un hardware secuencial no implica que las distintas instrucciones internas a un *process* se ejecuten secuencialmente.

Hardware secuencial

Para conseguir crear el HW secuencial esperado hay que cumplir una serie de reglas:

- Una sentencia *if* que tenga por condición una especificación de flanco no puede tener rama *else*, en caso contrario la rama *else* debería realizarse en todo momento menos en el preciso instante en el que el reloj cambia.
- En sentencias *if-then-elsif* la especificación de flanco sólo podrá ser la condición del último *elsif* (que no podrá tener rama else).
- Una sentencia if que tenga por condición una especificación de flanco puede tener internamente sentencias *if-else* encadenadas.
- En un *process* solo puede existir una única especificación de flanco, en caso contrario se estaría especificando HW secuencial sensible a varios relojes.

Se ilustra la manera de crear HW secuencial con los siguientes dos ejemplos. La diferencia entre el primer y el segundo ejemplo es el orden de asignación de los valores a b y c, en el primer caso esa asignación se haría como en cualquier lenguaje de programación, en el segundo caso parecería que la asignación es incorrecta (ver ejemplo 14). Para comprender los resultados hay que tener en cuenta dos propiedades de los *process* mencionadas con anterioridad:

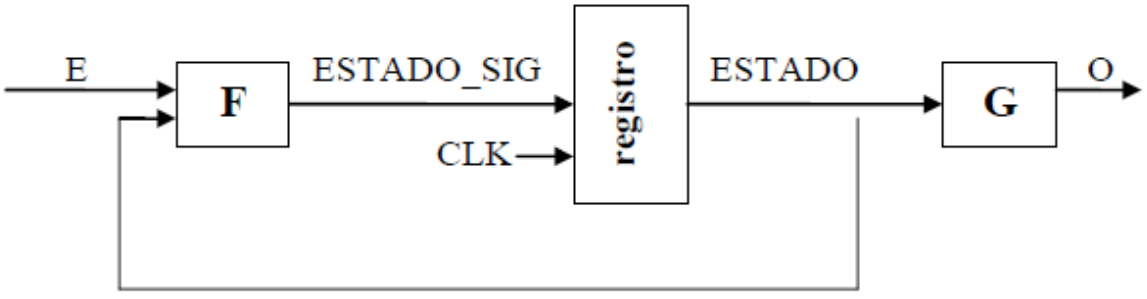
- Todas las asignaciones dentro de un *process* se hacen en paralelo, por lo que el orden de las asignaciones no altera el resultado final.
- No se actualizan los valores de las señales modificadas internamente en el *process* hasta que no se han terminado de ejecutar todas sus instrucciones internas. En este caso esto se ve agravado por el hecho de que dentro del if sólo se entra en el momento exacto en el cual se produce el flanco, lo que se corresponde con un único δ de simulación.

```
-------------------------------------------
-- Ejemplo 1
-------------------------------------------
process (clk, a, b, reset)
begin
   if reset = '1' then
      b <= '0';
      c <= '0';
   elsif clk'event and clk ='1' then
      b <= a;
      c <= b;
   end if;
end process;
-------------------------------------------
-- Ejemplo 2
-------------------------------------------
process (clk, a, b, reset)
begin
   if reset = '1' then
      b <= '0';
      c <= '0';
   elsif clk'event and clk ='1' then
      c <= b;
      b <= a;
   end if;
end process;
-------------------------------------------
```

Ejemplo 14. Hardware Secuencial con Biestables.

Descripción de una Maquina de Estados.

VHDL permite realizar descripciones algorítmicas de alto nivel de máquinas de estados. De esta forma, el diseñador se evita tareas como generar la tabla de transición de estados o la obtención de las ecuaciones de excitación basadas en un tipo de biestable (ver ejemplo15).



Ejemplo 15. Diagrama de las máquinas de estado.

Una Máquina de Estados Finita (FSM) se puede describir en VHDL de varias formas la que se propone en este capítulo es la forma estándar para cualquier herramienta que trabaje con VHDL.

En primer lugar, en la sección de declaraciones de la arquitectura, se define un tipo enumerado en el que se asignan identificadores a cada estado. Suele ser recomendable utilizar identificadores ilustrativos para los estados. La herramienta de síntesis será la encargada de codificar estos estados.

Posteriormente, en el cuerpo de la arquitectura se define la función de transición de estados (F) y la función de salida (G) en un process y el registro es decir el cambio de estado_sig a estado en otro process. Por lo tanto, se tiene:

- Un proceso secuencial que modela los biestables de estado; por lo tanto, que actualiza el estado (ESTADO).
- Un proceso combinacional que modela las funciones F y G; por lo tanto, deriva el siguiente estado (ESTADO_SIG) y actualiza las salidas (O).

Para ilustrar la descripción de máquinas de estados vamos a pensar en un ejemplo Se trata de diseñar una máquina de estados que active una salida S cuando se detecta la secuencia “001” en una línea de datos E de un bit sincronizada con un reloj. Este detector de secuencia se puede realizar con una máquina de Moore de cuatro estados.

- S1: Esperar el ler Cero de la secuencia.
- S2: Esperar el 2do Cero de la secuencia.
- S3: Esperar el uno de la secuencia.
- S4: Activar la salida.

Para la implementación en VHDL. Primero se define un tipo enumerado, formado por los nombres de los estados y se declaran dos señales de este tipo (ver ejemplo 16).

```
type ESTADOS is (S1, S2, S3, S4);
signal ESTADO, ESTADO_SIG: ESTADOS;
```

Ejemplo 16. Definición del tipo de enumerado

A continuación creamos un proceso combinacional en el que se determina el siguiente estado (ESTADO_SIG) y la salida S en función del estado actual (ESTADO, E). El programa nos quedaría de la siguiente manera (ver ejemplo 17).

```
library IEEE;
use IEEE.std_logic_1164.all;

entity FSM is
   port(
      reset, E, clk: in std_logic;
      O: out std_logic
      );
end FSM;

architecture ARCH of FSM is
   type ESTADOS is (S1, S2, S3,S4);
   signal ESTADO, SIG_ESTADO: ESTADOS;
begin

SINCRONO: process(clk,reset)
begin
   if reset ='1' then
      ESTADO<=S1;
   elsif clk'event and clk='1' then
      ESTADO<= SIG_ESTADO;
   end if;
end process SINCRONO;
```

```
COMBINACIONAL: process(ESTADO,E)
begin
case ESTADO is
   when S1 =>
         O <= '0';
      if (E='0') then
         SIG_ESTADO<=S2;
      else
         SIG_ESTADO<=S1;
      end if;
   when S2 =>
         O <= '0';
      if (E='0') then
         SIG_ESTADO<=S3;
      else
         SIG_ESTADO<=S1;
      end if;
   when S3 =>
         O <= '0';
      if (E='0') then
         SIG_ESTADO<=S3;
      else
         SIG_ESTADO<=S4;
      end if;
   when S4 =>
         O <= '1';
      if (E='0') then
         SIG_ESTADO<=S2;
```

```
        when S4 =>
              O <= '1';
           if (E='0') then
              SIG_ESTADO<=S2;
           else
              SIG_ESTADO<=S1;
           end if;
        end case;
    end process COMBINACIONAL;
    end ARCH;
```

Ejemplo 17. Descripción de una máquina de estado para la detección de secuencia

4.3 Diseño de sistemas digitales con FPGA para experiencias prácticas reproducibles en el laboratorio de microprocesadores de la UJAP (manejo de entradas / salidas digitales, comunicación de datos y aplicaciones de control).

Para el desarrollo del objetivo tres, se llevó a cabo el diseño de sistemas digitales con FPGA el cual se empleó un diseño jerárquico en VHDL para la elaboración de cada bloque digital. Y luego su posterior unión, con esto se buscó analizar y simular de manera individual cada uno de los bloques para obtener un mayor control al momento de una modificación del sistema digital, luego se relacionado cada bloque a través de un algoritmo de integración Top Level.

4.3.1 Descripción del diseño (Manejo de entradas / salidas digitales).

Antes de comenzar con la descripción del circuito, se van a comentar las especificaciones concretas establecidas que deben implementarse.

La pantalla LCD de 16x2 cuenta con dos filas y cada fila tiene la capacidad de mostrar 16 caracteres o símbolos, por lo general alfanuméricos. Al momento de energizar se requiere una inicialización para que logre mostrar de manera correcta los caracteres o símbolos en la pantalla, la pantalla cuenta con su propia tabla de caracteres y símbolos que deben ser tomados en cuenta para imprimir cualquiera de estos. Su modo de funcionamiento se puede establecer entre cuatro bits u ocho bits, para la implementación del mismo diseño se usa el modo de cuatro bits ya que nos permite el ahorro de puertos, igualmente cuenta con un pin de Habilitación (E) y un pin de escritura/lectura (R/S) que deben ser controlados por la FPGA, dando un total de seis pines requeridos. El pin de habilitación (E) no debe tener un periodo menor a 100us para que el microprocesador interno de la LCD sea capaz de captar la información transmitida.

Para la implementación del teclado la tarjeta ATLYS de XILINX ofrece un puerto USB (HOST J13) con entrada dedica a comunicación PS/2. El teclado solo inicia una comunicación cuando las líneas de datos y de reloj se encuentran inactivas y están en un valor lógico de ‘1’. Antes de enviar cualquier dato el teclado verifica si el anfitrión está utilizando las líneas. Para facilitar la identificación del uso de las líneas de comunicación, la señal de reloj se utiliza como indicador de: "Libre para enviar" ya que el reloj debe estar en un valor de ‘1’ lógico; si el dispositivo anfitrión desea que el teclado no inicie ninguna comunicación solo debe colocar el valor lógico de la señal de reloj a ‘0’.

El formato de datos que envía el teclado al dispositivo anfitrión es una trama de 11 bit, los cuales se detallan a continuación: el primer bit es el bit de arranque (START), los 8 bit siguiente es el dato correspondiente a código de teclas o códigos de control, el bit 10 es un bit de paridad y el

bit 11 es el bit de parada (STOP). En total son 11 transiciones de reloj a una frecuencia de 20 a 30 kHz y los datos son válidos en los flancos de bajada del reloj. El código enviado por el teclado está en un formato hexadecimal, así se implementó una lógica capaz de realizar el cambio hacia el formato binario requerido por la LCD. Cabe destacar que fue necesaria una lógica capaz de detectar caracteres especiales y en casos como Bloq Mayús el intercambio de carácter entre mayúscula y minúscula.

Arquitectura del módulo de control de comunicaciones PS/2

En este apartado se describe en detalle la jerarquía del sistema completo y su interfaz. El sistema completo engloba a todos los componentes que han sido necesarios para la realización del diseño, algunos bloques ya sea de forma parcial o total pueden ser utilizados en otros proyectos, ver figura 40.

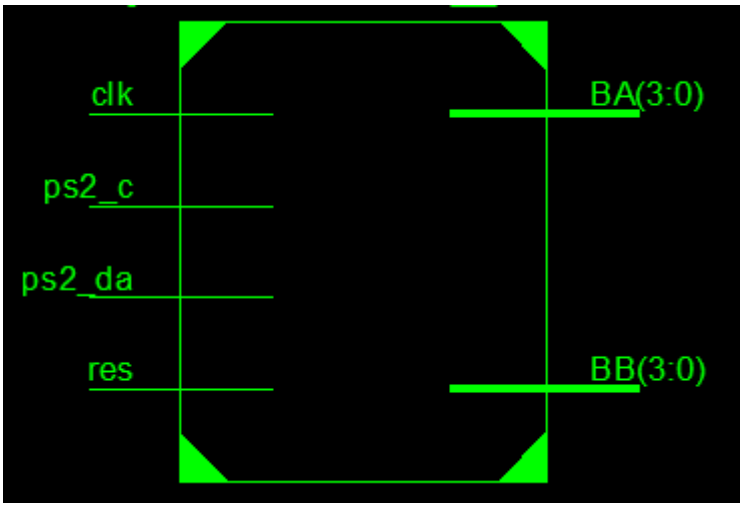


Figura 40. Esquema RTL de Hardware (manejo de entradas/salidas digitales).

Entradas:

- Clk: reloj interno de 100 MHz
- Ps2_c: Reloj externo transmitidos por el teclado para la captura de bits.
- Ps2_da: Puerto de datos transmitidos por el teclado.
- Res: Puerto para reset

Salidas:

- BA: bit de datos más significativos para entrada del LCD
- BB: bit de datos menos significativos para entrada del LCD.

En la figura 41 se muestra el esquemático RTL de bloques del sistema desarrollado para el teclado. El sistema está compuesto de cinco bloques claramente diferenciados.

- Bloque PS/2: se encarga de escanear los datos enviados por el teclado y verificar que la tecla pulsada fue soltada.
- Bloque de registro: se encarga de almacenar la tecla presionada para posteriormente ser transmitida al bloque codificador.
- Bloque codificador: se obtiene toda la lógica para obtener el cambio de letra mayúscula a minúscula o verificar que fue presionada alguna tecla especial.
- Bloque decodificador: por último, se tiene el decodificador que se evalúa que tecla fue presionada mediante su código hexadecimal y se obtiene su formato equivalente en el código de teclas para la LCD y ser mostrada en pantalla.

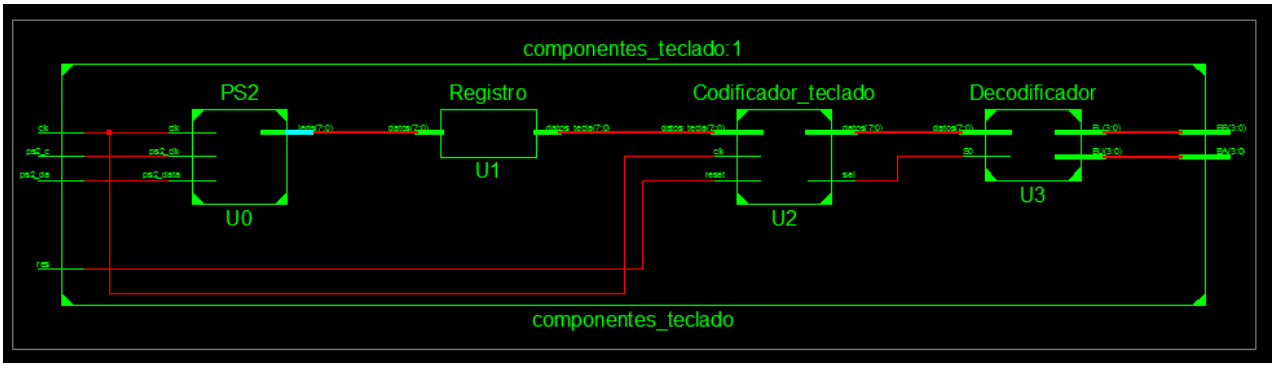


Figura 41. Esquema RTL del componente teclado internamente.

Bloque PS/2.

Este componente es el encargado de controlar la captura de los datos enviados por el teclado. Para este diseño se utilizan dos entradas de señal del tipo ***std_logic,*** las cuales serán ps2_data, ps2_clk estas señales son las encargadas de recibir el bit de información y el reloj externo entregado por el teclado, cabe destacar que este bloque tiene una entrada física dedicada que nos proporciona la tarjeta ATLYS de XILINX que será el puerto USB (HOST J13), también cuenta con una entrada de reloj interno para sincronizar la cargada de datos en su salida, luego para su salida se declaró una señal del tipo ***std_logic_vector*** de 8 bits con la información del carácter presionado en hexadecimal, este componente cuenta con dos ***process***, ver figura 42:

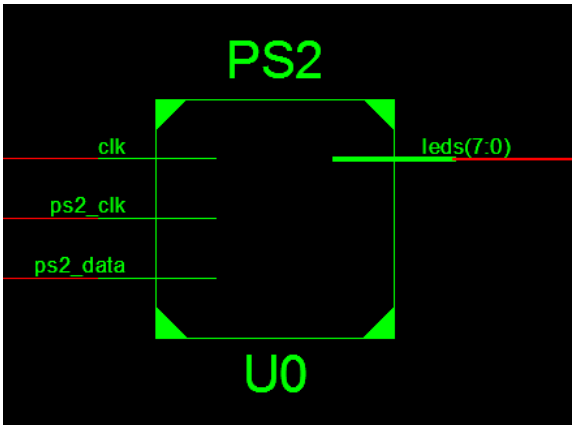


Figura 42. Esquema RTL del componente PS2

Entradas:

- ps2_data: Entrada para la obtención del tramo de 11 bits.
- ps2_clk: reloj para la obtención de los datos transmitidos.
- clk: reloj interno del sistema

Salida:

- Leds: Transmisión de la tecla presionada.

Estos procesos transcurren de la siguiente manera:

El primer ***process*** se encarga del puerto de reloj (ps2_c) observa un cambio de estado de '1' lógico a '0' lógico el cual habilita el proceso para cargar el bit que es trasmitido por el puerto de dato (ps2_da) e ir realizando un corrimiento en una señal que fue declara dentro de la arquitectura ***data_frame*** del tipo ***std_logic_vector*** de 22 bits para almacenar el carácter.

El segundo ***process*** se encarga de verificar que la tecla ha sido soltada, la lógica encargada de esto realiza una comparación con los bits del ***data_frame*** del (8vo bit al 1er bit) con la representación hexadecimal (F0) que confirma que la tecla fue soltada, para posteriormente transmitir el carácter pulsado al bloque del registro.

Bloque de Registro.

Este componente tiene como función almacenar el valor de la tecla pulsada y luego transmitir el dato al bloque codificador, ver figura 43.

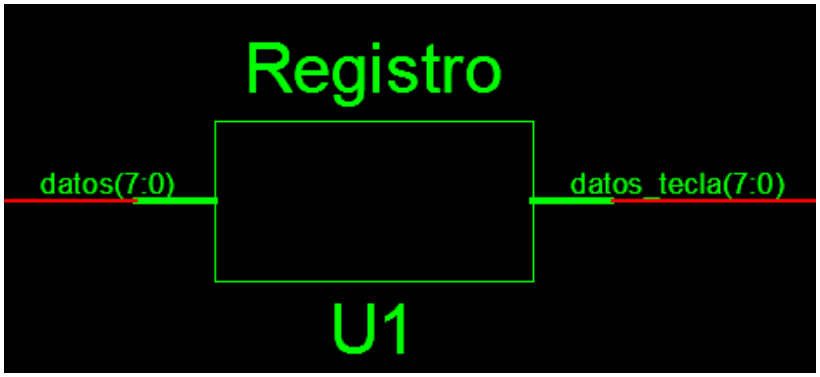


Figura 43. Esquema RTL del componente Registro

Entradas:

- Datos: Almacenar la tecla presionada y transmitirla.

Salida:

- Datos_tecla: Transmisión de la tecla pulsada.

Bloque Codificador _teclado.

En el siguiente modulo se encuentra la lógica para determinar en qué formato se encuentra el carácter, ya sea en mayúscula o minúscula. El componente fue diseño con tres señales de entrada las cuales son clk del tipo ***std_logic***, una entrada para el reinicio del sistema reset del tipo ***std_logic*** y datos_tecla del tipo ***std_logic_vector*** de 8 bits para almacenar el carácter pulsado, para sus salidas se declararon dos señales una para el carácter pulsado del tipo std_logic_vector de 8 bit y una para determinar si se encuentra en mayúscula o minúscula el carácter. El diseño cuenta con tres ***process*** esto debido que la lógica de control requería una máquina de estados para determinar en qué formato se encontraba el carácter, ver figura 44.

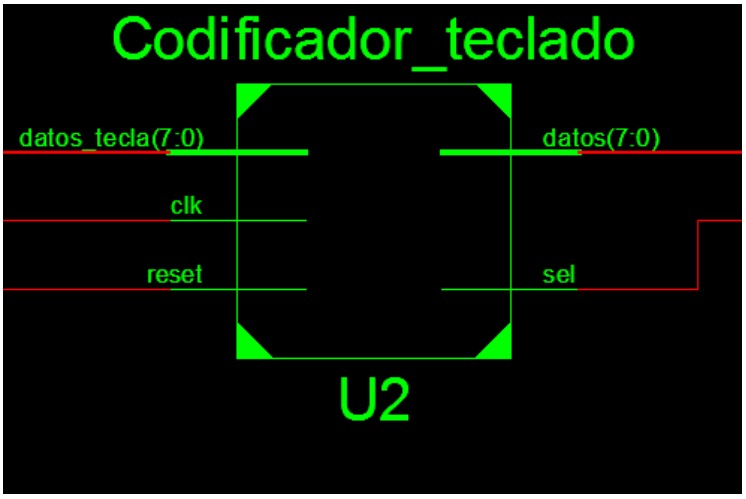


Figura 44. Esquema RTL del Codificador

Entradas:

- Datos_tecla: Carácter en código hexadecimal de la tecla pulsada.
- Clk: reloj interno del sistema
- Reset: reinicio o detección del sistema.

Salidas:

- Datos: Carácter en código hexadecimal para la transmisión de la tecla pulsada.
- Sel: Dato del carácter especial pulsado.

Estos procesos transcurren de la siguiente manera:

El primer ***process*** se encarga de la parte secuencial de la máquina de estados, la cual consta de cinco estados (A, B, C, D, E). El estado inicial de la secuencia es para un formato de carácter por defecto en minúscula, el cual es el comportamiento esperado para el teclado cuando enciende. A continuación, se ve el diagrama de estado (ver figura 44), luego de presionar el carácter especial (Bloq Máyus) el estado siguiente será el B, definiendo en su salida Sel un '1' lógico, esto hará que se interprete en el decodificador que los caracteres a continuación tendrán un formato en mayúsculas. Luego los siguientes estados se encargan de conservar su estado lógico y la transición entre mayúsculas y minúsculas.

En el segundo ***process*** se encarga sincronizar el estado presente con el estado futuro para la máquina de estado.

Por último, el tercer ***process*** tiene la función de un flip-flop con señal de habilitación y un reset asíncrono, esto nos permite reiniciar el sistema y detener la trasmisión de datos, la señal de habilitación va dirigida con la detención de los flancos de subida (pasar de '0' lógico a '1' lógico) con esto se transmite el carácter pulsado hacia el decodificador.

Bloque decodificador

Por último, en la arquitectura del decodificador se encuentra un multiplexor, que se encarga de transformar el formato hexadecimal que envía el teclado a un formato binario que sea capaz de procesar el microcontrolador de la pantalla LCD, Para el diseño fueron declaradas dos señales de entradas, para los datos del carácter es tipo ***std_logic_vector*** de 8 bits y una encargada de verificar en que formato (Mayúsculas o minúsculas) se encuentra el teclado. Para sus salidas contamos con dos señales las cuales son BL del tipo ***std_logic_vector*** de 4 bits como BU del mismo tipo de señal, ver figura 45.

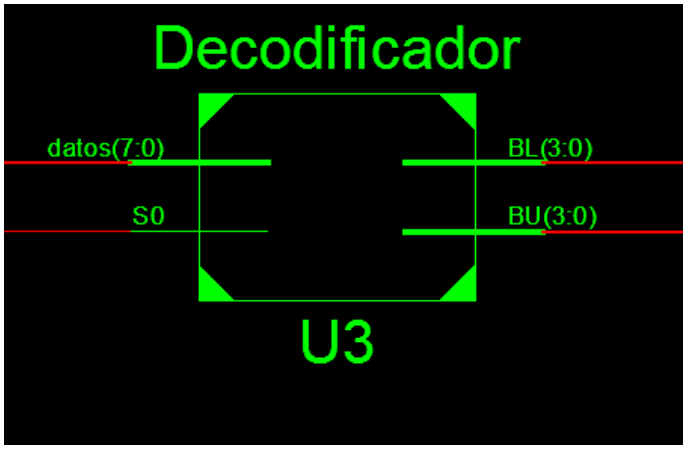


Figura 45. Esquema RTL del componente Decodificador

Entradas:

- Datos: Entrada del carácter en formato hexadecimal de la tecla pulsada
- S0: dato para determinar que tecla especial fue pulsada.

Salidas

- BL: Salida del dato binario menos significativo.
- BU: Salida del dato binario más significativo.

Estos procesos transcurren de la siguiente manera:

Para el diseño se declaró una estructura case que nos permite evaluar el formato del carácter, luego internamente se declaran condicionales lógicos que evalúan el valor del dato en hexadecimal y trasmite por su salida el dato binario para ser procesado correctamente por el LCD.

Arquitectura de control para el LCD 16x2.

La arquitectura de control para el LCD 16x2 está compuesta internamente por una máquina de estado (ver figura 46) que controla toda la lógica necesaria para el correcto funcionamiento del LCD como se ve en el diagrama de estados de la figura 46. Esta secuencia se diseñó para inicializar y lograr imprimir los caracteres en la pantalla, Al diseño se le incluye una entrada de reloj interna encargada de controlar la secuencia de la máquina de estado, cabe destacar que es necesario incluir un divisor de frecuencia para garantizar que los tiempo de la transmisión de datos están en el rango aceptado por la pantalla LCD y no ocurran errores de escritura en el microcontrolador interno de la pantalla, Al comprobar que la secuencia de inicialización se completa de manera correcta la máquina de estados quedara en un estado de reposo (IDLE) a la espera imprimir el carácter enviado por el teclado o instrucción, ver figura 47.

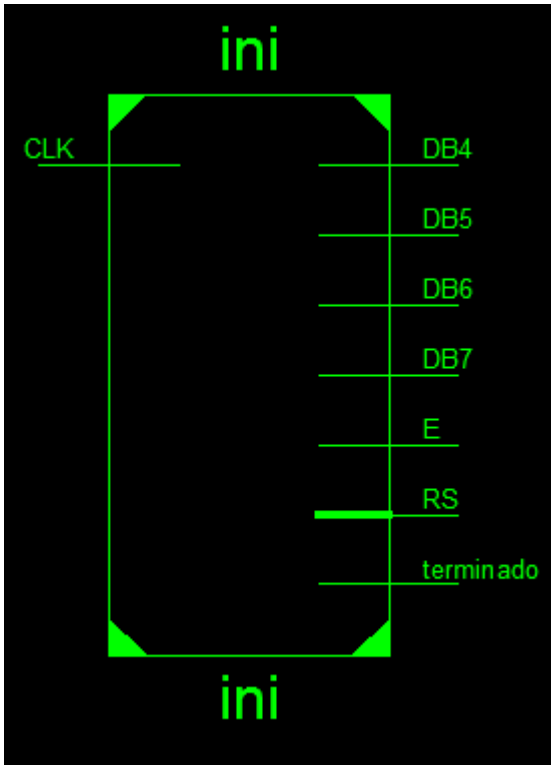


Figura 47. Esquema RTL del módulo LCD

Entradas:

- CLK: Reloj interno de 100 Mhz

Salidas:

- DB4: bit de control para el puerto DB4 de la LCD
- DB5: bit de control para el puerto DB5 de la LCD
- DB6: bit de control para el puerto DB6 de la LCD
- DB7: bit de control para el puerto DB7 de la LCD
- E: bit de control para el pin Enable de la LCD
- RS: bit de control para el pin de escritura o lectura R/S de la LCD
- Terminado: Bit de control para confirmar que se inicializo de manera correcta la LCD

Divisor de frecuencia:

Se implementó debido a que la frecuencia en estos bloques es diferente con respecto a la usada en la tarjeta de desarrollo ATLYS de XILINX que contiene un reloj de 100 Mhz. El diseño del divisor se basó en un contador que genera una señal cuadrada con la frecuencia deseada, La cuenta máxima del contador que se busca en el proyecto se calcula:

$$Frec.salida = \frac{1}{40\ Hz} = 25ms$$

$$\frac{Frec.Tarjeta}{Frec.Salida} = \frac{100\ Mhz}{40\ Hz} = 2500000$$

La elección de dicha frecuencia es para tener un alto margen de error al momento de ingresar un dato, dando un nivel de habilitación de 25 ms, que los requeridos por la LCD es un tiempo no menor a 50 us, con esta frecuencia nos da un gran margen para él envió de los mismos.

Diagrama de estado:

Máquina de estado principal compuesta por 24 estados que estarán detallados a continuación:

- A, B: Se encarga de trasmitir a los pines DB7, DB6, DB5, DB4 que el modo de trabajo a implementar en la LCD es a 4 bits y habilita E de '0' lógico a '1' lógico para procesar el código.
- C: Se encargar de habilitar la función set (para el modo del display).
- D: Termina de habilitar la función set y da paso a los siguientes comandos.
- Es: Habilita la función para el encendido y apagado del display.
- F: Finaliza el comando para la habilitación del display. Y se encuentra configurado
- G: El siguiente estado limpia el display para seguir ejecutando las demás instrucciones.
- H: finaliza la instrucción de limpieza.
- I: Se encarga de ejecutar la instrucción (Entry Mode Set) para habilitar la escritura de caracteres e imprimirlos en la pantalla.
- IDLE: Estado de espera, luego de que finaliza la secuencia de habilitación de la LCD, se ingresa en un modo de espera para la lectura de los caracteres transmitidos por el teclado.
- E1 – E11: Estados encargados de deshabilitar la señal E luego de cada transición de estado.

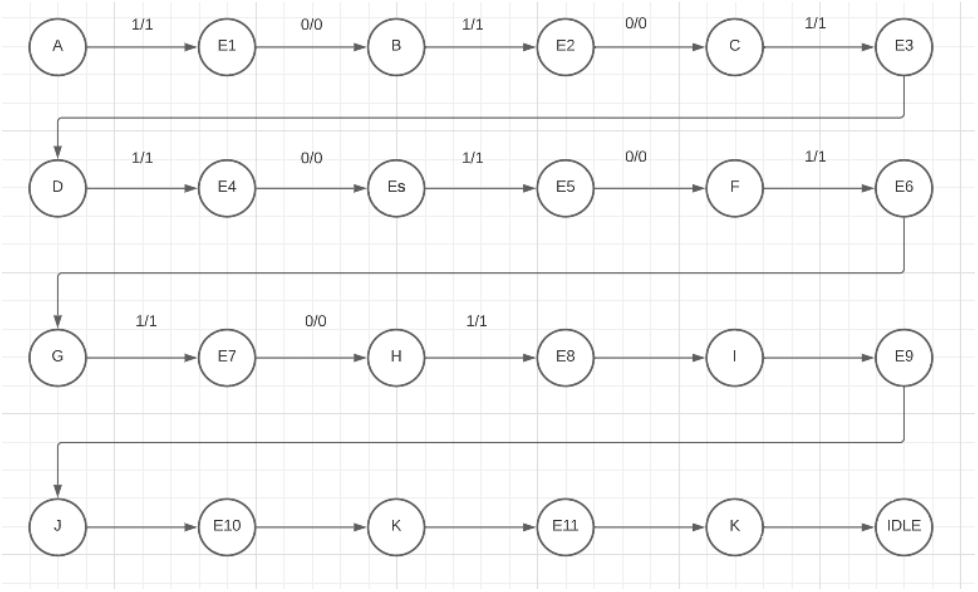


Figura 46. Diagrama de estado de la inicialización para la LCD

Simulación módulo de control para el teclado.

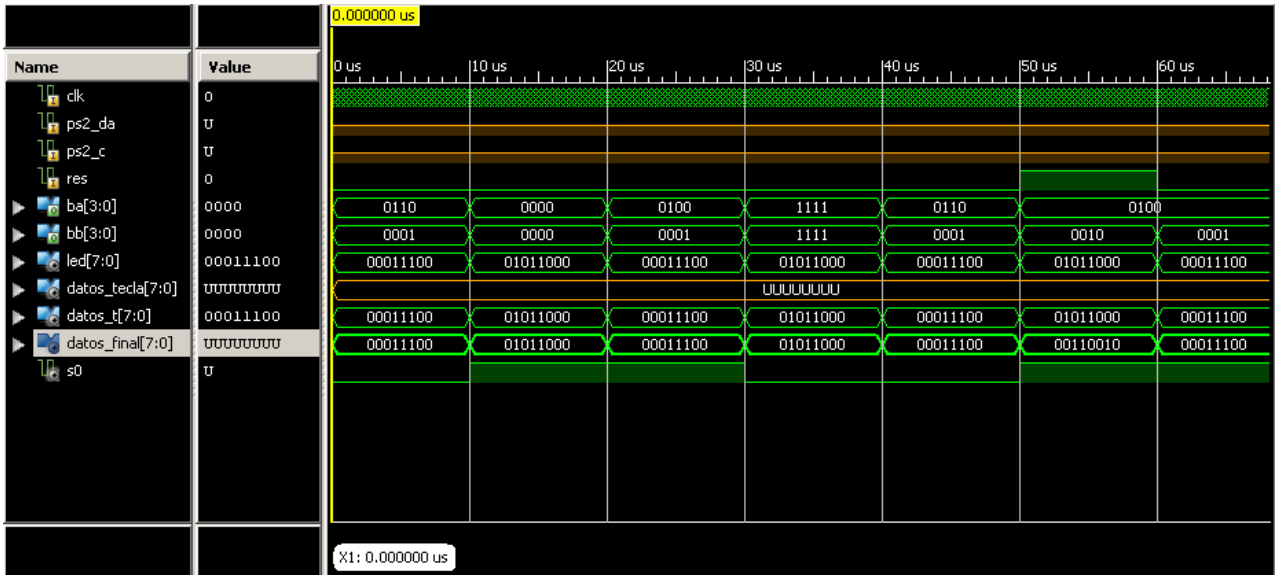


Figura 48. Simulación de la arquitectura del teclado.

Simulación del módulo de control del LCD.

.

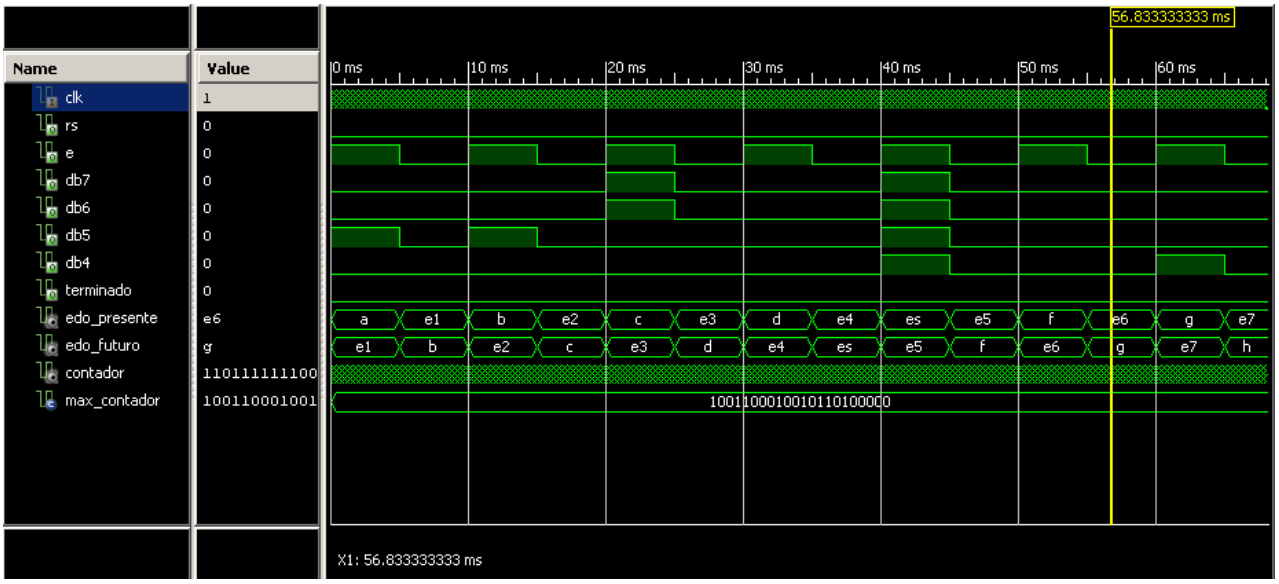


Figura 49. Simulación de la arquitectura del LCD.

4.3.2 Descripción del diseño (Comunicaciones de datos).

A continuación se comentaran de forma concreta las especificaciones y características que debe manejar el diseño.

El diseño consta de un sensor ultrasónico HC-SR04 que tiene la capacidad de medir la distancia mediante ondas de sonidos, en un rango de 2 a 450 cm. Se debe tener en cuenta el tiempo requerido por el pin de disparo (Trigger) que debe ser una señal de alto de 10 us. Luego el modulo automáticamente enviara una onda cuadrada de 40 kHz y esta misma es detecta al regreso de la señal por el sensor (Echo) esta señal se verá representada por este pin en el cual se debe leer constantemente cuando pase de un estado lógico ‘0’ a un estado lógico ‘1’ en donde comienza el tiempo que transcurre la emisión de la onda y recepción que acaba cuando ocurra un cambio de un ‘1’ lógico a un ‘0’ lógico este cambio nos ayuda a detectar cual fue el tiempo de duración y obtener la distancia del objeto. Esta información será mostrada por medio de los leds que tiene integrados la tarjeta ATLYS de XILINX.

Arquitectura del módulo de control de comunicaciones con el HC-SR04.

A continuación se describe en detalle la jerarquía del sistema completo y su interfaz. El sistema completo engloba a todos los componentes que han sido necesarios para la realización del

diseño, algunos bloques ya sea de forma parcial o total pueden ser utilizados en otros proyectos ver figura 50.

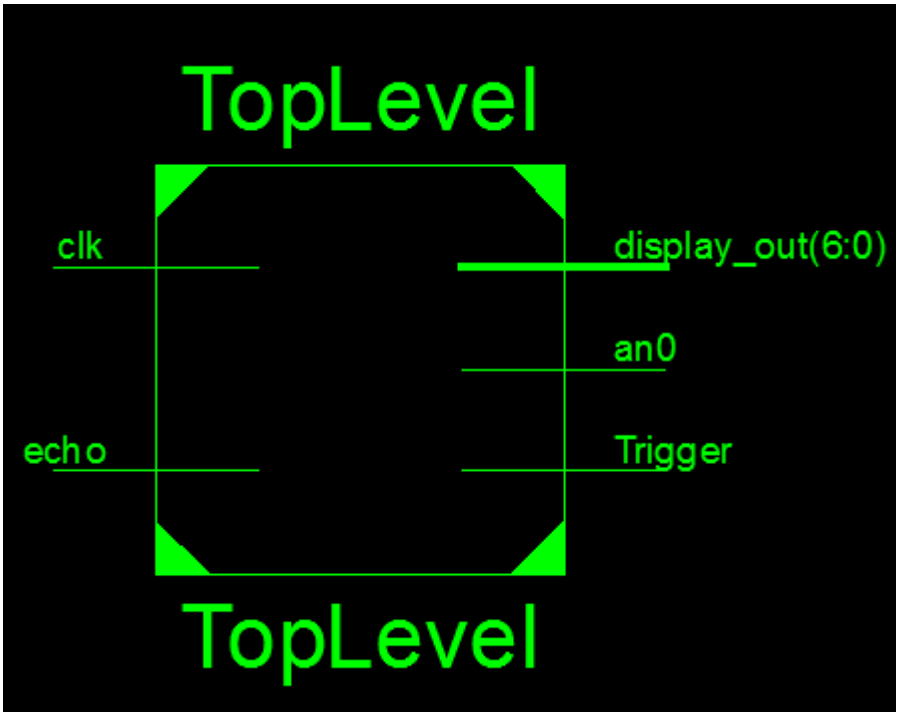


Figura 50. Esquema RTL del módulo de control para el HC-SR04

Entradas:

- Clk: reloj interno de 100 MHz
- Echo: Encargado de capturar la lectura del pin Echo del sensor.

Salidas:

- Trigger: Encargado de enviar el pulso de 10us al pin trigger del sensor
- Display_out: Encargado de suministrar los datos de distancia al módulo de la pantalla LCD de 16x2.
- An0: Encargado de suministrar que fue enviada a señal de distancia.

En la figura 51 se muestra el esquemático RTL del módulo para el sistema desarrollado sobre el sensor HC-SR04. El sistema está compuesto de cinco bloques claramente diferenciados.

- counter: se encarga de tomar el tiempo de duración de la señal echo
- fd: es un registro que se encarga de almacenar el tiempo que es trasmitido por el counter.
- Distance_calculation: modulo encargado de toda la lógica para obtener las distancias que se tienen del objeto.

- Distance_decoder modulo decodificador para transforma la distancia en una unidad capaz de ser leída por modulo del display LCD 16x2.
- TriggerGen: modulo encargado de emitir el pulso de 10us hacia el pin trigger del sensor HC-SR04.

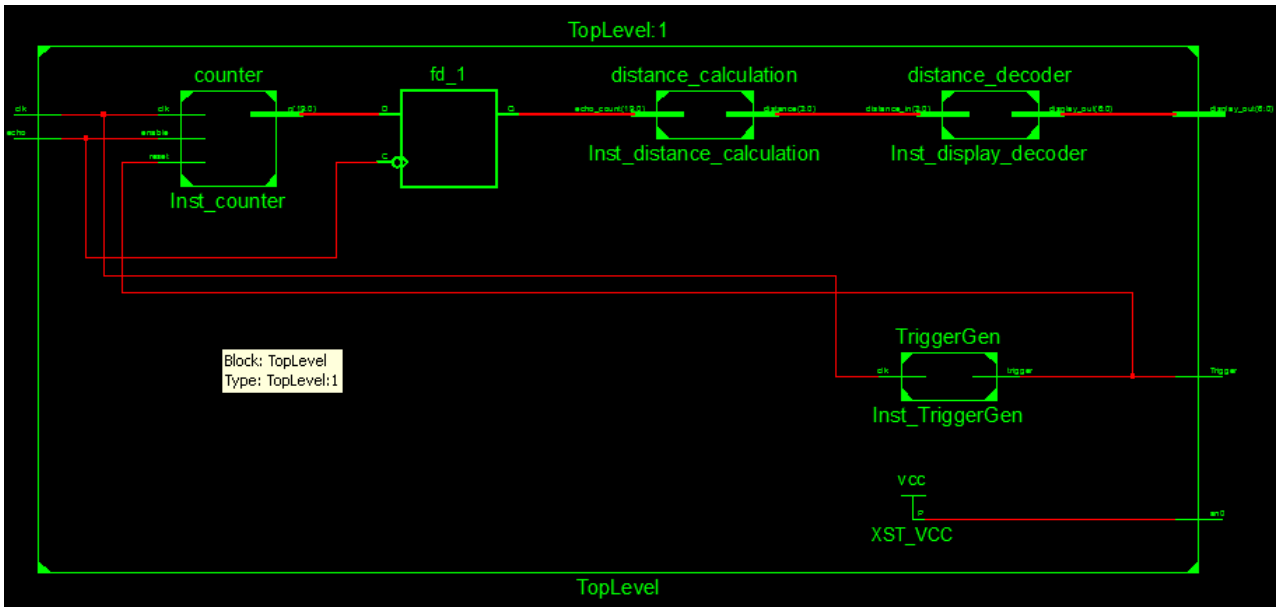


Figura 51. Esquema RTL de los módulos del controlador para el HC-SR04

Bloque counter

En el módulo de counter se encarga de contar el tipo de duración de la señal (echo) durante un estado lógico ‘1’, este conteo ocurre al momento del cambio de estado lógico de (echo), se debe tener en cuenta que el periodo del reloj es de 10ns, esto al momento de obtener el tiempo de (echo) debe multiplicar el número de pulsos del reloj por el periodo de cada pulso 10 ns. El componente fue diseñado con tres señales de entrada *std_logic* las cuales son clk, enable, reset y para su salida fue declarada una señal del tipo *std_logic_vector* de 20 bits, este número de bits nos servirá para almacenar el número de pulsos que fueron contados y así transformar posteriormente a una unidad de tiempo y distancia se detalla en la figura 52.

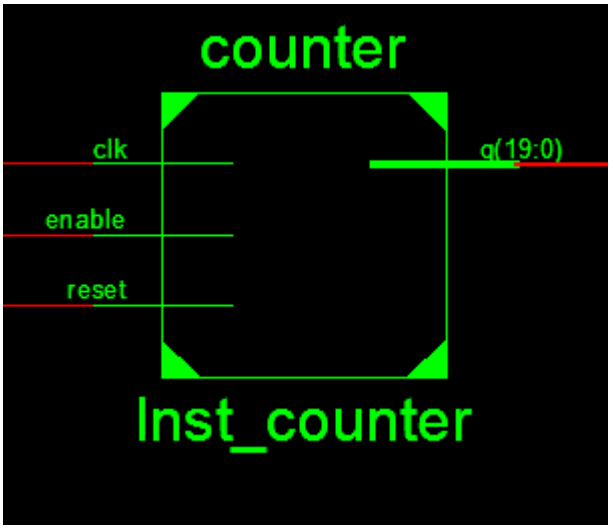


Figura 52. Esquema RTL del módulo counter.

Entradas:

- CLK: Reloj interno de 100 Mhz
- Enable: Encargado de recibir el pulso del pin echo
- Reset: Encargado de resetear la cuenta del contador por el pin trigger

Salidas:

- g: Encargada de almacenar el número de pulsos de reloj para determinar el tiempo del pin echo.

Los procesos internos transcurren de la siguiente manera:

El único proceso interno tiene la función de un flip-flop con señal de habilitación y reset asíncrono, con esto al recibir el cambio de estado lógico de '0' a '1' por el pin (echo) el flip-flop empieza su conteo mediante pulsos de 10ns. Luego de recibir el cambio de estado de '1' a '0' este conteo se detiene y se almacena en la señal de salida.

Bloque fd_1

En el módulo fd_1 tiene su función de registro de almacenamiento, el cual al detectar un flanco de bajada en el pin (echo) este va a almacenar el valor del counter hasta que se registre nuevamente un flanco de bajada en el pin (echo), El diseño consta de una señal de entrada del tipo *std_logic_vector* de 20 bits y una entrada *std_logic* para el pin (echo) que estará leyendo continuamente cuando se realice un flanco de bajada, para sus salidas se declaró una señal del tipo *std_logic_vector* la cual se encarga de trasmitir el valor al módulo de distancia ver figura 53.

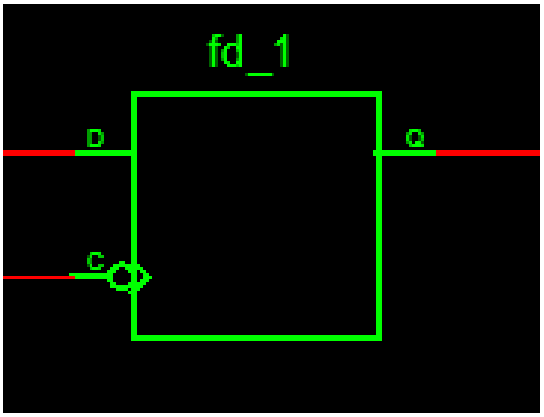


Figura 53. Esquema RTL del módulo fd_1

Entradas:

- D: Encargada de recibir el número de pulsos del counter.
- C: Encargada de recibir el pulso del pin (echo)

Salidas:

- Q: Encargada de trasmitir el número de pulsos del counter

El proceso interno transcurre de la siguiente manera:

Al detectar un flanco de bajada en el pin (echo) es decir detectar un cambio de estado lógico de '0' a '1' el registro transmitirá los datos de la señal (D) a la señal (Q) con esto evitamos que cada vez que (D) este aumentando su valor a medida que el counter realiza su conteo no haya ruido en la salida Q y no se vea afectada la pantalla LCD con estos cambios de valor.

Bloque distance_calculation:

En el módulo distance_calculation se lleva a cabo toda la lógica necesaria para determinar a qué distancia se encuentra el objeto, como también pueden definirse parámetros que sean necesarios en dicho modulo, como por ejemplo parámetros sobre rangos de medidas, o distancias prestablecidas. Para el diseño se declaró una señal del tipo ***std_logic_vector*** de 20 bits la cual recibe el número de pulsos del registro **fd_1** al momento de detectar un flanco de bajada, al obtener el número de pulsos debemos multiplicarlo por 10ns que es el periodo de cada pulso. Luego debemos dividir dicha cantidad entre 1000 esto se hace debido que la fórmula que suministra el fabricante del sensor HC-SR04 señala que el tiempo debe encontrarse en microsegundos (uS), Luego de hacer la conversión entre unidades de tiempo se divide entre 58, este valor es suministrado por el fabricante para obtener una unidad de distancia en centímetros.

Para su señal de salida se declaró una señal ***std_logic_vector*** de 4 bits la cual transmitirá al decodificador a que distancia se encuentra el objeto ver figura 55.

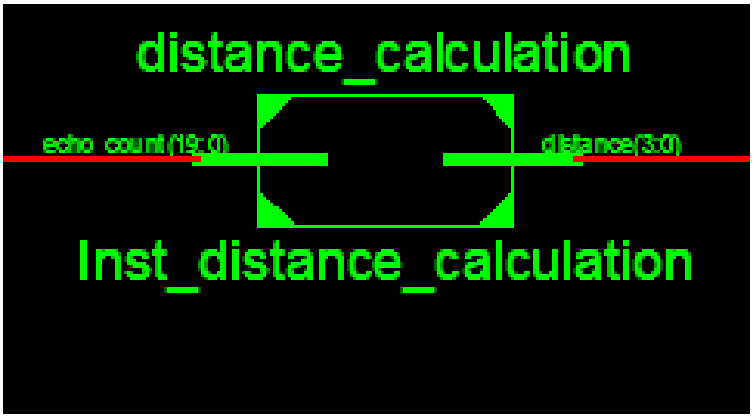


Figura 54. Esquema RTL del módulo distance_calculation.

Entradas:

- Echo_count: Encargada de recibir el número de pulsos del fd_1.

Salidas:

- distance: Encargada de trasmitir la distancia en centímetros.

La lógica interna transcurre de la siguiente manera:

Se declara un multiplexor esto para determinar de manera más simple la distancia a la que se encuentra el objeto, esto junto a comparados que tienen preestablecidos un número determinado de pulso a la cual corresponde una distancia en centímetros, ahorrando el tiempo de computo que requieren las operaciones aritméticas.

Bloque distance_decoder

En el módulo ***distance_decoder*** tiene su función de decodificar los datos transmitidos por el módulo de distancia, para así poder almacenar dicho valor y enviarlo en un formato que sea legible para la pantalla LCD de 16x2. Para su diseño se declaró una señal del tipo ***std_logic_vector*** de 4 bits la cual se encarga de recibir la distancia del objeto, para su salida se obtiene una señal ***std_logic_vector*** de 7 bits la cual es transmitida al módulo encargado del LCD de 16x2, ver figura 55.

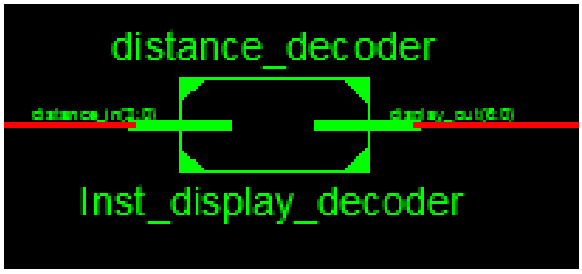


Figura 55. Esquema RTL del módulo distance_decoder.

Entradas:

- distance: Encargada de recibir la distancia que se encuentra el objeto en centímetros.

Salidas:

- display_out: Encargada de trasmitir la distancia en centímetros en un formato para la LCD de 16x2.

La lógica interna transcurre de la siguiente manera:

Consta de una memoria que tiene almacenada la distancia en un formato capaz de ser interpretada por la LCD, con esto al obtener la distancia del objeto mediante la señal (distance). Se usa como dirección en la misma y así obtener su equivalente en formato LCD.

Bloque TriggerGen

En el módulo TriggerGen se encuentra todo la lógica de control que se requiere para el pin (Trig) del sensor HC_SR04. Se encarga de transmitir un pulso de 10 us cada 20 ms, este tiempo de pulso es suministrado por el fabricante, este pulso de 10 us nos habilita cada 20 ms que el sensor sea capaz de enviar las ondas supersónicas de 40 kHz hacia el objeto. Para su diseño se declaró una señal del tipo ***std_logic*** para el (CLK) encargado de habilitar el pulso de 10 us, luego para su salida se declaró una señal del tipo ***std_logic*** (trigger) que va dirigida hacia el pin del sensor (Trig), ver figura 56.

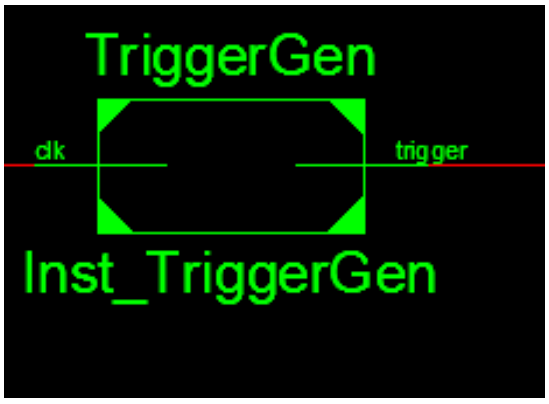


Figura 56. Esquema RTL del módulo TriggerGen

Entradas:

- clk: Encargada del reloj de 100 Mhz.

Salidas:

- trigger: Encargada de transmitir el pulso de 10 us al pin (Trig).

El proceso interno transcurre de la siguiente manera:

Dentro de la arquitectura del módulo, se declaran una ***signal*** del tipo **std_logic_vector** de 20 bits las cual estará encargada de controlar que el pulso emitido sea de 10 us. Esto por medio de un flip-flop y un contador que sea capaz de tener un periodo de 20 ms, luego se determina que la cantidad de pulsos requeridos por el flip-flop con un periodo de 10 ns se requieren 1000 pulsos que sería equivalente de 10us. Con esto se declara que los primeros mil pulsos la señal de salida (trigger) se encuentra en un estado lógico '1' al pasar esta cantidad de pulsos se mantendrá el estado lógico de la señal de salida (trigger) en '0'. Cumpliendo de esta manera el requerimiento exigido por el fabricante para la correcta manipulación del sensor.

4.3.3 Descripción del diseño (Control digital).

A continuación se comentaran de forma concreta las especificaciones y características que debe manejar el diseño.

El diseño consta de un servomotor de 9 g con un Angulo de giro máximo de 180 grados. Debe asegurarse con una fuente externa la alimentación por sus terminales VCD y GND, internamente su control será generado mediantes pulsos PWM con un periodo de 20 ms, los pulsos de dicha señal se encontraran en el rango de 1 ms a 2.5 ms que supondrán los 0 grados a 180 grados. Esto se tuvo en cuenta para el control PWM que será transmitida al servomotor. Para su control se contara con un componente anteriormente nombrado el PS/2 para el teclado e ingresar los grados al que se desea mover el servomotor.

Arquitectura del módulo de control digital

En este apartado se describe en detalle la jerarquía del sistema completo y su interfaz. El sistema completo engloba a todos los componentes que han sido necesarios para la realización del diseño, algunos bloques ya sea de forma parcial o total pueden ser utilizados en otros proyectos, ver figura 57.

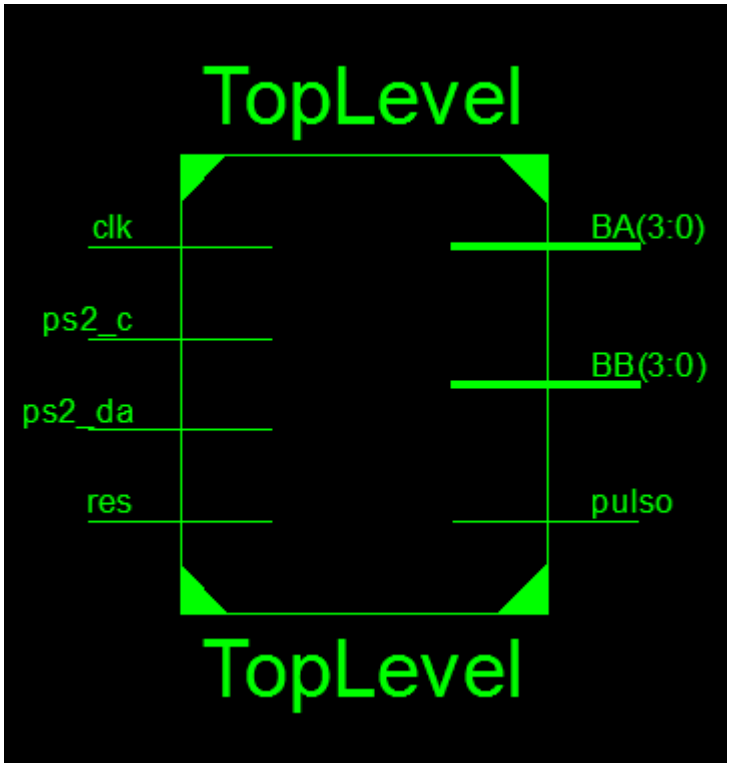


Figura 57. Esquema RTL del módulo de control digital.

Entradas:

- Clk: reloj interno de 100 MHz
- Ps2_c: Reloj externo transmitidos por el teclado para la captura de bits.
- Ps2_da: Puerto de datos transmitidos por el teclado.
- Res: Puerto para reset

Salidas:

- BA: bit de datos más significativos de LEDS de la placa
- BB: bit de datos menos significativos de LEDS de la placa
- Pulso: Trasmisor de la señal PWM que controla el servomotor.

En la figura 58 se muestra el esquemático RTL de bloques del sistema desarrollado para el control digital. Que está compuesto por cinco componentes incluyendo los compuesto por el componente PS/2 sobre el teclado que fue anteriormente utilizado

- Bloque PS/2: se encarga de escanear los datos enviados por el teclado y verificar que la tecla pulsada fue soltada.
- Bloque de registro: se encarga de almacenar la tecla presionada para posteriormente ser transmitida al bloque codificador.
- Bloque codificador: se obtiene toda la lógica para obtener el cambio de letra mayúscula a minúscula o verificar que fue presionada alguna tecla especial.
- Bloque decodificador: por último, se tiene el decodificador que se evalúa que tecla fue presionada mediante su código hexadecimal y se obtiene su formato equivalente.
- Bloque PWM: Encargado de toda la lógica que requiere el servomotor, internamente cuenta con un contador que sea capaz de determinar el tiempo del pulso PWM, como un decodificador capaz de traducir la tecla que fue presionada y mover el servomotor en los grados que le corresponde.

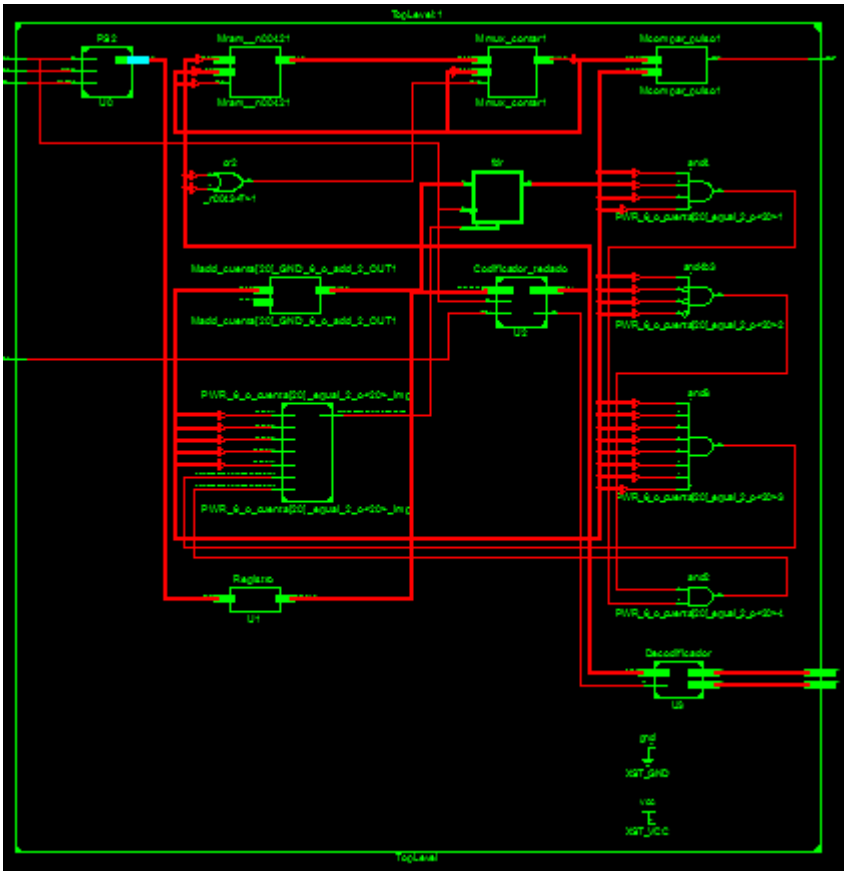


Figura 58. Esquema interno RTL del módulo de control digital.

Cabe destacar que como ya se ha mencionado en anteriores diseños el funcionamiento del componente PS/2 para el teclado, se enfocara en este apartado el componente para la generación de la señal PWM.

Bloque PWM

En el módulo PWM se encuentra toda la lógica para el control del servomotor, en donde fueron definidos los límites que dicho servomotor debe manejar, los cuales son señales con un periodo de 20 ms y pulsos lógicos positivos de 1 ms a 2.5 ms estos datos fueron consultados por la hoja de datos que es suministrada por el fabricante. Para su diseño fueron declaradas tres señales, las cuales se conforman por una señal del tipo ***intenger range (0 a 19999999)*** y otra señal del tipo ***intenger range (0 a 259999)*** que son capaces de mantener el límite que anteriormente fue mencionado para el periodo de 20 ms y pulsos positivos no mayores a 2.5 ms para su cálculo se realizó con la siguiente ecuación, para obtener el límite para un periodo de 20 ms:

$$Frec.salida = \frac{1}{50\ Hz} = 20\ ms$$

$$\frac{Frec.Tarjeta}{Frec.Salida} = \frac{100\ Mhz}{50\ Hz} = 2000000\ conteos$$

Para obtener el límite de pulsos de 2.5ms:

$$Frec.salida = \frac{1}{400\ Hz} = 2.5\ ms$$

$$\frac{Frec.Tarjeta}{Frec.Salida} = \frac{100\ Mhz}{400\ Hz} = 250000\ conteos$$

Y por último una señal del tipo ***std_logic_vector*** de 7 bits que se encarga de almacenar la tecla que fue presionada en el teclado, así internamente cuenta el diseño con un decodificador que define el pulso que será entregado al servomotor que ira de 1 ms a 2.5 ms respectivamente. Las teclas que pueden ser usadas para permitir el movimiento son los números del uno al nueve, el cual representa cada número 10 grados, ver figura 59.

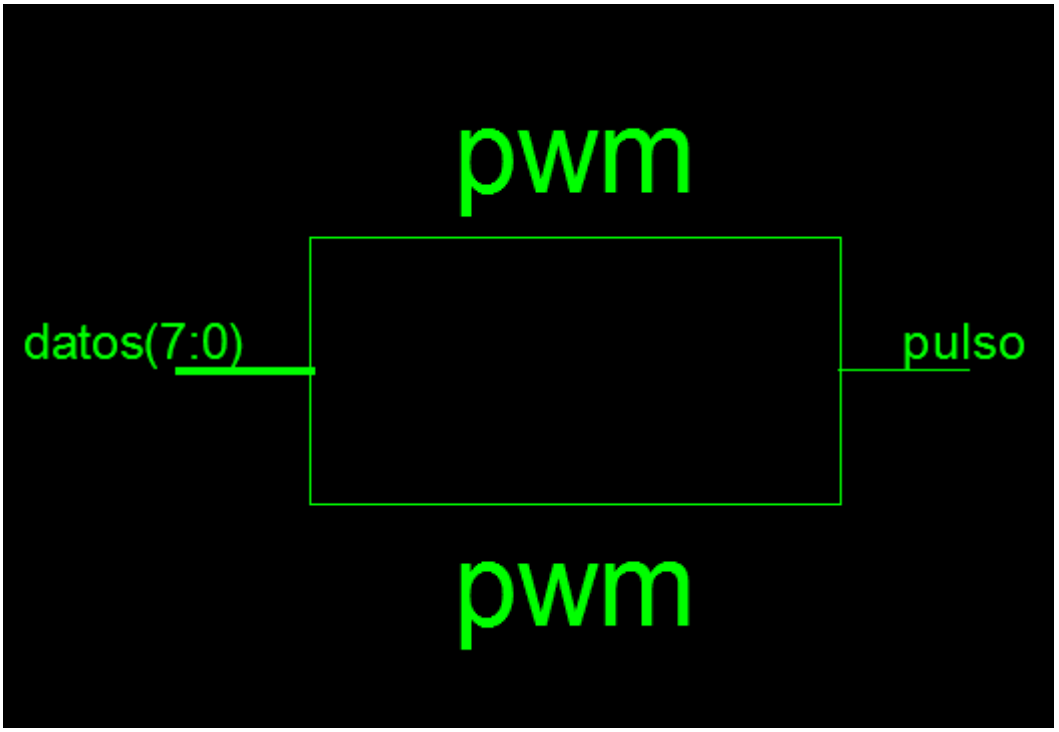


Figura 59. Esquema RTL del módulo PWM.

Entradas:

- datos: Encargada de almacenar los datos recibidos por el modulo PS/2.

Salidas:

- pulso: Encargada de transmitir el pulso de 1 ms a 2.5 ms al pin de señal del servomotor.

El proceso interno transcurre de la siguiente manera:

El proceso interno que dispone el módulo PWM se trata de un contador el cual está encargado de realizar cada uno de los conteos necesarios para entregar un periodo de 20 ms y pulsos de 1ms a 2.5 ms.

Simulación módulo de control digital.

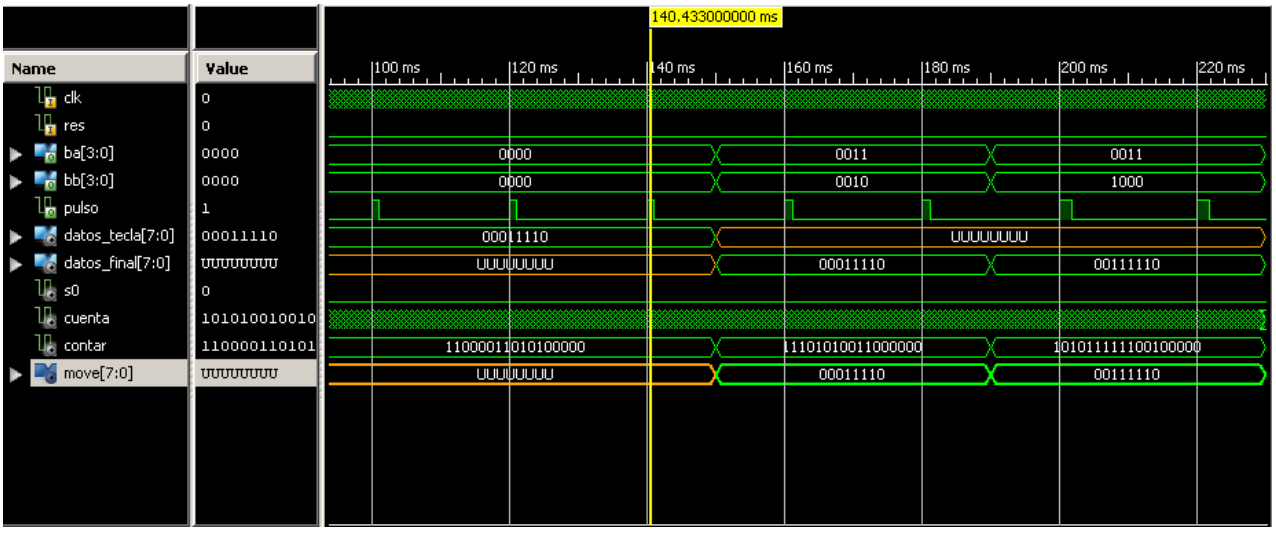


Figura 60. Simulación del funcionamiento del módulo digital.

4.3.4 Descripción del diseño (Comunicación serial I2C).

A continuación se comentaran de forma concreta las especificaciones y características que debe manejar el diseño.

Debe tenerse en cuenta la velocidad de transmisión, el protocolo I2C admite distintas velocidades de transmisión, que van desde los 100 Kbps hasta 3.4 Mbps. La velocidad de transmisión depende de los dispositivos conectados y de la longitud del bus. Es importante seleccionar una velocidad adecuada para garantizar una comunicación confiable, otro aspecto a considerar es las direcciones de los dispositivos, cada dispositivo en el bus I2C tiene una dirección única de 7 bits que se utiliza para identificarlo, al diseñar el modulo, es importante asegurarse de que su dirección no entre en conflicto con otras direcciones de dispositivos en el bus. Para el modo de operación que requiere emplear el protocolo de comunicación serial I2C admite dos modos de operación, que serán escritura y lectura.

La capacidades de transferencia juegan un papel fundamental al momento de la implementación de dicho diseño, donde el protocolo I2C admite dos tipos de transferencias de datos: bytes individuales y ráfagas de datos.

El nivel de voltaje para el protocolo I2C admite diferentes niveles de voltaje, incluidos 3.3 V y 5V. Es importante seleccionar el nivel de voltaje con las cuales trabajaran las resistencias Pull-Up y de que sean compatibles con los demás dispositivos.

Arquitectura del módulo de comunicación serial I2C.

En este apartado se describe en detalle la jerarquía del sistema completo y su interfaz. El sistema completo engloba a todos los componentes que han sido necesarios para la realización del diseño, algunos bloques ya sea de forma parcial o total pueden ser utilizados en otros proyectos ver figura 61.

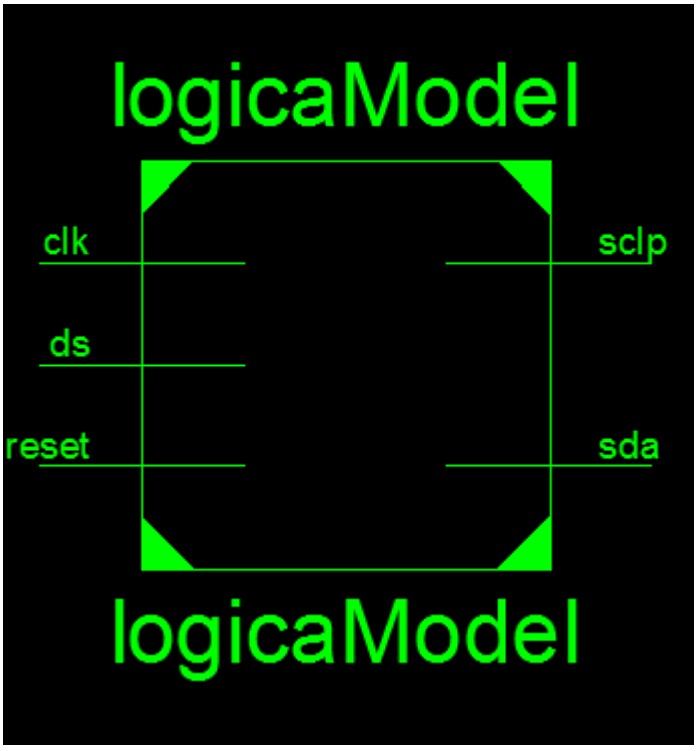


Figura 61. Esquema RTL del módulo I2C.

Entradas:

- Clk: reloj interno de 100 MHz
- ds: Bit de desplazamiento
- reset: Pin para reiniciar el sistema.

Salidas:

- sclp: Señal scl para la transmisión de datos.
- sda: Señal sda para la transmisión de datos.

En la figura 62 se muestra el esquemático RTL de bloques del sistema desarrollado para el módulo de comunicación serial I2C. Que está compuesto por cinco componentes incluyendo los cuales se describirán a continuación:

- inicioI2C: Se encarga de inicializar la comunicación del protocolo I2C la cual requiere que la señal (sda) cambie a un estado lógico '0' y luego de un tiempo de 4.7us la señal (scl) cambie a su estado lógico '0'. De esta manera el esclavo determinan que se ha iniciado una comunicación.
- StopI2C: Esta encargada de detener la comunicación del protocolo I2C la cual requiere que la señal (scl) cambie a un estado lógico '1' y luego de un tiempo de 4.7us la señal (sda) cambie a su estado lógico '1'. De esta manera el esclavo determina que la comunicación ha finalizado.
- SCL: Este módulo se encarga se controlar el periodo de la señal (scl).
- cambioI2C: Se encarga de realizar el cambio de dato en la señal sda, debido que este cambio debe realizarse cuando la señal (scl) se encuentra en un estado lógico '0'.
- contadorI2C: Se encarga de verificar el conteo de cada trama de datos, los cuales corresponden a los 7 bits de dirección, el bit de escritura/lectura y el bit ACK de confirmación.

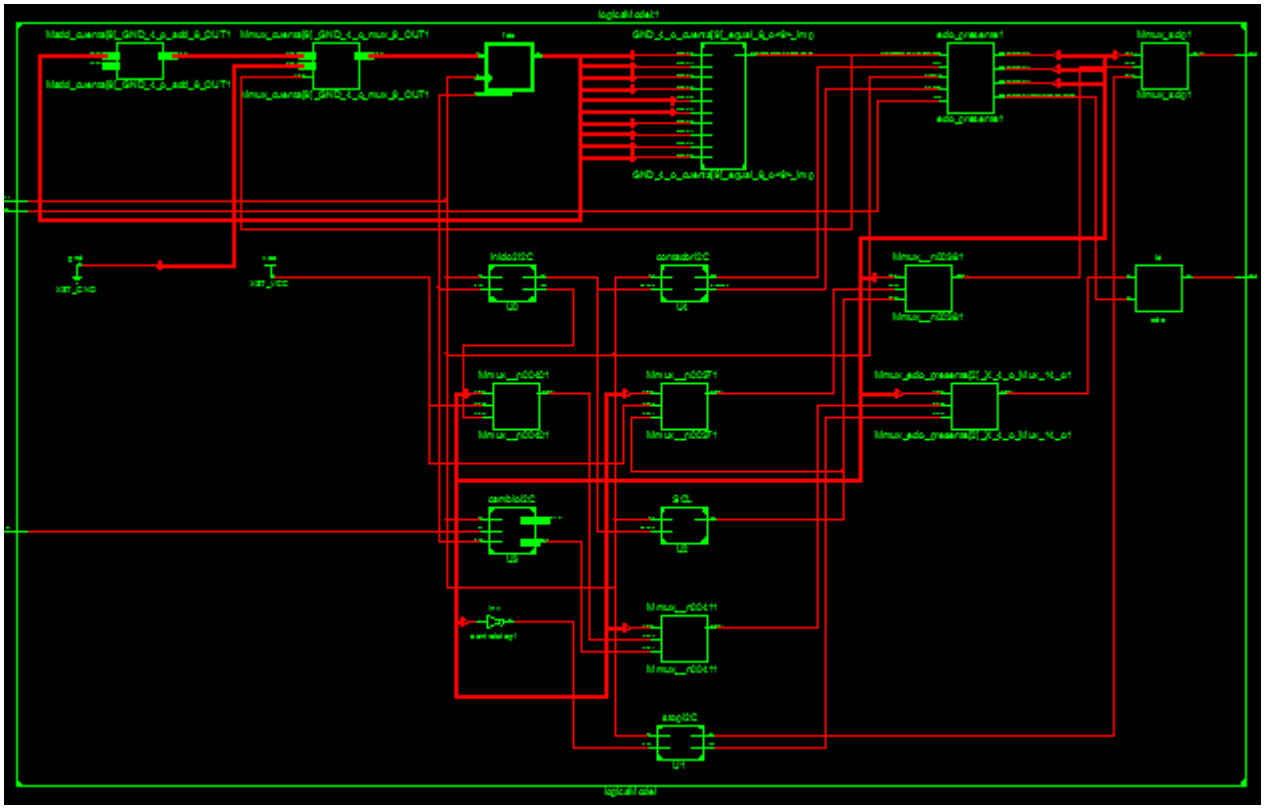


Figura 62. Esquema RTL de los componentes del módulo I2C.

Bloque inicioI2C

En el módulo inicio2I2C está definida la lógica de inicio que requiere modo de comunicación serial I2C, es decir, que la comunicación inicia cuando la señal sda tiene un cambio lógico de ‘1’ a ‘0’ y seguidamente luego de 4,7 us la señal de slc tiene un cambio lógico de ‘1’ a ‘0’. Luego de esto el esclavo detecta que está por iniciar una trasmisión de datos. Para su diseño fueron declaradas internamente cuatro señales las cuales constan de una señal de clk, una señal de reset y dos señales de salidas correspondiente a los pines de salida sda, slc, ver figura 63.

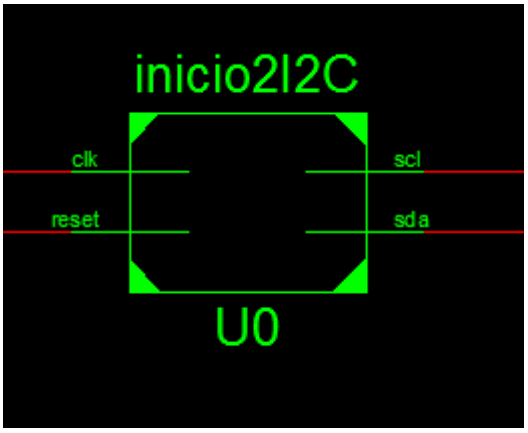


Figura 63. Esquema RTL del componente inicio2I2C.

Entradas:

- clk: encargado de obtener el reloj interno de la placa.
- Reset: obtiene la señal cuando se desea transmitir un dato

Salidas:

- sda: señal encargada de la transmisión de datos en el modo de comunicación I2C
- scl: señal encargada del trasmitir el reloj en el modo de comunicación I2C.

El proceso interno transcurre de la siguiente manera:

Consta de un circuito conformado por condicionales, las cuales son descritas mediantes flip-flops con señal de habilitación, al momento de iniciar la trasmisión de datos, la señal reset cambia a un estado lógico bajo '0' para dar inicio al proceso. Luego de trascurrido un tiempo la señal sda cambia a un estado bajo y así habilitar internamente un contador que será capaz de medir el tiempo de 4,7 us, así lograr cambiar en el tiempo específico la señal scl a un estado bajo.

Bloque de stopI2C.

A diferencia del bloque mencionado anteriormente, el siguiente bloque se encarga de detener la transmisión de datos. Esto se debe realizar para que los dispositivos esclavos logren procesar los datos que fueron enviados. Para que la transmisión de datos se detenga correctamente, debe realizar un cambio lógico de un estado bajo '0' a un estado alto '1' la señal scl, luego de transcurrido el tiempo de espera. La señal sda tendrá el mismo cambio lógico de '0' a '1'. Para el diseño interno del módulo mencionado, se describieron cuatro señal que estarán encargadas de controlar el comportamiento del mismo, estas señales son la señal sda y scl, las dos señales restantes son clk, reset en la figura 64 se muestra el esquema RTL del bloque.

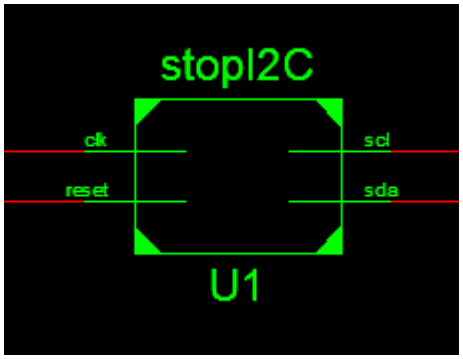


Figura 64. Esquema RTL del módulo Stop.

Entradas:

- clk: encargado de obtener el reloj interno de la placa.
- Reset: obtiene la señal cuando se desea transmitir un dato

Salidas:

- sda: señal encargada de la transmisión de datos en el modo de comunicación I2C
- scl: señal encargada del trasmitir el reloj en el modo de comunicación I2C.

El proceso interno transcurre de la siguiente manera:

Consta de un circuito conformado por condicionales, las cuales son descritas mediantes flip-flop con señal de habilitación, al momento de iniciar la trasmisión de datos, la señal reset cambia a un estado lógico bajo '0' para dar inicio al proceso. Luego de trascurrido un tiempo la señal scl cambia a un estado alto y así habilitar internamente un contador que será capaz de medir el tiempo de 4,7 us, así lograr cambiar en el tiempo específico la señal sda a un estado alto.

Bloque SCL.

Este bloque se encarga de controlar la frecuencia de la señal scl, debe recordarse que dicha señal puede variar de rango de frecuencia para realizar una transmisión de datos que se adapte a los dispositivos esclavos. El diseño consta internamente de flip-flop para realizar un divisor de frecuencia hasta obtener la frecuencia que es requerida por el esclavo, para su descripción fueron declaradas tres señales, clk, enable, scl a continuación se mostrara el esquema RTL en la figura 65.

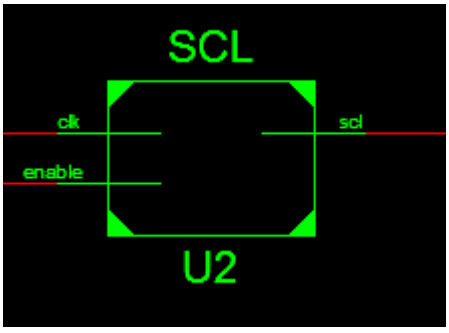


Figura 65. Esquema RTL del módulo SCL.

Entradas:

- clk: encargado de obtener el reloj interno de la placa.
- enable: señal de habilitación de reloj

Salidas:

- scl: señal encargada del trasmitir el reloj en el modo de comunicación I2C.

El proceso interno transcurre de la siguiente manera:

Consta de un contador que se encarga de almacenar el tiempo que ha transcurrido, luego de que el tiempo coincide con el tiempo del periodo deseado, este transmitirá la señal scl a la frecuencia que es requerida.

Bloque cambioI2C.

Este bloque se encarga de la transmisión de datos, cabe destacar que la transmisión de datos requiere que el cambio de un dato a otro deba ocurrir cuando la señal scl se encuentra en un nivel bajo, esto es para no ocasionar una interferencia en el esclavo o la lectura de un dato erróneo, ya que el estándar del I2C establece que la obtención del bit enviado debe realizarse siempre y cuando la señal scl se encuentre en un estado alto. Para su diseño interno se describe su funcionamiento mediante cinco señales, las cuales están definidas como clk, ds, reset, tiempo, data ver figura 66.

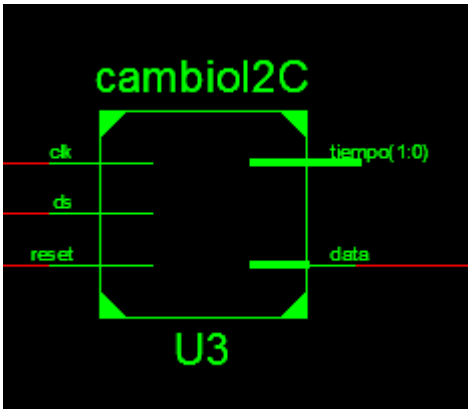


Figura 66. Esquema RTL del módulo cambioI2C

Entradas:

- clk: encargado de obtener el reloj interno de la placa.
- enable: señal de habilitación de reloj
- ds: señal encarga del dato a transmitir

Salidas:

- tiempo: señal encargada realizar el conteo de los datos trasmitido.
- Data: dato que es transmitido a la señal sda.

El proceso interno transcurre de la siguiente manera:

El diseño consta de un flip-flop con habilitación de señal, que es capaz de identificar si la señal scl se encuentra en un estado lógico bajo ‘0’, esto hará que el cambio se ejecute siempre y

cuando la señal scl este en un estado lógico ‘0’, que es lo que se espera para evitar errores futuros dentro de la transmisión de datos.

Bloque contadorI2C

Luego se tiene un bloque que se encarga de la señal ACK, la cual permite determinar que el esclavo ha recibido la información de una manera correcta, y si no es el caso será necesario reenviar la información. Para su diseño fue necesaria la descripción de cuatro señales que están definidas como clk, enable, ack, dirección, ver figura 67.

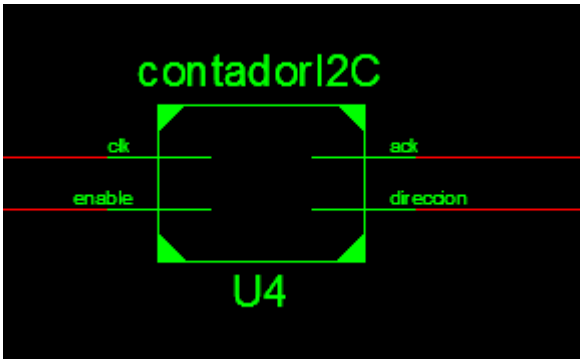


Figura 67. Esquema RTL del módulo contadorI2C

Entradas:

- clk: encargado de obtener el reloj interno de la placa.
- enable: señal de habilitación de reloj

Salidas:

- ack: señal encargada realizar el almacenamiento del ACK.
- Dirección: señal encargada de testear la dirección transmitida.

El proceso interno transcurre de la siguiente manera:

Consta de un contador que es capaz de determinar en qué momento debe obtenerse el dato que es suministrado por el esclavo. El cual está diseñado mediante flip-flop y la señal de habilitación scl la cual determina la frecuencia de transmisión y se realiza el conteo que es necesario en base a esta.

Simulación módulo de control digital con I2C.

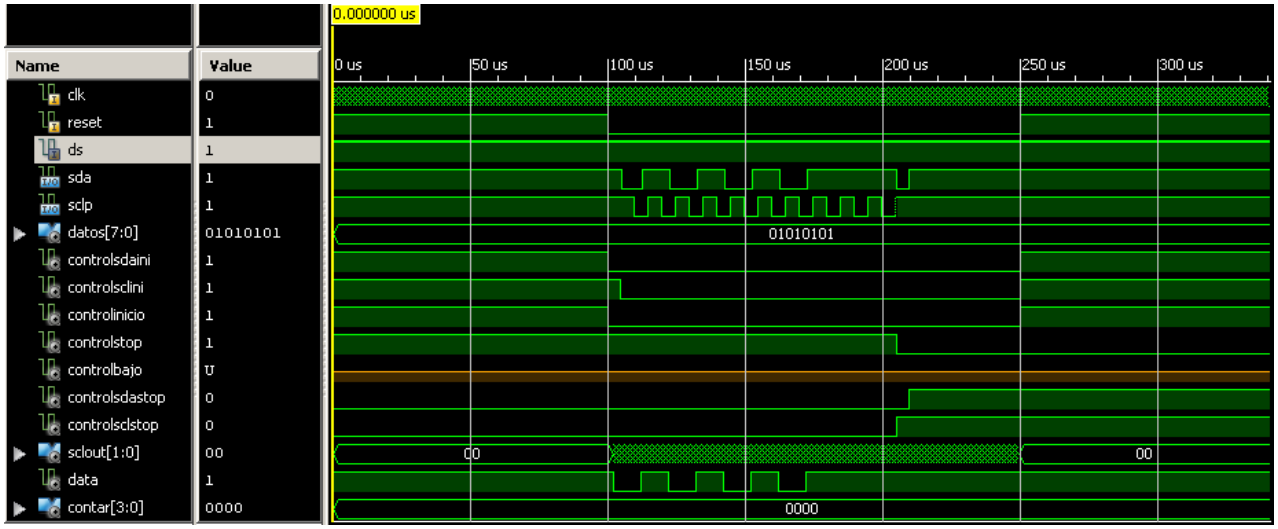


Figura 68. Esquema de la simulación del módulo I2C.

4.4 Elaboración de una guía de prácticas factibles basadas en el uso de FPGA para los estudiantes de la asignatura Microprocesadores de la UJAP.

4.4.1 Introducción

En esta guía práctica, exploraremos una serie de ejercicios que te ayudarán a familiarizarte con VHDL y desarrollar tus habilidades en la descripción y simulación de circuitos digitales. Los ejercicios están diseñados de manera progresiva, comenzando con conceptos básicos y avanzando hacia desafíos más complejos. Cada ejercicio irá acompañado de una descripción clara del problema a resolver, seguido de un enunciado detallado de los requisitos y especificaciones. Es importante tener en cuenta que esta guía no pretende ser exhaustiva, sino más bien un punto de partida para que puedas sumergirte en el mundo del diseño de hardware digital con VHDL. Una vez que hayas dominado los ejercicios presentados aquí, estarás preparado para enfrentar desafíos más avanzados y desarrollar tus propios proyectos.

4.4.2 Ejercicios propuestos

Entradas / Salidas digitales

Ejercicio 1. Generador de señal digital

Se pretende generar una señal digital S que tenga las siguientes características:

- Nivel de reposo: BAJO
- En el estado de reposo, si se activa la señal START durante al menos 1 ciclo de reloj, se generará un pulso de nivel ALTO de 10 us de duración. Al finalizar el pulso, se volverá al nivel BAJO y se esperará a que se active la señal G.
- Si se activa la señal G durante al menos un ciclo de reloj, se generará un pulso de nivel alto de 500 ns de duración. Al finalizar el pulso se volverá al estado de reposo.
- Las señales START y G sólo se tendrán en cuenta si se producen en el momento adecuado: la señal START sólo se atenderá en el estado de reposo y la señal G sólo se atenderá cuando se haya producido el pulso de START.

La frecuencia del reloj que debe manejar el hardware es de 10 Mhz, tendrá que declarar como entradas y salidas los siguientes pines.

Entradas:

- CLK: reloj del circuito, activo por flanco de bajada
- RESET: señal asíncrona que inicializa el circuito, activa por nivel bajo
- START: Activa por nivel alto.

- G: Activa por nivel alto

Salida:

- Señal generado por el circuito

Debe realizar la descripción de la entidad y la arquitectura en VHDL.

Ejercicio 2. Convertidor de binario a 7 segmentos.

En el siguiente ejercicio se pretende diseñar mediante un multiplexor de 4 bits de datos para que sea capaz de mostrar el dato seleccionado por los SWITCH de la tarjeta ATLYS de XILINX, muestre el dato en un DISPLAY de 7 segmentos.

El diseño debe tener las siguientes características:

- Debe ser capaz de suministrar los datos de entrada mediante los SWITCH que integra la tarjeta ATLYS de XILINX, los datos tienen que ser de 4 bits.
- Debe tener una entrada de RESET, que reinicie el sistema y en la salida del DISPLAY muestre un 0
- El valor ingresado por la entrada debe ser equivalente al valor que se muestre en el DISPLAY de 7 segmentos
- La declaración dentro de la arquitectura debe realizarse por sentencias concurrentes

Al momento del diseño debe tenerse en cuenta el funcionamiento del DISPLAY de 7 segmentos, es decir, si es un DISPLAY de 7 segmentos de ánodo común o cátodo común. De esta manera se evitara errores en pruebas finales.

A continuación se proporciona las entradas y salidas del diseño.

Entradas:

- Switch: señal de datos de 4 bits.
- RESET: señal asíncrona para el reinicio del sistema

Salidas:

- Segmentos: señal de datos de 7 bits

Comunicación digital

Ejercicio 3. Registro con desplazamiento serie/paralelo

En la figura 69 se muestra el esquema lógico de un registro de desplazamiento en serie/paralelo y salida serie. Realice un diseño en VHDL que realice la misma función.

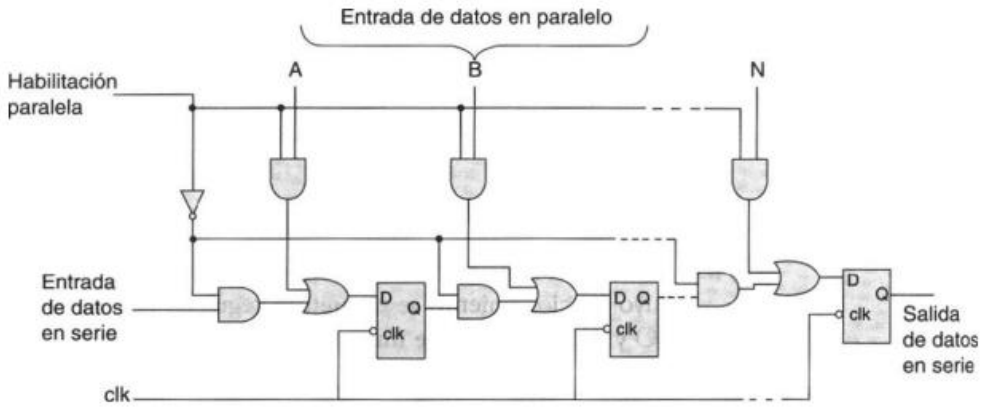


Figura 69. Esquema lógico de un registro de desplazamiento en serie/paralelo y salida serie

Ejercicio 4. Registro mediante compuertas lógicas

En la figura 70 se muestra el esquema lógico de un registro de desplazamiento con salida en paralelo. El circuito es representado mediante compuertas lógicas y flip-flop, de esta manera el estudiante deberá ejecutar un diseño con sentencia concurrente y sentencias secuenciales para los flip-flop del circuito lógico.

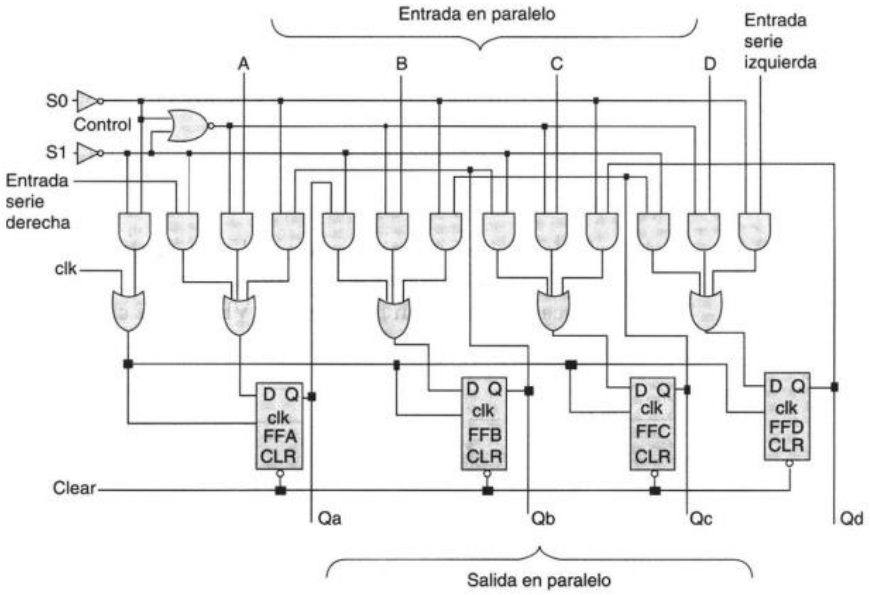


Figura 70. Esquema lógico de un registro de desplazamiento con salida en paralelo

Control digital

Ejercicio 5. Control luces traseras de un automóvil.

Diseñe un controlador digital que permita controlar las luces traseras de un automóvil. Hay tres luces en cada lado y operan en secuencia para mostrar la dirección en que se va a dar vuelta (Tabla I). La secuencia que se va a encender depende de la intención del conductor para dar vuelta a la izquierda o a la derecha a continuación se muestra la secuencia correspondiente.

Tabla I. Secuencia de luces, ejercicio 5

Luces izquierdas			***Luces derechas***		
L3	L2	L1	L3	L2	L1
0	0	0	0	0	0
0	0	1	1	0	0
0	1	1	1	1	0
1	1	1	1	1	1

Debe identificar las variables de entradas y salidas del controlador en conjunto con el diagrama RTL entregado por el Ise Design de Xilinx y un diagrama de flujos del mismo

Ejercicio 6. Trasmisor-Receptor de datos serie

Se debe diseñar un sistema de trasmisor-receptor de datos serie como el que se ilustra en la siguiente imagen. La descripción de cada uno de los elementos y señales correspondientes al trasmisor se muestra a continuación (figura 71):

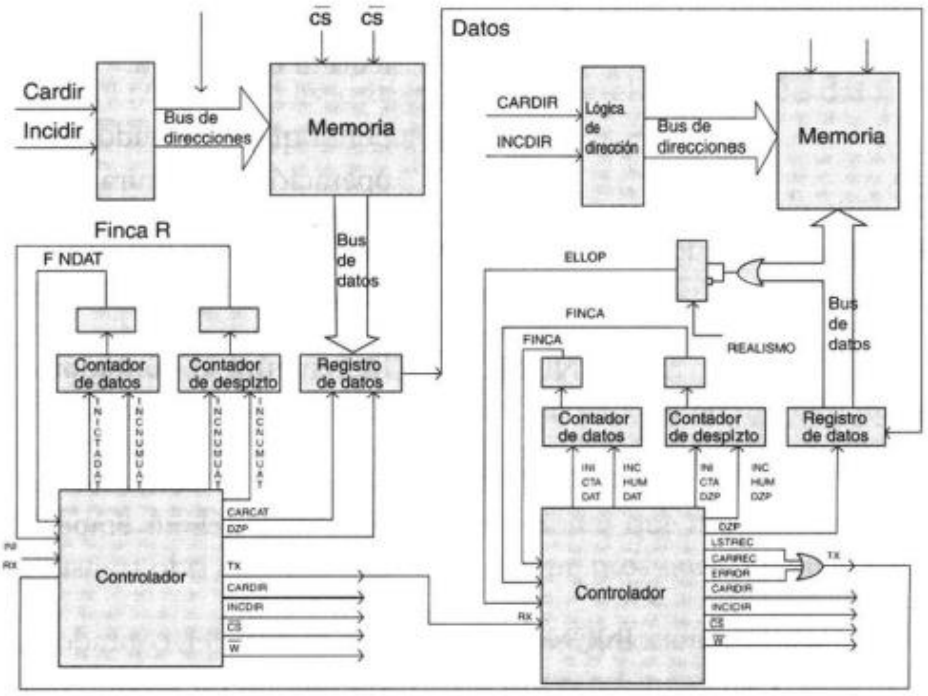


Figura 71. Transmisor-receptor de datos seriales

- Memoria: En este dispositivo se encuentran almacenados los datos que serán transmitidos.
- Lógica de dirección: Le indica a la memoria la localidad específica en dónde se encuentra almacenado el dato que se ha de transmitir.
- Contador de datos: Dispositivo que indica el número de datos que se han transmitido.
- Registro de datos: Dispositivo que almacena temporalmente el dato que se va a transmitir. El dato contenido en este registro se desplaza para realizar la transmisión en serie.
- Contador de desplazamiento: Le indica al controlador cuántos desplazamientos ha realizado el registro de datos.
- Controlador: Coordina las funciones que deberán de realizarse por cada uno de los elementos que forman el sistema.
- Cardir: Indica a la lógica de direccionamiento de memoria que debe cargar la dirección inicial a partir de la cual se encuentran almacenados los datos que van a ser transmitidos.
- Incdir: Señal de incremento de la dirección de memoria para leer el siguiente dato a transmitir.
- Cs: Señal de incremento de la dirección de memoria para leer el siguiente dato a transmitir.

- W: Señal que en estado alto indica a la memoria una operación de lectura.
- Inictada: Indica al contador de datos en qué momento deberá comenzar a contar el número de datos transmitidos.
- Incnumdat: Incrementa el contador de datos cada vez que un dato ha sido transmitido.
- Inictadzp: Le indica al contador de desplazamientos en qué momento deberá empezar a contar el número de bits que han sido transmitidos.
- Incnumdzp: Incrementa el contador de desplazamientos cada vez que un bit ha sido transmitido.
- Cardat: Le indica al registro de datos en que momento cargar el dato que se encuentra en el bus de datos de la memoria.
- Dzp: Señal que habilita al registro de datos para desplazar (trasmitir) el dato contenido en él.
- Fincar: Señal que indica cuando un dato ha sido trasmitido completamente.
- Findat: Señal que indica cuando un bloque de datos ha sido trasmitido.
- Ini: Señal que indica al sistema en qué momento se inicia la transmisión de datos.
- Tx: Línea a través de la cual viajan las señales de comunicación del transmisor hacia el receptor.
- Rx: Línea a través de la cual viajan las señales de comunicación del receptor hacia el transmisor.

La parte correspondiente al receptor se explica a continuación:

- Memoria: En este dispositivo se almacenarán los datos que se van a recibir.
- Lógica de direccionamiento: Le indica a la memoria la localidad específica en donde se almacenará el dato a recibir.
- Contador de datos: Dispositivo que indica el número de datos que se han recibido.
- Registro de datos: Dispositivo que almacena temporalmente el dato que se va a recibir. Este registro desplaza el bit que se encuentra en su entrada serie, para la realización de la recepción serie.
- Contador de desplazamiento: Le indica al controlador cuando un dato ha sido recibido por completo.
- Controlador: Coordina las funciones que deberán realizarse por cada uno de los elementos que forman al sistema.

- Cardir: Indica a la lógica de direccionamiento de memoria que debe cargar la dirección inicial a partir de la cual se almacenarán los datos que van a ser recibidos.
- Incdir: Señal de incremento de la dirección de memoria, para almacenar el nuevo dato recibido.
- Cs: Habilitación de la memoria para las operaciones de lectura o escritura.
- W: Señal que en estado bajo indica a la memoria una operación de escritura.
- Inictadat: Indica al contador de datos en que momento deberá comenzar a contar el número de datos recibidos.
- Incnumdat: Incrementa el contador de datos cada vez que un dato ha sido recibido.
- Inictadzp: Le indica al contador de desplazamientos en que momento deberá empezar contar el número de bits recibidos.
- Inicnumdzp: Incrementa el contador de desplazamiento cada vez que un bit ha sido recibido.
- Dzp: Señal que habilita al registro de datos para desplazar (recibir) el bit que se está transmitiendo.
- Fincar: Señal que indica cuando un dato ha sido recibido.
- Findat: Señal que indica cuando un bloque de datos ha sido recibido.
- Tx: Línea a través de la cual viajan las señales de comunicación del receptor al transmisor. Estas señales son:
- Lstrec: Listo a la recepción, indica que el sistema receptor está listo para recibir un dato.
- Carrec: Dato recibido, el sistema receptor indica que el dato que ha sido trasmitido fue recibido sin error y ha sido almacenado.
- Error: Esta señal se activa cada vez que ocurre un error de paridad cuando un dato ha sido recibido.
- Rx: Línea a través de la cual viajan las señales de comunicación del trasmisor hacia el receptor.
- Error: Señal que en estado alto indica al controlador que hay un error de paridad (par o impar).
- Paridad: Señal que indica el tipo de paridad que deberán tener los datos recibidos.

CAPÍTULO V

CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

Se puede concluir para el primer objetivo, que las capacidades que brinda la tarjeta ATLYS de XILINX ayudan a un correcto diseño, desarrollo, simulación e implementación del hardware que sea requerido, ofreciendo herramientas como Digilent Adept, el ISE Design Suite 14.7 que facilitan el desarrollo de hardware complejo y mejora la eficiencia del mismo.

Se realizó un estudio para demostrar las distintas capacidades que pueden obtenerse de dicho sistema, como el beneficio que obtendrá el estudiante al irse capacitando en esta tecnología. En donde se realizó una guía de estudio que servirán de apoyo para la introducción en la tarjeta ATLYS de XILINX y de las herramientas necesarias para el diseño de hardware, como también buenas practicas al momento de desarrollar su propio hardware.

En esta guía el estudiante dispondrá del manejo de herramientas para la depuración de hardware, síntesis y simulación. Donde se le explica las estructuras que son manejadas en VHDL, como lo son conceptos básicos como las máquinas de estado, flip-flop, contadores, registros, transmisión de datos y definición de las señales internas junto a ejemplos prácticos.

Se presentan diseños en la tarjeta ATLYS de XILINX, junto con el uso de VHDL, que permite desarrollar circuitos digitales para aplicaciones prácticas, como el manejo de entradas/salidas digitales, la comunicación de datos y el control digital. Estos diseños demuestran la versatilidad y el potencial de la tarjeta ATLYS para crear soluciones innovadoras en el laboratorio de microprocesadores.

Por último, A nivel estudiantil las FPGA son realmente un tema poco conocido y esto se ve reflejado en el conocimiento que se tienen hacer de ellas. Que a pesar de estar en un mundo globalizado y conectado, el interés que se muestra en Venezuela acerca de esta tecnología es nulo. Mientras universidades de alto prestigio capacitan y especializan a sus estudiantes en tecnologías del futuro, con este trabajo se buscar marcar un precedente del cual se espera la implementación del estudio de las FPGA en el laboratorio de microcontroladores.

5.2 Recomendaciones

Las recomendaciones para futuros trabajos a realizarse en el mismo campo que este, se pueden identificar en:

Realizar la adaptación de las guías en el lenguaje de descripción de hardware Verilog. De esta manera el estudiante contara con la capacidad de que lenguaje puede adaptarse mejor al diseño a implementar.

Proponer la apertura de una materia dedicada al Diseño de hardware con FPGA, para obtener un mayor aprovechamiento de los recursos y mayor capacitación a los estudiantes, debido al crecimiento que ha obtenido esta tecnología, La universidad José Antonio Páez marcaría un antes y después en Venezuela al momento de implementar dicha materia en su pensum.

REFERENCIAS BIBLIOGRÁFICAS

Arias, F. (2012). "**Proyecto de Investigación: Introducción a la Metodología**". Edición Nº 5. Caracas: Editorial Episteme.

Aguilucho, Alejandro (2018) Trabajo de grado "**Procesado de una señal de video para la detección de bordes en tiempo real sobre una FPGA".**

Balestrini, M. (2008). "**Cómo se elabora el Proyecto de Investigación**". Caracas. BL Consultores.

Bernal, C. (2006). Metodología de la Investigación. México, D.F., Pearson Educación.

Ruiz, Cecilia (2018) Trabajo de grado "**Modelado VHDL de control neuronal sobre tecnología FPGA orientado a Aplicaciones Sostenibles".**

Sabino, Carlos. (2010). "**El proceso de investigación**" Ed, Panamericana, Bogotá.

Damiani, L (2005). "**Epistemología y ciencia en la modernidad**" Edición de la biblioteca de la Universidad Central de Venezuela.

Maxinez, David y Alcala, Jessica. (2002). "**VHDL El arte de programar sistemas digitales**". 1era Edición.

Digilent (2018). **ATLYS FPGA Board Reference Manual.**

Pérez, Manuel (2017) Trabajo de grado "**Diseño de un microcontrolador de 32 bits con base en una FPGA".**

Martin, Caballero (2019) Trabajo de grado "**Desarrollo en FPGA de un enlace de comunicaciones serie para un convertidor de potencia distribuido**".

Morris, M y Charles R. (2005). Libro "**Fundamentos de diseño lógico y de computadoras**". 3era Edición.

Palella, S y Martins, F. (2012). Libro «**Metodología de la investigación cuantitativa**» 3ra Edición.

Caballero, Pedro. (2012). **Control visual con cámara FPGA basado en ViSP**. [En línea]. teralco.com/magazine/control-visual-con-camara-fpga-basado-en-visp/

Rusque, M. (2003). **De la diversidad a la unidad en la investigación cualitativa**. Caracas: Vadell Hermanos Editores

UPEL. (2016). **Manual de Trabajos de Grado, de Especialización y Maestría y Tesis Doctórales**. Caracas: Fondo Editorial de la Universidad Pedagógica Experimental Libertador.

APÉNDICE

Apéndice A

Código VHDL del Diseño de entradas y salidas Modulo Top-Level

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    19:24:09 04/05/2023
6   -- Design Name:
7   -- Module Name:    componentes_teclado - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  library UNISIM;
30  use UNISIM.VComponents.all;
31
32  entity componentes_teclado is
33  port(
34     clk: in std_logic;
35     ps2_da,ps2_c: in std_logic;
36     res: in std_logic;
37     BA, BB: out std_logic_vector(3 downto 0)
38  );
39  end componentes_teclado;
40
41  architecture Behavioral of componentes_teclado is
42  --Componentes inicio
43
44  component Decodificador port(
45     datos: in std_logic_vector(7 downto 0);
46     S0: in std_logic;
47     BU, BL: out std_logic_vector(3 downto 0)
48  );
49  end component;
50
51  component Codificador_teclado  port(
52     clk: in std_logic;
53     reset: in std_logic;
54     datos_tecla: in std_logic_vector(7 downto 0);
55     datos: out std_logic_vector(7 downto 0);
56     sel:out std_logic
57  );
```

```
58   end component;
59
60
61   component Registro port(
62      datos: in std_logic_vector(7 downto 0);
63      datos_tecla: inout std_logic_vector(7 downto 0)
64   );
65   end component;
66
67   component PS2 port(
68      ps2_data,ps2_clk,clk: in std_logic;
69      leds: out std_logic_vector(7 downto 0)
70   );
71   end component;
72
73
74   signal led: std_logic_vector(7 downto 0);
75   signal datos_tecla: std_logic_vector(7 downto 0);
76   signal datos_t: std_logic_vector(7 downto 0);
77   signal datos_final: std_logic_vector(7 downto 0);
78   signal S0: std_logic;
79
80   begin
81
82   U0: PS2 port map (ps2_da, ps2_c,clk, led);
83   U1: registro port map (led, datos_t);
84   U2: Codificador_teclado port map (clk , res, datos_t, datos_final, S0);
85   U3: Decodificador port map ( datos_final, S0, BA, BB);
86
87
88   end Behavioral;
89
90
```

Componente Decodificador

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    15:48:34 04/04/2023
6   -- Design Name:
7   -- Module Name:    Decodificador - teclado
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity Decodificador is
33  port(
34     datos: in std_logic_vector(7 downto 0);
35     S0: in std_logic;
36     BU, BL: out std_logic_vector(3 downto 0)
37  );
38  end Decodificador;
39
40  architecture teclado of Decodificador is
41  signal S:std_logic;
42
43  begin
44
45  deco: process(datos,S)begin
46  S <= S0;
47
48  case S is
49     when '1' => -- Mayúscula
50        if datos = "00011100" then -- A
51           BU <= "0100";
52           BL <= "0001";
53        elsif datos = "01011000" then -- mayus
54           BU <= "1111";
55           BL <= "1111";
56        elsif datos = "00110010" then -- B
57           BU <= "0100";
```

```
58            BL <= "0010";
59          elsif datos = "00100001" then -- C
60            BU <= "0100";
61            BL <= "0011";
62          elsif datos = "00100011" then -- D
63            BU <= "0100";
64            BL <= "0100";
65          elsif datos = "00100100" then -- E
66            BU <= "0100";
67            BL <= "0101";
68          elsif datos = "00101011" then -- F
69            BU <= "0100";
70            BL <= "0110";
71          elsif datos = "00110100" then -- G
72            BU <= "0100";
73            BL <= "0111";
74          elsif datos = "00110011" then -- H
75            BU <= "0100";
76            BL <= "1000";
77          elsif datos = "01000011" then -- I
78            BU <= "0100";
79            BL <= "1001";
80          elsif datos = "00111011" then -- J
81            BU <= "0100";
82            BL <= "1010";
83          elsif datos = "01000010" then -- K
84            BU <= "0100";
85            BL <= "1011";
86          elsif datos = "01001011" then -- L
87            BU <= "0100";
88            BL <= "1100";
89          elsif datos = "00111010" then -- M
90            BU <= "0100";
91            BL <= "1101";
92          elsif datos = "00110001" then -- N
93            BU <= "0100";
94            BL <= "1110";
95          elsif datos = "01000100" then -- O
96            BU <= "0100";
97            BL <= "1111";
98          elsif datos = "01001101" then -- P
99            BU <= "0101";
100           BL <= "0000";
101         elsif datos = "00010101" then -- Q
102           BU <= "0101";
103           BL <= "0001";
104         elsif datos = "00101101" then -- R
105           BU <= "0101";
106           BL <= "0010";
107         elsif datos = "00011011" then -- S
108           BU <= "0101";
109           BL <= "0011";
110         elsif datos = "00101100" then -- T
111           BU <= "0101";
112           BL <= "0100";
113         elsif datos = "00101100" then -- U
114           BU <= "0101";
```

```
115             BL <= "0101";
116         elsif datos = "00101100" then -- V
117             BU <= "0101";
118             BL <= "0110";
119         elsif datos = "00011101" then -- W
120             BU <= "0101";
121             BL <= "0111";
122         elsif datos = "00100010" then -- X
123             BU <= "0101";
124             BL <= "1000";
125         elsif datos = "00110101" then -- Y
126             BU <= "0101";
127             BL <= "1001";
128         elsif datos = "00011010" then -- Z
129             BU <= "0101";
130             BL <= "1010";
131         elsif datos = "01000101" then -- 0
132             BU <= "0011";
133             BL <= "0000";
134         elsif datos = "00010110" then -- 1
135             BU <= "0011";
136             BL <= "0001";
137         elsif datos = "00011110" then -- 2
138             BU <= "0011";
139             BL <= "0010";
140         elsif datos = "00100110" then -- 3
141             BU <= "0011";
142             BL <= "0011";
143         elsif datos = "00100101" then -- 4
144             BU <= "0011";
145             BL <= "0100";
146         elsif datos = "00101110" then -- 5
147             BU <= "0011";
148             BL <= "0101";
149         elsif datos = "00110110" then -- 6
150             BU <= "0011";
151             BL <= "0110";
152         elsif datos = "00111101" then -- 7
153             BU <= "0011";
154             BL <= "0111";
155         elsif datos = "00111110" then -- 8
156             BU <= "0011";
157             BL <= "1000";
158         elsif datos = "01000110" then -- 9
159             BU <= "0011";
160             BL <= "1001";
161         elsif datos = "01001100" then -- :
162             BU <= "0011";
163             BL <= "1010";
164         elsif datos = "01111111" then -- ; Tiene un erroneo para buscar como marcarla
165             BU <= "0011";
166             BL <= "1011";
167         elsif datos = "11111111" then -- <
168             BU <= "0011";
169             BL <= "1100";
170         elsif datos = "00111111" then -- =
171             BU <= "0011";
```

```
172             BL <= "1101";
173         elsif datos = "00111101" then -- >
174             BU <= "0011";
175             BL <= "1110";
176         elsif datos = "01111100" then -- ?
177             BU <= "0011";
178             BL <= "1111";
179         elsif datos = "01111001" then -- !
180             BU <= "0010";
181             BL <= "0001";
182         elsif datos = "00111010" then -- "
183             BU <= "0010";
184             BL <= "0010";
185         elsif datos = "00101001" then -- #
186             BU <= "0010";
187             BL <= "0011";
188         elsif datos = "00101111" then -- $
189             BU <= "0010";
190             BL <= "0100";
191         elsif datos = "00101100" then -- %
192             BU <= "0010";
193             BL <= "0101";
194         elsif datos = "00111011" then -- &
195             BU <= "0010";
196             BL <= "0110";
197         elsif datos = "00111100" then -- '
198             BU <= "0010";
199             BL <= "0111";
200         elsif datos = "00111011" then -- (
201             BU <= "0010";
202             BL <= "1000";
203         elsif datos = "01010011" then -- )
204             BU <= "0010";
205             BL <= "1001";
206         elsif datos = "00111111" then -- *
207             BU <= "0010";
208             BL <= "1010";
209         elsif datos = "01100010" then -- +
210             BU <= "0010";
211             BL <= "1011";
212         elsif datos = "01011011" then -- ,
213             BU <= "0010";
214             BL <= "1100";
215         elsif datos = "00111110" then -- -
216             BU <= "0010";
217             BL <= "1101";
218         elsif datos = "00110101" then -- .
219             BU <= "0010";
220             BL <= "1110";
221         elsif datos = "01011100" then -- /
222             BU <= "0010";
223             BL <= "1111";
224         else
225             BU <= "0000";
226             BL <= "0000";
227         end if;
228     when others =>
```

```
229         if datos = "00011100" then -- a
230            BU <= "0110";
231            BL <= "0001";
232         elsif datos = "00110010" then -- b
233            BU <= "0110";
234            BL <= "0010";
235         elsif datos = "00100001" then -- c
236            BU <= "0110";
237            BL <= "0011";
238         elsif datos = "00100011" then -- d
239            BU <= "0110";
240            BL <= "0100";
241         elsif datos = "00100100" then -- e
242            BU <= "0110";
243            BL <= "0101";
244         elsif datos = "00101011" then -- f
245            BU <= "0110";
246            BL <= "0110";
247         elsif datos = "00110100" then -- g
248            BU <= "0110";
249            BL <= "0111";
250         elsif datos = "00110011" then -- h
251            BU <= "0110";
252            BL <= "1000";
253         elsif datos = "01000011" then -- i
254            BU <= "0110";
255            BL <= "1001";
256         elsif datos = "00111011" then -- j
257            BU <= "0110";
258            BL <= "1010";
259         elsif datos = "01000010" then -- k
260            BU <= "0110";
261            BL <= "1011";
262         elsif datos = "01001011" then -- l
263            BU <= "0110";
264            BL <= "1100";
265         elsif datos = "00111010" then -- m
266            BU <= "0110";
267            BL <= "1101";
268         elsif datos = "00110001" then -- n
269            BU <= "0110";
270            BL <= "1110";
271         elsif datos = "01000100" then -- o
272            BU <= "0110";
273            BL <= "1111";
274         elsif datos = "01001101" then -- p
275            BU <= "0111";
276            BL <= "0000";
277         elsif datos = "00010101" then -- q
278            BU <= "0111";
279            BL <= "0001";
280         elsif datos = "00101101" then -- r
281            BU <= "0111";
282            BL <= "0010";
283         elsif datos = "00011011" then -- s
284            BU <= "0111";
285            BL <= "0011";
```

```
286         elsif datos = "00101100" then -- t
287            BU <= "0111";
288            BL <= "0100";
289         elsif datos = "00101100" then -- u
290            BU <= "0111";
291            BL <= "0101";
292         elsif datos = "00101100" then -- v
293            BU <= "0111";
294            BL <= "0110";
295         elsif datos = "00011101" then -- w
296            BU <= "0111";
297            BL <= "0111";
298         elsif datos = "00100010" then -- x
299            BU <= "0111";
300            BL <= "1000";
301         elsif datos = "00110101" then -- y
302            BU <= "0111";
303            BL <= "1001";
304         elsif datos = "00011010" then -- z
305            BU <= "0111";
306            BL <= "1010";
307         elsif datos = "01000101" then -- 0
308            BU <= "0011";
309            BL <= "0000";
310         elsif datos = "00010110" then -- 1
311            BU <= "0011";
312            BL <= "0001";
313         elsif datos = "00011110" then -- 2
314            BU <= "0011";
315            BL <= "0010";
316         elsif datos = "00100110" then -- 3
317            BU <= "0011";
318            BL <= "0011";
319         elsif datos = "00100101" then -- 4
320            BU <= "0011";
321            BL <= "0100";
322         elsif datos = "00101110" then -- 5
323            BU <= "0011";
324            BL <= "0101";
325         elsif datos = "00110110" then -- 6
326            BU <= "0011";
327            BL <= "0110";
328         elsif datos = "00111101" then -- 7
329            BU <= "0011";
330            BL <= "0111";
331         elsif datos = "00111110" then -- 8
332            BU <= "0011";
333            BL <= "1000";
334         elsif datos = "01000110" then -- 9
335            BU <= "0011";
336            BL <= "1001";
337         elsif datos = "01001100" then -- :
338            BU <= "0011";
339            BL <= "1010";
340         elsif datos = "01111111" then -- ; Tiene un erroneo para buscar como marcarla
341            BU <= "0011";
342            BL <= "1011";
```

```
343         elsif datos = "11111111" then -- <
344            BU <= "0011";
345            BL <= "1100";
346         elsif datos = "00111111" then -- =
347            BU <= "0011";
348            BL <= "1101";
349         elsif datos = "00111101" then -- >
350            BU <= "0011";
351            BL <= "1110";
352         elsif datos = "01111100" then -- ?
353            BU <= "0011";
354            BL <= "1111";
355         elsif datos = "01111001" then -- !
356            BU <= "0010";
357            BL <= "0001";
358         elsif datos = "00111010" then -- "
359            BU <= "0010";
360            BL <= "0010";
361         elsif datos = "00101001" then -- #
362            BU <= "0010";
363            BL <= "0011";
364         elsif datos = "00101111" then -- $
365            BU <= "0010";
366            BL <= "0100";
367         elsif datos = "00101100" then -- %
368            BU <= "0010";
369            BL <= "0101";
370         elsif datos = "00111011" then -- &
371            BU <= "0010";
372            BL <= "0110";
373         elsif datos = "00111100" then -- '
374            BU <= "0010";
375            BL <= "0111";
376         elsif datos = "00111011" then -- (
377            BU <= "0010";
378            BL <= "1000";
379         elsif datos = "01010011" then -- )
380            BU <= "0010";
381            BL <= "1001";
382         elsif datos = "00111111" then -- *
383            BU <= "0010";
384            BL <= "1010";
385         elsif datos = "01100010" then -- +
386            BU <= "0010";
387            BL <= "1011";
388         elsif datos = "01011011" then -- ,
389            BU <= "0010";
390            BL <= "1100";
391         elsif datos = "00111110" then -- -
392            BU <= "0010";
393            BL <= "1101";
394         elsif datos = "00110101" then -- .
395            BU <= "0010";
396            BL <= "1110";
397         elsif datos = "01011100" then -- /
398            BU <= "0010";
399            BL <= "1111";
```

```
400         else
401            BU <= "0000";
402            BL <= "0000";
403         end if;
404      end case;
405   end process deco;
406
407
408   end teclado;
409
410
```

Componente Codificador

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    14:17:48 04/02/2023
6   -- Design Name:
7   -- Module Name:    Codificador_teclado - teclado
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use ieee.std_logic_unsigned.ALL;
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  library UNISIM;
30  use UNISIM.VComponents.all;
31
32  entity Codificador_teclado is
33  port(
34     clk: in std_logic;
35     reset: in std_logic;
36     datos_tecla: in std_logic_vector(7 downto 0);
37     datos: out std_logic_vector(7 downto 0);
38     sel: out std_logic
39  );
40  end Codificador_teclado;
41
42  architecture teclado of Codificador_teclado is
43
44  type estados is (A,B,C,D,E);
45  signal edo_presente, edo_futuro: estados;
46
47  signal data: std_logic_vector(7 downto 0);
48  signal S : std_logic:= '0';
49  signal selec: std_logic;
50  begin
51
52  data <= datos_tecla;
53
54  with data select
55     selec <= '1' when "01011000",
56              '0' when others;
57
```

```
58   p_estados: process(edo_presente,S,selec)begin
59      case edo_presente is
60         when A =>
61            S <= '0';
62            if selec = '1' then
63            edo_futuro <= B;
64            else
65            edo_futuro <= A;
66            end if;
67         when B =>
68            S <= '1';
69            if selec = '0' then
70               edo_futuro <= C;
71            else
72               edo_futuro <= B;
73            end if;
74         when C =>
75            S <= '1';
76            if selec = '1' then
77               edo_futuro <= D;
78            else
79               edo_futuro <= C;
80            end if;
81         when D =>
82            S <= '0';
83            if selec = '0' then
84               edo_futuro <= E;
85            else
86               edo_futuro <= D;
87            end if;
88         when E =>
89            S <= '0';
90            if selec = '1' then
91               edo_futuro <= A;
92            else
93               edo_futuro <= E;
94            end if;
95      end case;
96   end process p_estados;
97
98   p_siguiente: process(clk)begin
99            if(clk'event and clk = '1')then
100              edo_presente <= edo_futuro;
101           end if;
102  end process p_siguiente;
103
104  final: process(S,datos_tecla,clk,reset)begin
105     if reset = '1' then
106        datos <= "00110010";
107        sel <= '1';
108     elsif( clk'event and clk = '1')then
109           datos <= datos_tecla;
110           sel <= S;
111     end if;
112  end process final;
113
114  end teclado;
```

Componente Registro

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    14:07:22 04/02/2023
6   -- Design Name:
7   -- Module Name:    Registro - registro_teclado
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity Registro is
33  port(
34     datos: in std_logic_vector(7 downto 0);
35     datos_tecla: inout std_logic_vector(7 downto 0)
36  );
37  end Registro;
38
39  architecture registro_teclado of Registro is
40
41  begin
42
43  registro : process(datos,datos_tecla)begin
44        datos_tecla <= datos;
45  end process registro;
46
47  end registro_teclado;
48
49
```

Componente PS2

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    14:15:06 12/14/2022
6   -- Design Name:
7   -- Module Name:    PS2 - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  -- Uncomment the following library declaration if using
26  -- arithmetic functions with Signed or Unsigned values
27  use IEEE.NUMERIC_STD.ALL;
28
29  -- Uncomment the following library declaration if instantiating
30  -- any Xilinx primitives in this code.
31  library UNISIM;
32  use UNISIM.VComponents.all;
33
34  entity PS2 is
35  port(
36     ps2_data,ps2_clk,clk: in std_logic;
37     leds: out std_logic_vector(7 downto 0)
38  );
39  end PS2;
40
41  architecture Behavioral of PS2 is
42  signal data_frame: std_logic_vector(21 downto 0);
43
44  begin
45
46
47
48  señal: process (ps2_clk,ps2_data,data_frame)
49  begin
50     if(ps2_clk'event and ps2_clk = '0')then
51           data_frame <= ps2_data & data_frame(21 downto 1);
52     end if;
53  end process señal;
54
55  tecla_presionada: process (data_frame,clk)
56  begin
57     if clk'event and clk = '1' then
```

```
58          if(data_frame(8 downto 1) = X"F0" ) then
59             leds <= data_frame(19 downto 12);
60          end if;
61       end if;
62    end process tecla_presionada;
63
64
65    end Behavioral;
66
67
```

Apéndice B

Código VHDL del diseño Comunicación digital, Componente Top-Level

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    14:29:38 04/16/2023
6   -- Design Name:
7   -- Module Name:    TopLevel - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity TopLevel is
33  Port ( clk : in  STD_LOGIC;
34             echo : in  STD_LOGIC;
35             Trigger : out  STD_LOGIC;
36             an0 : out std_logic;
37             display_out : out  STD_LOGIC_VECTOR (6 downto 0));
38  end TopLevel;
39
40  architecture Behavioral of TopLevel is
41   COMPONENT TriggerGen
42     PORT(
43        clk : IN std_logic;
44        trigger : OUT std_logic
45        );
46     END COMPONENT;
47
48  COMPONENT counter
49     PORT(
50        clk : IN std_logic;
51        reset : IN std_logic;
52        enable : IN std_logic;
53        q : OUT std_logic_vector(19 downto 0)
54        );
55     END COMPONENT;
56
57  COMPONENT distance_calculation
```

```
58      PORT(
59         echo_count : IN std_logic_vector(19 downto 0);
60         distance : OUT std_logic_vector(3 downto 0)
61         );
62      END COMPONENT;
63
64   COMPONENT distance_decoder
65      PORT(
66         distance_in : IN std_logic_vector(3 downto 0);
67         display_out : OUT std_logic_vector(6 downto 0)
68         );
69      END COMPONENT;
70
71   signal Trigger_out: std_logic;
72   signal echo_counter1 : STD_LOGIC_VECTOR (19 downto 0);
73   signal echo_count : STD_LOGIC_VECTOR (19 downto 0);
74   signal distance_bits : std_logic_vector(3 downto 0);
75
76   begin
77
78   Inst_TriggerGen: TriggerGen PORT MAP(
79         clk,
80         Trigger_out
81      );
82
83   Inst_counter: counter PORT MAP(
84         clk,
85         Trigger_out,
86         echo,
87         echo_counter1
88      );
89
90      process(echo) begin
91         if falling_edge(echo) then
92            echo_count <= echo_counter1;
93         end if;
94
95      end process;
96
97   Inst_distance_calculation: distance_calculation PORT MAP(
98         echo_count,
99         distance_bits
100     );
101
102  Inst_display_decoder: distance_decoder PORT MAP(
103        distance_bits,
104        display_out
105     );
106
107  Trigger <= Trigger_out;
108  an0 <= '1';
109  end Behavioral;
110
111
```

Componente TriggerGen

```
1  ----------------------------------------------------------------------------------
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:    14:20:40 04/16/2023
6  -- Design Name:
7  -- Module Name:    TriggerGen - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 -- Uncomment the following library declaration if using
24 -- arithmetic functions with Signed or Unsigned values
25 use IEEE.NUMERIC_STD.ALL;
26 use ieee.std_logic_unsigned.all;
27
28 -- Uncomment the following library declaration if instantiating
29 -- any Xilinx primitives in this code.
30 --library UNISIM;
31 --use UNISIM.VComponents.all;
32
33 entity TriggerGen is
34  generic (n: integer :=20);
35     Port ( clk : in  STD_LOGIC;
36            trigger : out  STD_LOGIC);
37 end TriggerGen;
38
39 architecture Behavioral of TriggerGen is
40  signal tick: std_logic_vector(n-1 downto 0) := (others =>'1');
41 constant nclks: integer := 1000000; --1000000; --it corresponds to 20ms
42 begin
43   process (clk) begin
44    if clk'event and clk = '1' then
45     if tick < nclks-1 then
46      tick <= tick + 1;
47     else
48      tick <= (others => '0');
49      end if;
50    end if;
51   end process;
52  trigger <= '1' when (tick < 1000) else '0';
53 end Behavioral;
54
55
```

Componente Counter

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    14:31:16 04/16/2023
6   -- Design Name:
7   -- Module Name:    counter - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use ieee.std_logic_unsigned.all;
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity counter is
33   generic (N: integer := 20);
34      Port ( clk : in  STD_LOGIC;
35             reset : in  STD_LOGIC;
36             enable : in  STD_LOGIC;
37             q : out  STD_LOGIC_VECTOR (N-1 downto 0));
38  end counter;
39
40  architecture Behavioral of counter is
41   signal tick: std_logic_vector(N-1 downto 0);
42
43  begin
44
45  process (reset, clk, enable) begin
46
47     if reset = '1' then
48        tick <= (others => '0');
49     elsif clk'event and clk = '1' then
50        if enable = '1' then
51           tick <= tick + 1;
52        end if;
53     end if;
54  end process;
55
56  q <= tick;
57  end Behavioral;
```

Componente Distance_calculation

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    14:37:19 04/16/2023
6   -- Design Name:
7   -- Module Name:    distance_calculation - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use ieee.std_logic_unsigned.all;
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity distance_calculation is
33   port(
34  echo_count : in STD_LOGIC_VECTOR (19 downto 0);
35  distance : out  STD_LOGIC_VECTOR (3 downto 0)
36  );
37  end distance_calculation;
38
39  architecture Behavioral of distance_calculation is
40
41  begin
42
43  Distance <="0000" when (echo_count < 2900) else --0cm
44             "0001" when (echo_count > 2900 and echo_count < 8700) else --3cm
45             "0010" when (echo_count > 8700 and echo_count < 14500) else --3-5cm
46             "0011" when (echo_count > 14500 and echo_count < 21750) else --5-7.5cm
47             "0100" when (echo_count > 21750  and echo_count < 27550) else --7.5-9.5cm
48             "0101" when (echo_count > 27550  and echo_count < 30450) else --9.5-10.5cm
49             "0110" when (echo_count > 30450 and echo_count < 33350) else --10.5-11.5cm
50             "0111" when (echo_count > 33350 and echo_count < 37700) else --11.5-13cm
51             "1000" when (echo_count > 37700 and echo_count < 40600) else --13-14cm
52             "1001" when (echo_count > 40600 and echo_count < 46400) else --14-16cm
53             "1010" when (echo_count > 46400 and echo_count < 49300) else --16-17cm
54             "1011" when (echo_count > 49300 and echo_count < 52200) else --17-18cm;
55             "1100" when (echo_count > 52200 and echo_count < 55100) else --18-19cm;
56             "1101" when (echo_count > 55100 and echo_count <58000) else --19-20cm;
57             "1110" when (echo_count > 115000 and echo_count < 116300) else --20-22cm;

58              "1111";--more than 22cm
59   end Behavioral;
60
61
```

Componente Distance_Decoder

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    14:44:48 04/16/2023
6   -- Design Name:
7   -- Module Name:    distance_decoder - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity distance_decoder is
33    Port ( distance_in : in  STD_LOGIC_VECTOR (3 downto 0);
34           display_out : out  STD_LOGIC_VECTOR (6 downto 0));
35  end distance_decoder;
36
37  architecture Behavioral of distance_decoder is
38  begin
39   display_out <="1000000" when distance_in = "0000" else
40                 "1111001" when distance_in = "0001" else
41                 "0100100" when distance_in = "0010" else
42                 "0110000" when distance_in = "0011" else
43                 "0011001" when distance_in = "0100" else
44                 "0010010" when distance_in = "0101" else
45                 "0000010" when distance_in = "0110" else
46                 "0111000" when distance_in = "0111" else
47                 "1110111" when distance_in = "1110" else
48                 "1010101" when distance_in = "1101" else
49                 "0000110";
50  end Behavioral;
51
52
```

Asignación de pines

```
1   NET "display_out<0>" LOC = U18 | IOSTANDARD=LVCMOS33; # Bank = 1, Pin name =
    IO_L52N_M1DQ15,       Sch name = LD0
2   NET "display_out<1>" LOC = M14 | IOSTANDARD=LVCMOS33; # Bank = 1, Pin name =
    IO_L53P,           Sch name = LD1
3   NET "display_out<2>" LOC = N14 | IOSTANDARD=LVCMOS33; # Bank = 1, Pin name =
    IO_L53N_VREF,      Sch name = LD2
4   NET "display_out<3>" LOC = L14 | IOSTANDARD=LVCMOS33; # Bank = 1, Pin name =
    IO_L61P,           Sch name = LD3
5   NET "display_out<4>" LOC = M13 | IOSTANDARD=LVCMOS33; # Bank = 1, Pin name =
    IO_L61N,           Sch name = LD4
6   NET "display_out<5>" LOC = D4  | IOSTANDARD=LVCMOS33; # Bank = 0, Pin name =
    IO_L1P_HSWAPEN_0,     Sch name = HSWAP/LD5
7   NET "display_out<6>" LOC = P16 | IOSTANDARD=LVCMOS33; # Bank = 1, Pin name =
    IO_L74N_DOUT_BUSY_1, Sch name = LD6
8   #NET "display_out<3>" LOC = N12 | IOSTANDARD=LVCMOS33; # Bank = 2, Pin name =
    IO_L13P_M1_2,        Sch name = M1/LD7
9
10  NET "clk"   LOC = "L15"; # Bank = 1, Pin name = IO_L42P_GCLK7_M1UDM, Type = GCLK, Sch
    name = GCLK
11
12  # PMOD Connector
13  NET "echo"  LOC = "T3"; # Bank = 2,  Pin name = IO_L62N_D6,     Sch name = JA-D0_N
14  NET "echo" CLOCK_DEDICATED_ROUTE = false;
15  NET "trigger"  LOC = "R3"; # Bank = 2,  Pin name = IO_L62P_D5,     Sch name = JA-D0_P
```

Apéndice C

Código VHDL del diseño Control Digital, Componente Top-Level

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    17:42:53 04/23/2023
6   -- Design Name:
7   -- Module Name:    TopLevel - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.NUMERIC_STD.ALL;
23  use ieee.std_logic_unsigned.all;
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  --use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity TopLevel is
34  port(
35     clk: in std_logic;
36     ps2_da,ps2_c: in std_logic;
37     res: in std_logic;
38     BA, BB: out std_logic_vector(3 downto 0);
39     pulso: out std_logic
40  );
41  end TopLevel;
42
43  architecture Behavioral of TopLevel is
44  --Componentes inicio
45
46  component Decodificador port(
47     datos: in std_logic_vector(7 downto 0);
48     S0: in std_logic;
49     BU, BL: out std_logic_vector(3 downto 0)
50  );
51  end component;
52
53  component Codificador_teclado  port(
54     clk: in std_logic;
55     reset: in std_logic;
56     datos_tecla: in std_logic_vector(7 downto 0);
57     datos: out std_logic_vector(7 downto 0);
```

```
58      sel:out std_logic
59    );
60    end component;
61
62
63    component Registro port(
64       datos: in std_logic_vector(7 downto 0);
65       datos_tecla: inout std_logic_vector(7 downto 0)
66    );
67    end component;
68
69    component PS2 port(
70       ps2_data,ps2_clk,clk: in std_logic;
71       leds: out std_logic_vector(7 downto 0)
72    );
73    end component;
74
75
76    signal led: std_logic_vector(7 downto 0);
77    signal datos_tecla: std_logic_vector(7 downto 0);
78    signal datos_t: std_logic_vector(7 downto 0);
79    signal datos_final: std_logic_vector(7 downto 0);
80    signal S0: std_logic;
81
82    signal cuenta: integer range 0 to 1999999 := 0;
83    signal contar: integer range 0 to 199999 := 0;
84    signal move: std_logic_vector(7 downto 0);
85
86    begin
87
88
89    U0: PS2 port map (ps2_da, ps2_c,clk, led);
90    U1: registro port map (led, datos_t);
91    U2: Codificador_teclado port map (clk , res, datos_t, datos_final, S0);
92    U3: Decodificador port map ( datos_final, S0, BA, BB);
93
94    move<=datos_final;
95
96    process (clk) begin
97       if clk'event and clk = '1' then
98        if cuenta  = 1999999 then
99         cuenta <= 0;
100       else
101        cuenta <= cuenta + 1;
102        end if;
103      end if;
104   end process;
105
106   contar <= 110000 when move = "00010110" else ---- 1
107             120000 when move = "00011110" else ---- 2
108             130000 when move = "00100110" else ---- 3
109             140000 when move = "00100101" else ---- 4
110             150000 when move = "00101110" else ---- 5
111             160000 when move = "00110110" else ---- 6
112             170000 when move = "00111101" else ---- 7
113             180000 when move = "00111110" else ---- 8
114             100000;
```

```
115
116   pulso <= '1' when cuenta < contar else '0';
117
118   end Behavioral;
119
120
```

Asignación de pines

```
1   NET "BA<0>" LOC = U18 | IOSTANDARD=LVCMOS33; # Bank = 1, Pin name =
    IO_L52N_M1DQ15,       Sch name = LD0
2   NET "BA<1>" LOC = M14 | IOSTANDARD=LVCMOS33; # Bank = 1, Pin name =
    IO_L53P,           Sch name = LD1
3   NET "BA<2>" LOC = N14 | IOSTANDARD=LVCMOS33; # Bank = 1, Pin name =
    IO_L53N_VREF,      Sch name = LD2
4   NET "BA<3>" LOC = L14 | IOSTANDARD=LVCMOS33; # Bank = 1, Pin name =
    IO_L61P,           Sch name = LD3
5   NET "BB<0>" LOC = M13 | IOSTANDARD=LVCMOS33; # Bank = 1, Pin name =
    IO_L61N,           Sch name = LD4
6   NET "BB<1>" LOC = D4  | IOSTANDARD=LVCMOS33; # Bank = 0, Pin name =
    IO_L1P_HSWAPEN_0,     Sch name = HSWAP/LD5
7   NET "BB<2>" LOC = P16 | IOSTANDARD=LVCMOS33; # Bank = 1, Pin name =
    IO_L74N_DOUT_BUSY_1, Sch name = LD6
8   NET "BB<3>" LOC = N12 | IOSTANDARD=LVCMOS33; # Bank = 2, Pin name =
    IO_L13P_M1_2,         Sch name = M1/LD7
9
10  ## onBoard SWITCHES
11  NET "res"  LOC = "A10" | IOSTANDARD=LVCMOS33; # Bank = 0, Pin name =
    IO_L37N_GCLK12,         Sch name = SW0
12
13
14  NET "clk"   LOC = "L15"; # Bank = 1, Pin name = IO_L42P_GCLK7_M1UDM, Type = GCLK, Sch
    name = GCLK
15
16  # keyboard
17  NET "ps2_c" LOC = P17  |  SLEW = SLOW |  DRIVE = 2  |  IOSTANDARD = LVCMOS33 | PULLUP;
18  NET "ps2_da" LOC = N15  |  SLEW = SLOW  |  DRIVE = 2  |  IOSTANDARD = LVCMOS33 |
    PULLUP;
19  NET "ps2_c" CLOCK_DEDICATED_ROUTE = false;
20
21
22
23  # PMOD Connector
24  NET "pulso"  LOC = "T3" |  IOSTANDARD=LVCMOS33; # Bank = 2,  Pin name =
    IO_L62N_D6,     Sch name = JA-D0_N
25  #NET "pulso"  LOC = "T3"; # Bank = 2,  Pin name = IO_L62N_D6,     Sch name = JA-D0_N
26
```

Apéndice D

Código VHDL del diseño del módulo I2C, componente Top-Level

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    02:13:30 01/01/2013
6   -- Design Name:
7   -- Module Name:    logicaModel - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity logicaModel is
33  port(
34     clk,reset,ds: in std_logic;
35     sda,sclp: inout std_logic
36
37  );
38  end logicaModel;
39
40  architecture Behavioral of logicaModel is
41
42  --Componente de cambio de estado
43
44  component cambioI2C port(
45     clk, reset,ds: in std_logic;
46     data: out std_logic;
47     tiempo: out std_logic_vector(1 downto 0)
48  );
49  end component;
50
51  -- componente de Inicio
52  component inicio2I2C port(
53     clk,reset: in std_logic;
54     scl,sda: out std_logic
55  );
56  end component;
57
```

```
58
59    -- componente de SCL
60    component SCL port(
61       clk,enable: in std_logic;
62       scl: out std_logic
63    );
64    end component;
65
66    -- componente de stop
67    component stopI2C port(
68       clk,reset: in std_logic;
69       scl,sda: out std_logic
70    );
71    end component;
72
73    --componente de contador
74    component contadorI2C port(
75       clk,enable: in std_logic;
76       ack,direccion: out std_logic
77    );
78    end component;
79
80    --componente de contadorBytes
81    component contadorBytes port(
82       clk,enable: in std_logic;
83       confirmacion: out std_logic_vector(1 downto 0)
84    );
85    end component;
86
87    component contadorInicio
88    port(
89       clk,enable: in std_logic;
90       start: out std_logic
91    );
92    end component;
93
94    --Señales
95    signal datos: std_logic_vector(17 downto 0) := "010101011111000110";
96    signal controlsdaini,controlsclini,controlinicio,controlstop,controlbajo: std_logic;
97    signal controlsdastop, controlsclstop: std_logic;
98    signal sclout: std_logic_vector(1 downto 0);
99    signal data:std_logic;
100   signal contar : std_logic_vector(3 downto 0) := "0000";
101   signal cuenta: integer range 0 to 999 := 0;
102   signal ackp,datoack: std_logic;
103   signal direccion: std_logic;
104   signal señalscl:std_logic;
105   signal confirmar: std_logic_vector(1 downto 0);
106   signal start: std_logic;
107
108   type estados is (libre,inicio,cambio,ack,stop);
109   signal edo_presente, edo_futuro: estados;
110   begin
111
112   U0: inicio2I2C port map (clk, controlinicio, controlsclini,controlsdaini);
113   U1: stopI2C port map (clk, controlstop, controlsclstop,controlsdastop);
114   U2: SCL port map (clk, controlsclini, señalscl);
```

```
115   U3: cambioI2C port map (clk,start,ds,data,sclout);
116   U4: contadorI2C port map (clk,controlsclini,ackp,direccion);
117   U5: contadorBytes port map (clk,controlsclini,confirmar);
118   U6: contadorInicio port map (clk,controlsclini,start);
119
120   controlinicio <= reset;
121
122
123   p_estados: process(clk,reset)begin
124      case edo_presente is
125         when libre =>
126            sclp <= '1';
127            sda <= '1';
128            controlstop <= '1';
129         if reset <= '0' then
130            edo_futuro <= inicio;
131         else
132            edo_futuro <= libre;
133         end if;
134         when inicio =>
135            sda <= controlsdaini;
136            sclp <= señalscl;
137            controlstop <= '1';
138            if start = '1' then
139               edo_futuro <= cambio;
140            else
141               edo_futuro <= inicio;
142            end if;
143         when cambio =>
144            sclp <= señalscl;
145            sda <= data;
146            controlstop <= '1';
147         if confirmar = "00" or confirmar = "01" then
148            edo_futuro <= ack;
149         else
150            edo_futuro <= cambio;
151         end if;
152         when ack =>
153            datoack <= sda;
154            sclp <= señalscl;
155            controlstop <= '1';
156         if confirmar = "01" then
157            edo_futuro <= stop;
158         elsif confirmar = "00" then
159            edo_futuro <= cambio;
160         else
161            edo_futuro <= ack;
162         end if;
163         when stop =>
164            controlstop <= '0';
165            sda <= controlsdastop;
166            sclp <= controlsclstop;
167            edo_futuro <= stop;
168      end case;
169
170
171      if reset = '1' then
```

```
172         cuenta <= 0;
173      elsif clk'event and clk = '1' then
174         if cuenta = 499 then
175            cuenta <= 0;
176            edo_presente <= edo_futuro;
177         else
178            cuenta <= cuenta + 1;
179         end if;
180      end if;
181   end process p_estados;
182
183
184   end Behavioral;
185
186
```

Componente CambioI2C

```
1  ----------------------------------------------------------------------------------
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:    11:58:24 05/04/2023
6  -- Design Name:
7  -- Module Name:    cambioI2C - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use ieee.std_logic_unsigned.all;
23
24 -- Uncomment the following library declaration if using
25 -- arithmetic functions with Signed or Unsigned values
26 --use IEEE.NUMERIC_STD.ALL;
27
28 -- Uncomment the following library declaration if instantiating
29 -- any Xilinx primitives in this code.
30 --library UNISIM;
31 --use UNISIM.VComponents.all;
32
33 entity cambioI2C is
34 port(
35    clk, reset,ds: in std_logic;
36    data: out std_logic;
37    tiempo: out std_logic_vector(1 downto 0)
38 );
39 end cambioI2C;
40
41 architecture Behavioral of cambioI2C is
42 signal contar: std_logic_vector(1 downto 0) := "00";
43 signal cuenta: integer range 0 to 249 := 0;
44 signal datos: std_logic_vector(17 downto 0) := "001010101000011110";
45
46 begin
47
48 process(clk,reset) is
49 begin
50    if reset = '0' then
51       contar <= "00";
52       cuenta <= 0;
53    elsif clk'event and clk = '1' then
54       if cuenta = 249 then
55          cuenta <= 0;
56          contar <= contar + 1;
57          if contar = "00" then

58             datos <= ds & datos(17 downto 1);
59          end if;
60       else
61          cuenta <= cuenta + 1;
62       end if;
63    end if;
64 end process;
65
66 tiempo <= contar;
67 data <= datos(0);
68
69 end Behavioral;
70
```

Componente InicioI2C

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    12:54:59 05/04/2023
6   -- Design Name:
7   -- Module Name:    inicio2I2C - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use ieee.std_logic_unsigned.all;
23
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  --use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity inicio2I2C is
34  port(
35     clk,reset: in std_logic;
36     scl,sda: out std_logic
37  );
38  end inicio2I2C;
39
40  architecture Behavioral of inicio2I2C is
41  signal control: std_logic;
42  signal controlsda: std_logic;
43  signal cuenta: integer range 0 to 469 := 0;
44  begin
45
46  process(clk,reset) is
47  begin
48     if reset = '1' then
49        control <= '1';
50        controlsda <= '1';
51        cuenta <= 0;
52     elsif clk'event and clk = '1' then
53        if cuenta = 496 then
54           control <= '0';
55        else
56           control <= '1';
57           controlsda <= '0';
```

```
58            cuenta <= cuenta + 1;
59          end if;
60       end if;
61   end process;
62
63
64   sda <= controlsda;
65   scl <= control;
66
67   end Behavioral;
68
69
```

Componente SCL

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    12:38:49 05/04/2023
6   -- Design Name:
7   -- Module Name:    SCL - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use ieee.std_logic_unsigned.all;
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity SCL is
33  port(
34     clk,enable: in std_logic;
35     scl: out std_logic
36  );
37  end SCL;
38
39  architecture Behavioral of SCL is
40  signal control: std_logic;
41  signal cuenta: integer range 0 to 999 := 0;
42
43  begin
44
45  process(clk,enable) is
46  begin
47     if enable = '1' then
48        control <= '1';
49        cuenta <= 0;
50     elsif clk'event and clk = '1' then
51        if cuenta = 499 then
52           cuenta <= 0;
53           control <= not control;
54        else
55           cuenta <= cuenta + 1;
56        end if;
57     end if;

58   end process;
59
60   scl <= control;
61   end Behavioral;
62
63
```

Componente StopI2C

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    13:52:47 05/04/2023
6   -- Design Name:
7   -- Module Name:    stopI2C - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use ieee.std_logic_unsigned.all;
23
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  --use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity stopI2C is
34  port(
35     clk,reset: in std_logic;
36     scl,sda: out std_logic
37  );
38  end stopI2C;
39
40  architecture Behavioral of stopI2C is
41  signal control: std_logic;
42  signal controlsda: std_logic;
43  signal cuenta: integer range 0 to 949 := 0;
44  begin
45
46  process(clk,reset) is
47  begin
48     if reset = '1' then
49        control <= '0';
50        controlsda <= '0';
51        cuenta <= 0;
52     elsif clk'event and clk = '1' then
53        if cuenta = 949 then
54           controlsda <= '1';
55           control <= '1';
56        elsif cuenta >= 469 then
57           control <= '1';
```

```
58            controlsda <= '0';
59            cuenta <= cuenta + 1;
60         else
61            control <= '0';
62            controlsda <= '0';
63            cuenta <= cuenta + 1;
64         end if;
65      end if;
66   end process;
67
68   sda <= controlsda;
69   scl <= control;
70
71   end Behavioral;
72
```

Componente ContadorI2C

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    21:32:43 05/04/2023
6   -- Design Name:
7   -- Module Name:    contadorI2C - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use ieee.std_logic_unsigned.all;
23
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  --use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity contadorI2C is
34  port(
35     clk,enable: in std_logic;
36     ack,direccion: out std_logic
37  );
38  end contadorI2C;
39
40  architecture Behavioral of contadorI2C is
41
42  signal control: std_logic_vector(3 downto 0);
43  signal cuenta: integer range 0 to 999 := 0;
44  signal validacion:std_logic:= '0';
45  signal bits: std_logic:='0';
46
47  begin
48
49  process(clk,enable) is
50  begin
51     if enable = '1' then
52        control <= "0000";
53        cuenta <= 0;
54     elsif clk'event and clk = '1' then
55        if cuenta = 999 then
56           cuenta <= 0;
57           control <= control + 1;
```

```
58            if control = "1000" then
59               bits <= '1';
60            elsif control  = "1001" then
61               validacion <= '1';
62               bits <= '0';
63            elsif control = "1010" then
64               bits <= '0';
65               validacion <= '0';
66               control <= "0000";
67            else
68               bits <= '0';
69               validacion <= '0';
70            end if;
71         else
72            cuenta <= cuenta + 1;
73         end if;
74      end if;
75   end process;
76
77
78   direccion <= bits;
79   ack <= validacion;
80   end Behavioral;
81
82
```

Componente ContadorBytes

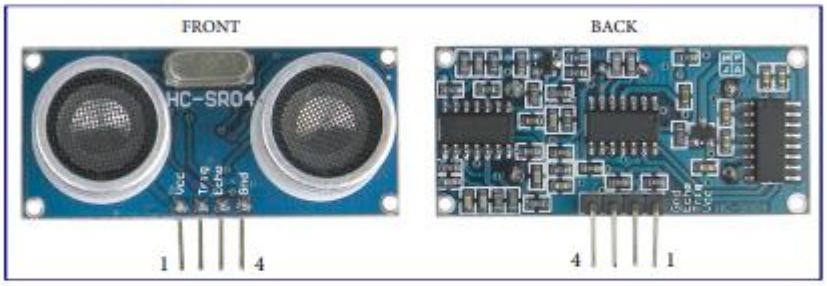
```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    05:39:12 05/20/2023
6   -- Design Name:
7   -- Module Name:    contadorBytes - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use ieee.std_logic_unsigned.all;
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity contadorBytes is
33  port(
34     clk,enable: in std_logic;
35     confirmacion: out std_logic_vector(1 downto 0)
36  );
37  end contadorBytes;
38
39  architecture Behavioral of contadorBytes is
40
41  signal control: std_logic_vector(4 downto 0);
42  signal cuenta: integer range 0 to 999 := 0;
43  signal validacion:std_logic_vector(1 downto 0):= "11";
44
45  begin
46
47  process(clk,enable) is
48  begin
49     if enable = '1' then
50        control <= "00000";
51        cuenta <= 0;
52     elsif clk'event and clk = '1' then
53        if cuenta = 999 then
54           cuenta <= 0;
55           control <= control + 1;
56           if control  = "01000" then
57              validacion <= "00";
```

```
58            elsif control = "10001" then
59               validacion <= "01";
60            else
61               validacion <= "11";
62            end if;
63         else
64            cuenta <= cuenta + 1;
65         end if;
66      end if;
67   end process;
68
69   confirmacion <= validacion;
70   end Behavioral;
71
72
```

ANEXOS

Anexo A. Hoja de datos del sensor HC-SR04

3. Product Views



4. Module Pin Asignments

	Pin Symbol	Pin Function Description
1	VCC	5V power supply
2	Trig	Trigger Input pin
3	Echo	Receiver Output pin
4	GND	Power ground

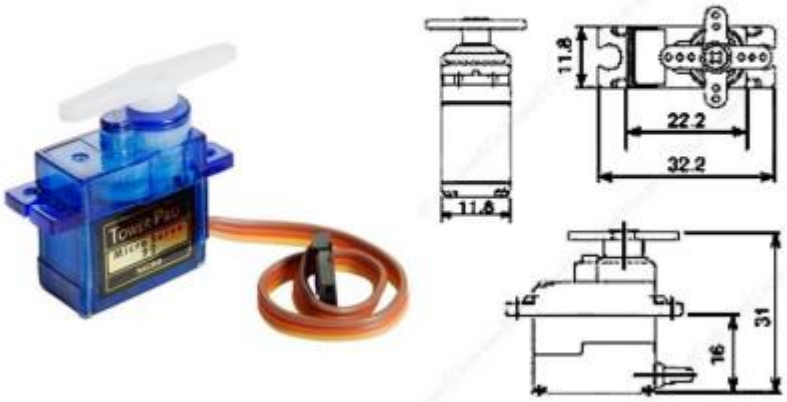
5. Electrical Specifications

WARARNING
Do Not connect Module with Power Applied! Always apply power after connecting Connect "GND" Terminal first

Electrical Parameters	HC-SR04 Ultrasonic Module
Operating Voltage	5VDC
Operating Current	15mA
Operating Frequency	40KHz
Max. Range	4m
Nearest Range	2cm
Measuring Angle	15 Degrees
Input Trigger Signal	10us min. TTL pulse
Output Echo Signal	TTL level signal, proportional to distance
Board Dimensions	1-13/16" X 13/16" X 5/8"
Board Connections	4 X 0.1" Pitch Right Angle Header Pins

Anexo B. Hoja de datos del servomotor

SG90 9 g Micro Servo



Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but *smaller*. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.

Specifications

- Weight: 9 g
- Dimension: 22.2 x 11.8 x 31 mm approx.
- Stall torque: 1.8 kgf·cm
- Operating speed: 0.1 s/60 degree
- Operating voltage: 4.8 V (~5V)
- Dead band width: 10 µs
- Temperature range: 0 ºC – 55 ºC

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is all the way to the left. ms pulse) is all the way to the right, ""-90" (~1ms pulse) is all the way to the left.

Anexo C. Hoja de datos de la pantalla LCD 16x2

Crystalfontz America, Inc.
15611 East Washington Road
Valleyford, WA 99036

Phone: (509) 291-3514
Fax: (509) 291-3345

http://www.crystalfontz.com
email: sales@crystalfontz.com

7.Interface Pin Function

Pin No.	Symbol	Level	Description
1	V_{SS}	0V	Ground
2	V_{DD}	5.0V	Supply Voltage for logic
3	VO	(Variable)	Operating voltage for LCD
4	RS	H/L	H: DATA, L: Instruction code
5	R/W	H/L	H: Read(MPU→Module) L: Write(MPU→Module)
6	E	H,H→L	Chip enable signal
7	DB0	H/L	Data bit 0
8	DB1	H/L	Data bit 1
9	DB2	H/L	Data bit 2
10	DB3	H/L	Data bit 3
11	DB4	H/L	Data bit 4
12	DB5	H/L	Data bit 5
13	DB6	H/L	Data bit 6
14	DB7	H/L	Data bit 7
15	A	—	LED +
16	K	—	LED—

Anexo D. Encuestas

Entrevista estructurada a especialistas

Primer entrevistado

Entrevistador: Autor

Entrevistado: Wendy García, Ing. Electricista MSc en instrumentación.

Fecha: 10.06.2023

1. En su opinión ¿Cuáles son los tipos de FPGA necesarias en el sector industrial?

Respuesta, Se requieren FPGAs robustos, capaces de operar en condiciones de temperaturas variables y extremas (altas y bajas) que soporten caídas, humedad, etc.

2. Desde su experiencia en el ramo industrial ¿Cuáles son las características más comunes de las FPGA en la industria?

Respuesta, Alta velocidad en procesamiento, carga y descarga de información, bajo consumo, flexibilidad para cambio en los diseños.

3. Las FPGA se encuentran comúnmente utilizadas en el DSP; Podría describir usted ¿Qué beneficios otorgan al sector industrial las FPGA en el procesamiento digital de señales?

Respuesta, Permiten aumentar la velocidad de procesamiento, por lo que aumenta el rendimiento, a nivel industrial, esto es muy útil en aplicaciones que involucren el movimiento, el control de motores, entre otras.

4. Describa desde su experiencia, ¿Cuáles son las estrategias que utiliza cuando se encuentra ante trabajos con FPGA y diseño de hardware?

Respuesta, Se aplica diseño Top-Down:

Se parte de los requerimientos

Se realiza el diagrama general de entradas y salidas del sistema que se va a diseñar,

Se realiza la descomposición funcional y una descripción en diagrama de bloques

Posteriormente, se realiza un diagrama de tiempos, en el cual se describe el sistema, en este caso en un lenguaje de descripción de hardware de los bloques internos del sistema y, por ende, del sistema total.

Se verifica el funcionamiento del mismo

5. En su opinión, ¿Cuál es la mejora en cuando a diseño que se obtiene con una descripción estructural en VHDL?

Respuesta, Permite que más de una persona trabaje en el mismo proyecto, unificando al final los resultados de cada uno. Flexibiliza el cambio en los diseños, sin que esto involucre cambios drásticos de hardware.

6. Desde el punto de vista de diseñador de hardware, ¿Podría usted enumerar las variables a considerar para ejecutar un correcto diseño de hardware con las herramientas de diseño?

Respuesta, Variables para caracterización Vectorial del sistema (Declaración de los vectores del módulo o subsistema, entradas y salidas, para la entidad VHDL. Variables para la estructura interna o comportamiento del módulo o del sistema, para la arquitectura VHDL.

Segundo entrevistado

Entrevistador: Autor

Entrevistado: Carlos Blanchard, Ing. Electrónico

Fecha: 10.06.2023

1. En su opinión ¿Cuáles son los tipos de FPGA necesarias en el sector industrial?

Respuesta, En las industrias Tradicionales como lo son la gran mayoría de las industrias venezolanas, se requieren FPGAS pequeños, compactos y relativamente sencillos. En industrias más modernas y automatizadas (con mayor cantidad de robots) si encajan perfectamente los FPGA más complejos, ya que los robots siguen siendo hoy en día, sistemas en constante evolución, y con esos FPGA eso se pueden perfectamente lograr.

2. Desde su experiencia en el ramo industrial ¿Cuáles son las características más comunes de las FPGA en la industria?

Respuesta, En la industria he encontrado FPGAS como decodificadores de direcciones de sistemas micro programados, como controladores de buses en sistemas de alarma contra incendio, incluso en computadores de vehículos (ECU) y todos eran relativamente sencillos, con pocos pines, y trabajaban a 5 V.

3. Las FPGA se encuentran comúnmente utilizadas en el DSP; Podría describir usted ¿Qué beneficios otorgan al sector industrial las FPGA en el procesamiento digital de señales?

Respuesta, Las FPGA pueden procesar señales a velocidades mayores que los sistemas micro programados, por lo cual se pueden implementar en FPGA estrategias de control que de otra manera no serían factibles, también se puede obtener un mejor ancho de banda en dispositivos como generadores de señales de forma de onda arbitraria.

4. Describa desde su experiencia, ¿Cuáles son las estrategias que utiliza cuando se encuentra ante trabajos con FPGA y diseño de hardware?

Respuesta,

1) Definir el Tipo y Número de entradas y salidas del diseño.

2) Definir los requerimientos funcionales (comportamiento del diseño deseado)

3) Seleccionar el Hardware (Composición interna, cantidad y composición de los bloques programables necesarios)

4) Realizar un diseño previo y validarlo mediante las herramientas disponibles.

5) Realizar las correcciones necesarias al diseño.

6) Realizar de ser posible la implementación de un prototipo. (Puede ser mediante tarjetas de desarrollo existentes)

7) Realizar las correcciones necesarias al diseño.

8) Realizar la construcción del producto final.

5. En su opinión, ¿Cuál es la mejora en cuando a diseño que se obtiene con una descripción estructural en VHDL?

Respuesta, Al emplear descripción estructural el diseñador puede ir formando su propia biblioteca de componentes los cuales ya previamente haya probado y este seguro de su comportamiento, lo que le permitiría dedicar su mayor esfuerzo al desarrollo de los nuevos componentes que no existan, y a establecer las conexiones entre los componentes que forman el diseño, por lo que se tendría modularidad y reusabilidad en el diseño, dos características muy deseadas cuando se diseña.

6. Desde el punto de vista de diseñador de hardware, ¿Podría usted enumerar las variables a considerar para ejecutar un correcto diseño de hardware con las herramientas de diseño?

Respuesta,

1) El Tipo y Número de entradas y salidas del diseño.

2) La Composición interna (cantidad y composición de los bloques programables).

Tercer entrevistado

Entrevistador: Autor

Entrevistado: Aida Pérez, Ing. Electricista MSc en instrumentación.

Fecha: 10.06.2023

1. En su opinión ¿Cuáles son los tipos de FPGA necesarias en el sector industrial?

Respuesta, No tengo suficiente experiencia para indicar los tipos de FPGA en la industria No entiendo lo de tipos de FPGA Hay distintos tipos lógicos programables y las FPGA son un tipo, así como lo son los PLD y los CPLD. Pero no conozco los tipos de FPGA en la industria Creo que se debe plantear la pregunta de forma diferente, hacia sus campos de aplicaciones

Se pueden aplicar en sectores como:

- Aeroespacial.
- Audio.
- Automotriz.
- Broadcast.
- Electrónica.
- Centros de datos.
- Computación de alto rendimiento.
- Industrial y médica.

2. Desde su experiencia en el ramo industrial ¿Cuáles son las características más comunes de las FPGA en la industria?

Respuesta, No tengo suficiente experiencia respecto a características más comunes de las FPGA en la industria Se puede hablar de las características de las FPGA: flexibilidad, pueden hacer cambios físicos sin hacer modificaciones costosas en la placa que lo soporta. Las FPGA de alto rendimiento están ayudando a los procesadores mediante aceleraciones de carga y descarga de información, con lo que se aumenta el rendimiento del sistema.

3. Las FPGA se encuentran comúnmente utilizadas en el DSP; Podría describir usted ¿Qué beneficios otorgan al sector industrial las FPGA en el procesamiento digital de señales?

Respuesta, Esta pregunta se parece a la 2 Flexibilidad, aceleración, simplificar el proceso de diseño y reducir el tiempo de comercialización, pueden utilizarse para ejecutar tareas de procesamiento de la visión e inteligencia artificial de alta intensidad computacional,

4. Describa desde su experiencia, ¿Cuáles son las estrategias que utiliza cuando se encuentra ante trabajos con FPGA y diseño de hardware?

Respuesta, Diseño inicial del sistema en bloques, programación, simulación, síntesis, implementación y configuración.

5. En su opinión, ¿Cuál es la mejora en cuando a diseño que se obtiene con una descripción estructural en VHDL?

Respuesta, Se pueden describir las funciones del hardware de forma más completa y las funciones que realizará.

6. Desde el punto de vista de diseñador de hardware, ¿Podría usted enumerar las variables a considerar para ejecutar un correcto diseño de hardware con las herramientas de diseño?

Respuesta, Entradas, salidas, señales.