UNIVERSIDAD CATÓLICA DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA DE INGENIERÍA ELECTRÓNICA EN CONTROL Y AUTOMATISMO

TÍTULO:

DESARROLLO DE ALGORITMOS EN VHDL SOBRE UNA FPGA DE0-NANO PARA PRÁCTICAS DE LABORATORIO DE DIGITALES EN LA CARRERA DE INGENIERÍA ELECTRÓNICA EN CONTROL Y AUTOMATISMO

AUTOR:

JUAN CARLOS ALVARADO BONILLA

Previa la obtención del Título

INGENIERO ELECTRÓNICO EN CONTROL Y AUTOMATISMO

TUTOR:

MsC. Luis Orlando Philco Asqui

Guayaquil, Ecuador

2016

UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA DE INGENIERÍA ELECTRÓNICA EN CONTROL Y AUTOMATISMO

CERTIFICACIÓN

Certificamos que el presente trabajo fue realizado en su totalidad por el Sr. **Juan Carlos Alvarado Bonilla** como requerimiento parcial para la obtención del título de INGENIERO ELECTRÓNICO EN CONTROL Y AUTOMATISMO.

TUTOR

MsC. Luis Orlando Philco Asqui

DIRECTOR DE CARRERA

MsC. Miguel A. Heras Sánchez.

Guayaquil, a los 17 días del mes de Marzo del año 2016

UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA DE INGENIERÍA ELECTRÓNICA EN CONTROL Y AUTOMATISMO

DECLARACIÓN DE RESPONSABILIDAD

Yo, **Juan Carlos Alvarado Bonilla**

DECLARO QUE:

El trabajo de titulación "DESARROLLO DE ALGORITMOS EN VHDL SOBRE UNA FPGA DE0-NANO PARA PRÁCTICAS DE LABORATORIO DE DIGITALES EN LA CARRERA DE INGENIERÍA ELECTRÓNICA EN CONTROL Y AUTOMATISMO" previa a la obtención del Título de Ingeniero Electrónico en Control y Automatismo, ha sido desarrollado respetando derechos intelectuales de terceros conforme las citas que constan al pie de las páginas correspondientes, cuyas fuentes se incorporan en la bibliografía. Consecuentemente este trabajo es de mi autoría.

En virtud de esta declaración, me responsabilizo del contenido, veracidad y alcance científico del Trabajo de Titulación referido.

Guayaquil, a los 14 días del mes de Marzo del año 2016

EL AUTOR

JUAN CARLOS ALVARADO BONILLA

UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA DE INGENIERÍA ELECTRÓNICA EN CONTROL Y AUTOMATISMO

AUTORIZACIÓN

Yo, **Juan Carlos Alvarado Bonilla**

Autorizamos a la Universidad Católica de Santiago de Guayaquil, la publicación, en la biblioteca de la institución del Trabajo de Titulación: "DESARROLLO DE ALGORITMOS EN VHDL SOBRE UNA FPGA DE0-NANO PARA PRÁCTICAS DE LABORATORIO DE DIGITALES EN LA CARRERA DE INGENIERÍA ELECTRÓNICA EN CONTROL Y AUTOMATISMO", cuyo contenido, ideas y criterios es de mi exclusiva responsabilidad y autoría.

Guayaquil, a los 14 días del mes de Marzo del año 2016

EL AUTOR

JUAN CARLOS ALVARADO BONILLA

DEDICATORIA

Dedico este trabajo a todas esas horas de esfuerzo y desvelo, a esos sacrificios e infinidad de pruebas y bendiciones que me llevaron hasta este punto, el de poder titularme como Ingeniero Electrónico en Control y Automatismo. También le dedico este trabajo a quienes directa o indirectamente fueron participes de mis fracasos y esta gran victoria, que no es más que un peldaño adicional en este camino llamado vida.

EL AUTOR

JUAN CARLOS ALVARADO BONILLA

AGRADECIMIENTO

Como fiel creyente quiero dar gracias en primer lugar a Dios, quien es El responsable de todas las experiencias vividas y por vivir. Le doy gracias a mi madre, el ser humano más importante en mi vida, que con su esfuerzo y lágrimas logró infundir en mí el coraje para alcanzar cada una de mis metas, y porque fue quien me levantó cuando caí. También le quiero agradecer a quienes colaboraron de alguna manera u otra en la consecución del presente trabajo.

EL AUTOR

JUAN CARLOS ALVARADO BONILLA

Índice General

Índice de Figuras

Capítulo 2: Fundamentación Teórica de FPGAs.

Capítulo 3: Aplicaciones Prácticas en VHDL sobre una FPGA DE0-NANO.

Índice de Tablas

Capítulo 2: Fundamentación Teórica de FPGAs.

Capítulo 3: Aplicaciones Prácticas en VHDL sobre una FPGA DE0-NANO.

Resumen

Los Sistemas Electrónicos Digitales son de gran importancia en la formación de futuros profesionales de la Carrera de Ingeniería Electrónica en Control y Automatismo. Los avances logrados en la electrónica digital, permitieron que se desarrollen tecnologías, tal como, dispositivos lógicos programables simples y complejos. En este último, dispositivo complejo se lo conoce como CPLD, y se incluyen al arreglo de compuertas programables en campo (FPGA). Este dispositivo FPGA se pueden implementar diversas aplicaciones de compuertas lógicas, circuitos combinacionales, circuitos secuenciales, diseño de controladores, entre otras, cuyos contenidos cumplen con los programas de estudio de Sistemas Digitales I y II y Laboratorio de Digitales. También, sirven para desarrollar proyectos de investigación y se incluyan aplicaciones de robótica. El dispositivo FPGA escogido fue DE0-Nano de Altera, en el mismo se implementaron aplicaciones para demostrar la utilidad y funcionalidad de la FPGA. Para el desarrollo del capítulo 3, se utilizó el software Quartus II para programar en VHDL. Los algoritmos de programación fueron realizados en VHDL e implementados en la DE0-Nano cuyo integrado es el Cyclone IV. Finalmente, este trabajo de titulación cumplió con el propósito planteado, que fue desarrollar algoritmos en VHDL y evaluados en la FPGA DE0-Nano.

CAPÍTULO 1: GENERALIDADES DEL TRABAJO DE TITULACIÓN

1.1. Introducción.

Los arreglos de compuertas programables en campo (FPGA), se convierten en una de las más exitosas tecnologías de hoy en día para el desarrollo de los sistemas electrónicos digitales que requieren un funcionamiento en tiempo real. Aunque, los circuitos integrados para aplicaciones específicas *(Application Specific Integrated Circuit, ASIC)* son dispositivos personalizados que también se utilizan para este propósito, pero las FPGAs proporcionan flexibilidad adicional.

Las FPGAs colocan celdas lógicas fijas en la oblea y los diseñadores de FPGAs pueden construir funciones más complejas de estas celdas. El término campo programable resalta la personalización del circuito integrado *(Integrated Circuit, IC)* por el usuario. Varios profesionales en sus investigaciones discuten el diseño de sistemas digitales o hardware. Las numerosas investigaciones realizadas con las FPGAs resaltan aplicaciones de control, y pretenden obtener mejores respuestas de control.

Las FPGAs son matrices bidimensionales de bloques lógicos y biestables (Flip-Flops) con una interconexión programable eléctricamente entre bloques lógicos. Las interconexiones constan de conmutadores programables eléctricamente, razón por la cual las FPGAs difieren de los ICs personalizados, es decir, que se programan utilizando la tecnología de

fabricación de circuitos integrados para formar conexiones entre los bloques lógicos.

1.2. Antecedentes.

En un dispositivo FPGA los bloques lógicos se implementan utilizando múltiples niveles de compuertas de bajo nivel, lo que le da un diseño más compacto en comparación con una implementación en dos niveles lógicos AND-OR. Las FPGAs proporciona a los usuarios una forma de configuración y estas especificaciones ofrece la “lógica difusa”, es decir, una gran cantidad de lógica en un solo IC (intersección entre los bloques lógicos y la función de cada bloque lógico).

Los bloques lógicos de una FPGA se pueden configurar de tal manera que puede proporcionar una funcionalidad tan simple como la de un transistor o tan complejo como la de un microprocesador. Se puede utilizar para implementar diferentes combinaciones de funciones lógicas combinacionales y secuenciales.

En el ámbito educativo, las instituciones de educación superior realizan mejoras en sus laboratorios, en especial los de electrónica digital. La mayoría de carreras de Ingeniería Electrónica, aún utilizan protoboard, circuitos de compuertas lógicas, en sus laboratorios de digitales. Algunas IES han logrado equipar con dispositivos lógicos programables, tal como los FPGAs.

La ventaja de utilizar dispositivos FPGAs, es que no necesitan de ICs, diodos, switch, entre otras cosas, basta con desarrollar algoritmos de programación en HDL, VHDL, captura esquemática y máquinas de estados.

1.3. Justificación del Problema.

A través de algoritmos de programación en VHDL se podrán desarrollar prácticas de sistemas digitales y que están orientadas a mejorar el proceso de aprendizaje de los estudiantes de V y VI de ciclo de la Carrera de Ingeniería Electrónica en Control y Automatismo. La intención del presente trabajo de titulación, es demostrar la funcionalidad y operatividad de la tarjeta DE0-Nano de Altera.

1.4. Definición del Problema.

Necesidad de desarrollar algoritmos de programación en VHDL sobre un dispositivo programable FPGA de la familia Altera DE0-Nano para prácticas de Laboratorio de Digitales en la Carrera de Ingeniería Electrónica en Control y Automatismo.

1.5. Objetivos del Problema de Investigación.

1.5.1 Objetivo General.

Desarrollar algoritmos de programación en VHDL sobre dispositivos programables FPGAs para prácticas de Laboratorio de Digitales en la Carrera de Ingeniería Electrónica en Control y Automatismo.

1.5.2 Objetivos Específicos.

- Describir la fundamentación teórica de los diferentes dispositivos de lógica programable y del arreglo de compuertas programables de campo FPGA.
- Diseñar sistemas combinatoriales y secuenciales mediante el desarrollo de algoritmos en un lenguaje de descripción de hardware de circuitos integrados de muy alta velocidad.
- Evaluar los algoritmos VHDL desarrollados en Quartus II sobre el dispositivo de lógica programable FPGA DE0-Nano.

1.6. Hipótesis.

Los algoritmos de programación en VHDL permitirán evaluar la funcionalidad de la tarjeta DE0-Nano para el desarrollo de aplicaciones básicas en sistemas digitales y de implementar prácticas para el Laboratorio de Digitales.

1.7. Metodología de Investigación.

Para (Davison, 2015) la forma en que se efectúa la investigación puede ser concebida en términos de la filosofía de investigación suscrita a, la estrategia de investigación empleada y a los instrumentos de investigación utilizada (y tal vez desarrollado) en la búsqueda de un objetivo (el objetivo de investigación), la búsqueda de la solución de un problema.

La investigación se clasifica ampliamente en dos clases principales: la investigación fundamental o básica y la investigación aplicada. La investigación básica, según (Rajasekar, Philominathan, & Chinnathambi, 2016) es una investigación sobre los principios y razones básicas de la ocurrencia de un evento en particular o proceso o fenómeno, también conocida como investigación teórica.

La investigación básica algunas veces puede no dar lugar a un uso inmediato o aplicación. No tiene que ver con la solución de los problemas prácticos de interés inmediato. Pero es original o de carácter básico. Proporciona una visión sistemática y profundamente en un problema y facilita la extracción de la explicación científica y lógica. Los resultados de la investigación básica forman la base de muchas investigaciones aplicadas. Los investigadores que trabajan en la investigación aplicada tienen que hacer uso de los resultados de la investigación básica y explorar la utilidad de ellas.

La investigación aplicada, según (Rajasekar, Philominathan, & Chinnathambi, 2016) se pueden resolver ciertos problemas que emplean teorías y principios bien conocidos y aceptados. La mayor parte de la investigación experimental, estudios de caso y la investigación interdisciplinaria son la investigación aplicada en esencia. La investigación aplicada es útil para la investigación básica. Una investigación, cuyo resultado tiene una aplicación inmediata también se denomina como la investigación aplicada.

Una vez revisada la información del tipo de metodología de investigación, el presente trabajo utilizará el método descriptivo y explicativo con enfoque cuantitativo.

.

CAPÍTULO 2: Fundamentación Teórica de FPGAs.

2.1. Introducción a los dispositivos lógicos programables.

En esta sección se realiza una pequeña introducción de dispositivos programables de campo *(Field Programmable Devices, FPDs)* también conocidos como dispositivos lógicos programables *(Programmable Logic Devices, PLD)*. En la figura 2.1 se muestra la clasificación de los dispositivos SPLDs, CPLDs y FPGAs.

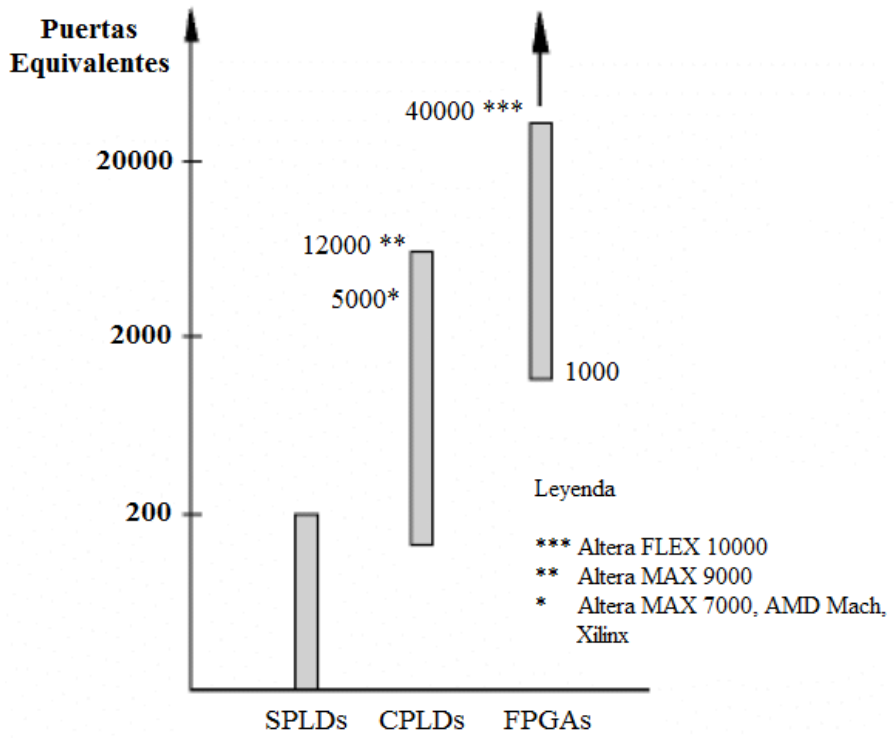


Figura 2. 1: Clasificación de dispositivos programables SPLDs, CPLDs y FPGAs.
Fuente: (Bozich, 2005)

De acuerdo al trabajo de (Bozich, 2005) se procederá a describir en las siguientes secciones del capítulo 2, las aplicaciones de matrices lógicas programables *(Programmable Logic Arrays, PLAs)*, posteriormente se describen los PLDs simples (SPLDs) y PLDs complejos (CPLDs), y se

concluye con la descripción del arreglo de compuertas programables de campo *(Field Programmable Gate Arrays, FPGAs)*.

2.2. Matrices Lógicas Programables - PLA.

Una matriz lógica programable (PLA), es una pequeña FPD que contiene dos niveles de lógica, AND y OR, comúnmente conocida como nivel AND y nivel OR respectivamente. En concepto, una PLA es similar a una ROM, pero no proporciona decodificación completa de las variables, en otras palabras, un PLA no genera todos los términos producto (conocido como minterms) como lo hace una memoria ROM.

Desde luego, una ROM se puede diseñar como un circuito combinacional con cualquier palabra no utilizada y no les importa las condiciones, pero las palabras no utilizadas darían lugar a un mal diseño. Consideremos, por ejemplo, la conversión de un código de 16 bits en un código de 8 bits. En este caso, tendríamos que $2^{16} = 65536$ combinaciones de entrada y sólo $2^8 = 256$ combinaciones de salida, y debido a que sólo necesitaría 256 entradas válidas para el código de 16 bits, tendríamos 65280 palabras desperdiciadas. Con una PLA podemos evitar este tipo de palabras desperdiciadas.

El tamaño de una PLA se define por el número de entradas, el número de suma de productos (SOP o minterms) en el nivel AND, y el número de productos de suma (POS o maxterms) en el nivel OR. En la figura 2.2 se

muestra el diagrama de bloques de un PLA típico, que se compone de entradas, salidas, SOP y POS. El grupo de los términos constituye el nivel de compuertas AND, y el grupo de términos constituye el nivel de compuertas OR.

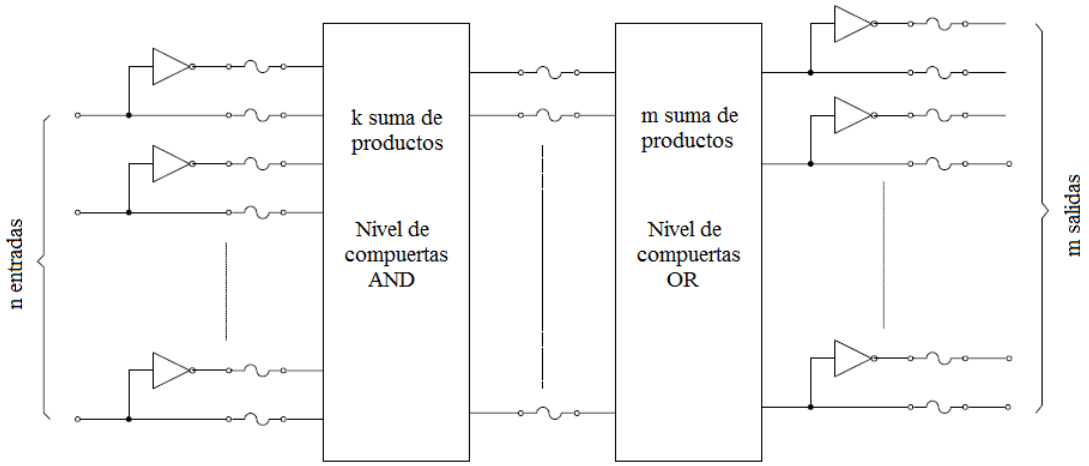


Figura 2. 2: Diagrama de bloques de un típico PLA
Fuente: (Karris, 2010)

Podemos observar los enlaces en la figura 2.2, que aparecen como fusibles, insertados en todas las entradas y sus valores añaden a cada una de las puertas AND. También, se proporcionan enlaces entre las salidas de las compuertas AND y las entradas de las compuertas OR. Otro conjunto de enlaces aparecen en las líneas de salida que permite a la función de salida aparecer, ya sea en la forma AND-OR o la forma AND-OR invertida. Estas formas se muestran en la figura 2.3. Los inversores en la sección de salida de la PLA son dispositivos de tres estados.

Una PLA puede ser de máscara programable o de campo programable. Con una máscara programable PLA el fabricante presenta una tabla (véase la

tabla 2.1 que hace referencia a la figura 2.4) de programación PLA. Con un campo programable PLA el usuario configura el mismo PLA. En la figura 2.4 se muestra un ejemplo de cómo programar un PLA, sea un circuito combinacional definido por las funciones $F_1 = A\overline{B}\,\overline{C} + A\overline{B}\,C + ABC$ y $F_2 = \overline{A}\,BC + A\overline{B}\,C + ABC$, es posible diseñar el circuito digital (véase la figura 2.4) con una PLA de 3 entradas, 3 SOP y 2 salidas

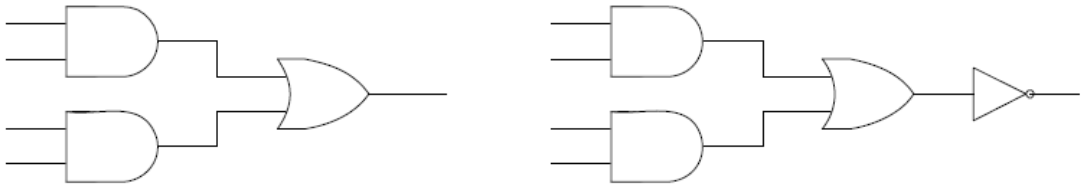


Figura 2. 3: Formas de configuración AND-OR y AND-OR invertidas.
Fuente: (Karris, 2010)

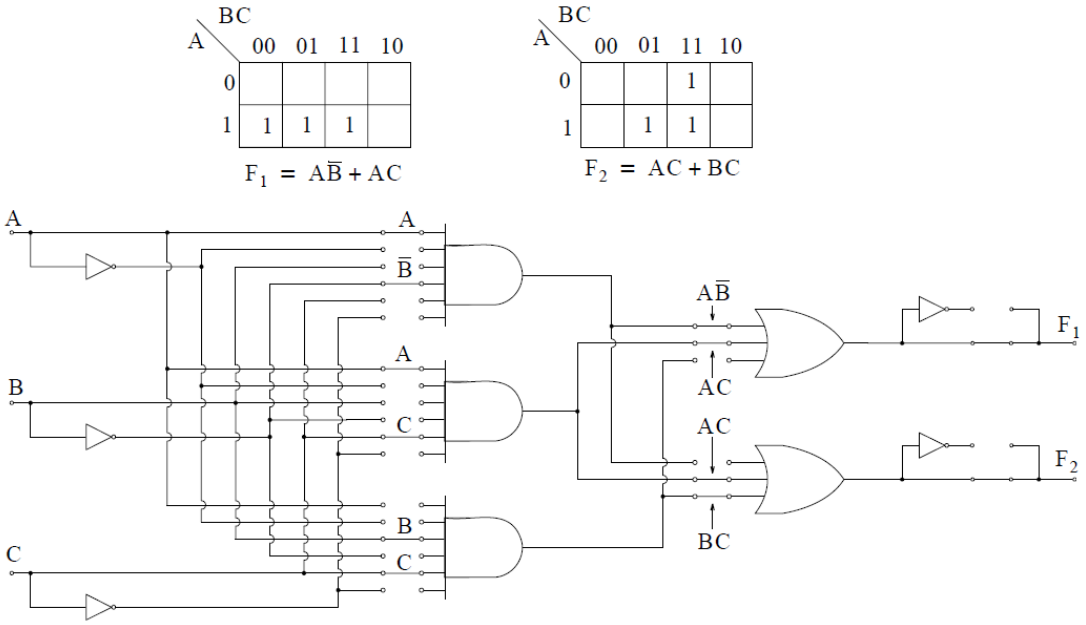


Figura 2. 4: Mapas de Karnaugh y diseño de una PLA.
Fuente: (Karris, 2010)

Tabla 2. 1: Términos de productos para el diseño de PLA.

	Términos de Productos	ENTRADAS			SALIDAS		
		A	B	C	F_1	F_2	
$A\bar{B}$	1	1	0	—	1	—	
AC	2	1	—	1	1	1	
BC	3	—	1	1	—	1	
					T	T	T/C

Fuente: (Karris, 2010)

2.3. Dispositivos de matriz programable - PAL.

De acuerdo a (Karris, 2010) los dispositivos de matriz programable *(Programmable Arrays Logic, PAL)* se han desarrollado para superar las deficiencias de los PLAs. Una diferencia importante entre los PLAs y PALs, es que este último está diseñado con un cable programable por nivel AND seguido por un número fijo de compuertas OR y de flip flops de manera que los cirucitos secuenciales también puedan ser implementados.

Mientras que (Montejo R., 2016) sostiene que las PALs tienen una arquitectura diferente a las PLAs, en la que internamente consiste de compuertas AND programables que alimentan compuertas OR fijas. Es decir, que se pueden combinar compuertas AND entre si mismo, pero otras AND especificas se proponen compuertas OR especificas.

Las variantes de los PALs originales fueron diseñados con diferentes entradas, y varios tamaños de compuertas OR, y que forman actualmente la

base de los diseños de hardware digital. En la figura 2.5 se muestra el diagrama de bloques de una PAL típica.

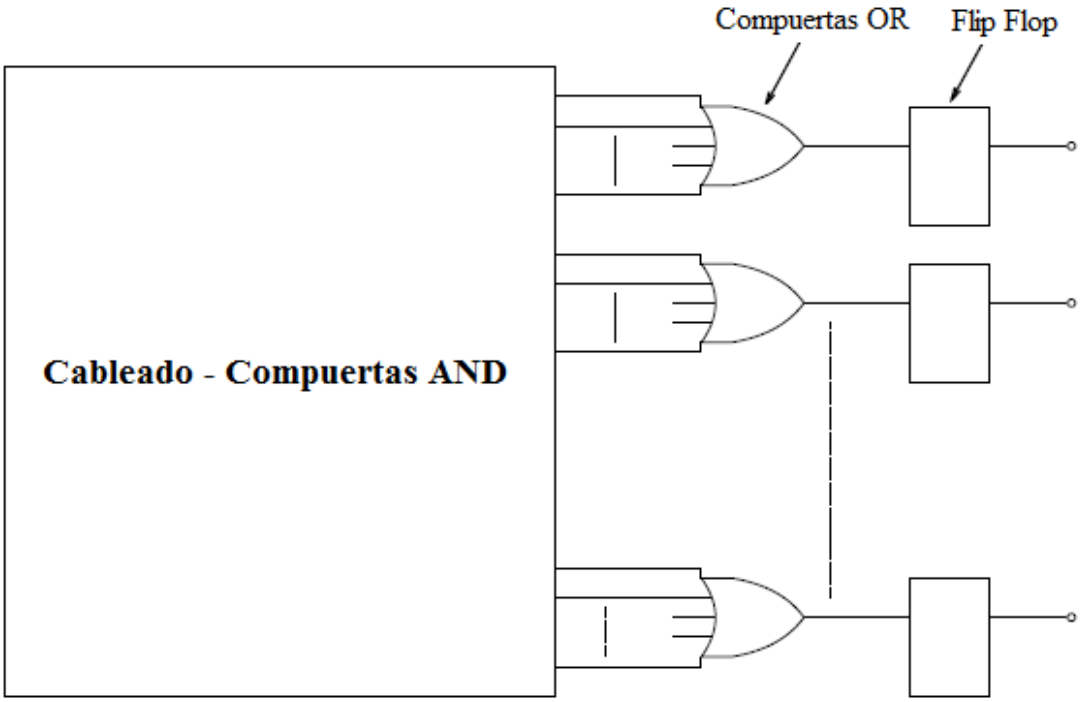


Figura 2. 5: Diagrama de bloques de una PAL típica.
Fuente: (Karris, 2010)

Las variantes de PLAs, PALs y PAL se refieren generalmente a los Dispositivos Lógicos Programables Simples *(Simple Programmable Logic Devices, SPLDs)*. Cuatro destacadas SPLDs son la serie 16R8, 16R6, 16R4, y 16L8. Estos dispositivos tienen 16 entradas potenciales y 8 salidas configurables por el usuario. Las configuraciones de salida de 8 registros (flip flops), 8 combinacionales, 6 registros y 2 combinacionales, 4 registros y 4 combinacionales, o bien 8 combinacionales se proporcionan donde la letra R se refiere a la cantidad de salidas "registradas" y la letra L no registrada, es decir, salidas combinacionales.

La figura 11.7 muestra las variaciones funcionales Cypress Semiconductor de esta serie.

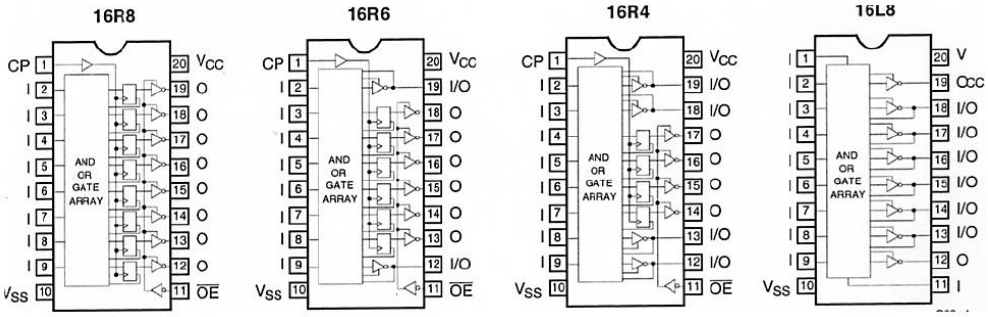


Figura 2. 6: Diferentes SPLDs.
Fuente: (Karris, 2010)

2.4. Dispositivos Lógicos Programables Complejos – CPLDs.

Los dispositivos CPLDs integran múltiples SPLDs en un solo chip. Un típico CPLD proporciona una capacidad lógica equivalente a 50 SPLDs simples. Altera, es el segundo mayor fabricante del mundo (actualmente Xilinx es el fabricante más grande) de semiconductores programables, entre los dispositivos CPLDs más destacados están las series MAX 5000, 7000, y 9000. MAX 5000 se basa en tecnología más antigua, la familia MAX 7000 es un PLD de alto rendimiento y fabricado con tecnología CMOS avanzada, y MAX 9000 es similar a MAX 7000 excepto que tiene mayor capacidad lógica.

2.4.1. Dispositivo Altera MAX 7000.

Debido a su versatilidad y popularidad, nuestra discusión sobre Altera CPLD se basará en la familia de CPLDs el MAX 7000. La familia MAX 7000 consta de 7 PLDs basados en memorias EEPROM proporcionando 600, 1250, 1800, 2500, 3200, 3750, 5000 compuertas utilizables.

Los dispositivos MAX 7000 utilizan celdas CMOS EEPROM para implementar funciones lógicas y acomodar una variedad de funciones combinacionales y secuenciales independientes. Los dispositivos pueden ser reprogramados para interacciones rápidas y eficientes durante los ciclos de desarrollo de diseño y depuración, y pueden programarse y borrarse hasta alrededor de 100 veces.

Debido a su complejidad, el diseño de circuitos digitales para su aplicación con FPDs, es necesario emplear programas de Diseño Asistido por Ordenador *(Computer Aided Design, CAD)*. Para la familia MAX 7000 de PLDs, el software de apoyo al diseño es proporcionado por Altera para plataformas basadas en Windows, entre otras estaciones de trabajo que son compatibles con Altera.

La figura 2.7 muestra la arquitectura básica de Altera MAX 7000 familia de CPLDs. Como se muestra, está contiene un conjunto de bloques denominados bloques de matriz lógica *(LABs)*, una interconexión de matriz programable (PIA) que contiene los cables de interconexión, y bloques de control de E/S. Cada LAB contiene 16 macroceldas y los PIA pueden conectarse a cualquier entrada o salida LAB a cualquier otro LAB. También incluye cuatro entradas dedicadas que se pueden utilizar como entradas, señales de control global (reloj, clear), y dos salidas de señales de habilitación.

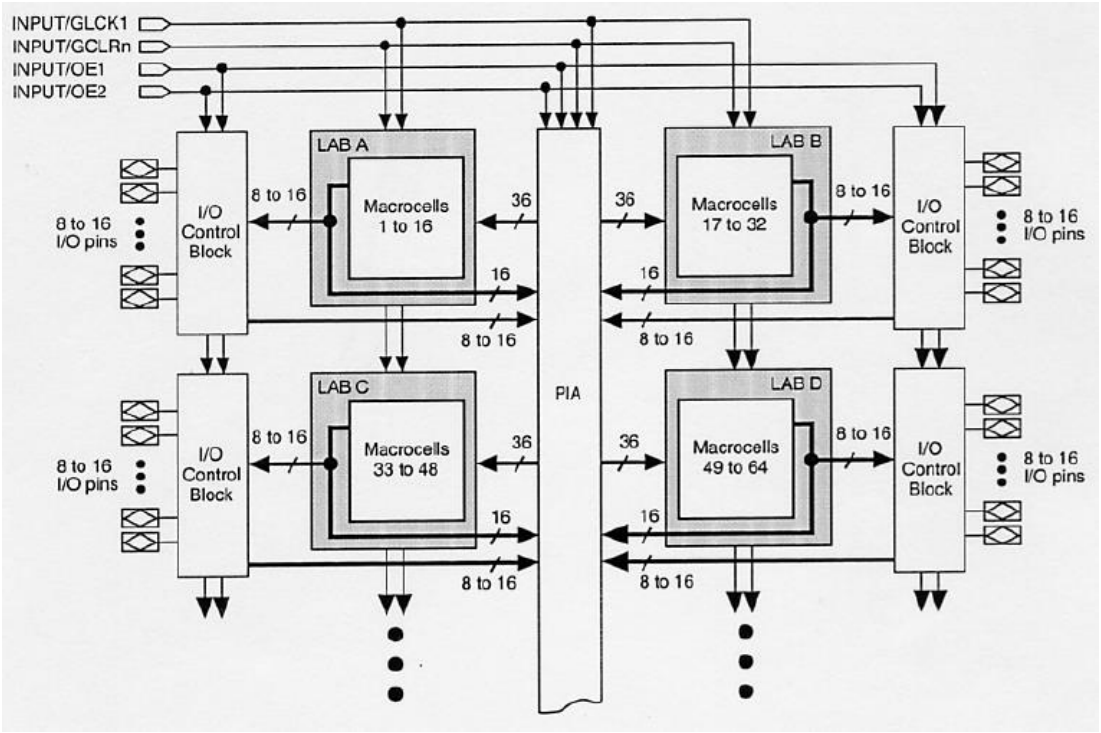


Figura 2. 7: Diagrama de bloques del dispositivo programable MAX 7000.
Fuente: (Karris, 2010)

La figura 2.8 muestra una matriz macrocelda del dispositivo MAX 7000, podemos observar que en la parte izquierda se proporcionan 5 términos de productos por macrocelda y el producto asigna estos términos de productos para su uso como entradas lógicas primarias (compuertas OR y XOR) para implementar funciones combinacionales, o como entradas secundarias de las macroceldas de registros, clear, preestablecido, reloj y permiten funciones de control.

Hay dos tipos de términos producto de expansión, conocidos como expansores. Los expansores del MAX 7000 como CPLDs más eficientes en el área de chip, ya que las funciones lógicas normalmente no requieren más de 5 términos de productos. Pero esta arquitectura es compatible con más de

5 si es necesario. A continuación se describen dos tipos de expansores: compartibles y paralelos.

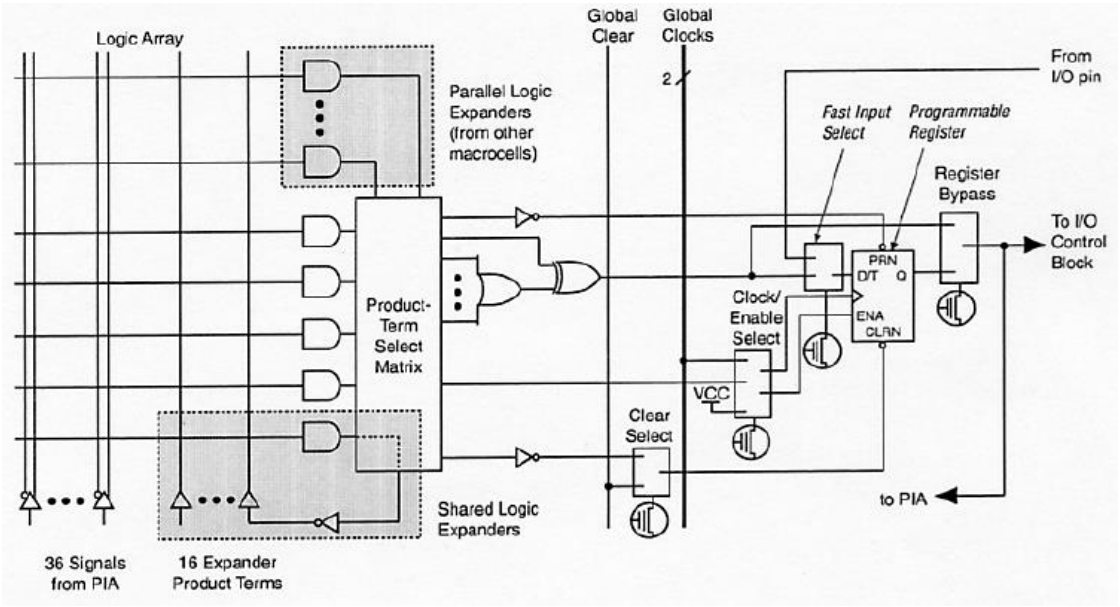


Figura 2. 8: Macroceldas de Altera MAX 7000.
Fuente: (Karris, 2010)

a. Expansores compartibles.

Cada LAB tiene 16 expansores compartibles que pueden visualizarse como un conjunto de términos de productos individuales no comprometidas (uno de cada macrocelda) con salidas invertidas que se alimentan de nuevo en la red lógica. Cada uno de los expansores compartibles pueden ser utilizadas y compartidas por cualquiera o todas las macroceldas en el LAB para construir funciones lógicas complejas. En la figura 2.9 se muestra cómo compartir los expansores para alimentar múltiples microelementos.

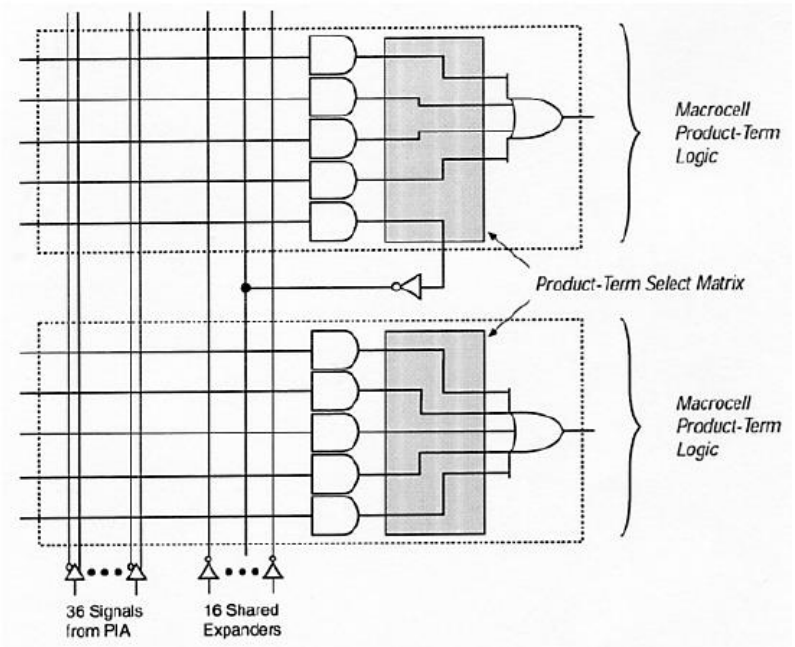


Figura 2. 9: Expansores compartibles para el dispositivo MAX 7000.
Fuente: (Karris, 2010)

b. Expansores paralelos.

Los expansores paralelos son términos de productos no utilizados que se pueden asignar a una macrocelda vecina para implementar funciones lógicas rápidas y complejas. Los expansores paralelos permiten hasta 20 términos de productos para alimentar directamente a una macrocelda lógica OR, con cinco términos de productos proporcionados por la macrocelda y 15 expansores paralelos previstos por la macrocelda vecina en el LAB.

El sistema de desarrollo Altera puede asignar hasta tres grupos de hasta cinco expansores paralelos de forma automática en las macroceldas que requieren términos de productos adicionales. Por ejemplo, si una macrocelda requiere 14 términos de producto, el sistema utiliza los 5 términos de productos dedicados dentro de la macrocelda y asigna 2 conjuntos de

expansores paralelos; el primer conjunto incluye 5 términos de productos, y el segundo conjunto incluye 4 términos de productos. Dos grupos de 8 macroceldas dentro de cada LAB (por ejemplo, macroceldas de 1 a 8 y 9 a 16) forman dos cadenas para prestar o pedir prestado expansores paralelos.

Una macrocelda toma prestado expansores paralelos a partir del número más bajo de macroceldas. Por ejemplo, 8 macroceldas toma prestado expansores paralelos de 7 macroceldas, o a partir de macroceldas 7 y 6, o desde macroceldas 7, 6 y 5. Dentro de cada grupo el número más bajo de macroceldas sólo puede prestar expansores paralelos y el número más alto de macroceldas sólo puede tomarlos prestados. En la figura 2.10 muestra cómo expansores paralelos puede tomar prestado a partir de una macrocelda vecina.

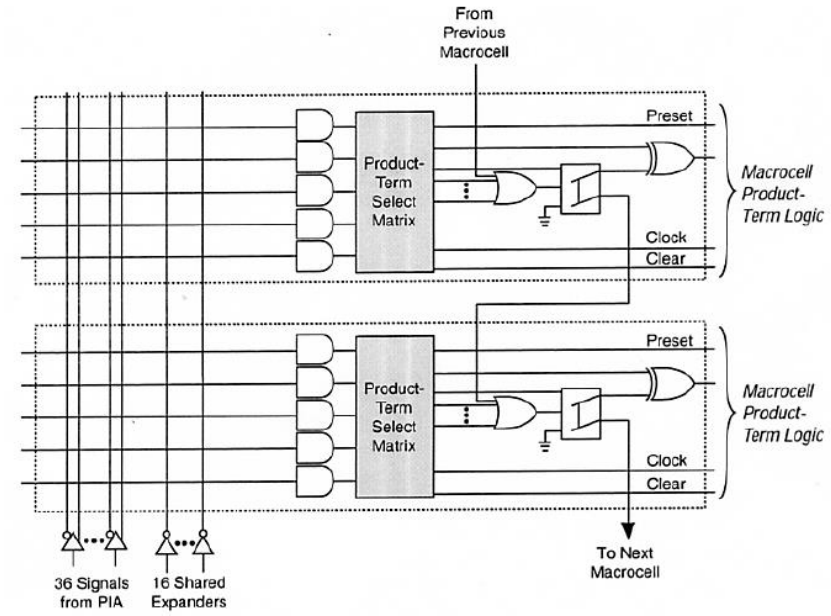


Figura 2. 10: Expansores paralelos para el dispositivo MAX 7000.
Fuente: (Karris, 2010)

2.4.2. Familia AMD Mach.

La familia de los micro dispositivos avanzados *(Advanced Micro Devices, AMD)* consiste en las series Mach 1 a Mach 5, donde cada dispositivo Mach comprende varios PALs. Mach 1 y Mach 2 se componen de PALs 22V16, Mach 3 y Mach 4 se componen de PALs 34V16, y Mach 5 también está hecho de PALs 34V16 pero es más rápido. Todos los dispositivos Mach se basan en tecnología EEPROM. Sólo vamos a discutir el Mach 4, ya que es el más popular entre esta serie, su arquitectura se muestra en la figura 2.11.

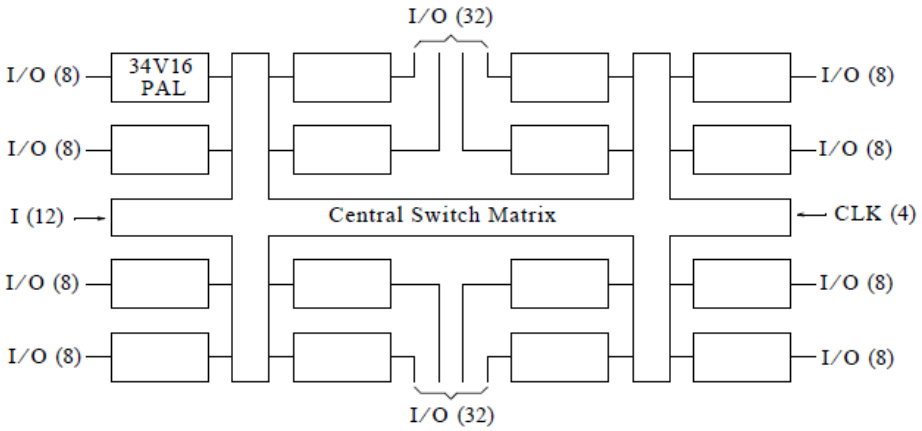


Figura 2. 11: Arquitectura del AMD Mach 4.
Fuente: (Karris, 2010)

La designación 34V16 indica que este PAL tiene 34 entradas y 16 salidas, y la letra V significa versátil, es decir, cada salida se puede configurar ya sea como registros (flip flops) o de circuitos combinacionales. La matriz de conmutación central, es un bus de interconexión que permite la conexión de todos PALs 34V16 juntos y por lo tanto los retardos de tiempo son predecibles. En la figura 2.12 se muestra el diagrama de bloques de la PAL 34V16.

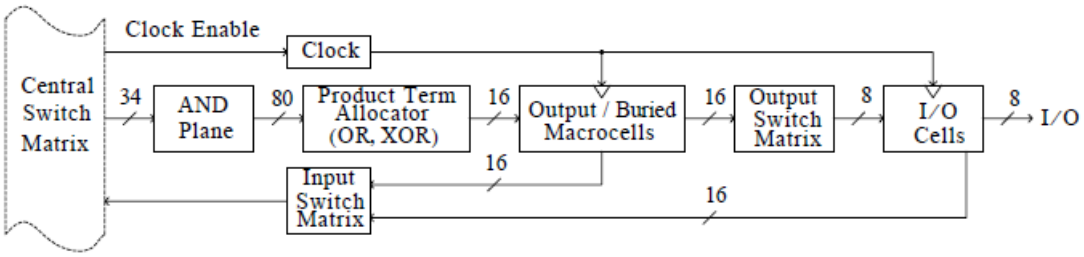


Figura 2. 12: Diagrama de bloques de una PAL 34V16.
Fuente: (Karris, 2010)

En la figura 2.12, el término producto asigna, distribuye y comparte términos de productos a partir del nivel AND a las compuertas OR que lo requieran. Una PAL convencional tiene un tamaño fijo de compuertas OR, por lo tanto, el PAL 34V16 proporciona más flexibilidad. Las macroceldas pueden ser tanto macroceldas de E/S, que incluyen una celda de E/S que se asocia con un pin de E/S, o macroceldas enterrados, que no se conectan a un dispositivo de E/S. La combinación de macroceldas de E/S y macroceldas enterradas varían de un dispositivo a otro.

La macrocelda enterrada cuenta con un registro que puede ser configurado como combinatorial, un flip flop D, un flip flop T, o un latch. También, las macroceldas enterradas son compatibles con la capacidad de registro de entrada. Además, la macrocelda enterrada se puede configurar para actuar como una entrada. La matriz de salida del interruptor hace posible que cualquier salida de microceldas (compuertas OR o flip flops) para conducir cualquiera de los pines de E/S conectados al bloque PAL.

2.4.3. La Familia Lattice.

Lattice Semiconductor Corporation, es considerado el inventor de los productos lógicos programables en sistema, diseña, desarrolla y comercializa una amplia gama de FPGAs, así como Chips de Sistema Programable de Campo *(Field Programmable System Chips, FPSCs)* y de los PLDs. Su línea de productos incluye las familias pLSI basados en EEPROM, los dispositivos ispLSI son básicamente el mismo que el pLSI excepto que incluyen programación en el sistema, el Lattice ispMACH 4256V CPLD, y las series Lattice de CPLDs 3000, 4000 y 5000. La arquitectura del Lattice ispLSI se muestra en la figura 2.13.

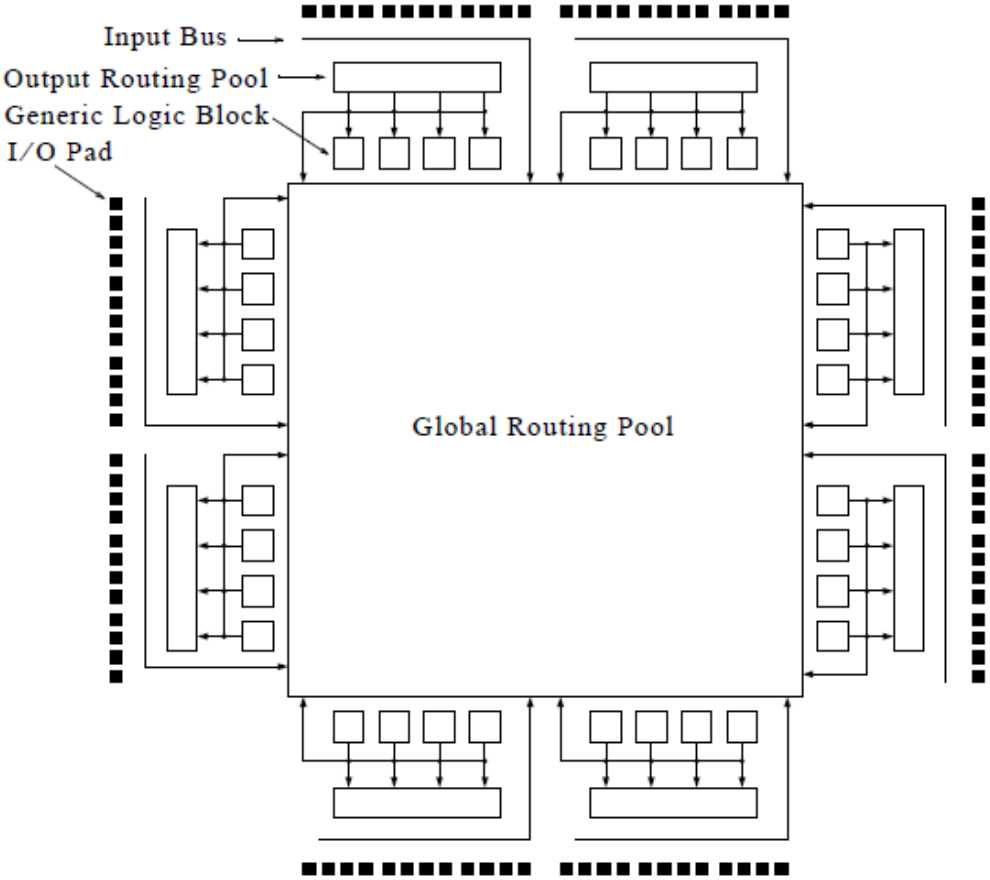


Figura 2. 13: Arquitectura Lattice ispLSI.
Fuente: (Karris, 2010)

En la figura 2.13 los Pads de E/S son dispositivos bidireccionales que están conectados a ambos Global Routing Pool (GRP) y los bloques lógicos genéricos. La GRP es un conjunto de cables que interconectan el dispositivo completo, incluyendo los bloques lógicos genéricos de entradas y salidas. Debido a que todas las interconexiones se enrutan a través del GRP, los retardos de tiempo son predecibles en cuanto a los dispositivos de AMD Mach. Cada uno de los bloques lógicos genéricos consiste básicamente cada PAL en un nivel AND, términos de productos asignados, y macroceldas tal como se muestra en la figura 2.14.

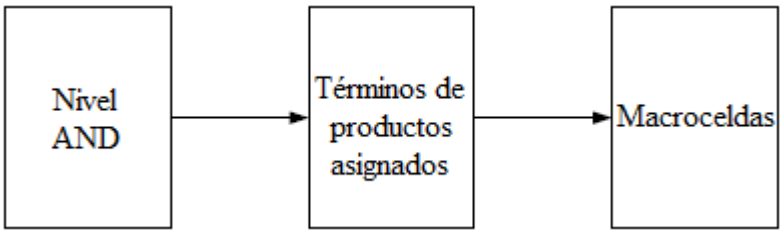


Figura 2. 14: Diagrama del bloque lógico genérico de Lattice.
Fuente: (Karris, 2010)

2.4.4. Cypress FLASH370.

Cypress FLASH 370 de la familia CPLDs, se basa en la tecnología EEPROM, y cuenta con una amplia variedad de densidades y cantidades de pines para elegir. En cada densidad hay dos opciones de empaquetado, una que es de E/S intensiva y otro de registro intensivo. Los dispositivos de la familia CY7C374 y CY7C375 (véase la figura 2.15), ambos cuentan con 128 macroceldas. El primer dispositivo CY7C375 dispone de un número igual de macroceldas y celdas de E/S que se muestra en la figura 2.16, esta arquitectura es la más adecuada para aplicaciones de E/S intensivas.

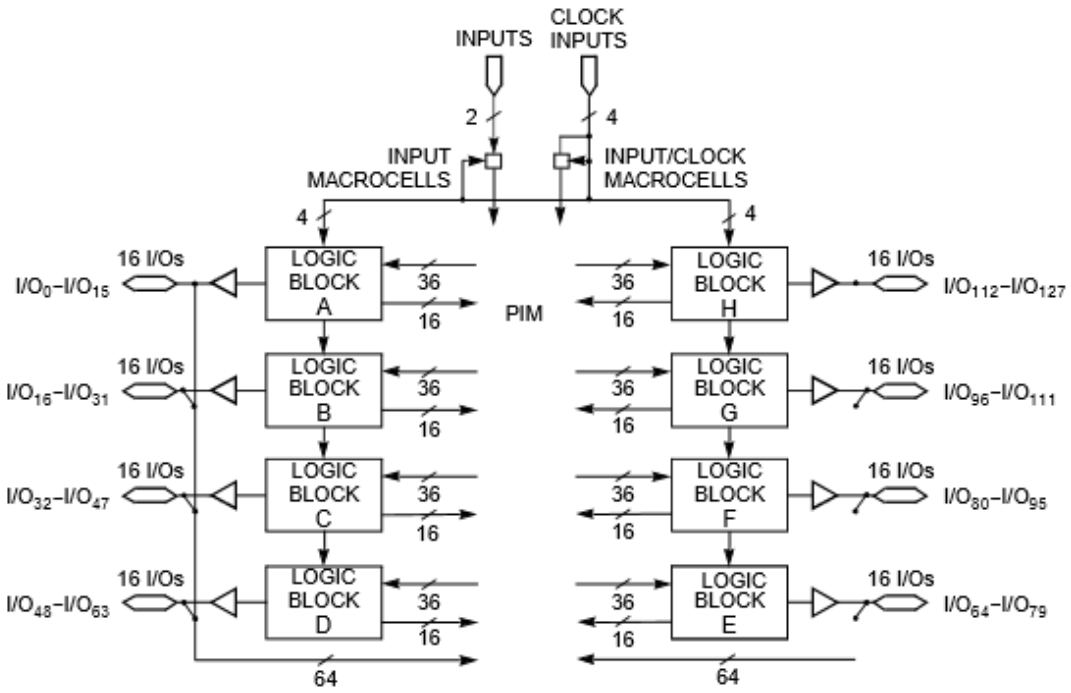


Figura 2. 15: Diagrama del bloques del dispositivo CY7C375.
Fuente: (Cypress, 2016)

En la figura 2.17 se muestra la arquitectura más adecuada para aplicaciones de registros intensivos.

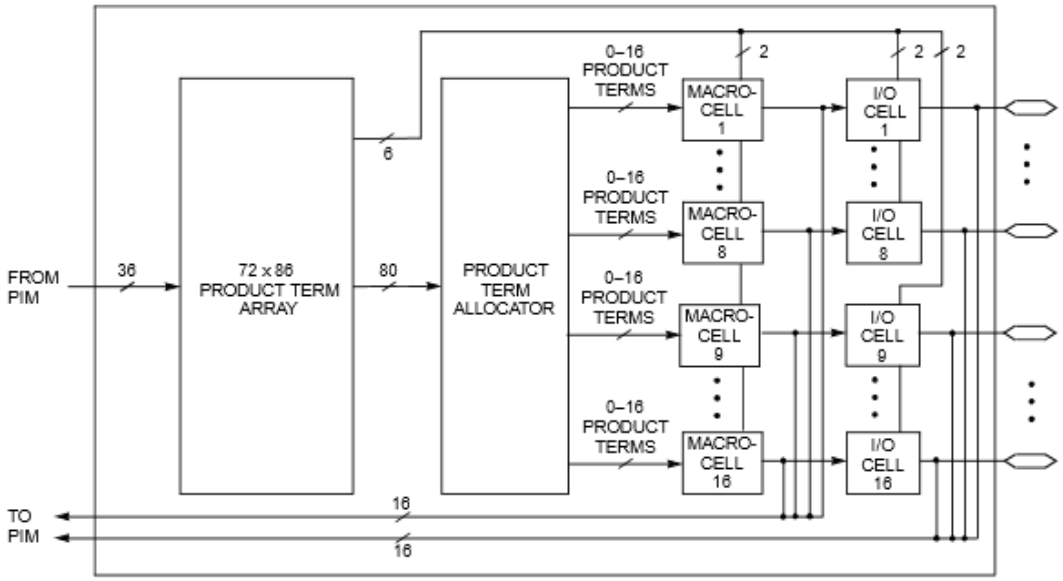


Figura 2. 16: Arquitectura para dispositivos de E/S intensivas.
Fuente: (Cypress-1, 2016)

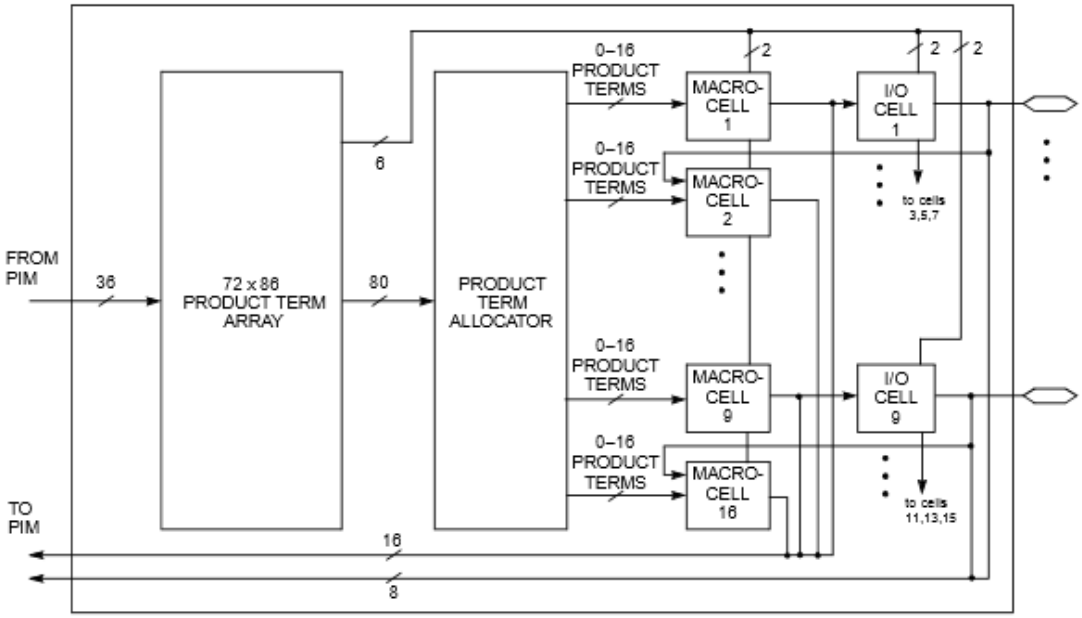


Figura 2. 17: Arquitectura para registros intensivos.
Fuente: (Cypress-1, 2016)

2.4.5. Xilinx XC9500.

La serie XC9500 es una de las familias programables de Xilinx dentro del sistema de CPLDs. Todos los dispositivos de esta familia están diseñados para un mínimo de 10000 ciclos de programa/borrado, e incluyen IEEE 1149.1 (JTAG). La arquitectura del Xilinx XC9500 se muestra en la figura 2.18. La densidad de la lógica de los dispositivos XC9500 varía de 800 a más de 6.400 puertas utilizables con 36 a 288 registros, respectivamente. La familia XC9500 es totalmente compatible pin-lo que permite la migración fácil del diseño a través de múltiples opciones de densidad en un tamaño determinado paquete.

Las características de la arquitectura XC9500 de Xilinx abordan los requisitos de capacidad de programación en el sistema. La capacidad de pines de bloqueo ha mejorado evita la reanudación de tarjetas costosas. Un

conjunto de instrucciones JTAG ampliado permite el control de versiones de los patrones de programación y depuración en el sistema.

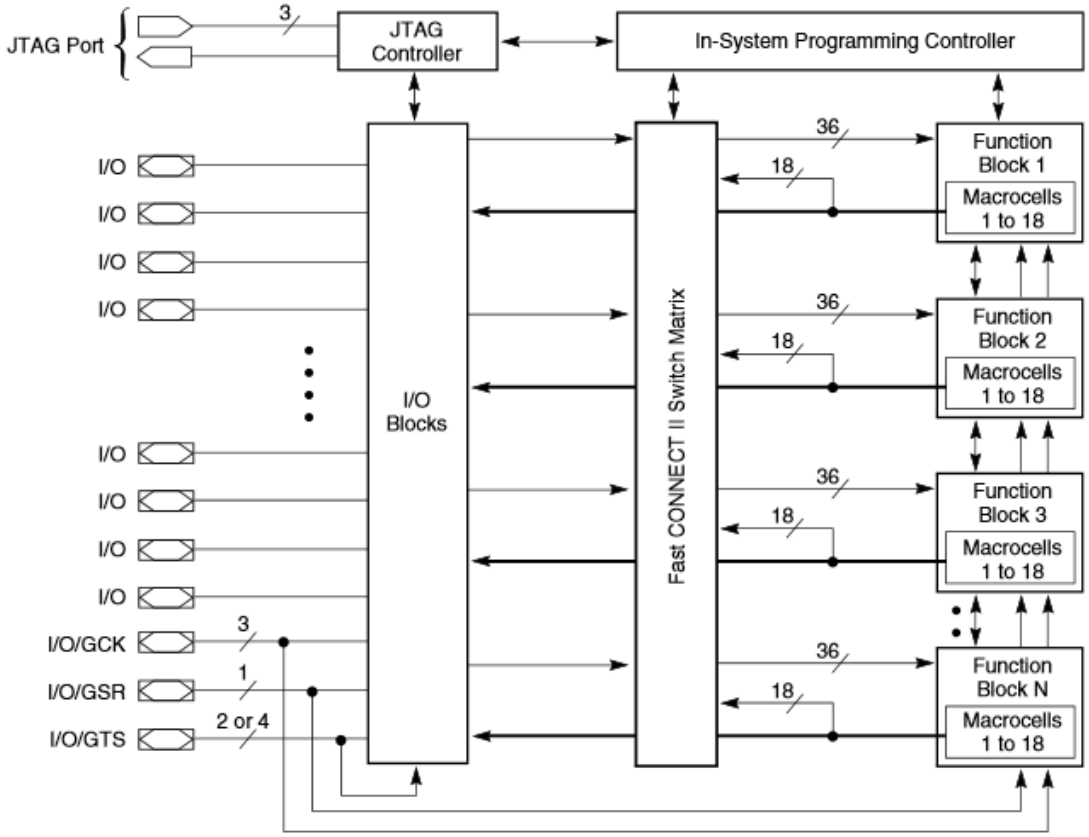


Figura 2. 18: Arquitectura del dispositivo XC9500.
Fuente: (Xilinx, 2016)

Cada uno de los bloques de funciones (véase la figura 2.19) se compone de 18 macroceldas independientes, cada uno capaz de implementar una función combinatorial o registros. El bloque de función también recibe un reloj global, salida de habilitación, y señales de set/reset. Cada uno de las macroceldas XC9500 se puede configurar de forma individual para una función combinatorial o de registros. La macrocelda y el bloque de función lógica asociada se muestran en la figura 2.20.

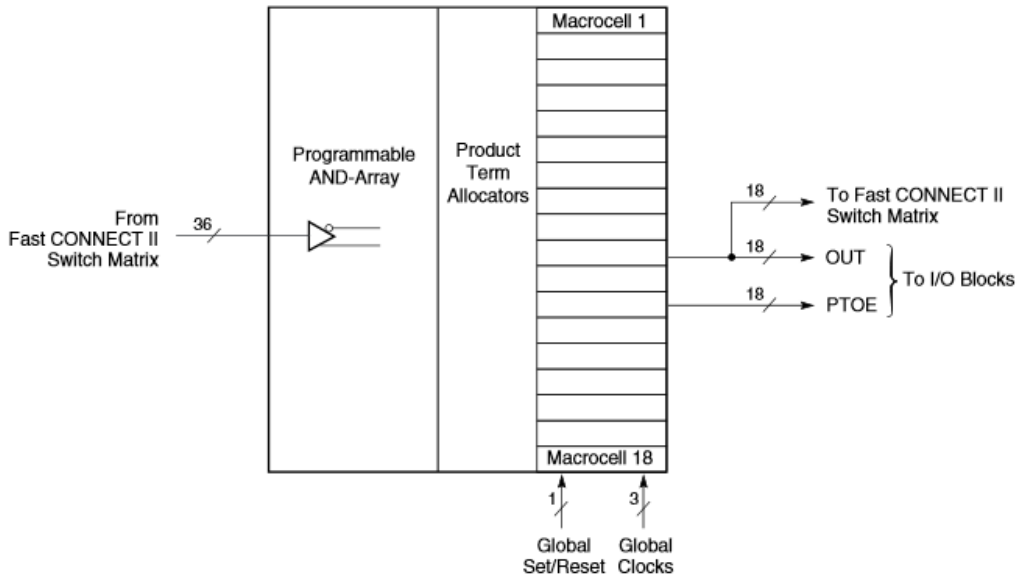


Figura 2. 19: Bloque de funciones del dispositivo XC9500.
Fuente: (Xilinx, 2016)

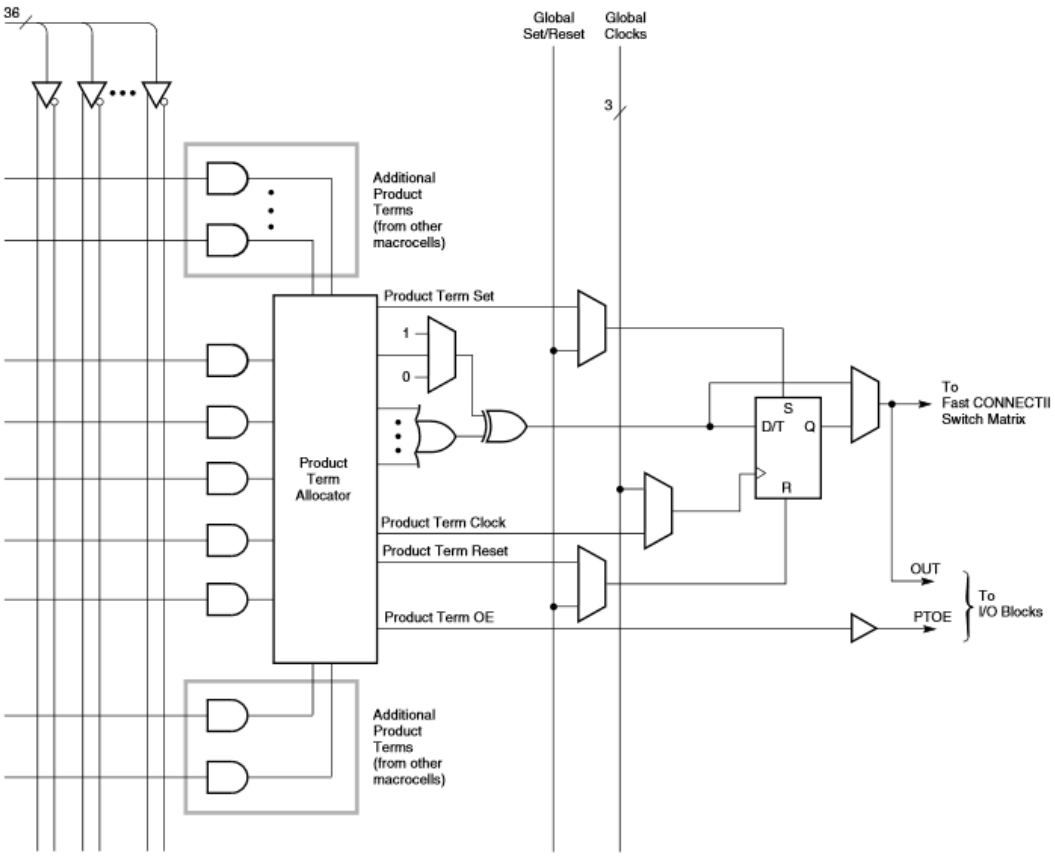


Figura 2. 20: Macrocelda dentro del bloque de funciones del dispositivo XC9500.
Fuente: (Xilinx, 2016)

Todas las señales de control (véase la figura 2.21) están disponibles para cada macrocelda individual, incluyendo señales de reloj, set/reset, y la salida de señales de habilitación. Observamos que la macrocelda de reloj se origina a partir de cualquiera de tres relojes globales o un reloj término producto.

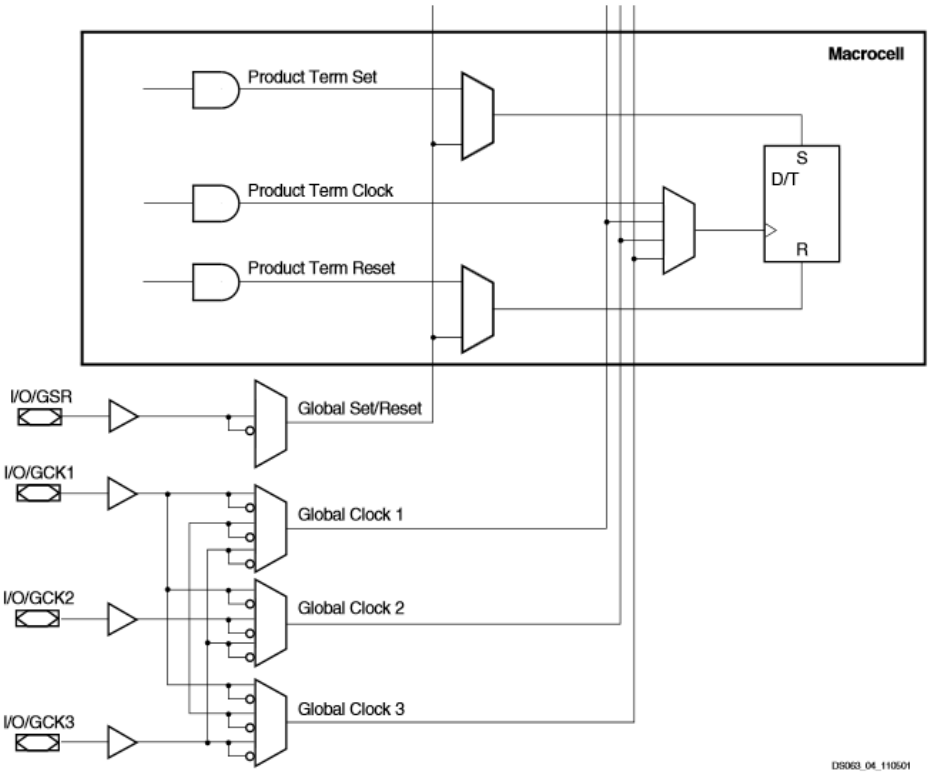


Figura 2. 21: Macrocelda de Reloj y capacidad de Set/Reset del XC9500.
Fuente: (Xilinx, 2016)

El término producto asignado determina cómo los 5 términos de productos directos se asignan a cada macrocelda. En la figura 2.22 se muestran 5 términos directos que conducen a la compuerta OR. El término producto asignado puede volver asignar otros términos de productos dentro del bloque de funciones, logrando aumentar la capacidad lógica de una macrocelda más allá de 5 términos directos.

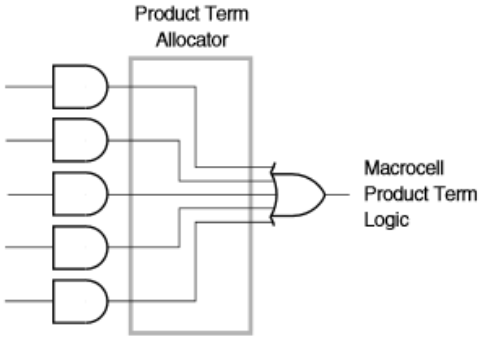


Figura 2. 22: Macrocelda lógica utilizando términos de productos.
Fuente: (Xilinx, 2016)

En las siguientes secciones se discutirán el arreglo de compuertas programables de campo conocidos como FPGAs, posteriormente se describirán los dispositivos Xilinx y Altera, este último servirá como base fundamental para utilizar la tarjeta DE0-Nano de Altera y mediante programación VHDL y diseño esquemático, se desarrollarán aplicaciones prácticas para la asignatura de Laboratorio de Digitales.

2.5. Arreglo de compuertas programables de campo - FPGA.

Una FPGA es similar a un PLD, pero los PLDs se limitan generalmente a cientos de compuertas, mientras que las FPGAs soportan miles de compuertas. Las FPGAs son especialmente populares para los prototipos de diseños de circuitos integrados. Una vez que el diseño se establece, los chips alámbricos son producidos para un rendimiento más rápido.

Las FPGAs se dividen en dos categorías principales:

a. FPGAs basados en SRAM.

b. FPGAs basados en antifuse.

2.5.1. FPGAs basados en SRAM.

Hay una amplia gama de FPGAs proporcionados por muchos proveedores de semiconductores incluyendo Xilinx, Altera, Atmel, y Lattice. Cada fabricante de FPGAs proporciona una propia arquitectura. Una FPGA típica consiste en una serie de elementos lógicos programables de enrutamiento y recursos utilizados para proporcionar la conectividad entre elementos lógicos, pines de E/S, y otros recursos como la memoria en el chip de la FPGA.

La estructura y la complejidad de los elementos lógicos, así como la organización y funcionalidad soportan jerarquía de interconexión, es lo que diferencia los diferentes dispositivos entre sí. Otras características, tales como memoria de bloque y tecnología de retardo de bucle cerrado son también factores importantes que influyen en la complejidad y rendimiento de un algoritmo ejecutado en las FPGAs.

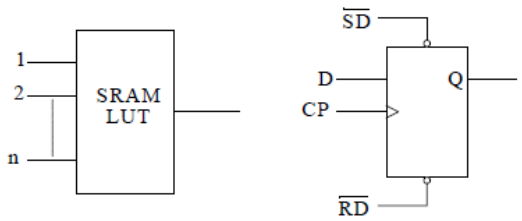


Figura 2. 23: Típico elemento lógico en una FPGA basada en SRAM.
Fuente: (Karris, 2010)

Un elemento lógico, por lo general consiste en una o más tablas de consulta (LUT) basados en n entradas de RAM, donde n es un número entre 3 y 6, y uno o más flip flops. Las LUTs se utilizan para implementar lógica combinacional, tal como se muestra en la figura 2.23. Una arquitectura típica

de FPGA basada en SRAM (véase la figura 2.24) en la cual las líneas negras indican cómo realizar conexiones entre dos o más elementos lógicos y puertos de E/S.

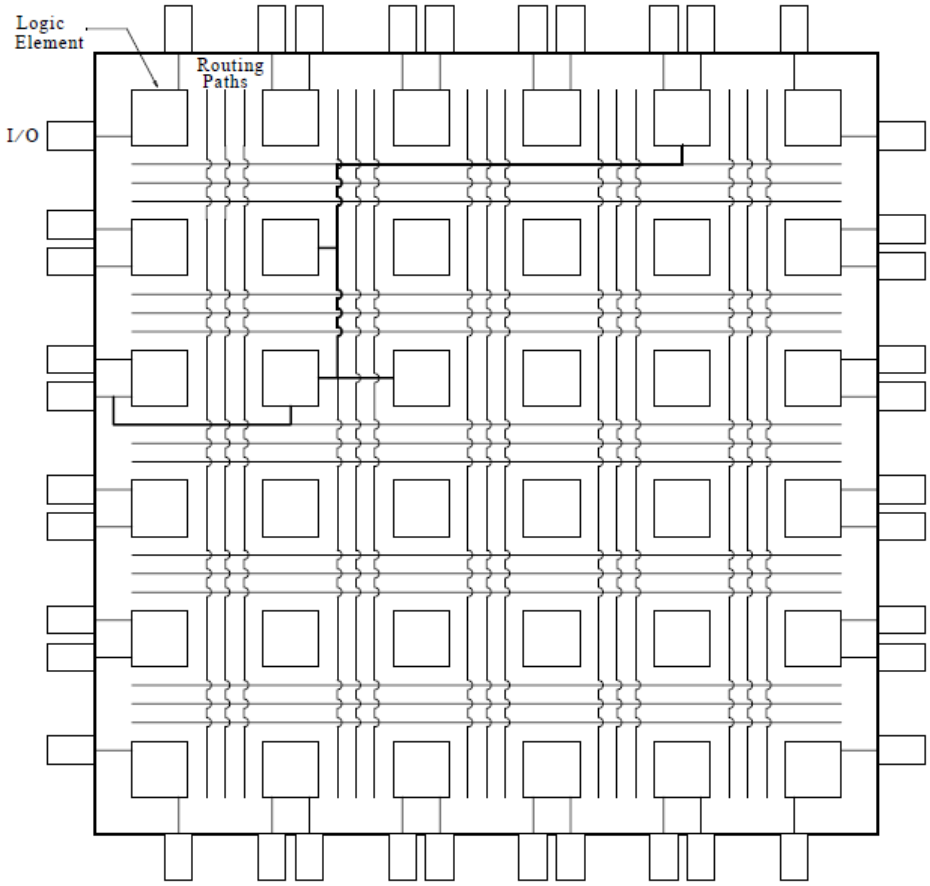


Figura 2. 24: LUTs se utilizan para implementar lógica combinacional
Fuente: (Karris, 2010)

2.5.2. FPGAs basados en antifuse.

Como se dijo anteriormente, el método antifuse cierra el circuito mediante la creación de una trayectoria conductora. Dos capas de metal sándwich una capa de no conductor, el silicio amorfo. Cuando se aplica un voltaje a esta capa intermedia, el silicio amorfo se convierte en polisilicio, que es conductora.

2.6. Fundamentos de VHDL.

VHDL es un lenguaje para describir los sistemas electrónicos digitales. Este lenguaje de descripción de hardware de circuitos integrados de muy alta velocidad (VHDL), surgió a partir del programa VHSIC en Estados Unidos en 1980. En el desarrollo este programa, se hizo evidente que había la necesidad de un lenguaje estándar para describir la estructura y función de los ICs. Por lo tanto, se desarrolló VHSIC Hardware Description Language (VHDL), y posteriormente adoptado como un estándar del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) en los Estados Unidos.

VHDL fue elaborado para cubrir una serie de necesidades en el proceso de diseño. En primer lugar, permite que la descripción de la estructura de un diseño, que se descompone en sub-diseños, y cómo esos sub-diseños están interconectados. En segundo lugar, permite la especificación de la función de diseños utilizando formas de lenguaje de programación familiarizado. Finalmente, como resultado, permite diseñar a través de simulación antes de ser implementado o fabricado, por lo que la mayoría de diseñadores pueden comparar alternativas y pruebas para corregir retardo y gastos en la creación de prototipos de hardware.

De acuerdo a (Pérez L., Soto C., & Fernández G., 2010) generalmente, VHDL tiene por lo menos tres elementos (véase la figura 2.25) que son: (a) Libraries – Biblioteca, (b) Entities – Entidades, y (c) Architectures – Arquitecturas.

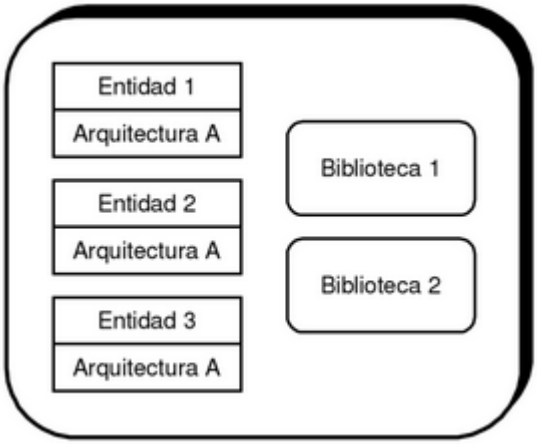


Figura 2. 25: Componentes principales para VHDL.
Fuente: (Pérez L., Soto C., & Fernández G., 2010)

En la figura 2.26 se muestra la sintaxis de una declaración básica de entidad (entities).

```
entity entity-name is
    port (signal-names : mode signal-type;
          signal-names : mode signal-type;
          ...
          signal-names : mode signal-type);
end entity-names;
```

Figura 2. 26: Componentes principales para VHDL.
Fuente: (Pérez L., Soto C., & Fernández G., 2010)

Además, de las expresiones, una declaración de entidad tiene los siguientes elementos:

- ✓ entity-name: es un identificador seleccionado por el usuario para nombrar la entidad.
- ✓ signal-names: lista separada por comas de uno o más identificadores seleccionados por el usuario para nombrar las señales de la interfaz externa.

✓ mode: dispone de cuatro palabras reservadas, especificando la dirección de la señal. En la figura 2.27 se muestra los modos in, out, buffer e inout.

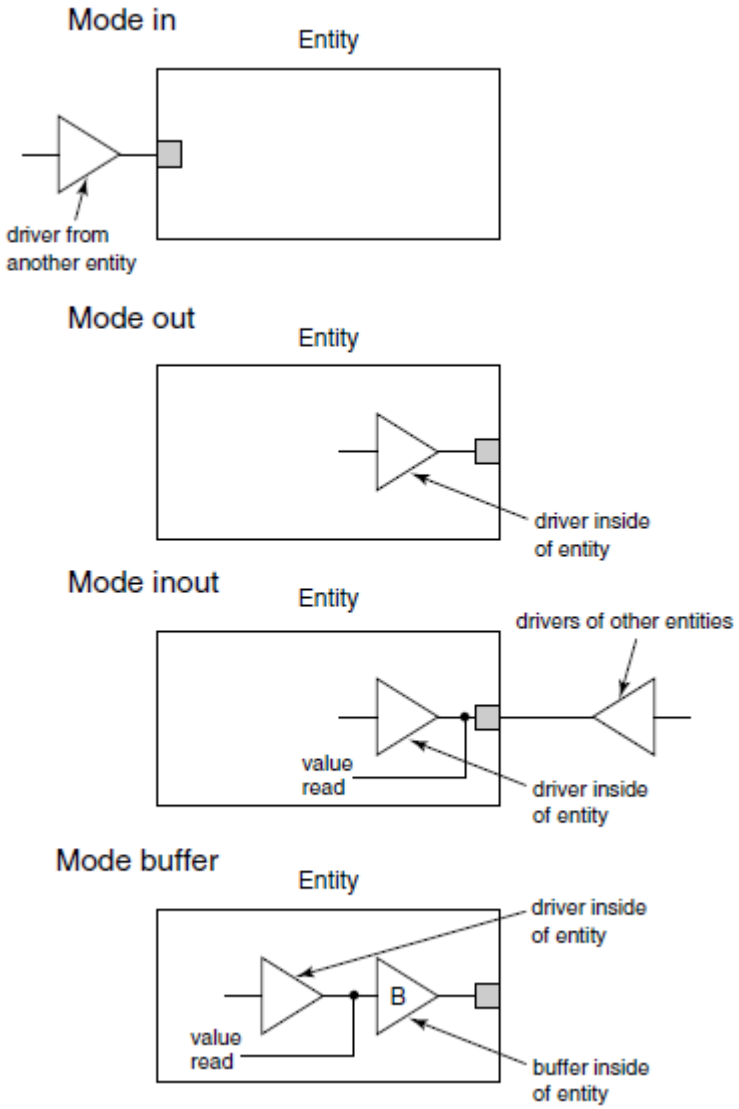


Figura 2. 27: Modos de in, out, inout y buffer para un puerto.
Fuente: (Short , 2010)

A continuación, se describen los modos:

➢ in: la señal es una entrada de la entidad.

- out: la señal es una salida de la entidad. El valor de una señal de este tipo no se puede “leer” dentro de la arquitectura de la entidad, sino únicamente por otras entidades que lo utilizan.
- buffer: la señal es una salida de la entidad, y su valor también se puede leer en el interior de la arquitectura de la entidad.
- inout: la señal se puede utilizar como una entrada o salida de la entidad. Este modo se utiliza típicamente para tres pines de estado de entrada/salida en los PLDs.

✓ signal-type: tipo de señal incorporada o definida por el usuario.

En la figura 2.28 se muestra la sintaxis de una definición básica de arquitecturas (architectures).

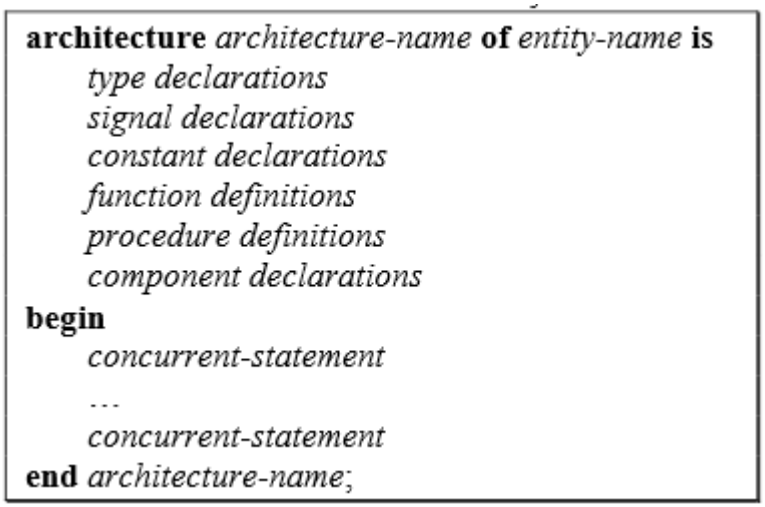
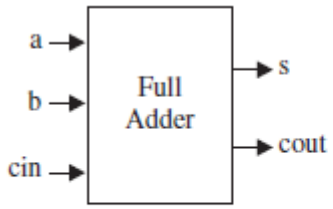


Figura 2. 28: Componentes principales para VHDL.
Fuente: (Pérez L., Soto C., & Fernández G., 2010)

En el texto de (Pedroni, 2004) se muestra el ejemplo de un sumador completo (véase la figura 2.29), donde, a y b representan los bits de entrada del sumador; Cin, es el bit de acarreo de entrada, s es el bit de la suma; y

Cout, es el bit de acarreo de salida. La tabla de verdad mostrada en la figura 2.29, s debe ser alta cuando el número de entradas que son altos sean impares; mientras que Cout, debe ser alta cuando dos o más entradas son altas.



a b cin	s cout
0 0 0	0 0
0 1 0	1 0
1 0 0	1 0
1 1 0	0 1
0 0 1	1 0
0 1 1	0 1
1 0 1	0 1
1 1 1	1 1

Figura 2. 29: Diagrama de bloques y tabla de verdad de un sumador completo.
Fuente: (Pedroni, 2004)

El código VHDL para el sumador completo se muestra en la 2.30. Se trata de una ENTITY, que es una descripción de pines (puertos) del circuito; y de una ARCHITECTURE, que describe cómo el circuito debe funcionar (Pedroni, 2004).

```
ENTITY full_adder IS
PORT (a, b, cin: IN BIT;
      s, cout: OUT BIT);
END full_adder;
-------------------------------------
ARCHITECTURE dataflow OF full_adder IS
BEGIN
   s <= a XOR b XOR cin;
   cout <= (a AND b) OR (a AND cin) OR
           (b AND cin);
END dataflow;
```

Figura 2. 30: Diagrama de bloques y tabla de verdad de un sumador completo.
Fuente: (Pérez L., Soto C., & Fernández G., 2010)

CAPÍTULO 3: APLICACIONES PRÁCTICAS EN VHDL SOBRE UNA FPGA DE0-NANO.

3.1. Introducción.

Para el desarrollo de las aplicaciones prácticas se utilizará el software de programación Quartus II de Altera. Las aplicaciones prácticas serán diseñadas a partir de sistemas combinatoriales y secuenciales para comprobar el funcionamiento de la tarjeta Altera DE0-Nano que se utilizarán en las prácticas de Laboratorio de Digitales para la formación de Ingenieros Electrónicos en Control y Automatismo. Antes del desarrollo de prácticas se describirá en la siguiente sección el software de programación Quartus II.

3.2. Desarrollo de Aplicación Práctica #1: Semáforo simple.

En la práctica, las tarjetas de Terasic DE0-NANO con un microprocesador Altera Cyclone IV puede facilitar el desarrollo de soluciones para problemas, en este caso presentaremos el diseño de un semáforo simple de dos intersecciones aplicando el lenguaje de programación VHDL.

Para analizar el problema, simplemente nos enfocaremos en una intersección simple de dos carriles cada una, este debe variar de estados cada determinado tiempo, en la figura 3.1 especificaremos la variación del mismo.

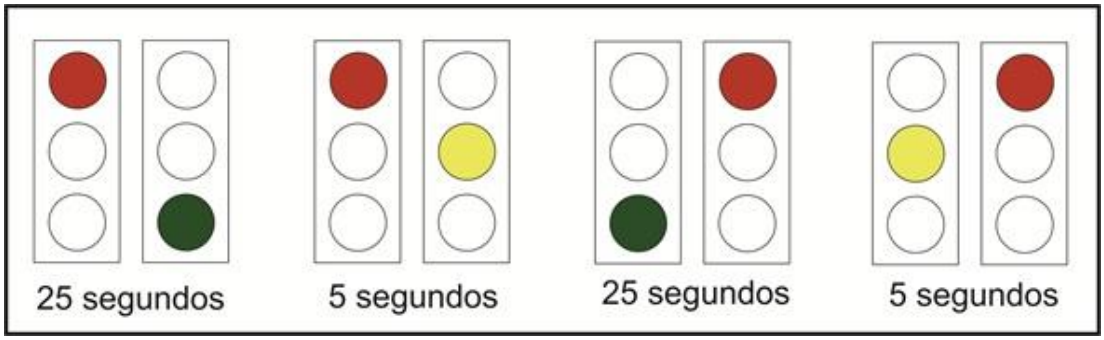


Figura 3. 1: Tiempos relativos de cambio de luces de un semáforo simple.
Elaborado por: El Autor.

Esta aplicación se desarrollará en un sencillo programa en VHDL para implementarlo en la tarjeta descrita anteriormente. Antes que nada, lo primero es iniciar un proyecto en Quartus II, tal como se muestra en la figura 3.2.

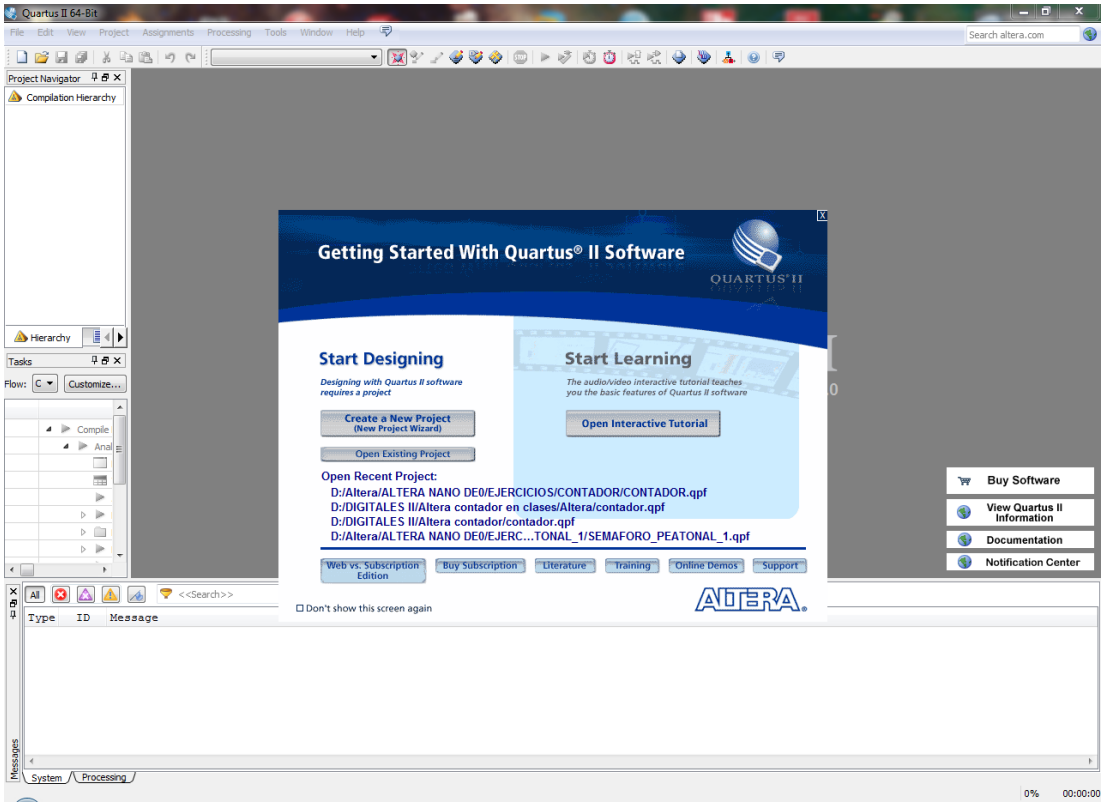


Figura 3. 2: Pantalla de bienvenida del programa Quartus II
Elaborado por: El Autor.

En nuestra pantalla de bienvenida, seleccionamos en crear un nuevo proyecto, en esta opción nos presentaran diferentes opciones como con nombre del proyecto y lugar a ubicar la carpeta de este, además se presentaran las distintas características y entre ellas tendremos que seleccionar la tarjeta en el que vamos a desarrollar nuestra práctica.

A continuación presentaremos con las siguientes imágenes las opciones que se llegaron a determinar para el desarrollo de nuestra práctica, en la figura 3.3 nos muestra la ubicación de la carpeta y nombre del proyecto, que para nuestro caso es “SEMAFOROSIMPLE”.

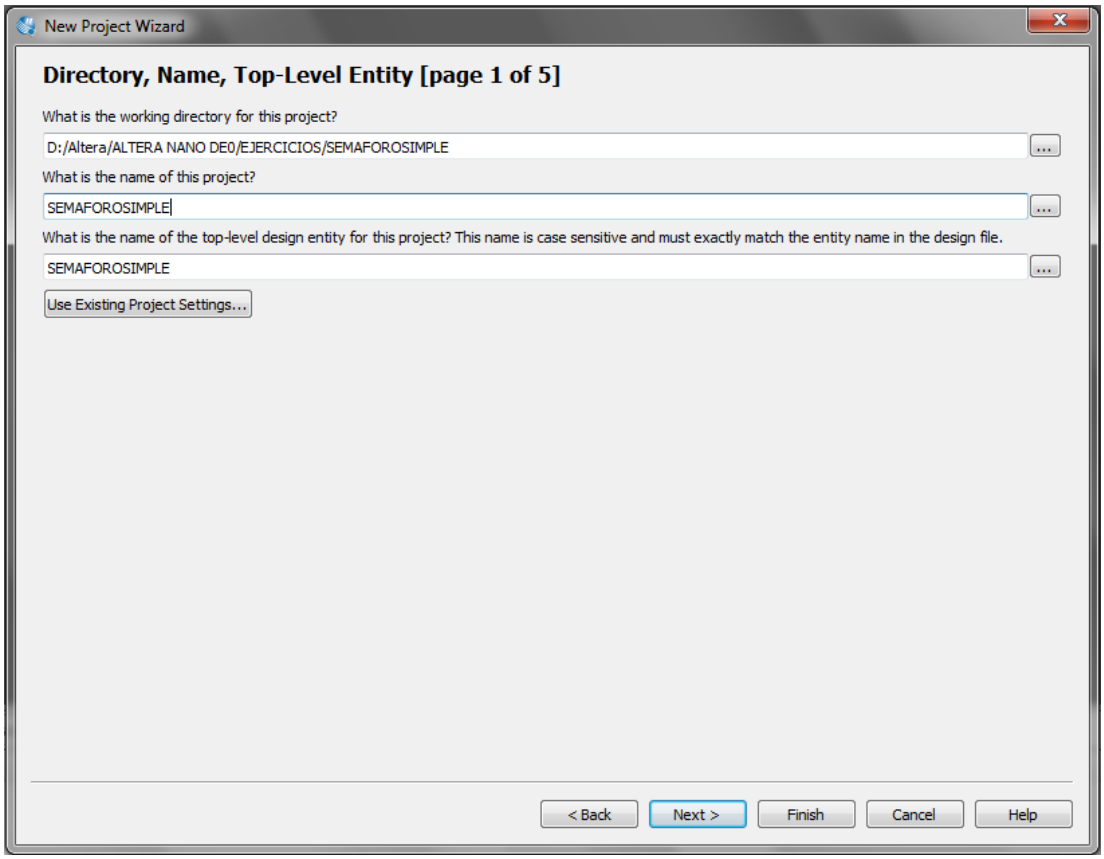


Figura 3. 3: Directorio y nombre del proyecto y entidad
Elaborado por: El Autor.

En la figura 3.4 se muestra una gran variedad de productos Altera, con la que se podrían desarrollar diferentes aplicaciones, en nuestro caso seleccionamos la familia Cyclone IV E y su variable EP4CE22F17C6.

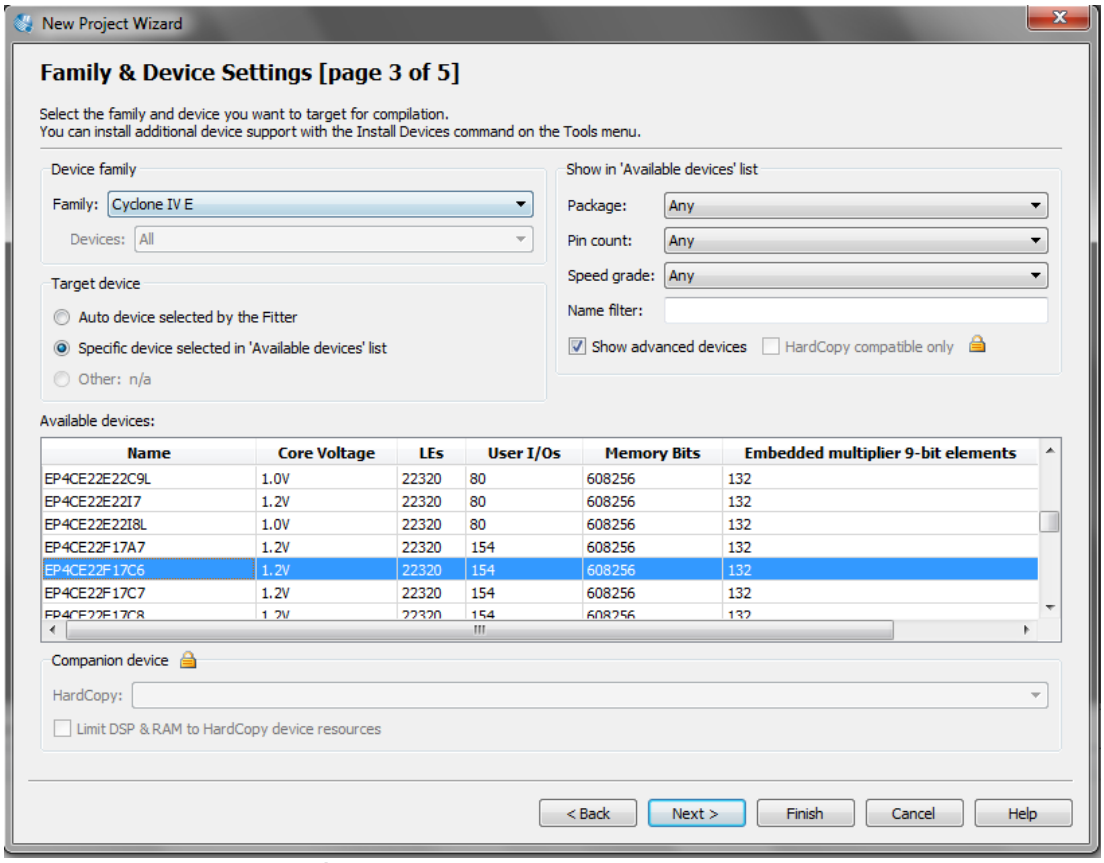


Figura 3. 4: Selección de la familia y dispositivo a utilizar para su desarrollo
Elaborado por: El Autor.

Una vez ya determinado todos los parámetros, antes de finalizar en la figura 3.5 se presenta la pantalla en el que está el resumen del dispositivo y sus características que seleccionamos anteriormente para la implementación de la práctica a realizar sobre la placa de desarrollo Terasic DE0-NANO.

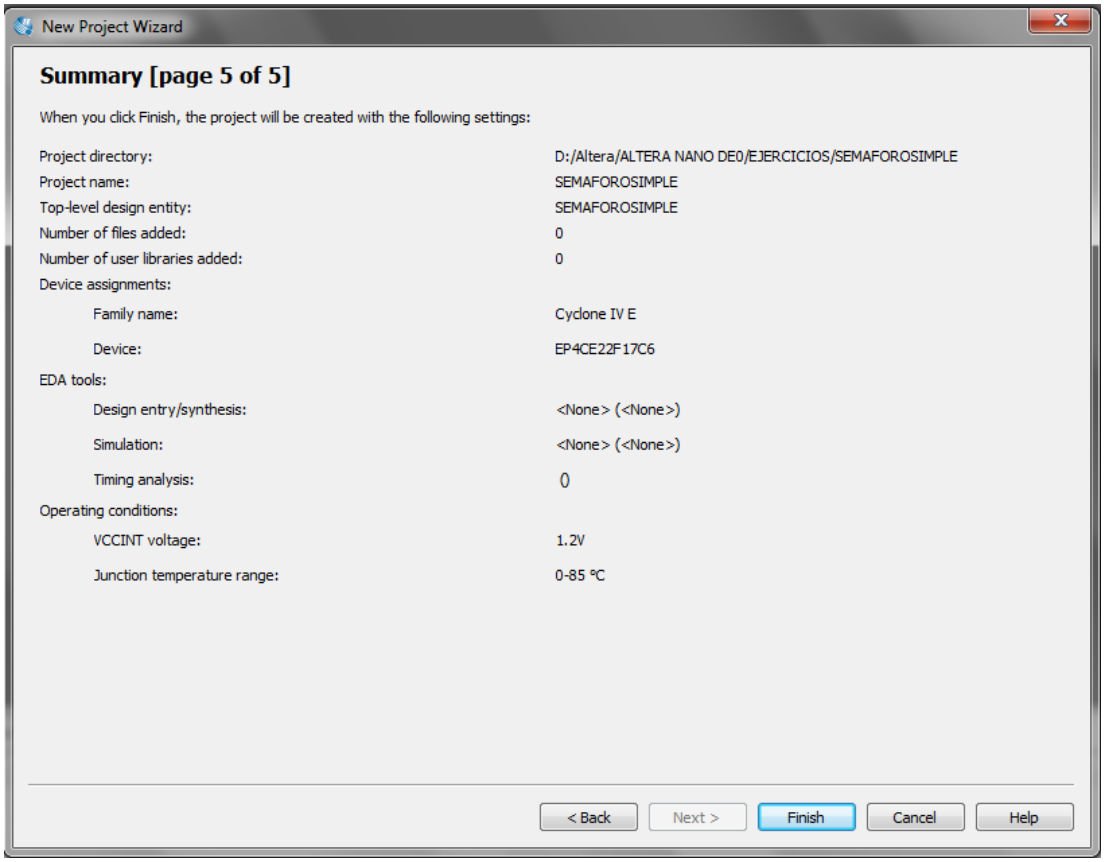


Figura 3. 5: Resumen de las opciones seleccionadas para la implementación de la práctica.
Elaborado por: El Autor.

Cabe recalcar que la herramienta Quartus, es muy robusta para la implementación de cualquier tipo de proyecto en la FPGA Altera, en ella al tener listo nuestro proyecto nos presentara que tipo de diseño deseamos implementar, ya que no solo es posible realizarlos mediante VHDL, sino también por un diagrama de bloques o captura esquemática, programación en AHDL, y distintos modos tal como se muestra en la figura 3.6. Para esta aplicación práctica lo desarrollamos en VHDL, la misma que nos permitirá medir los conocimientos adquiridos en las clases de la Carrera de Ingeniería Electrónica en Control y Automatismo.

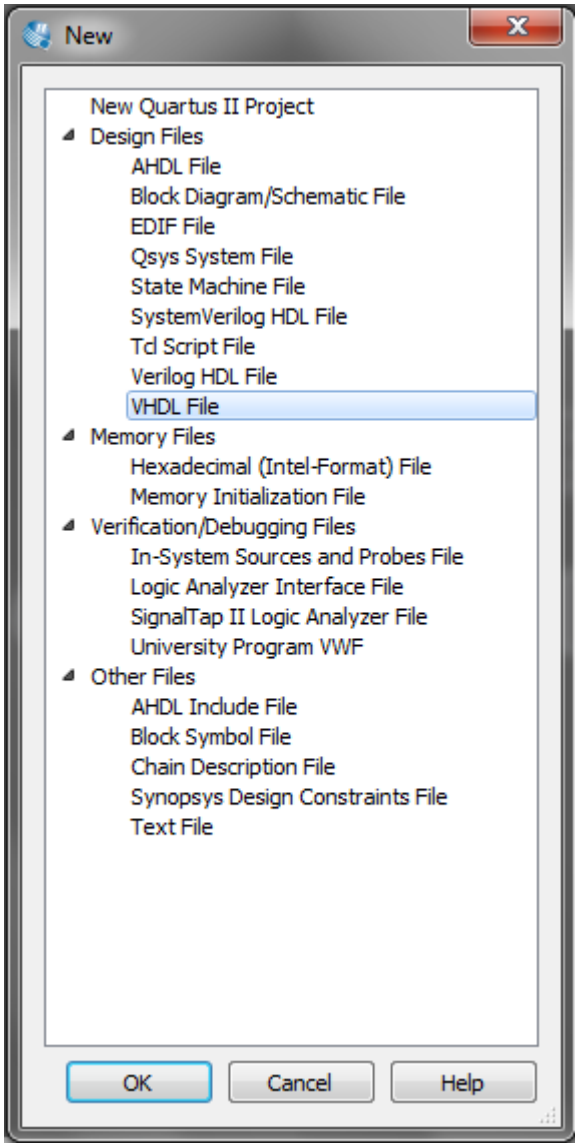


Figura 3. 6: Distintas opciones para la implementación de un proyecto en Quartus II
Elaborado por: El Autor.

Una vez seleccionado el archivo tipo VHDL, Quartus II nos proporciona una interfaz de trabajo para el desarrollo del mismo, en el que nos dedicaremos a escribir las diferentes líneas de instrucciones de nuestro programa del semáforo simple. En la figura 3.7 se muestra la estructura del programa en VHDL, que a continuación describiremos su funcionamiento por bloques de líneas.

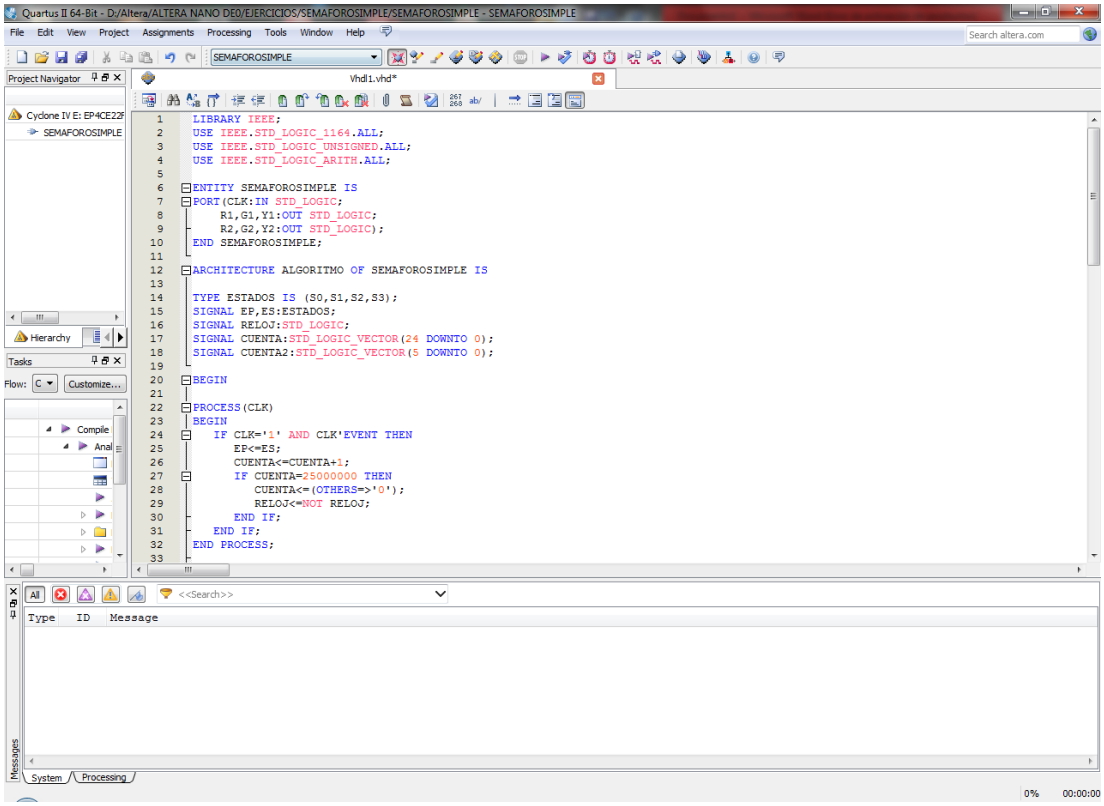


Figura 3. 7: Presentación del programa finalizado en la interfaz de trabajo de Quartus II
Elaborado por: El Autor.

En el desarrollo de una implementación para los FPGA Altera con instrucciones en VHDL se necesita estructurarlos con determinadas especificaciones, para ello primero se llaman las librerías, después la entidad, y por ultimo su arquitectura. A continuación se presentará en la figura 3.8 las librerías utilizadas y su entidad, describiendo las diferentes variables utilizadas en nuestra aplicación.

En la presente aplicación práctica llegaremos a utilizar 7 variables tipo STD_LOGIC, 6 de salidas lógicas, y una de entrada. Las 6 variables de salidas lógicas, R1, G1, Y1, R2, G2, Y2, son las que nos ofrecerán el encendido de

los leds a colores de nuestro semáforo, en cambio la variable CLK será la señal de reloj de entrada para nuestra FPGA, que en nuestra tarjeta DE0_NANO está definido por un cristal de 50MHZ en el PIN_R8 de acuerdo al pin planner de la tarjeta DE0-Nano.

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   USE IEEE.STD_LOGIC_UNSIGNED.ALL;
4   USE IEEE.STD_LOGIC_ARITH.ALL;
5
6   ENTITY SEMAFOROSIMPLE IS
7   PORT(CLK:IN STD_LOGIC;
8        R1,G1,Y1:OUT STD_LOGIC;
9        R2,G2,Y2:OUT STD_LOGIC);
10  END SEMAFOROSIMPLE;
11
```

Figura 3. 8: Librerías utilizadas y su entidad implementadas para esta practica
Elaborado por: El Autor.

La arquitectura es parte fundamental de la estructura de un programa en VHDL, este lleva en su interior todos los procesos que puede realizar una FPGA simultáneamente, utilizando las variables que anteriormente describimos. Utilizaremos cuatro estrados S0, S1, S2 y S3, estos de tipo Type, y también varias variables tipo Signal que pertenecerán a Estados, tal como se ilustra en la figura 3.9. Después de escribir el inicio o cabecera de nuestra arquitectura, empezaremos a describir los procesos que son parte de la aplicación práctica.

```
12  ARCHITECTURE ALGORITMO OF SEMAFOROSIMPLE IS
13
14  TYPE ESTADOS IS (S0,S1,S2,S3);
15  SIGNAL EP,ES:ESTADOS;
16  SIGNAL RELOJ:STD_LOGIC;
17  SIGNAL CUENTA:STD_LOGIC_VECTOR(24 DOWNTO 0);
18  SIGNAL CUENTA2:STD_LOGIC_VECTOR(5 DOWNTO 0);
19
20  BEGIN
```

Figura 3. 9: Cabecera de la arquitectura de nuestra practica en FPGA Alterna DE0_NANO
Elaborado por: El Autor.

EL proceso CLK presentado en la figura 3.10 nos demuestra las líneas su secuencia para poder trabajar con el oscilador de 50MHZ que tenemos en la tarjeta DE0_NANO, contando los flacos o pulsaciones.

```
21
22  PROCESS(CLK)
23  BEGIN
24     IF CLK='1' AND CLK'EVENT THEN
25        EP<=ES;
26        CUENTA<=CUENTA+1;
27        IF CUENTA=25000000 THEN
28           CUENTA<=(OTHERS=>'0');
29           RELOJ<=NOT RELOJ;
30        END IF;
31     END IF;
32  END PROCESS;
33
```

Figura 3. 10: Proceso CLK realizado para nuestro contador de 25 segundos
Elaborado por: El Autor.

El proceso Reloj toma el juego de datos que realiza el proceso anterior CLK y los transforma en segundos, así cada variación de la variable reloj de 1 a 0 lógico se convertirá en un segundo el numero binario 111011 (véase la figura .13).

```
34  PROCESS(RELOJ)
35  BEGIN
36     IF RELOJ='1' AND RELOJ'EVENT THEN
37        CUENTA2<=CUENTA2+1;
38        IF CUENTA2="111011" THEN
39           CUENTA2<="000000";
40        END IF;
41     END IF;
42  END PROCESS;
43
```

Figura 3. 11: Demostración del proceso RELOJ para tomar su tiempo estimado.
Elaborado por: El Autor.

A continuación, se presentará los cambios de estados que existen dependiendo de un estado anterior a uno futuro, estos harán que se enciendan

o apaguen los leds dependiendo de los valores de las variables EP, ES, S0, S1, S2, S3, tal como se muestra en la figura 3.12.

```
44  PROCESS(EP,CUENTA2)
45  BEGIN
46     ES<=EP;
47     CASE EP IS
48           WHEN S0=>R1<='0';
49                 Y1<='0';
50                 G1<='1';
51                 R2<='1';
52                 Y2<='0';
53                 G2<='0';
54                 IF CUENTA2="011001" THEN
55                   ES<=S1;
56                 ELSE
57                   ES<=S0;
58                 END IF;
59           WHEN S1=>R1<='0';
60                 Y1<='1';
61                 G1<='0';
62                 R2<='1';
63                 Y2<='0';
64                 G2<='0';
65                 IF CUENTA2="011110" THEN
66                   ES<=S2;
67                 ELSE
68                   ES<=S1;
69                 END IF;
70           WHEN S2=>R1<='1';
71                 Y1<='0';
72                 G1<='0';
73                 R2<='0';
74                 Y2<='0';
75                 G2<='1';
76                 IF CUENTA2="110111" THEN
77                   ES<=S3;
78                 ELSE
79                   ES<=S2;
80                 END IF;
81           WHEN S3=>R1<='1';
82                 Y1<='0';
83                 G1<='0';
84                 R2<='0';
85                 Y2<='1';
86                 G2<='0';
87                 IF CUENTA2="000000" THEN
88                   ES<=S0;
89                 ELSE
90                   ES<=S3;
91                 END IF;
```

Figura 3. 12: Secuencia de condiciones tomando en cuenta sus estados para su desarrollo.
Elaborado por: El Autor.

Por último, para todo tipo de proyecto diseñado en VHDL es necesario terminar con el o los procesos realizados, y finalizar con su arquitectura, tal como se muestra en la figura 3.13.

```
92      END CASE;
93    END PROCESS;
94
95  END ALGORITMO;
96
```

Figura 3. 13: Finalización de la arquitectura del algoritmo de semáforo.
Elaborado por: El Autor.

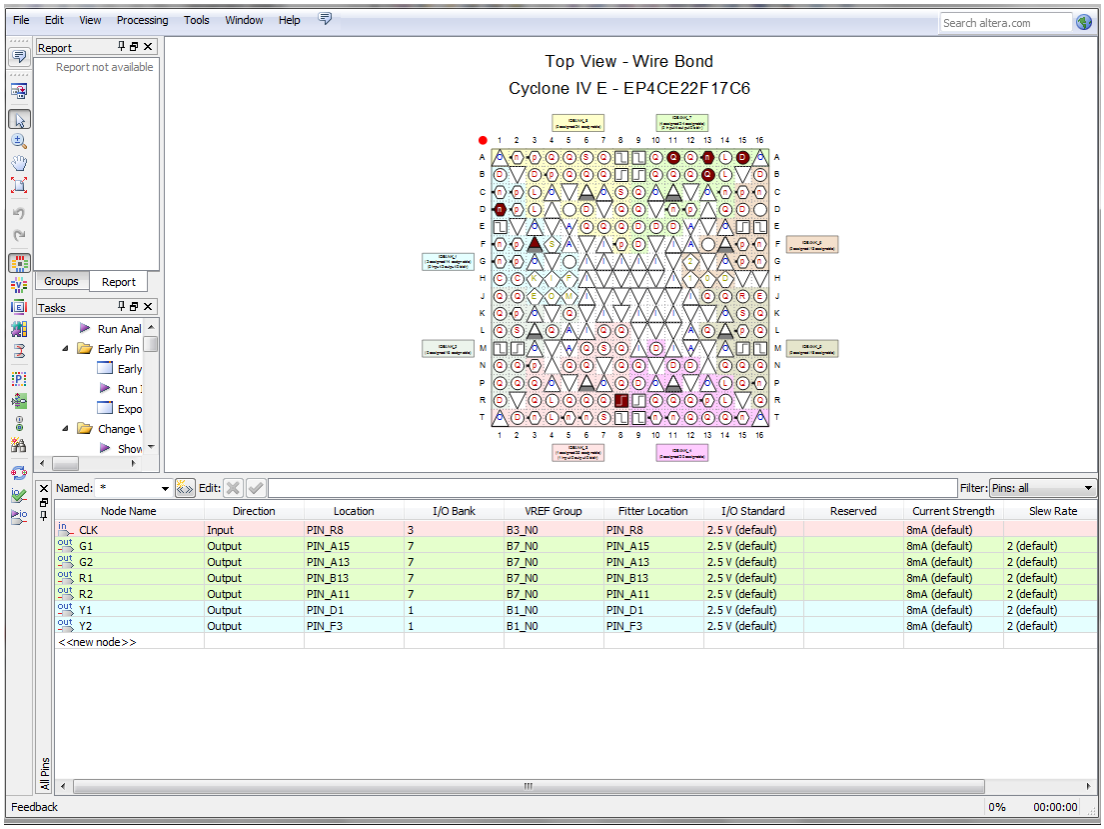


Figura 3. 14: Asignación de los pines de entradas y salidas para nuestro semáforo
Elaborado por: El Autor.

Después, de finalizar un proyecto en VHDL, siempre se debe compilar el algoritmo, y de existir errores se deberá corregir. Posterior a las correcciones, se debe compilar nuevamente y de no presentar errores, se procede a

seleccionar los pines de acuerdo al pin planner, este último se configuran los pines de entradas y salidas para ser implementado sobre la tarjeta DE0-Nano. En la figura 3.14 se muestra los pines escogidos para verificar que el algoritmo desarrollado funcione correctamente en la tarjeta DE0-Nano.

En resumen contaremos con la presentación de un diagrama de estados para nuestro semáforo en la siguiente figura 3.15 y también su funcionamiento en la foto 1.

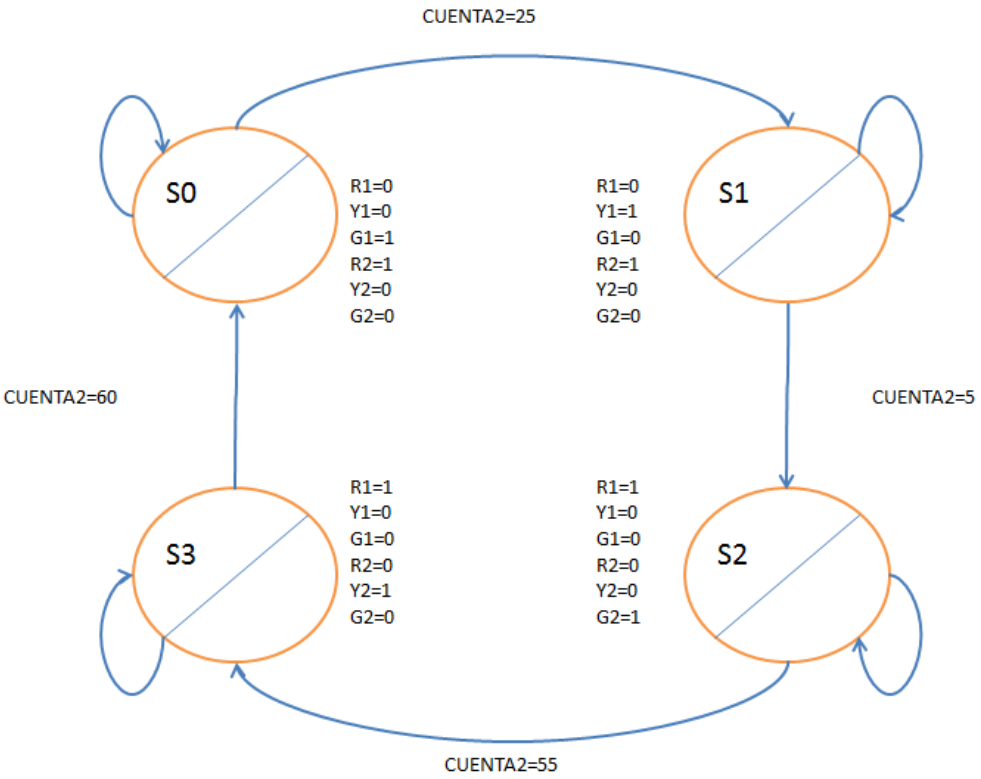


Figura 3. 15: Diagramas de estado para la aplicación práctica de un semáforo.
Elaborado por: El Autor.

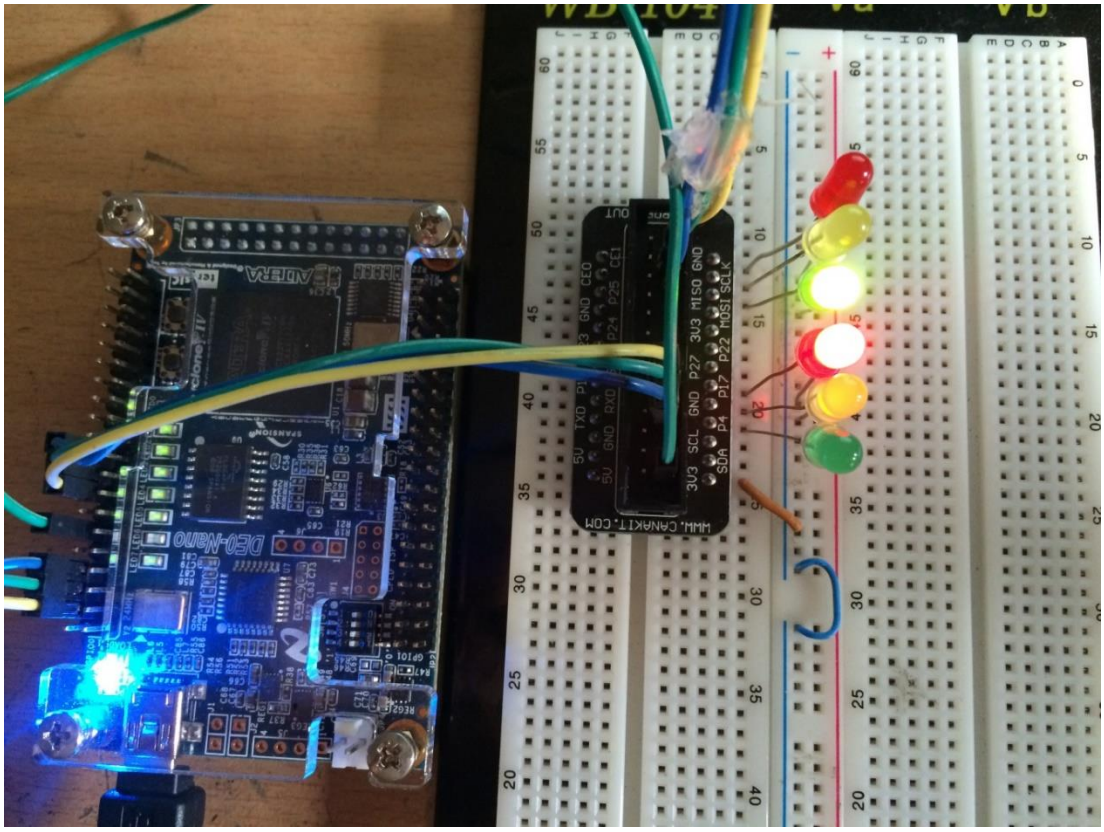


Figura 3. 16: Funcionamiento de la aplicación práctica 1.
Elaborado por: El Autor.

3.3. Desarrollo de Aplicación Práctica #2: Máquinas de estados “ASM”.

Para nuestra siguiente práctica, propusimos realizar una máquina de estados algorítmica.

Inicialmente realizamos un diagrama como se muestra en la figura 3.17 y una tabla que nos servirá a referenciar nuestra posición en el momento de comprobar el funcionamiento de nuestro código, tabla 3.1.

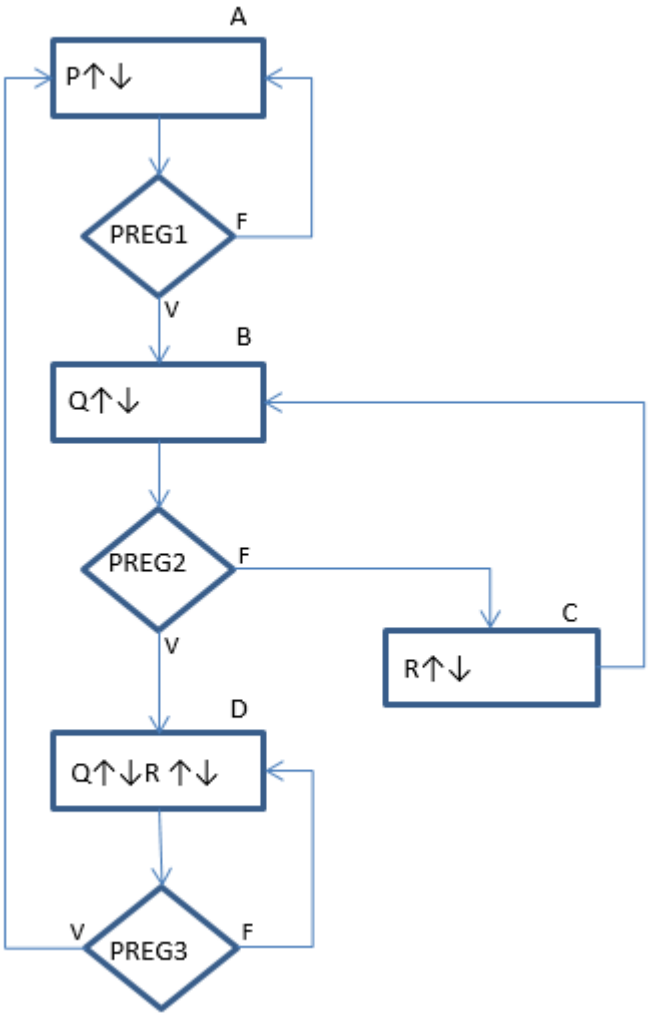


Figura 3. 17: Diagrama de flujo para una máquina de estados.
Elaborado por: El Autor.

Tabla 3. 1: Estados correspondientes para la aplicación práctica 2.

ESTADOS	P	Q	R	Ta	Tb	Tc	Td
A	1	0	0	1	0	0	0
B	0	1	0	0	1	0	0
C	0	0	1	0	0	1	0
D	0	1	1	0	0	0	1

Elaborado por: El Autor.

En el diagrama tenemos los estados A, B, C y D con sus respectivas variables que en nuestro caso se activaran en alto. En la tabla describimos como variables Ta, Tb, Tc, Td, para poder visualizarlos en la placa físicamente encendiendo un led y poder tener conocimiento en qué estado nos encontramos en el momento del test.

Una vez conseguido establecer el ejercicio a realizar, procedemos a utilizar Quartus II para describir el codigo VHDL según lo requerido, en la siguiente figura 3.18 demuestra un capture de cómo queda impreso nuestro programa

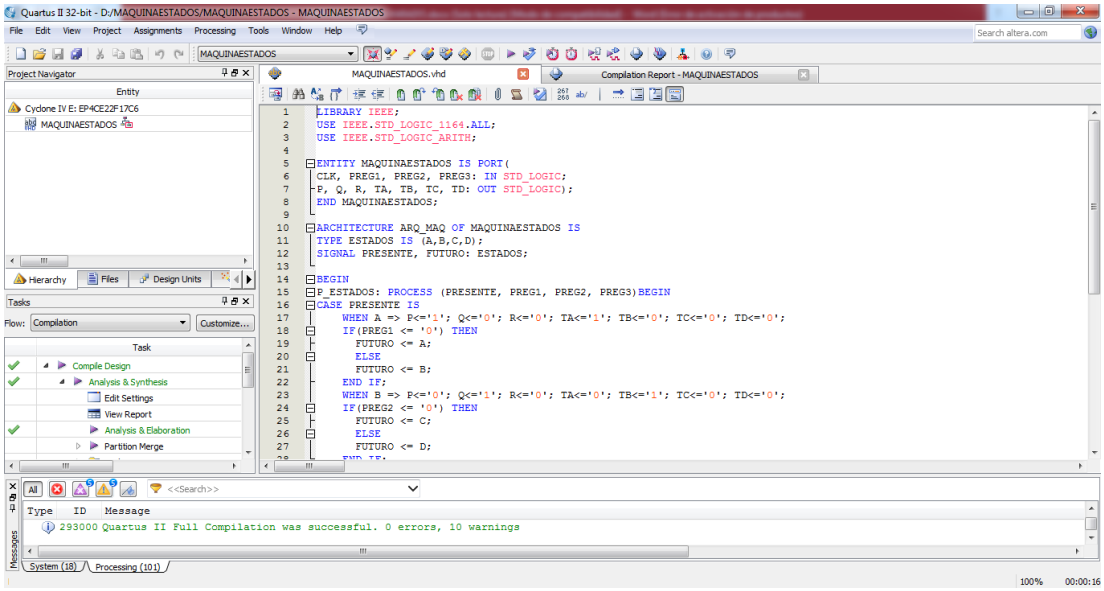


Figura 3. 18: Captura de pantalla del proyecto 2 compilado sin errores.
Elaborado por: El Autor.

A continuación describiremos la estructura de nuestro programa por bloques.

En la figura 3.19 se presenta como encabezado del código el llamado de las librerías que se llegaran a utilizar en nuestro programa VHDL, y a continuación describimos la entidad conteniendo así las variables y de qué

tipo son para nuestra práctica, en nuestro caso asignamos CLK, PREG1, PREG2, PREG3, como variables de entrada lógica, y P, Q, R, TA, TB, TC y TD como variables de salida de estado lógico, estas nos presentaran en el encendido de un led de nuestra placa DE0-NANO el estado en el que se encuentra en ese momento el programa y su respuesta antes las variaciones de reloj.

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   USE IEEE.STD_LOGIC_ARITH;
4
5   ENTITY MAQUINAESTADOS IS PORT(
6   CLK, PREG1, PREG2, PREG3: IN STD_LOGIC;
7   P, Q, R, TA, TB, TC, TD: OUT STD_LOGIC);
8   END MAQUINAESTADOS;
9
```

Figura 3. 19: Descripción de librerías y entidades para el logaritmo del proyecto 2.
Elaborado por: El Autor.

Nuevamente como en la práctica anterior, escribimos el código de la arquitectura, en el que se verá presente variables tipo estados A, B, C, D y de tipo signal PRESENTE, FUTURO, ESTADOS, para su desarrollo de diferentes procesos.

El proceso P_ESTADOS, es simplemente para la asignación de resultados dependiendo de su estado PRESENTE o FUTURO. Utilizamos la sentencia condicional CASE para comparar los estados presentes en determinado tiempo, como se presenta a continuación en la figura 3.20.

```
10  ARCHITECTURE ARQ_MAQ OF MAQUINAESTADOS IS
11    TYPE ESTADOS IS (A,B,C,D);
12    SIGNAL PRESENTE, FUTURO: ESTADOS;
13  BEGIN
14  P_ESTADOS: PROCESS (PRESENTE, PREG1, PREG2, PREG3)BEGIN
15  CASE PRESENTE IS
16      WHEN A => P<='1'; Q<='0'; R<='0'; TA<='1'; TB<='0'; TC<='0'; TD<='0';
17      IF(PREG1 <= '0') THEN
18        FUTURO <= A;
19        ELSE
20        FUTURO <= B;
21      END IF;
22      WHEN B => P<='0'; Q<='1'; R<='0'; TA<='0'; TB<='1'; TC<='0'; TD<='0';
23      IF(PREG2 <= '0') THEN
24        FUTURO <= C;
25        ELSE
26        FUTURO <= D;
27      END IF;
28      WHEN C => P<='0'; Q<='0'; R<='1'; TA<='0'; TB<='0'; TC<='1'; TD<='0';
29      FUTURO <= B;
30      WHEN D => P<='0'; Q<='1'; R<='1'; TA<='0'; TB<='0'; TC<='0'; TD<='1';
31      IF(PREG3 <= '0') THEN
32        FUTURO <= D;
33        ELSE
34        FUTURO <= A;
35      END IF;
36  END CASE;
37  END PROCESS P_ESTADOS;
```

Figura 3. 20: Descripción de arquitectura cabecera y proceso de estados para el proyecto 2.
Elaborado por: El Autor.

Para poder tener un progreso en el tiempo es necesario crear un proceso que funcione simultáneamente con el descrito anteriormente, y es ahí cuando creamos un reloj manual, el cual por cada pulsación de un botón decimos al programa que se continua al siguiente evento de tiempo, claro que si en los dip switch no se varia ninguna condición este no cambiara de estado.

En la figura 3.21 describe la estructura de nuestro proceso para un reloj manual, a este se lo llamó REL.

```
38
39  REL: PROCESS (CLK) BEGIN
40  IF(CLK'EVENT AND CLK='1') THEN
41    PRESENTE <= FUTURO;
42  END IF;
43  END PROCESS REL;
44
45  END ARQ_MAQ;
46
47
48
```

Figura 3. 21: Proceso de la señal CLK manual para el proyecto 2.
Elaborado por: El Autor.

Para que nuestro código pueda tener efecto en nuestra placa DE0_NANO es necesario describir cuáles serán los pines de entrada y salida, siendo los de entrada los dip switch que viene ya instalado en la misma y los de salida los leds.

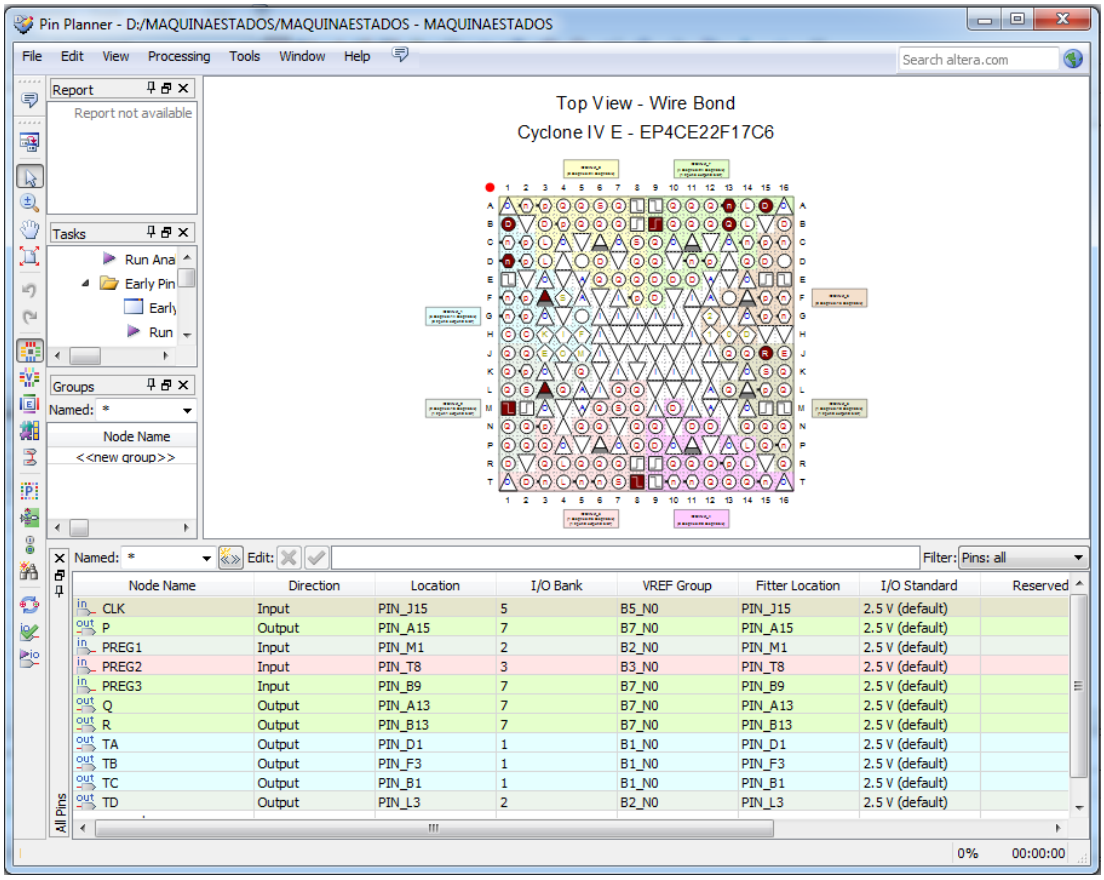


Figura 3. 22: Configuración del pin planner DE0-Nano del proyecto 2.
Elaborado por: El Autor.

Para ello es necesario describirlos mediante el PIN PLANNER que se muestra en la figura 3.22.

Pasamos el programa a la tarjeta mediante la herramienta Programmer que posee el software Quartus II, y por último este se ejecutara automáticamente en ella. En la figura 3.23 se puede observar cómo es posible realizar este procedimiento.

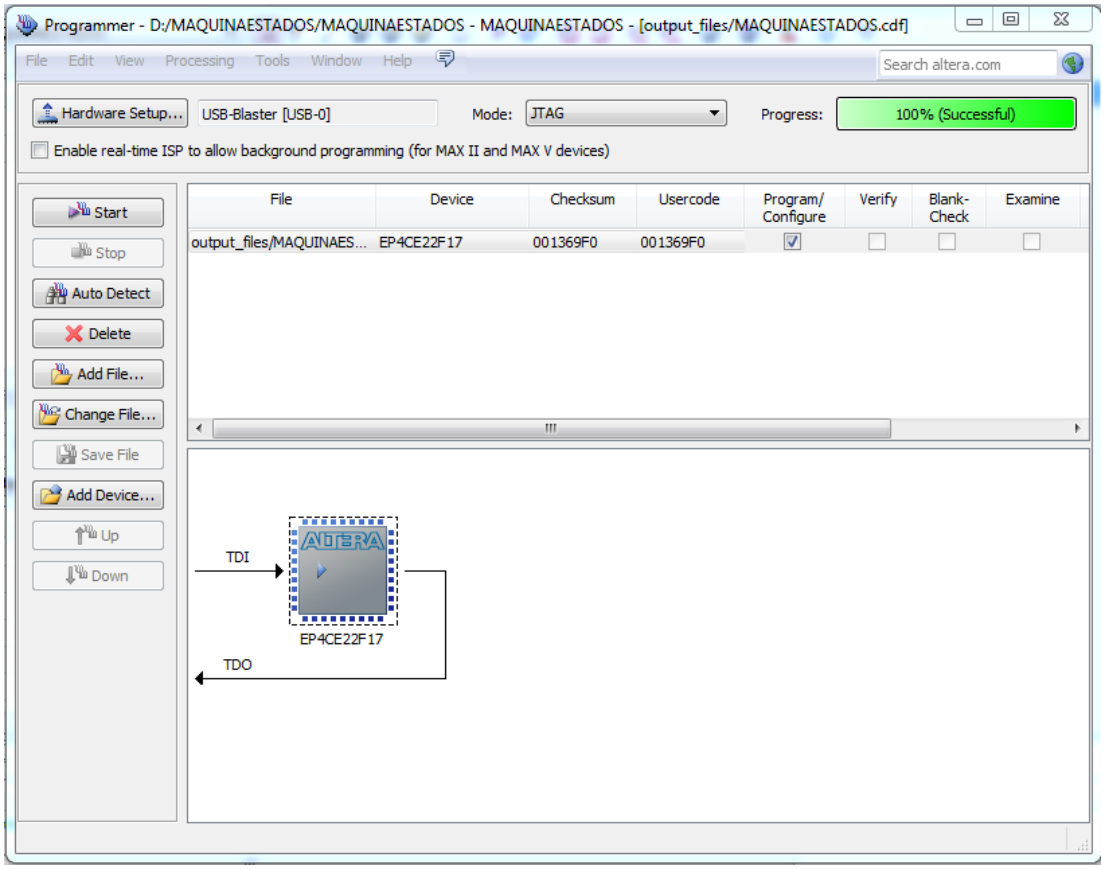


Figura 3. 23: Herramienta de programación del software Quartus II.
Elaborado por: El Autor.

Físicamente se puede comprobar el funcionamiento de nuestro código VHDL después de haber seguido un riguroso procedimiento de ensayo y error. En la figura 3.24 se puede observar la tarjeta DE0_NANO ejecutando la

práctica, pudimos capturar la imagen en el último estado D, donde tenemos encendido dos leds que representan a Q y R, y así también el ultimo led del extremo izquierdo que representa el estado D.

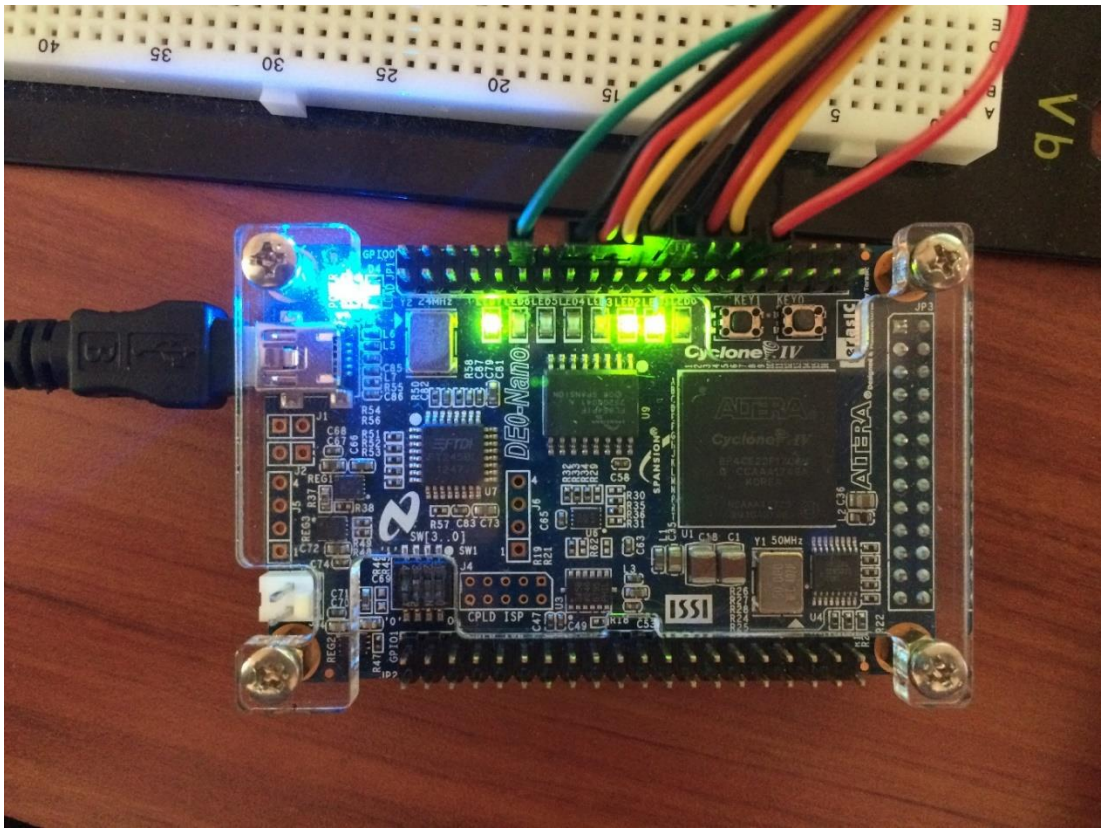


Figura 3. 24: Funcionamiento de la aplicación práctica 2.
Elaborado por: El Autor.

CAPÍTULO 4: CONCLUSIONES Y RECOMENDACIONES.

4.1. Conclusiones.

- La descripción realizada en la fundamentación teórica permitió evidenciar las potencialidades de los dispositivos lógicos programables, tales como PLAs, PALs, CPLDs y en especial de las FPGAs, esta última es la más robusta de los dispositivos programables cuyos mayores fabricantes en el mundo son Xilinx y Altera.

- El dispositivo programable a utilizar es la FPGA DE0-Nano de Altera, está resulto ser muy robusta a pesar del tamaño que tiene. Aunque se pudo utilizar otros dispositivos de Altera, se escogió a DE0-Nano para evidenciar sus potencialidades y que servirá como ayuda académica para el proceso de aprendizaje de los estudiantes de la Carrera de Ingeniería Electrónica en Control y Automatismo.

- En el capítulo 3 se realizaron el diseño de sistemas combinatoriales y secuenciales a través de algoritmos en VHDL y programados en la FPGA DE0-Nano de Altera, los cuales funcionaron correctamente al momento de verificar la operatividad y robustez de la tarjeta DE0-Nano.

4.2. Recomendaciones.

- Utilizar los dispositivos lógicos programables como la FPGA DE0-Nano de Altera para las asignaturas de Sistemas Digitales y en especial para prácticas de Laboratorio de Digitales.

- Aprovechar las potencialidades del dispositivo DE0-Nano que permitió tanto a Docentes desarrollar trabajos de investigación aplicada con fines de acreditación y para los estudiantes que pueden proponer nuevas aplicaciones prácticas de futuros trabajos de titulación y porque no para implementar robots que servirán de mucho en competencias de robótica nacionales e internacionales.

REFERENCIAS BIBLIOGRÁFICAS

Aparicio O., A., & Ponluisa M., N. (2014). *Estudio comparativo de los lenguajes HDL y su aplicación en la implementación del laboratorio de sistemas digitales avanzados mediante FPGAs en la EIE-CRI.* Riobamba: Repositorio de la Escuela Superior Politécnica de Chimborazo.

Bozich, E. (2005). *Introducción a los Dispositivos FPGA: Análisis y ejemplos de diseño.* La Plata, Argentina: Repositorio de la Universidad Nacional de la Plata.

Brown, S., & Rose, J. (2010). *FPGA and CPLD Architectures: A Tutorial.* Toronto: Universidad de Toronto.

Cypress. (10 de Enero de 2016). *IC72.* Obtenido de http://www.ic72.com/pdf_file/7/74327.pdf

Cypress-1. (11 de Enero de 2016). *Advanced Digital Electronics.* Obtenido de http://eent3.lsbu.ac.uk/units/ade/warpdocs/CDrom/cdrom/docs/product/cplds/flsh370.pdf

Davison, R. (17 de Noviembre de 2015). *Department of Information Systems.* Obtenido de http://www.is.cityu.edu.hk/staff/isrobert/phd/ch3.pdf

De la A S., L. (2015). *Módulo de Control aplicando tecnología FPGA para el análisis del proceso de mezcla de potabilización del agua, nivel de PH*

y control de flujo. La Libertad, Ecuador: Repositorio de la Universidad Estatal Península de Santa Elena.

Karris, S. T. (2010). *Introduction to CPLDs and FPGAs.* Orchard.

Kuon, I., Tessier, R., & Rose, J. (2008). FPGA Architecture: Survey and Challenges. *Foundations and Trends in Electronic Design Automation*, 135-253.

Montejo R., M. (10 de Enero de 2016). *Cursos de Electrónica*. Obtenido de PLD's - Dispositivos de Lógica Programable: http://pablin.com.ar/electron/cursos/intropld/index.htm

Pedroni, V. (2004). *Circuit Design with VHDL.* TLFeBook.

Pérez L., S., Soto C., E., & Fernández G., S. (2010). *Diseño de Sistemas Digitales con VHDL.* Madrid: Paraninfo.

Rajasekar, S., Philominathan, P., & Chinnathambi, V. (19 de Noviembre de 2016). *arXiv.* Obtenido de http://arxiv.org/pdf/physics/0601009.pdf

Short , K. (2010). *VHDL for Engineers.* New Jersey: Pearson Education.

Xilinx. (13 de Enero de 2016). *Xilinx All Programmable.* Obtenido de http://www.xilinx.com/support/documentation/data_sheets/DS063.pdf

DECLARACIÓN Y AUTORIZACIÓN

Yo, Alvarado Bonilla Juan Carlos, con C.C: # 092479716-0 autor del trabajo de titulación: Desarrollo de Algoritmos en VHDL sobre una FPGA DE0-NANO para prácticas de Laboratorio de Digitales en la carrera de Ingeniería Electrónica en Control y Automatismo previo a la obtención del título de **INGENIERO ELECTRÓNICO EN CONTROL Y AUTOMATISMO** en la Universidad Católica de Santiago De Guayaquil.

1.- Declaro tener pleno conocimiento de la obligación que tienen las instituciones de educación superior, de conformidad con el Artículo 144 de la Ley Orgánica de Educación Superior, de entregar a la SENESCYT en formato digital una copia del referido trabajo de titulación para que sea integrado al Sistema Nacional de Información de la Educación Superior del Ecuador para su difusión pública respetando los derechos de autor.

2.- Autorizo a la SENESCYT a tener una copia del referido trabajo de titulación, con el propósito de generar un repositorio que democratice la información, respetando las políticas de propiedad intelectual vigentes.

Guayaquil, 17 de marzo de 2016

REPOSITORIO NACIONAL EN CIENCIA Y TECNOLOGÍA			
FICHA DE REGISTRO DE TESIS/TRABAJO DE TITULACIÓN			
TÍTULO Y SUBTÍTULO:	DESARROLLO DE ALGORITMOS EN VHDL SOBRE UNA FPGA DE0-NANO PARA PRÁCTICAS DE LABORATORIO DE DIGITALES EN LA CARRERA DE INGENIERÍA ELECTRÓNICA EN CONTROL Y AUTOMATISMO		
AUTOR(ES) (apellidos/nombres):	Alvarado Bonilla, Juan Carlos		
REVISOR(ES)/TUTOR(ES) (apellidos/nombres):	Philco Asqui, Luis Orlando MsC.		
INSTITUCIÓN:	Universidad Católica de Santiago de Guayaquil		
FACULTAD:	Facultad de Educación Técnica Para el Desarrollo		
CARRERA:	Escuela de Ingeniería Electrónica en Control y Automatismo		
TITULO OBTENIDO:	Ingeniero Electrónico en Control y Automatismo		
FECHA DE PUBLICACIÓN:	17 de marzo de 2016	**No. DE PÁGINAS:**	72
ÁREAS TEMÁTICAS:	Desarrollo de Algoritmos, Lenguaje VHDL, Sistemas Digitales		
PALABRAS CLAVES/ KEYWORDS:	VHDL, ALGORITMOS, ALTERA, LABORATORIO DE DIGITALES, ELECTRÓNICA, PROGRAMACIÓN		

RESUMEN/ABSTRACT (150-250 palabras):

Los Sistemas Electrónicos Digitales son de gran importancia en la formación de futuros profesionales de la Carrera de Ingeniería Electrónica en Control y Automatismo. Los avances logrados en la electrónica digital, permitieron que se desarrollen tecnologías, tal como, dispositivos lógicos programables simples y complejos. En este último, dispositivo complejo se lo conoce como CPLD, y se incluyen al arreglo de compuertas programables en campo (FPGA). Este dispositivo FPGA se pueden implementar diversas aplicaciones de compuertas lógicas, circuitos combinacionales, circuitos secuenciales, diseño de controladores, entre otras, cuyos contenidos cumplen con los programas de estudio de Sistemas Digitales I y II y Laboratorio de Digitales. También, sirven para desarrollar proyectos de investigación y se incluyan aplicaciones de robótica. El dispositivo FPGA escogido fue DE0-Nano de Altera, en el mismo se implementaron aplicaciones para demostrar la utilidad y funcionalidad de la FPGA. Para el desarrollo del capítulo 3, se utilizó el software Quartus II para programar en VHDL. Los algoritmos de programación fueron realizados en VHDL e implementados en la DE0-Nano cuyo integrado es el Cyclone IV. Finalmente, este trabajo de titulación cumplió con el propósito planteado, que fue desarrollar algoritmos en VHDL y evaluados en la FPGA DE0-Nano.

ADJUNTO PDF:	☒ SI	☐ NO
CONTACTO CON AUTOR/ES:	**Teléfono:** +593-4-2473442/ +593993065632	E-mail: juan.alvarado01@cu.ucsg.edu.ec / juan.alvaradob@outlook.com
CONTACTO CON LA INSTITUCIÓN:	**Nombre:** Philco Asqui, Luis Orlando MsC.	
	Teléfono: +593980960875	
	E-mail: orlando.philco@cu.ucsg.edu.ec / orlandophilco_@hotmail.com	

SECCIÓN PARA USO DE BIBLIOTECA	
Nº. DE REGISTRO (en base a datos):	
Nº. DE CLASIFICACIÓN:	
DIRECCIÓN URL (tesis en la web):	http://repositorio.ucsg.edu.ec/handle/123456789/232