UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

TÍTULO:

SISTEMA DE ENTRENAMIENTO DE1 DE ALTERA: DESARROLLO DE APLICACIONES PRÁCTICAS PARA EL LABORATORIO DE DIGITALES

AUTORAS:

Sandra María Sosa Calero

Yadira María Valdez Jiménez

Previa la obtención del Título

INGENIERO EN TELECOMUNICACIONES

TUTORA:

MsC. Luzmila Ruilova Aguirre

Guayaquil, Ecuador

2015

UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

CERTIFICACIÓN

Certificamos que el presente trabajo fue realizado en su totalidad por las Srtas. **Sandra María Sosa Calero** y **Yadira María Valdez Jiménez** como requerimiento parcial para la obtención del título de INGENIERO EN TELECOMUNICACIONES.

TUTORA

_____

MsC. Luzmila Ruilova Aguirre

DIRECTOR DE CARRERA

_____

MsC. Miguel Heras Sánchez.

Guayaquil, a los 18 días del mes de Febrero del año 2015

UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

DECLARACIÓN DE RESPONSABILIDAD

Nosotras, **Sandra María Sosa Calero** y **Yadira María Valdez Jiménez**

DECLARAMOS QUE:

El trabajo de titulación "SISTEMA DE ENTRENAMIENTO DE1 DE ALTERA: DESARROLLO DE APLICACIONES PRÁCTICAS PARA EL LABORATORIO DE DIGITALES" previa a la obtención del Título de Ingeniero en Telecomunicaciones, 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 nuestra autoría.

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

Guayaquil, a los 18 del mes de Febrero del año 2015

LAS AUTORAS

_____ _____

SANDRA MARÍA SOSA CALERO YADIRA MARÍA VALDEZ JIMÉNEZ

UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

AUTORIZACIÓN

Nosotras, **Sandra María Sosa Calero** y **Yadira María Valdez Jiménez**

Autorizamos a la Universidad Católica de Santiago de Guayaquil, la publicación, en la biblioteca de la institución del Trabajo de Titulación: "SISTEMA DE ENTRENAMIENTO DE1 DE ALTERA: DESARROLLO DE APLICACIONES PRÁCTICAS PARA EL LABORATORIO DE DIGITALES", cuyo contenido, ideas y criterios es de nuestra exclusiva responsabilidad y autoría.

Guayaquil, a los 18 del mes de Febrero del año 2015

LAS AUTORAS

_____ _____

SANDRA MARÍA SOSA CALERO YADIRA MARÍA VALDEZ JIMÉNEZ

DEDICATORIA

Dedico este trabajo a mis padres, Dr. Alfredo Sosa Rodríguez y de manera muy particular a mi madre Dra. Carmen Calero de Sosa, que han sido mi apoyo incondicional en los momentos difíciles y placenteros, sin ella no hubiera podido cumplir con el objetivo de obtener mi título profesional, a mis hermanos, a mi esposo Isaac Jiménez y a mi Hij@ que pronto nacerá.

A mis padres y hermanos por su amor su cariño y sus cuidados, por haberme dado la dicha de vivir en una familia llena de amor y respeto.

A mi tía por ser mi ejemplo a seguir y ser esa persona que me alentó a luchar, me lleno de motivación y me enseñó a enfrentar los problemas con paciencia y siempre confiando en Dios.

A Daniel que es mi cómplice y mi compañero por su apoyo y amor que me ofreció en todos estos 4 años.

A todas aquellas personas que hicieron mi paso por esta Universidad de prestigio sea de mayor alegría y un grato recuerdo.

LAS AUTORAS

SANDRA MARÍA SOSA CALERO YADIRA MARÍA VALDEZ JIMENENEZ

AGRADECIMIENTO

Al culminar la etapa #3 de estudios y obtener un título universitario dejamos constancia de agradecimiento a Dios, a nuestros Padres, Hermanos, a mi Esposo Isaac Jiménez y a cada uno de los profesores que con sus conocimientos hicieron posible que obtengamos este título y de manera muy especial al Coordinador de Titulación, MSc. Fernando Palacios y nuestra Tutora, MSc. Luzmila Ruilova y a todos las personas que de una u otra forma participaron en mi formación profesional.

Gracias, muchas gracias.

LAS AUTORAS

SANDRA MARÍA SOSA CALERO YADIRA MARÍA VALDEZ JIMÉNEZ

Índice General

Índice de Figuras

Capítulo 2

Capítulo 3

Índice de Tablas

Capítulo 2

Resumen

En trabajo de titulación presente se dan varios procedimientos por los cuales los circuitos digitales pueden ser diseñados e implementados en la tarjeta DE1 de ALTERA. Se demostró la funcionalidad de la misma mediante el diseño de determinadas aplicaciones que se desarrollan en el pensum académico de Laboratorio de Digitales.

La plataforma de programación QUARTUS II es una herramienta robusta y que permite modelar circuitos a través de captura esquemática, máquinas de estados, programación HDL, VHDL y Verilog HDL. La aplicabilidad de estos fueron comprobados en el capítulo 3.

CAPÍTULO 1: GENERALIDADES DEL TRABAJO DE TITULACIÓN

1.1. Antecedentes.

Los circuitos digitales se pueden dividir en dos grupos: circuitos combinacionales y secuenciales. Los circuitos digitales combinacionales tienen sus salidas cambiadas tan pronto como sus entradas sean cambiadas. Por otra parte, los circuitos secuenciales se rigen por una señal de reloj maestro de tal manera que los cambios de salida se producen sólo en el flanco ascendente o descendente, del reloj maestro. Además, las salidas serán dependientes de la entrada sino también del estado actual del circuito. En general, un sistema digital típico consta de dos circuitos combinacionales y secuenciales.

La importancia del diseño digital y su aplicación está aumentando día a día. Los circuitos digitales tienen aplicaciones en todos los ámbitos de la vida. Pruebas experimentales de circuitos digitales pueden hacerse utilizando FPGAs y que los circuitos digitales diseñados se puedan implementar sobre la FPGA DE1 de ALTERA.

La implementación de circuitos digitales VLSI es una tarea difícil. En particular, la aplicación de circuitos de lógica difusa, incluyen aplicaciones para implementarse sobre FPGAs, así como el diseño VLSI; en particular, se da el diseño de circuitos a través de códigos Verilog. Se espera que el presente trabajo de titulación realizado pueda recorrer un largo camino para

mejorar las simulaciones y diseño de circuitos digitales para la asignatura de Laboratorio de Digitales en la formación de Ingenieros en Telecomunicaciones.

1.2. Justificación del Problema.

Los circuitos digitales mediante compuertas lógicas son implementados en protoboards en prácticas que se desarrollan en Laboratorio de Digitales, pero en la actualidad la mayoría de instituciones de educación superior, cuentan en sus laboratorios con modernas tarjetas de entrenamiento como por ejemplo Xilinx y Altera. Estas dos últimas se encuentran en el Laboratorio de Electrónica de la Facultad de Educación Técnica para el Desarrollo en la Universidad Católica de Santiago de Guayaquil.

El trabajo de titulación implementa circuitos digitales sobre la tarjeta de entrenamiento DE1 de ALTERA de FPGAs. Como aporte nuestro, dejamos 5 tarjetas para ser utilizadas en las prácticas del Laboratorio de Digitales. En total se dispondrá de aproximadamente 15 tarjetas, para lo cual cada estudiante podrá trabajar de manera individual y no como hasta la presente en grupos de 2 o 3 estudiantes.

1.3. Definición del Problema.

Necesidad de incorporar aplicaciones prácticas de circuitos digitales sobre la tarjeta DE1 de ALTERA como ayuda de enseñanza para los estudiantes de la Carrera de Ingeniería en Telecomunicaciones,

específicamente en la asignatura de Laboratorio de Digitales y también para otras asignaturas, tales como: Sistemas de Microprocesadores y Diseño Electrónico Digital.

1.4. Objetivos del Problema de Investigación.

El trabajo de titulación cuya modalidad es de implementación de circuitos digitales sobre DE1 de ALTERA.

1.4.1. Objetivo General.

Desarrollar e implementar una serie de aplicaciones prácticas para la asignatura de Laboratorio de Digitales.

1.4.2. Objetivos Específicos.

- Describir el estado del arte del arreglo de compuertas programables en el campo – FPGAs.
- Realizar diseños de circuitos digitales combinacionales y secuenciales mediante programación VHDL y captura esquemática.
- Implementar sobre la tarjeta de entrenamiento DE1 de ALTERA los diseños de circuitos digitales combinacionales y secuenciales.

1.5. Hipótesis.

A través del desarrollo de aplicaciones prácticas sobre la tarjeta de entrenamiento DE1 de Altera se pretenderá dotar de una herramienta tecnológica para obtener un mejor aprendizaje y resultados en la asignatura

de Laboratorio de Digitales de la Carrera de Ingeniería en Telecomunicaciones y también para los que estudian Ingeniería Electrónica en Control y Automatismo.

1.6. Metodología de Investigación.

La metodología de investigación utilizada para el desarrollo del trabajo de titulación es el paradigma empírico – analítico con enfoque cuantitativo. El método es experimental, ya que se realizará aplicaciones prácticas sobre la tarjeta DE1 de Altera.

CAPÍTULO 2: Estado del Arte del Arreglo de Compuertas Programables en el Campo.

El presente capítulo se describirá el arreglo de compuertas programables en el campo, conocido como FPGA *(Field Programmable Gate Array)*.

2.1. Introducción a las FPGAs.

Según Zambrano G., F. A., (20014) en su investigación describió a una FPGA como un microchip que se diseñó pensando en que puede ser reconfigurable una vez que ha sido fabricado. Mediante una FPGA se realizan implementaciones de circuitos digitales a través de compuertas lógicas, muy similares a los ASICs (circuitos integrados de aplicaciones específicas) se pueden ejecutar.

La reconfiguración de las FPGAs, se debe a sus componentes reconfigurables denominados bloques lógicos, que están interconectadas por una red de enrutamiento reconfigurable. Zambrano G., F. A., (20014) manifiesta que para la interconexión de enrutamiento existen dos topologías, que son redes de enrutamiento basados en árbol y en malla.

Por ejemplo, una FPGA basada en una arquitectura de árbol está diseñada a través de conexión de bloques lógicos agrupados. En otras palabras, las agrupaciones se conectan de manera recursiva formando así

una estructura jerárquica. Por el contrario, Zambrano G., F. A., (20014) manifiesta que una FPGA con arquitectura en malla admite enlaces lógicos en mallas 2-D en la misma red de enrutamiento.

Mientras, que una FPGA con arquitectura en árbol no requiere mayor área como en una topología de malla. Pero una FPGA con arquitectura en árbol tiene una desventaja en la escalabilidad de diseñar circuitos lógicos, frente a una FPGA en malla que es totalmente escalable y muy utilizado por la mayoría de fabricantes de FPGA tales como Xilinx y Altera. La figura 2.1 muestra una FPGA con arquitectura en malla.

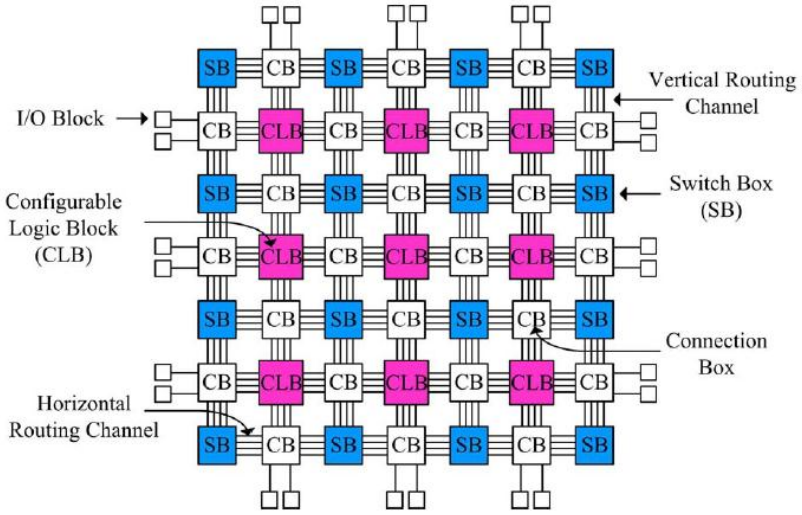


Figura 2. 1: Resumen de la arquitectura FPGA
Fuente: http://repositorio.ucsg.edu.ec/handle/123456789/2660.

Mientras que las cajas de conexión, permiten conectar los bloques lógicos y los bloques de los pines de E/S con pistas de enrutamiento adyacentes. Mediante el diseño de diagramas ASM o de flujo se pueden convertir circuitos digitales en una CLB que tiene conexiones a dispositivos

de E/S, para luego ser asignados en la FPGA. ASM, permite generar flujos de datos binarios (bits) programadas para ser ejecutados por el hardware de destino.

2.1.1. Bloques lógicos configurables.

Un bloque lógico configurable (CLB), es un componente básico de una FPGA que implementa funcionalidad lógica de un diseño de aplicación de destino. Una CLB puede comprender de un solo elemento de lógica básica *(Basic Logic Element, BLE)* o un grupo de BLES interconectados localmente. Un BLE simple consiste en una tabla de consulta *(Look Up Table, LUT)* y un Flip-Flop. Una LUT con k entradas (LUT-k) contiene los bits de configuración 2^k; donde se puede implementar cualquier función booleana de k-entradas. La figura 2.2 muestra un BLE sencillo, que consta de una entrada de 4 tablas de consulta (Look-Up Table, LUT-4) y un Flip-Flop tipo D.

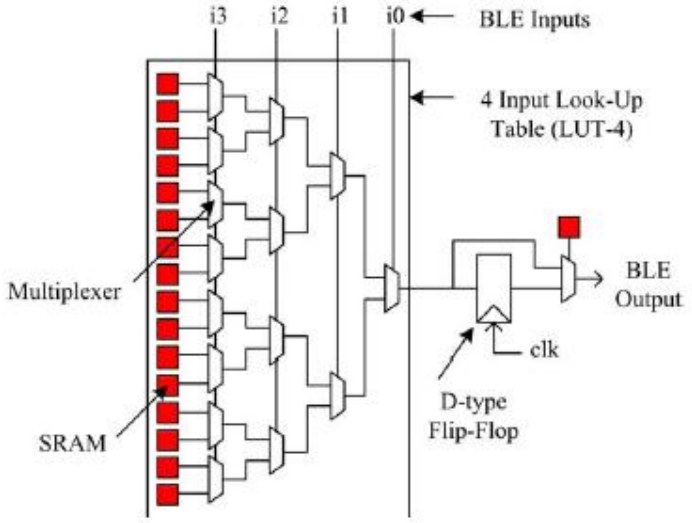


Figura 2. 2: Esquemático del elemento de lógica básica (BLE)
Fuente: http://repositorio.ucsg.edu.ec/handle/123456789/2660.

Una CLB puede contener un conjunto de BLEs conectados a través de una red de enrutamiento local. La figura 2.3 muestra un conjunto de 4 BLES; donde cada BLE contiene una LUT-4 y un Flip-Flop. La salida BLE es accesible a otros BLEs del mismo clúster a través de una red de enrutamiento local.

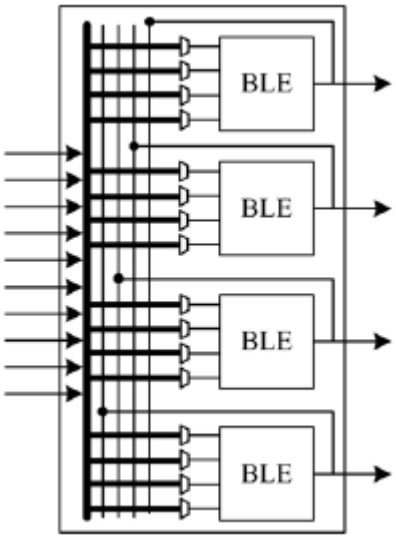


Figura 2. 3: Esquemático del bloque lógico configurable (CLB) con cuatro BLES.
Fuente: http://repositorio.ucsg.edu.ec/handle/123456789/2660.

2.1.2. Red de enrutamiento o encaminamiento.

La red de encaminamiento de una FPGA ocupa entre un 80% y 90% del área del chip de FPGA, mientras que el área lógica ocupa entre el 10% y 20% del FPGA. La flexibilidad de una FPGA depende principalmente de su red de encaminamiento programable. Una red de enrutamiento basado en FPGA de malla consta de pistas de enrutamiento horizontales y verticales que están interconectadas a través de las cajas de interruptores (SB).

La figura 2.4 muestra una pista bidireccional y una caja de conmutación unidireccional con Fs igual a 3. La figura 2.5 muestra un ejemplo de las diferentes longitudes de cable. Los segmentos de cable más largos abarcan múltiples bloques y requieren menos interruptores, lo que reduce el área de enrutamiento y los retrasos.

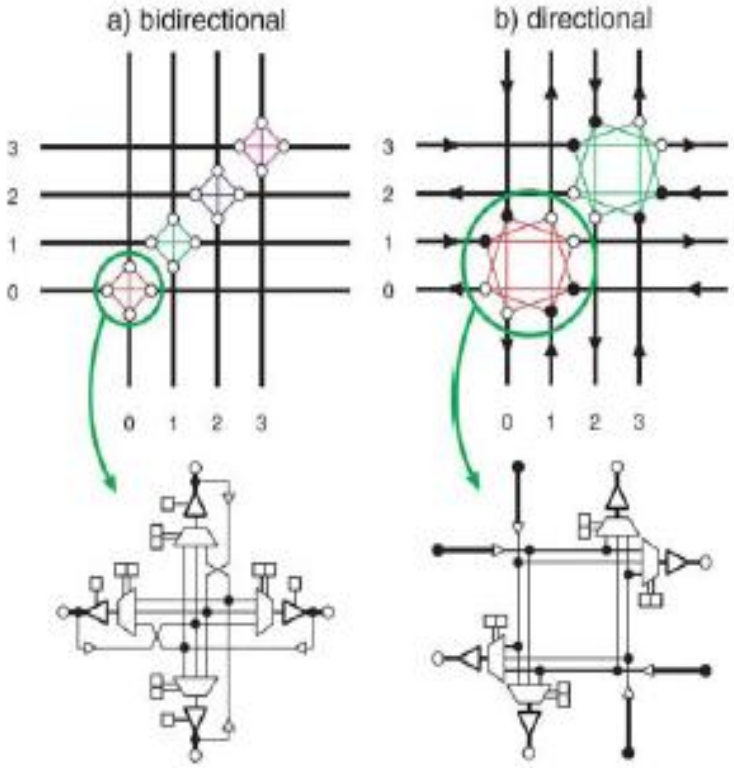


Figura 2. 4: Cambie Block, longitud 1 cables
Fuente: http://repositorio.ucsg.edu.ec/handle/123456789/2660.

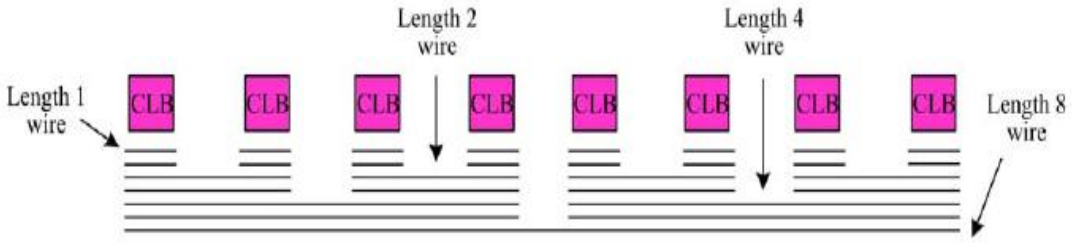


Figura 2. 5: Canal de distribución de segmento
Fuente: http://repositorio.ucsg.edu.ec/handle/123456789/2660.

2.2. Diseño Digital con EDA y FPGAs.

Los diseños digitales modernas se caracterizan por el uso del diseño electrónico automatizado *(Electronic Design Automation, EDA)* y arreglo de compuertas programables en el campo (FPGA) también denominada matrices de puertas programables (FPGAs). Una herramienta EDA puede tener una descripción de alto nivel de un circuito digital y traducir eso en hardware real.

Mediante "alto nivel" nos referimos a que muchos detalles, como las ecuaciones booleanas se pueden omitir y que las funciones o comportamientos de los circuitos digitales pueden ser descritas directamente. La mayoría de las herramientas EDA apoyan la captura de esquemáticos como un método de entrada de diseño. Sin embargo, el dibujo un esquema de un circuito digital bastante grande puede ser una tarea desalentadora; por no mencionar que un dibujo de cables y puertas lógicas puede ser imposible de depurar si aumenta la complejidad del circuito.

Por otro lado, el lenguaje de descripción de hardware *(Hardware Description Language, HDL)* puede describir un circuito digital en un lenguaje de alto nivel. La descripción de alto nivel del hardware digital, permite el desarrollo de sistemas digitales mucho más complejos debido a su enfoque estructural y la relativa facilidad para ser depurados. En los diseños digitales modernos, el hardware digital se describe en HDL y es sintetizada por una herramienta de EDA.

Los FPGAs pueden realizar los circuitos digitales en el hardware casi al instante. Muchos FPGAs están basados en memorias de acceso aleatorio *(Random Access Memory, RAM)*, lo que significa que la configuración pueda descargarse en la FPGA como si escribiéramos en una RAM.

La configuración de una FPGA es el equivalente a una plataforma de programación en un computador, que resuelva el circuito digital que se está ejecutando actualmente. Las FPGAs están disponibles en muchas compañías desarrolladoras de dispositivos FPGA, con diversas capacidades, estructuras internas y funciones integradas.

Sin embargo, la tendencia general es hacia FPGAs cada vez más grandes con número de características cada vez mayor. Un sistema digital completo se puede realizar con facilidad en un solo chip FPGA moderno. Sin embargo, se requieren herramientas EDA para manejar la mayor parte del proceso de diseño.

Los diseños digitales pueden ser sumamente complejos en su hardware y muy atractivo en sus funcionalidades. Los sistemas integrados simples, tales como las que se encuentran en los electrodomésticos, los dispositivos de adquisición de datos y dispositivos de E/S de computadoras; potencialmente puede ser implementado como sistemas digitales personalizados.

Tal cambio paradigmático o migración ascendente de hardware digital personalizado, es posible con los continuos avances de las herramientas EDA y FPGAs. Los diseños digitales modernos dependen en gran medida de las tres herramientas EDA, HDL y FPGAs que se muestra en la figura 2.6.

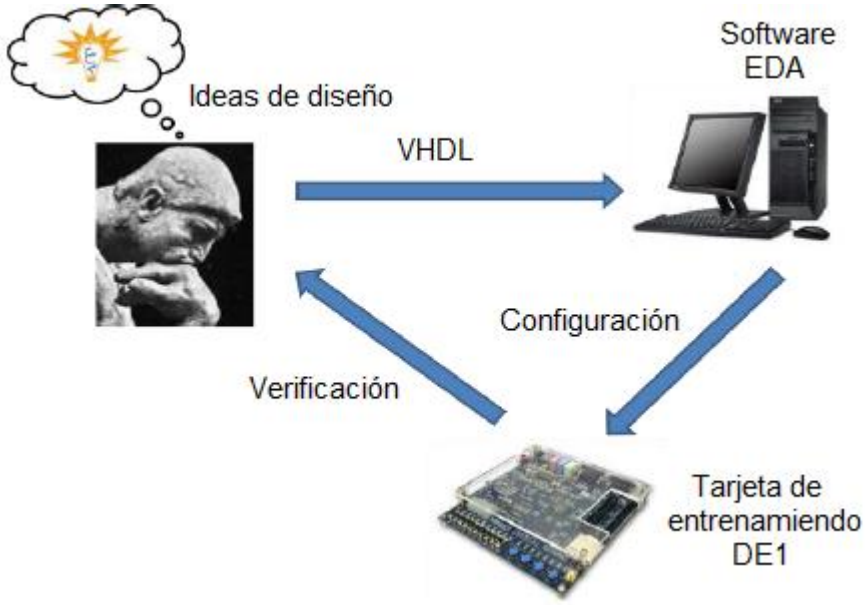


Figura 2. 6: Proceso del diseño digital moderno con EDA, VHDL y FPGA.
Elaborado por: Las Autoras

Las ideas de diseño se describieron por primera vez en HDL de tal manera que la herramienta EDA pueda entenderlo. La herramienta EDA entonces sintetiza las descripciones VHDL y genera la configuración de la FPGA. Una vez que la configuración se carga en la FPGA, la FPGA "se da cuenta" del circuito digital.

Con la ayuda de la placa de desarrollo FPGA DE1 de ALTERA, el circuito digital poner en el medio ambiente de la vida real y el diseñador puede verificar que el diseño se realiza según lo previsto.

2.3. Nivel de Transferencia de Registro (RTL).

Desde el punto de vista físico, los circuitos digitales se pueden describir en diversos niveles de abstracción basado en los componentes de hardware dominantes. La figura 2.7 muestra los diferentes niveles de componentes de hardware y abstracciones (aislamiento). El componente de hardware en cada nivel es típicamente construido a partir de los componentes de un nivel inmediatamente inferior.

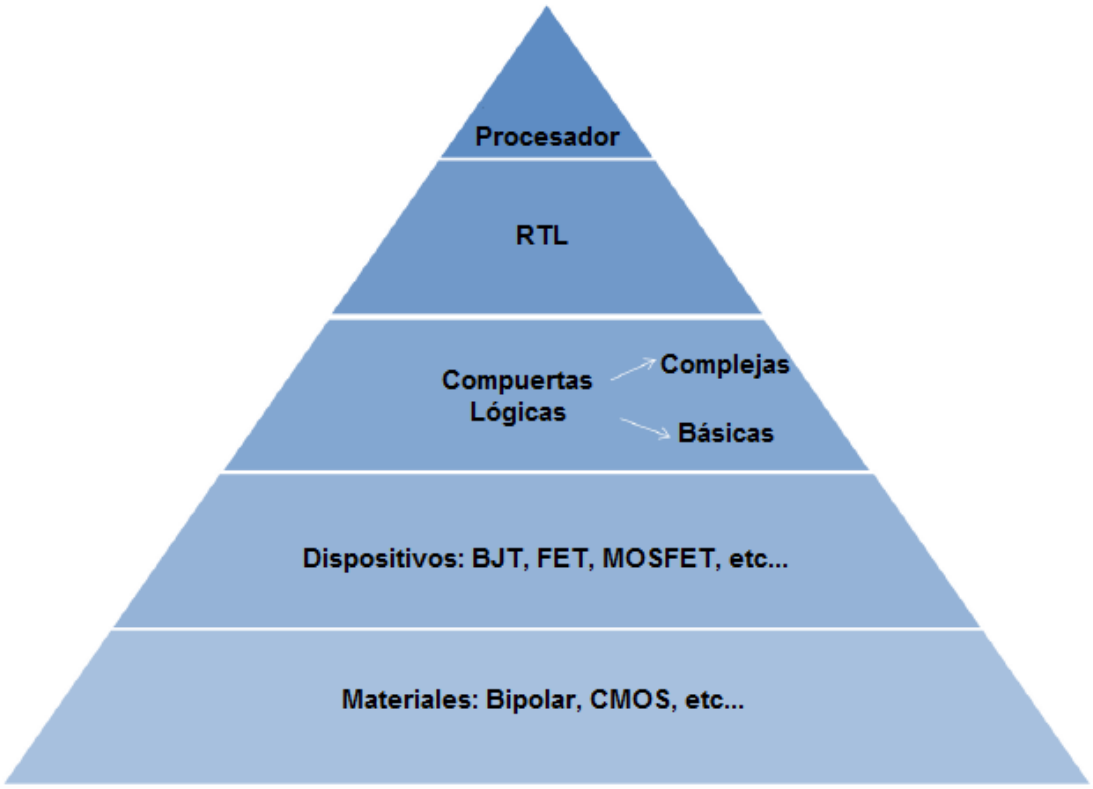


Figura 2. 7: Proceso del diseño digital moderno con EDA, VHDL y FPGA.
Elaborado por: Las Autoras

A medida que se asciende en la pirámide, el hardware es más complejo y por lo tanto los bloques de construcción básica, o componentes principales, se hacen más grandes en tamaño y más atractivo en sus funcionalidades. Todos los sistemas digitales modernos se basan en

materiales semiconductores. Los dispositivos semiconductores se construyen utilizando estos materiales.

Las puertas lógicas podrían ser básicas o complejas, pero, todos ellos están construidos a partir de dispositivos semiconductores. Vamos a utilizar las puertas lógicas para construir componentes RTL o bloques de construcción con funciones conocidas.

RTL, como se muestra en la figura 2.10, está en una posición central en los diseños digitales modernas. Las construcciones de componentes RTL se basan en la lógica booleana básica y los diseños de circuitos digitales en técnicas básicas. Los componentes de RTL, a su vez, se utilizan para construir los sistemas digitales a nivel de procesador. Los componentes RTL conocidos como bloques de construcción son: registros, contadores, registros de desplazamiento, codificadores, decodificadores, multiplexores, etc.

Los componentes de memoria RAM y ROM (memoria de sólo lectura) por algunos se denominan componentes RTL, mientras que otros pueden llamarlos componentes del procesador. Dicha distinción no es necesaria ya que RTL no es un término técnico bien definido. Todos los componentes RTL antes mencionados están bien definidos en sus funciones. Sin embargo, las flexibilidades en las realizaciones de hardware reales son todavía posibles.

Por ejemplo, un contador decimal codificado en binario (BCD) es un contador que cuenta de 0 a 9. Sin embargo, algunos contadores BCD pueden tener características opcionales tales como carga en paralelo, el recuento de ondulación de entrada y salida, etc.

2.4. Fundamentos de VHDL.

El diseño digital moderno depende en gran medida de EDA, en la cual tanto el lenguaje de descripción de hardware (VHDL) y Verilog HDL son esenciales para comunicar ideas del diseño para el software EDA. El objetivo principal del software EDA, es realizar el diseño de hardware real.

En general, los primeros HDL, como Verilog HDL, tienen muchas características específicamente para el modelado de simulación. Modernas herramientas informáticas EDA suelen ser lo suficientemente potente como para traducir estas declaraciones "no sintetizables" a hardware real.

Sin embargo, en esos casos, las ideas de diseño de los diseñadores pueden no ser interpretados con precisión por la EDA. Dado que el presente trabajo de titulación consiste en modelar el diseño digital, nos centraremos más en las técnicas y ejemplos de diseño.

2.4.1. Modelación Básica.

VHDL es estrictamente un "tipo" de lenguaje de tal manera que cada cable (alambre) y todos los elementos o componentes deben estar definidos

claramente. Nosotros usamos "código" VHDL para modelar un circuito lógico. El "código" se "compila" por el software EDA, que en este caso es QUARTUS II de ALTERA y "montado" en el hardware real.

a. Elementos léxicos de VHDL: los VHDLs originales (VHDL-87) solo utilizan los caracteres del código ASCII. Los VHDLs posteriores aceptan todos los caracteres del juego de caracteres de la ISO 8859 Latin-1 de 8 bits. Es decir, que incluyen letras mayúsculas y minúsculas, letras con signos diacríticos, como "à", "ä", etc., los dígitos del 0 al 9, puntuación y otros caracteres especiales.

A continuación en detalle los elementos utilizados en VHDL:

- **COMENTARIOS**
  - ✓ Comentario de una sola línea: cualquier texto dentro de una línea sigue guiones de arrastre.
  - ✓ Comentario delimitados: cualquier texto que sigue a "/*" y prosigue "*/". (No compatible con Quartus II)

- **IDENTIFICADORES**
  - ✓ Solo puede contener letras alfabéticas (mayúsculas y minúsculas), números decimales y subrayado ("_")
  - ✓ Debe comenzar con una letra del alfabeto,
  - ✓ No puede terminar con subrayado; y
  - ✓ No puede incluir dos subrayados sucesivas.

- **SÍMBOLOS ESPECIALES**
  - ✓ Uno de los siguientes caracteres:

    " # & ' ( ) * + - , . / : ; < = > ? @ [ ] ` |

  - ✓ Par de caracteres:

    => ** := /= >= <= <> ?? ?= ?/= ?> ?< ?>= ?<= << >>

- **NÚMEROS**
  - ✓ Enteros: todos los números enteros positivos en base 10.
  - ✓ Reales: 4.15336, 17.66e-12, etc.
  - ✓ Enteros diferente a base 10: 3#11023#, 78#B3ED#, etc.

- **CARACTERES**
  - ✓ Los caracteres literales están encerrados entre comillas simples.

- **CADENAS (STRINGS) O SECUENCIAS**
  - ✓ Las cadenas son secuencias de caracteres (incluyendo espacios) encerrados entre comillas dobles.

- **CADENAS (STRINGS) O SECUENCIAS DE BITS.**

  "Bit" es el tipo predefinido de VHDL; que permite valores de '0' y '1'. En la práctica, a menudo utilizamos el tipo "std_logic" definido en la biblioteca IEEE (necesitamos invocar la librería explícitamente),

que permite los nueve valores diferentes para un bit o un solo cable.

- **PALABRAS RESERVADAS.**

  La tabla 2.1 muestra las palabras reservadas en VHDL.

Tabla 2. 1: Listado de palabras reservadas en VHDL.

abs	disconnect	is	out	sli
access	downto	label	package	sra
after	else	library	port	srl
alias	elsif	linkage	postponed	subtype
all	end	literal	procedure	then
and	entity	loop	process	to
architecture	exit	map	pure	transport
array	file	mod	range	type
assert	for	nand	record	unaffected
attribute	function	new	register	units
begin	generate	next	reject	until
block	generic	nor	return	use
body	group	not	rol	variable
buffer	guarded	null	ror	wait
bus	if	of	select	when
case	impure	on	severity	while
component	in	open	signal	with
configuration	inertial	or	shared	xnor
constant	inout	others	sla	xor

Fuente: Altera.

2.4.2. Tipos de datos VHDL.

A continuación se describen los diferentes tipos de datos para programación en VHDL:

a. TIPOS ESCALAR.

Un objeto en VHDL se define como una "constante", "variable" o "señal". En primer lugar, la "constante" representa cables o buses con valores lógicos fijos. La "variable" representa un soporte de valor temporal para la práctica de VHDL.

Por último, la "señal" representa en los circuitos digitales a cables, buses de datos, o registros. Los valores escalares predefinidos en VHDL son: bit (0 y 1), booleano (verdadero y falso), enteros, de coma flotante (real), o tiempo. A continuación ejemplos de valores escalares:

```
Constant  year : integer := 2009;
Constant  delay : time  := 200 ps;
Variable  average : real;
Variable  is_active : Boolean := false;
Signal    output : bit;
```

b. TIPOS DE ENUMERACIÓN.

En VHDL, los usuarios pueden crear sus propios tipos de datos mediante el uso de la declaración "tipo". La definición de tipo de enumeración es una manera de crear valores, como por ejemplo:

```
Type date is (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
variable  operation_day : date ;
```

Del ejemplo anterior, la variable "operation_day" puede tomar uno de los siete valores definidos en el tipo "fecha". Por supuesto, el compilador VHDL eventualmente da estos valores de su significado físico para la realización de hardware.

Sin embargo, el diseñador no tendrá que preocuparse acerca de sus verdaderos valores mientras que los utiliza para describir el diseño. Los tipos de enumeración más importante se definen en la biblioteca IEEE para "std_logic".

c. Matrices (Arrays).

Para los tipos de datos bidimensionales, VHDL permite al usuario definir "matrices". Para el tipo de datos multidimensionales, se puede utilizar en varias ocasiones la declaración

```
array_type_definition <=
        array ( discrete_range { , ... } ) of element_subtype_indication

type bit_vector is array (natural range <>) of bit;
signal reg : bit_vector(31 downto 0);

type matrix is array (1 to 3, 1 to 3) of integer;
variable transform : matrix;
```

d. Atributos.

VHDL proporciona información a través de "atributos" acerca de variables o señales. Esto se hace mediante la adición de un apóstrofe (') a una variable o señal seguido por los nombres de los atributos. Existen varios atributos, el más usado es ‘event, para la señal de reloj. "Clock'event" devuelve un valor booleano (verdadero o falso) para indicar si un evento (cambio) se produce en la señal de reloj.

e. Registros.

Los Arrays es el tipo de elementos de datos homogéneos. Para los tipos de datos heterogéneos es posible la agrupación entre "registro" utilizados. Dado que todos los elementos son diferentes, deben tener identificadores separados. Estos elementos pueden ser o pueden pertenecer a los mismos tipos de datos.

El siguiente ejemplo muestra dos formas de asignar valores a los elementos en un registro: posicional asociación (primer ejemplo constante).

```
record_type_definition <=
        record
                ( identifier { , ... } : subtype_indication ; )
                { ... }
        end record [ identifier ]
```

```
type time_stamp is record
      seconds  : integer range 0 to 59;
      minutes  : integer range 0 to 59;
      hours    : integer range 0 to 23;
end record time_stamp;
```

f. Librerías y uso de cláusulas.

Como se muestra en Example_basic.vhd, el código VHDL comienza con la "librería y uso de cláusulas". VHDL fue diseñado con un conjunto muy limitado de tipos predefinidos. Todos los tipos útiles se definen en la librería y los paquetes incluidos en la librería.

Vamos a utilizar tres tipos de bibliotecas aquí: IEEE, de trabajo, y "altera_mf". En el ejemplo (ver siguiente página), se utiliza la librería IEEE. La librería “work” o “trabajo” se está refiriendo al directorio de trabajo (el directorio o carpeta en la que se diseña un proyecto). En otras palabras, puede colocar un archivo de diseño VHDL (véase la figura 2.8) en el directorio del proyecto y luego invocarlo haciendo referencia a la biblioteca de "trabajo". La biblioteca "altera_mf" es la

biblioteca propietaria del Altera. Se utiliza para invocar circuitos integrados que son descritas en Megafunctions propiedad de Altera.

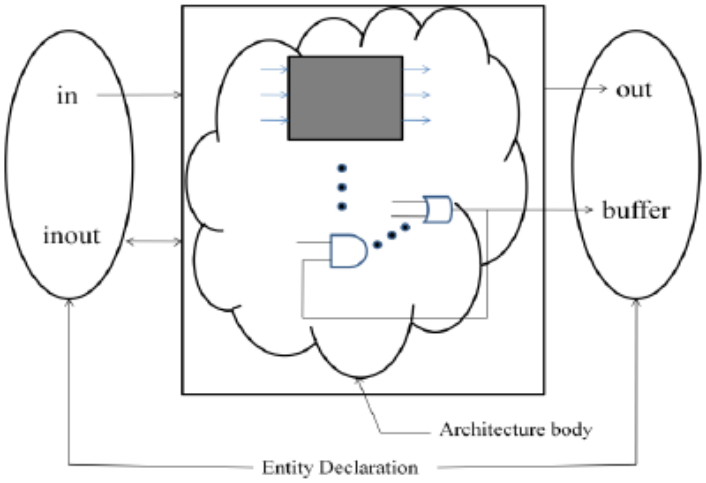


Figura 2. 8: Modelado de un diseño lógico en VHDL.
Fuente: Altera.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY example_basic IS
      PORT( CLOCK_50 : IN std_logic;
            KEY  : IN std_logic_vector(3 DOWNTO 0);
            hex0 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
          );
END example_basic;

ARCHITECTURE ex1 of example_basic IS
      signal ain: unsigned(3 downto 0);
BEGIN
  --The following is a Hex to 7-segment decoder.
  hex0 <= "1000000" when ain="0000" else --0
          "1111001" when ain="0001" else --1
          "0100100" when ain="0010" else --2
          "0110000" when ain="0011" else --3
          "0011001" when ain="0100" else --4
          "0010010" when ain="0101" else --5
          "0000010" when ain="0110" else --6
          "1111000" when ain="0111" else --7
          "0000000" when ain="1000" else --8
          "0011000" when ain="1001" else --9
          "0001000" when ain="1010" else --A
          "0000011" when ain="1011" else --B
```

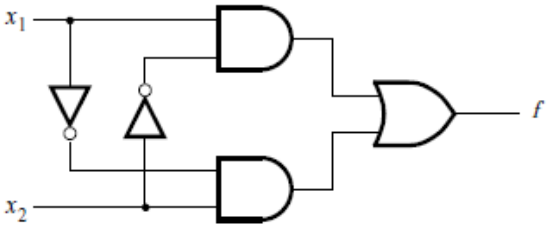
```
            "1000110" when ain="1100" else --C
            "0100001" when ain="1101" else --D
            "0000110" when ain="1110" else --E
            "0001110" when ain="1111" else --F
            "1111111"; -- default

  counter: PROCESS(clock_50, KEY(0))
              variable loopcount:integer range 0 to 1000000;
       BEGIN
              IF(KEY(0) = '0') THEN
                    loopcount := 0;
                    ain <= "0000";
              ELSIF(clock_50'event AND clock_50 = '1') THEN
                    if(loopcount = 1000000) then
                          loopcount := 0;
                          ain <= ain + 1;
                    else
                          loopcount := loopcount + 1;
                    end if;
              END IF;
       END PROCESS;
END ARCHITECTURE ex1;
```

2.5. Diseño mediante captura esquemática.

Mediante un ejemplo de diseño de un circuito controlador de luz de dos vías (véase la figura 2.9) se entenderá como realizar el diseño por captura esquemática o edición gráfica de circuitos digitales. El circuito puede ser utilizado para controlar una sola luz de cualquiera de los dos interruptores, x1 y x2, donde un interruptor cerrado corresponde al valor de la lógica 1.

La tabla de verdad para el circuito también se muestra en la figura 2.9. Hay que tener en cuenta que esto es sólo la función OR Exclusiva de entradas x1 y x2 pero vamos a hacerlo operativo mediante las compuertas lógicas. El editor gráfico del software Quartus II se utiliza para especificar un circuito mediante diagrama de bloques. Seleccione File> New y aparecerá la

ventana de la figura 2.10, elegimos *Block Diagram/Schematic File*, y haga clic en OK para abrir la ventana del editor gráfico.



x_1	x_2	f
0	0	0
0	1	1
1	0	1
1	1	0

Figura 2. 9: Diseño esquemático del circuito controlador de luz.
Fuente: Altera.

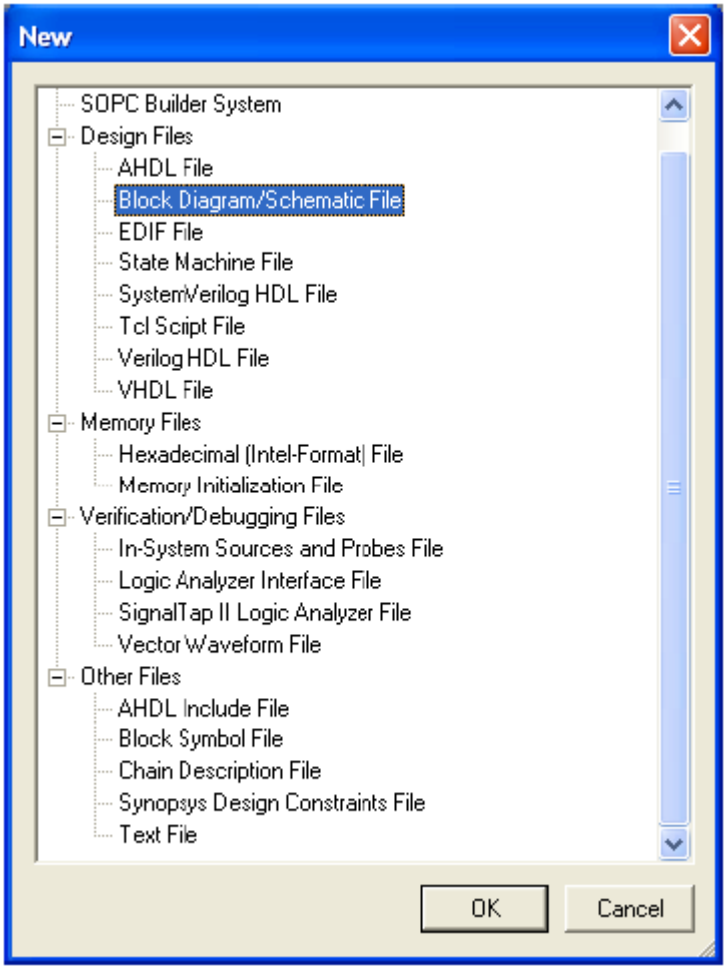


Figura 2. 10: Elección para diseño de circuitos mediante diagrama de bloques.
Fuente: Altera.

El primer paso es especificar un nombre para generar el archivo y posterior a esto se mostrará la ventana del editor de gráficas.

Importación de símbolos de compuertas lógicas.

El editor gráfico proporciona un número de bibliotecas que incluyen elementos de circuito que se pueden importar en un esquema. Haga doble clic en el espacio en blanco en la ventana del editor gráfico, o haga clic en el icono de una compuerta AND de la barra de herramientas. Aparecerá una ventana emergente en la figura 2.11.

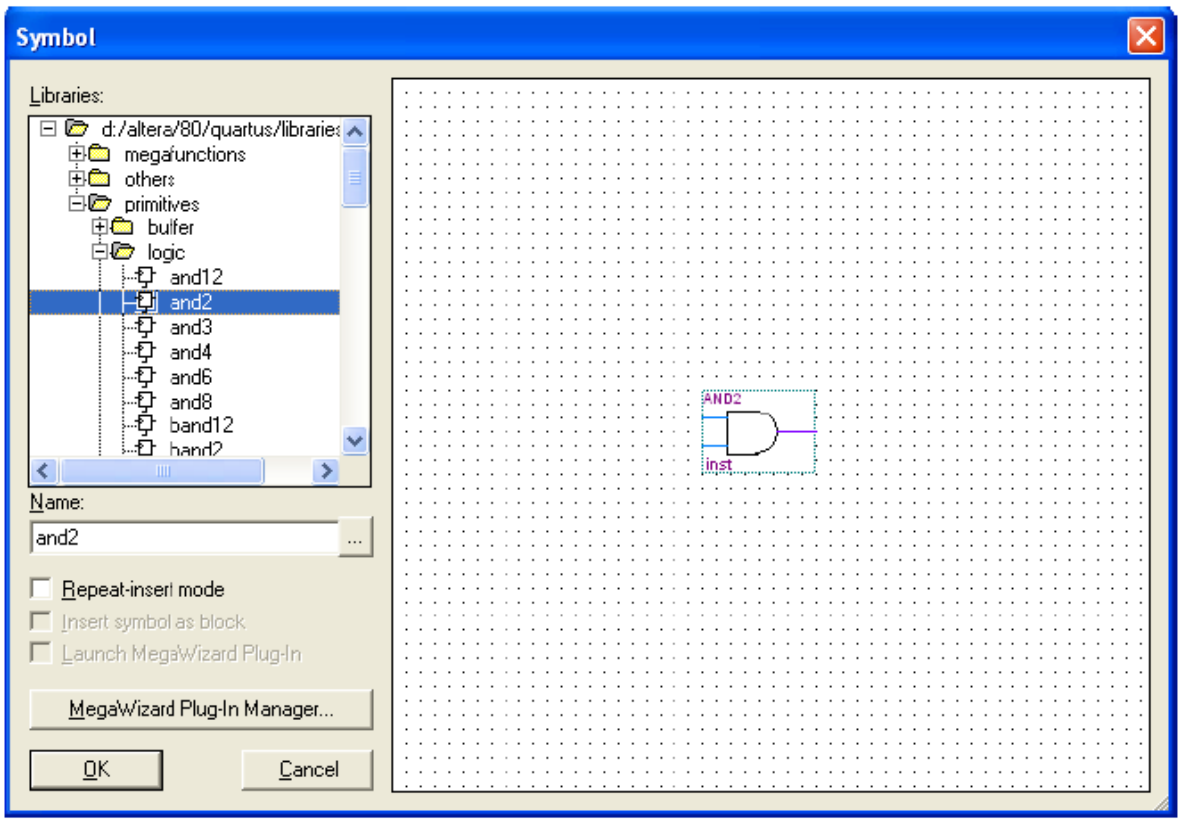


Figura 2. 11: Elección de compuertas lógicas de la librería.
Fuente: Altera.

Ampliar la jerarquía en el cuadro de librerías de la figura 2.11, primero ampliamos la librería y a continuación, expandimos las primitivas de librerías,

seguido por la expansión de la lógica de la biblioteca que comprende las puertas lógicas. Seleccione AND2, que es una compuerta lógica de dos entradas y una salida, tal como se muestra en la figura 2.11.

Ahora, la compuerta y el símbolo aparecerán en la ventana del editor gráfico. Con el ratón, movemos el símbolo de una ubicación conveniente y haga clic para colocarlo allí. Importamos la segunda compuerta, que se puede hacer simplemente colocando el puntero del ratón sobre el símbolo de la puerta AND existente, mediante un clic y arrastrando el mouse podemos hacer una copia del símbolo.

Un símbolo en la ventana del editor gráfico se puede mover haciendo clic en él y arrastrándolo a una nueva ubicación con el botón pulsado del ratón. A continuación, seleccionamos OR2 de la librería y la importación de la compuerta OR en el diagrama. A continuación, seleccione la compuerta NOT e importamos dos instancias de la misma NOT, tal como se muestra en la figura 2.12.

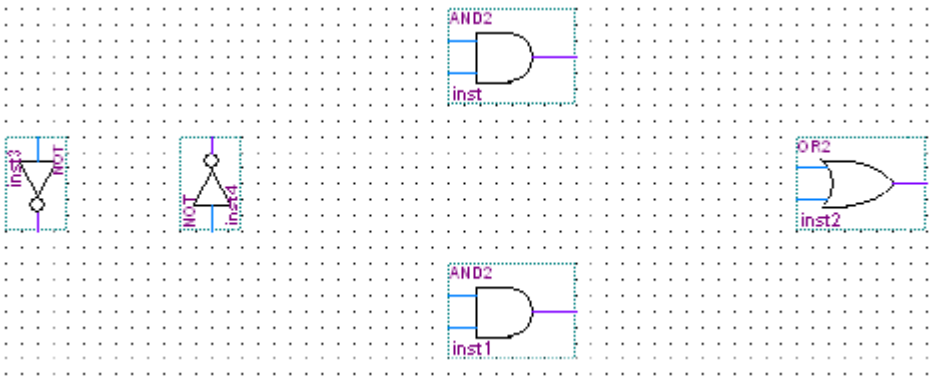


Figura 2. 12: Importación de símbolos de compuertas lógicas de la librería.
Fuente: Altera.

Después de haber ingresado los símbolos de compuertas lógicas, ahora es necesario introducir los símbolos que representan los puertos de entrada y de salida del circuito. Utilizando el mismo procedimiento de la importación de compuertas, pero elegimos símbolos de puerto o pines primitivos de la librería. La importación de dos instancias de puertos de entrada y una instancia de puertos de salida son mostrados en la figura 2.13.

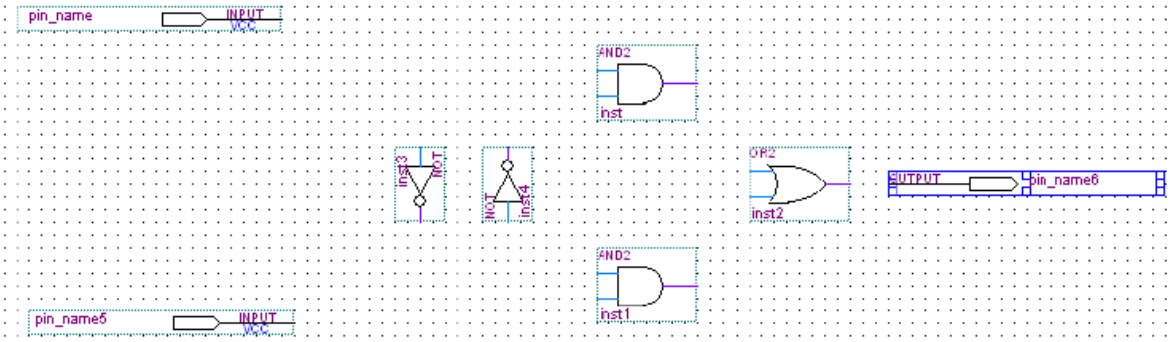


Figura 2. 13: Importación de puertos de entrada y salidas de la librería.
Fuente: Altera.

2.6. Procesadores basados en FPGA

Cantidad considerable de área FPGA se puede reducir mediante la incorporación de un microprocesador en una FPGA. Un microprocesador puede ejecutar cualquier tarea menos cálculos intensivos, mientras que las tareas de cálculo intensivo se pueden ejecutar en una FPGA. Del mismo modo, una aplicación basada en un microprocesador puede tener enormes ganancias de aceleración si una FPGA se une con él.

Un FPGA adjunto con un microprocesador puede ejecutar cualquier funcionalidad de cálculo intensivo como una instrucción de hardware personalizado. Estas ventajas han obligado a los proveedores comerciales

de FPGAs incorporar microprocesadores a un FPGA para que el sistema completo pueda ser programado en un solo chip.

Pocos vendedores han integrado procesadores fijos en su FPGA (como procesador AVR integrado en Atmel FPSLIC o procesadores PowerPC embebido en Xilinx Virtex-4). Otros proporcionan núcleos de procesadores sencillos que son altamente optimizados para ser asignada en los recursos programables de FPGA. Nios de Altera y de Xilinx Microblaze, son procesadores suaves significaba para los diseños FPGA que permiten instrucciones de hardware personalizado.

Las unidades reconfigurables también se puede conectar con microprocesadores para alcanzar el tiempo de ejecución más veloz en los programas de software. Hay una famosa regla 90/10, que establece que el 90% del tiempo de ejecución del programa se gasta en el 10% del código. Así que el objetivo principal es la de convertir el código de 10% en lógica de hardware y aplicar como una función de hardware para lograr avances muy grandes.

Considerando que el 90% restante del código se ejecuta en un microprocesador, se incorpora una unidad reconfigurable con microprocesadores para lograr aceleración en tiempo de ejecución. Para aquellos se ha propuesto un procesador VLIW (véase la figura 2.14) que

puede soportar instrucciones personalizadas específicos de la aplicación a través de un bloque de hardware reconfigurable.

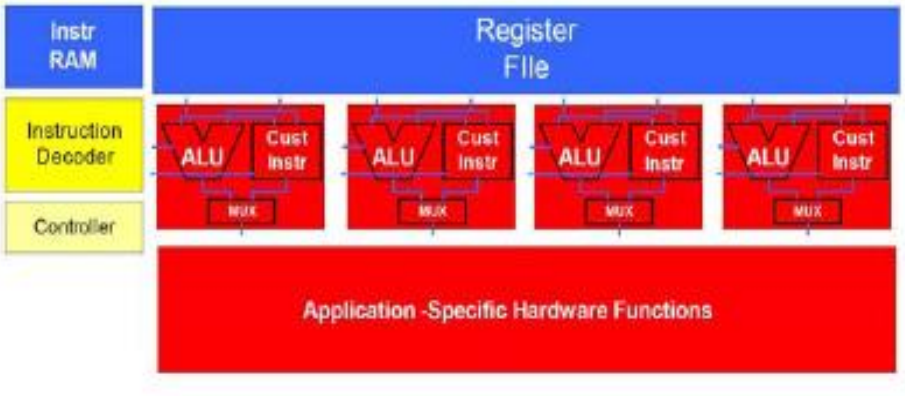


Figura 2. 14: Una arquitectura de procesador VLIW.
Fuente: http://repositorio.ucsg.edu.ec/handle/123456789/2660**.**

Los procesadores VLIW tienen un solo controlador instrucción que despacha diferentes operaciones a varias unidades funcionales que comparten un único archivo de registro. Todas estas unidades funcionales se ejecutan en paralelo. Idealmente, muchas instrucciones pueden ejecutar en paralelo. Pero la aplicación también debe exhibir Nivel alto Instrucción Paralelismo (ILP), por lo que el control de datos y dependencias no limitan las mejoras de rendimiento.

La arquitectura propuesta, que se ilustra en la figura 2.15, es un procesador de 4 vías VLIW (4 unidades funcionales), con recursos de hardware destinados a la implementación de funciones de hardware específico de aplicación. Todos los cuatro bloques funcionales y las funciones de hardware están unidos entre sí a través del archivo de registro.

El bloque de hardware es capaz de leer 16 operandos desde cualquiera de los 32 registros y escribir de nuevo 8 resultados en cualquiera de los registros. Un proceso de compilación se desarrolla para el procesador VLIW con funciones de hardware. El código C para ser implementado se perfila para encontrar los núcleos intensivos computacionales.

La síntesis de comportamiento o de alto nivel se les aplica transformar automáticamente en la lógica combinatoria. El código restante se transforma por el compilador VLIW y ensamblador, e implementado en el procesador VLIW. El procesador VLIW y el bloque de hardware se implementan en Altera Stratix II FPGA debido a su apoyo a los bloques DSP de alta velocidad. El procesador VLIW con funciones de hardware ha mostrado una aceleración máxima de 230 veces, y un aumento de velocidad promedio de 63 veces para los núcleos de cálculo de un conjunto de puntos de referencia DSP.

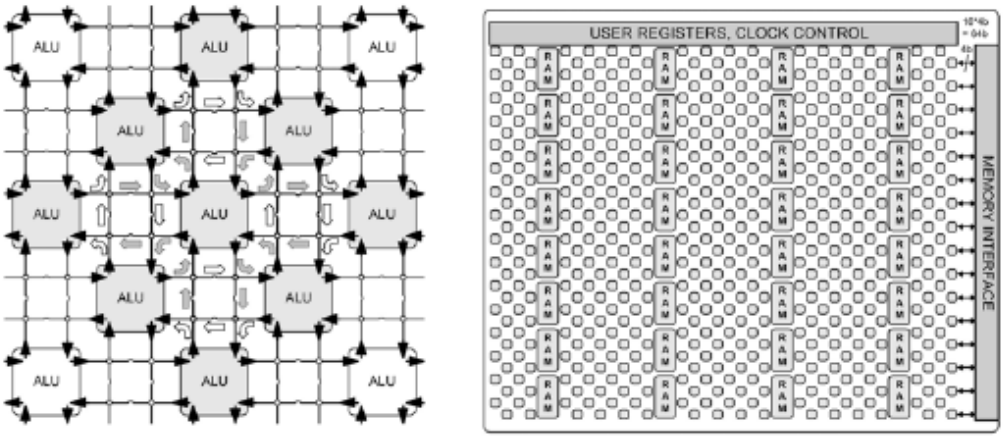


Figura 2. 15: Una matriz reconfigurable Aritmética para Aplicaciones Multimedia
Fuente: http://repositorio.ucsg.edu.ec/handle/123456789/2660.

CAPÍTULO 3: APLICACIONES PRÁCTICAS PARA LABORATORIO DE DIGITALES.

En el presente capítulo se presenta el aporte práctico del trabajo de titulación, en la que se desarrollan experiencias prácticas sobre las tarjetas de entrenamiento DE1 de ALTERA para ser utilizadas en el Laboratorio de Digitales en la formación de Ingenieros en Telecomunicaciones y Electrónica en Control y Automatismo.

3.1. Aplicación Práctica #1: Máquinas de estados.

Cuando iniciamos la aplicación de cualquier modelo de programación a una FPGA, en nuestro caso con Altera, podemos acceder a la creación de diferentes tipos de proyectos. Podemos crear programas mediante el lenguaje de programación VHDL o también crear como para nuestro ejemplo un proyecto mediante el diseño de máquinas de estado.

Para iniciar el diseño digital mediante máquina de estados, nos dan una ventana con varias viñetas como podemos apreciar en la figura 3.1, en las cuales debemos ir configurando y personalizando de acuerdo a nuestras necesidades, que para este caso es diseñar una máquina de estado para **detectar secuencias de bits**.

En el puerto de salida especificamos el tipo de salida <<Pulso>> (véase la figura 3.1).

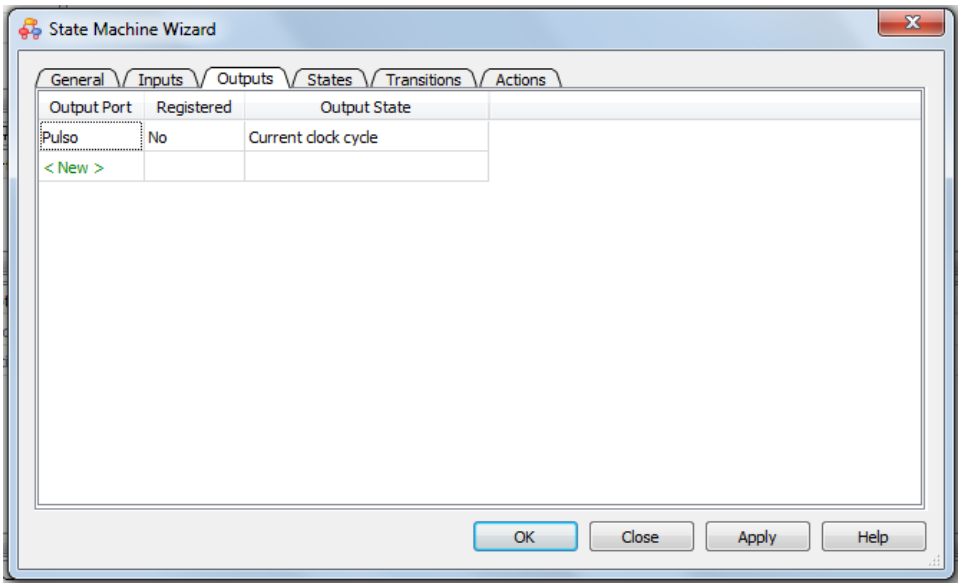


Figura 3. 1: Configuración de los puertos de salida para el diseño de máquinas de estados.
Elaborado por: Las Autoras.

Posteriormente, en la viñeta <<States>> configuramos los estados del detector de secuencias para 5 bits, que serían los estados A, B, C, D y E; en el primer estado “A” se escoge <<Reset>> para inicializar el detector de secuencias tal como se muestra en la figura 3.2.

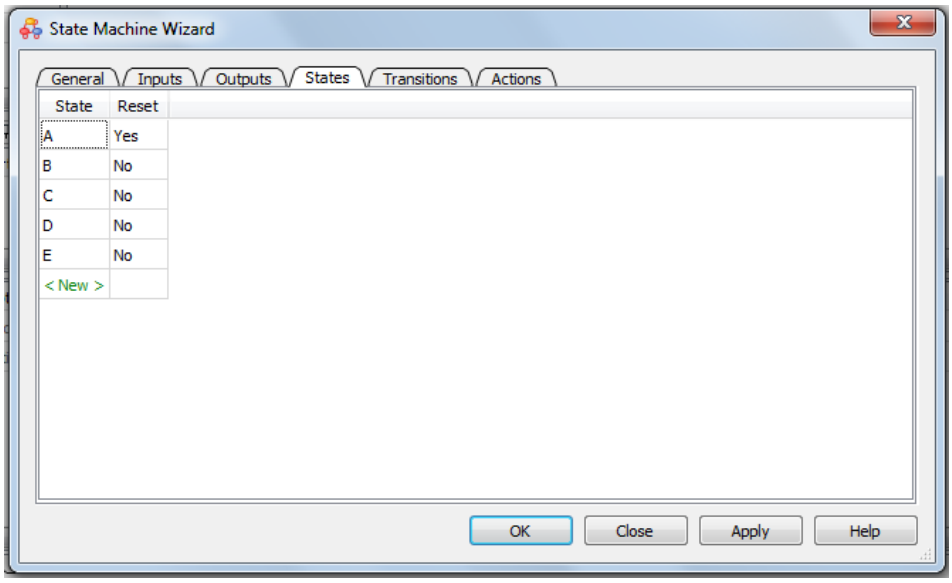


Figura 3. 2: Configuración de los estados para el diseño de máquinas secuenciales.
Elaborado por: Las Autoras.

Una vez realizado la configuracion de las salidas y estados, debemos realizar la configuracion de las transiciones en la viñeta “Transitions”, ya que con esto damos las condiciones para que nuestra maquina de estado vaya evaluando bit por bit (figura 3.3).

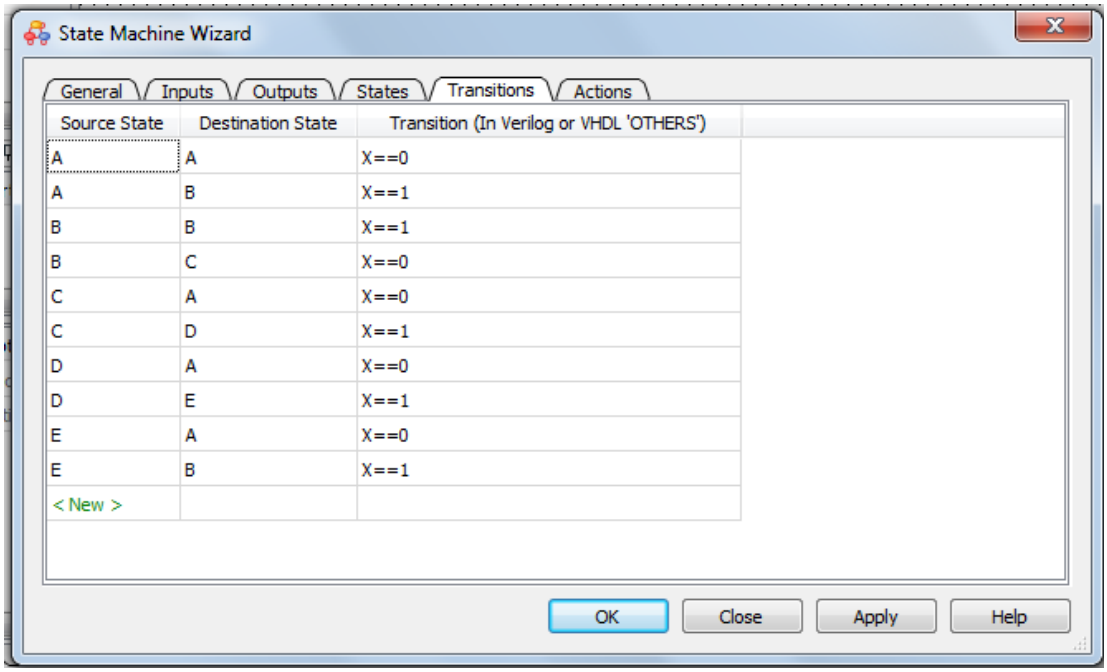


Figura 3. 3: Configuración de transiciones de estados para el diseño de máquinas secuenciales.
Elaborado por: Las Autoras.

Al terminar toda la secuencia de procedimientos realizados anteriormente daremos inicio a las acciones que deseemos que se realice como podemos apreciar en la figura a continuacion (vease figura 3.4), en la viñeta “Actions”.

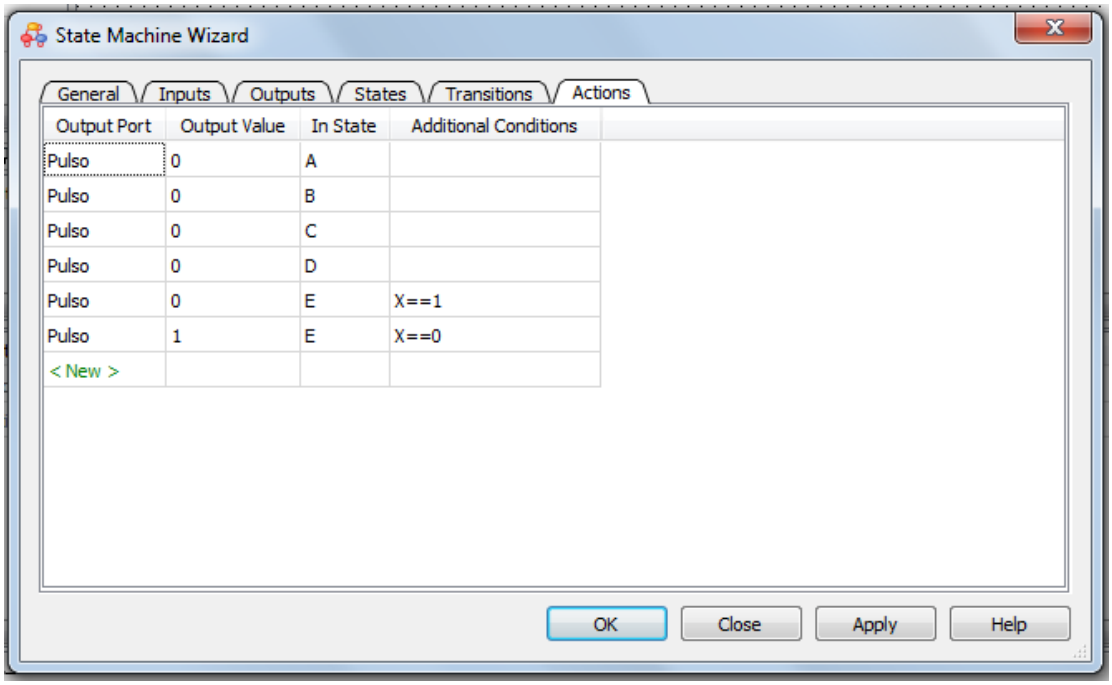


Figura 3. 4: Configuración de los estados para el diseño de máquinas de estados.
Elaborado por: Las Autoras.

Al terminar el procedimiento de creacion de nuestra maquina de estado, Quartus en nuestro proyecto nos proyectara el resultado, como podemos observar en la siguiente figura (vease figura 3.5).

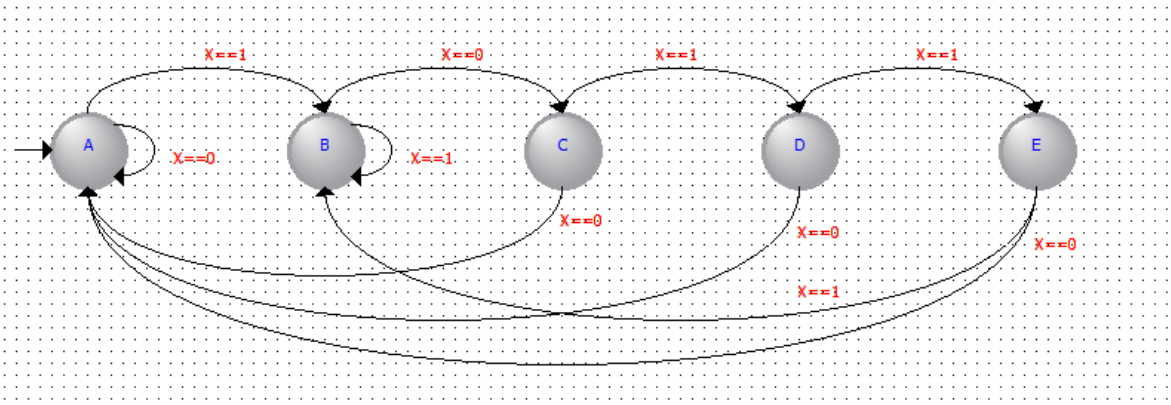


Figura 3. 5: Configuración de los estados para el diseño de máquinas de estados.
Elaborado por: Las Autoras.

Simbólicamente podemos tener los resultados de nuestro problema en el esquema de la figura 3.5, aunque ciertamente Quartus es una herramienta muy potente, este nos permite obtener el código fuente de la programación en VHDL como se muestra en la figura 3.6 en la cual nos describe rápidamente el encabezado de nuestra solución aplicada en la tarjeta FPGA a las tarjetas Altera, este encabezado va desde la linea 1 hasta la linea 17.

Siempre en VHDL, se deben declarar primeramente las librerias (ver las líneas 18 y 19) para después proceder a ensamblar el Entity donde tendremos declarados las entradas y salidas del módulo.

```
1  -- Copyright (C) 1991-2013 Altera Corporation
2  -- Your use of Altera Corporation's design tools, logic functions
3  -- and other software and tools, and its AMPP partner logic
4  -- functions, and any output files from any of the foregoing
5  -- (including device programming or simulation files), and any
6  -- associated documentation or information are expressly subject
7  -- to the terms and conditions of the Altera Program License
8  -- Subscription Agreement, Altera MegaCore Function License
9  -- Agreement, or other applicable license agreement, including,
10 -- without limitation, that your use is for the sole purpose of
11 -- programming logic devices manufactured by Altera and sold by
12 -- Altera or its authorized distributors.  Please refer to the
13 -- applicable agreement for further details.
14
15 -- Generated by Quartus II Version 13.0.1 Build 232 06/12/2013 Service Pack 1 SJ Web Edition
16 -- Created on Mon Feb 23 13:16:45 2015
17
18 LIBRARY ieee;
19 USE ieee.std_logic_1164.all;
20
21 ENTITY Detectar IS
22     PORT (
23         reset : IN STD_LOGIC := '0';
24         clock : IN STD_LOGIC;
25         x : IN STD_LOGIC := '0';
26         pulso : OUT STD_LOGIC
27     );
28 END Detectar;
29
```

Figura 3. 6: Configuración de los estados para el diseño de máquinas de estados.
Elaborado por: Las Autoras.

Posteriormente, se muestra la aquitectura del programa cuya descripción detallada sería la estructura interna del módulo, en esta

evaluamos por medio de condicionales dadas por las variables A, B, C, D, E, es decir, que estas variables son conocidas como estados tal como se muestran en las figuras 3.7 y 3.8.

```
30  ARCHITECTURE BEHAVIOR OF Detectar IS
31      TYPE type_fstate IS (A,B,C,D,E);
32      SIGNAL fstate : type_fstate;
33      SIGNAL reg_fstate : type_fstate;
34      SIGNAL reg_pulso : STD_LOGIC := '0';
35  BEGIN
36      PROCESS (clock,reset,reg_fstate)
37      BEGIN
38          IF (reset='1') THEN
39              fstate <= A;
40          ELSIF (clock='1' AND clock'event) THEN
41              fstate <= reg_fstate;
42          END IF;
43      END PROCESS;
44
45      PROCESS (fstate,x,reg_pulso)
46      BEGIN
47          reg_pulso <= '0';
48          pulso <= '0';
49          CASE fstate IS
50              WHEN A =>
51                  IF ((x = '0')) THEN
52                      reg_fstate <= A;
53                  ELSIF ((x = '1')) THEN
54                      reg_fstate <= B;
55                  -- Inserting 'else' block to prevent latch inference
56                  ELSE
57                      reg_fstate <= A;
58                  END IF;
59
60                  reg_pulso <= '0';
61              WHEN B =>
62                  IF ((x = '1')) THEN
63                      reg_fstate <= B;
64                  ELSIF ((x = '0')) THEN
65                      reg_fstate <= C;
66                  -- Inserting 'else' block to prevent latch inference
67                  ELSE
68                      reg_fstate <= B;
69                  END IF;
70
71                  reg_pulso <= '0';
```

Figura 3. 7: Configuración de los estados A y B para el diseño de máquinas de estados.
Elaborado por: Las Autoras.

```
72              WHEN C =>
73                  IF ((x = '0')) THEN
74                      reg_fstate <= A;
75                  ELSIF ((x = '1')) THEN
76                      reg_fstate <= D;
77                  -- Inserting 'else' block to prevent latch inference
78                  ELSE
79                      reg_fstate <= C;
80                  END IF;
81
82                  reg_pulso <= '0';
83              WHEN D =>
84                  IF ((x = '0')) THEN
85                      reg_fstate <= A;
86                  ELSIF ((x = '1')) THEN
87                      reg_fstate <= E;
88                  -- Inserting 'else' block to prevent latch inference
89                  ELSE
90                      reg_fstate <= D;
91                  END IF;
92
93                  reg_pulso <= '0';
94              WHEN E =>
95                  IF ((x = '0')) THEN
96                      reg_fstate <= A;
97                  ELSIF ((x = '1')) THEN
98                      reg_fstate <= B;
99                  -- Inserting 'else' block to prevent latch inference
100                 ELSE
101                     reg_fstate <= E;
102                 END IF;
103
104                 IF ((x = '1')) THEN
105                     reg_pulso <= '0';
106                 ELSIF ((x = '0')) THEN
107                     reg_pulso <= '1';
108                 -- Inserting 'else' block to prevent latch inference
109                 ELSE
110                     reg_pulso <= '0';
111                 END IF;
112             WHEN OTHERS =>
113                 reg_pulso <= 'X';
114                 report "Reach undefined state";
115         END CASE;
116         pulso <= reg_pulso;
117     END PROCESS;
118 END BEHAVIOR;
```

Figura 3. 8: Configuración de los estados C, D y E para el diseño de máquinas de estados.
Elaborado por: Las Autoras.

En otras palabras, Quartus II de Altera no solamente es para programar en VHDL, si no también mediante diseño de máquinas secuenciales o estados, captura esquemática (que se verá en la sección 3.2), está última es

una solucion empleada por Quartus para aquellas personas que desconocen como programar en VHDL y sería de gran ayuda para los estudiantes de Laboratorio de Digitales.

A continuacion tendremos que configurar el Pin Planner (véase la figura 3.9), para poder llevarlo a nuestras tarjetas Altera. El Pin Planner no es más que la ubicacion o plano de la tarjeta para especificar las entradas, salidas, y demas pines según lo realizado en la programacion o este caso por el diseño de máquinas de estados.

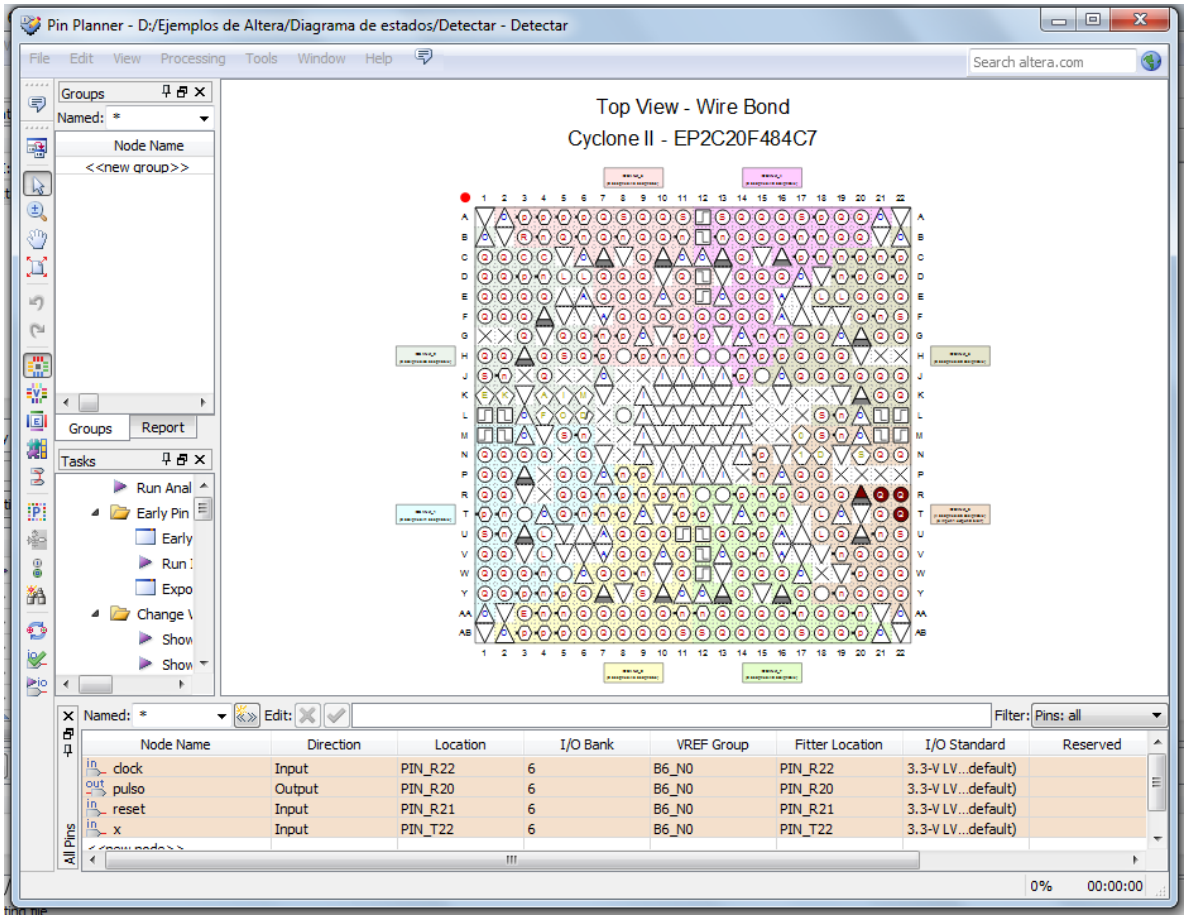


Figura 3. 9: Configuración del Pin Planner de la aplicación práctica #1.
Elaborado por: Las Autoras.

3.2. Aplicación Práctica #2: Circuito Restador.

Como se mencionó en la sección anterior, Quartus II nos permite diseñar circuitos digitales mediante captura esquemática, muy similar al simuladors ISIS PROTEUS. La captura esquemática es más fácil para algunas personas que no saben programar. La figura 3.10 muestra la ventana para el diseño de un nuevo proyecto en Quartus II, para lo cual se debe seleccionar *Block Diagram/Schematic File.*

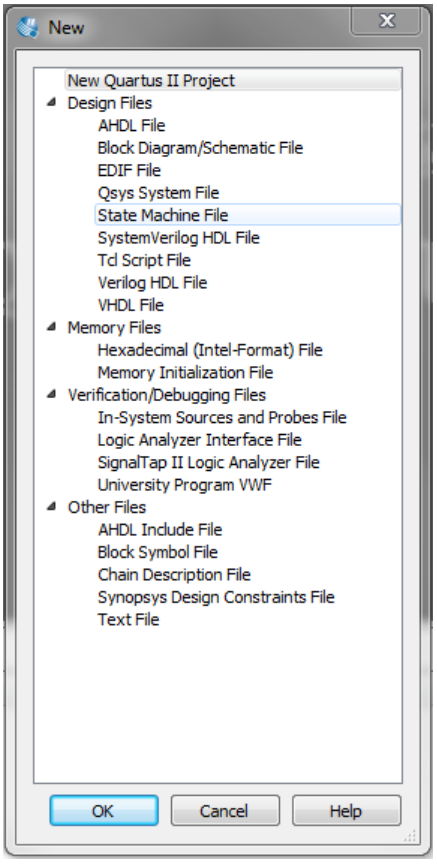


Figura 3. 10: Selección del tipo de proyecto en Quartus.
Elaborado por: Las Autoras.

Para esta aplicación práctica se ha escogido diseñar mediante captura esquemática de un Circuito Restador de 4 bits. La figura 3.11 muestra el circuito FULL ADDER para luego pasar al multiplexor (véase la figura 3.12) y

finalmente, dependiendo del valor que resulta de la operación se convierte el valor de binario a través de un decodificador BCD que será visualizado en un display de 7 segmentos (véase la figura 3.13).

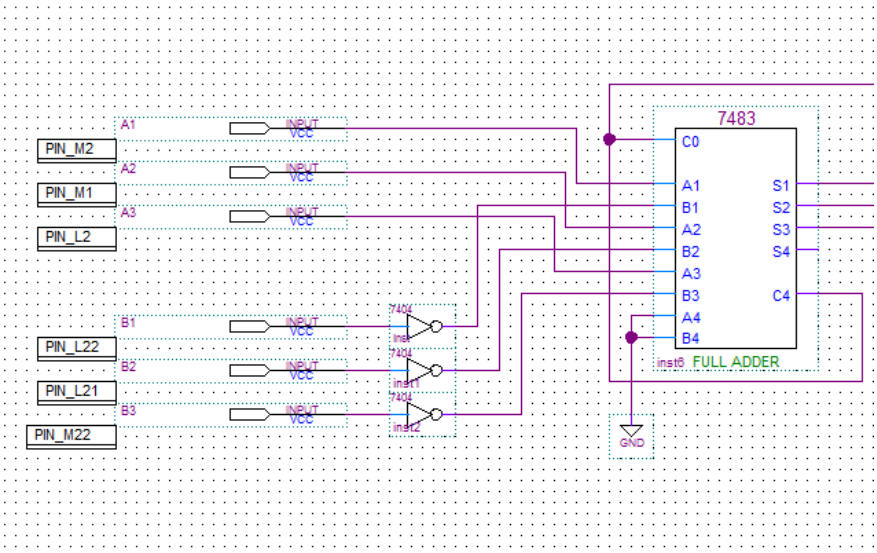


Figura 3. 11: Ingreso de variables a un restador de 4 bits.
Elaborado por: Las Autoras.

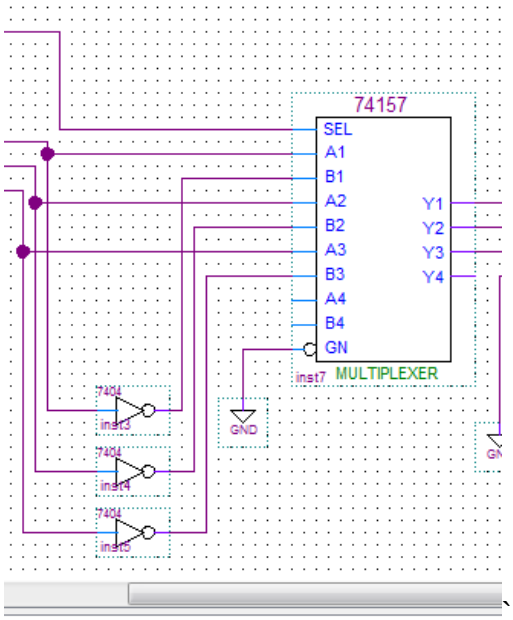


Figura 3. 12: Ingreso de resultado del restador a un multiplexor.
Elaborado por: Las Autoras.

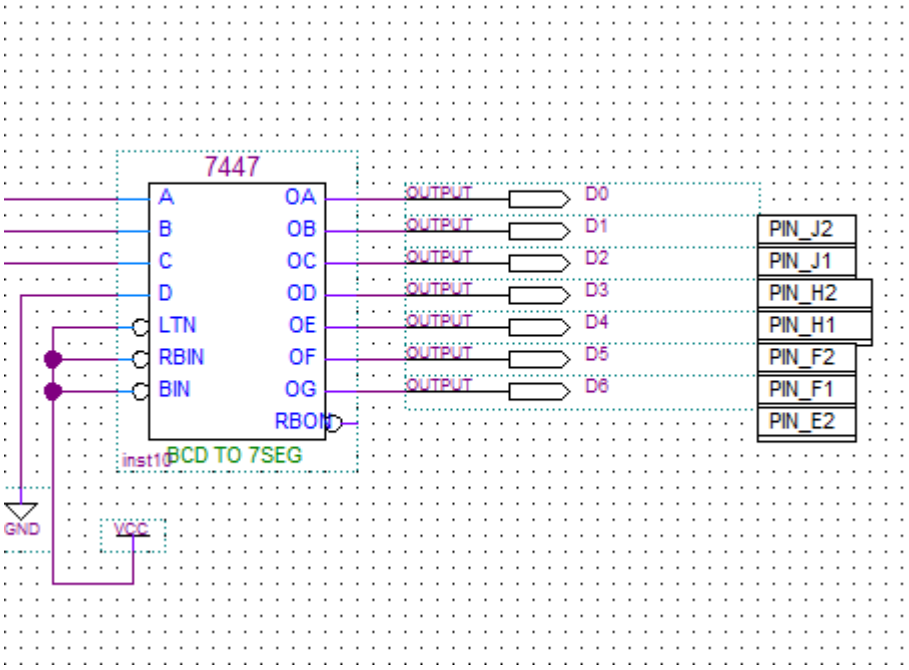


Figura 3. 13: Decodificador de binario a decimal.
Elaborado por: Las Autoras.

Una vez finalizado el diseño esquemático, se realizará la simulacion para constatar el funcionamiento, pero antes se deberá configurar el pin planner. Los pines de salida del circuito BCD están conectados a un display de 7 segmentos de la tarjeta DE1 de Altera. La figura 3.14 muestra el circuito ensamblado en Quartus y que será cargado a la tarjeta DE1 de Altera.

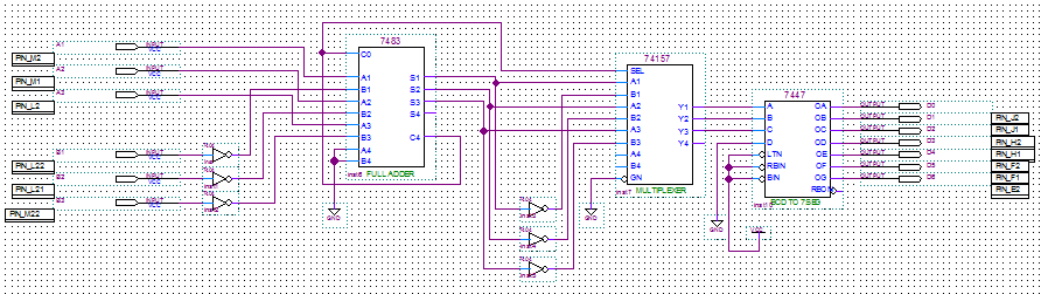


Figura 3. 14: Decodificador de binario a decimal.
Elaborado por: Las Autoras.

A continuacion configuramos el Pin Planner de acuerdo a las necesidades del usuario (véase la figura 3.15).

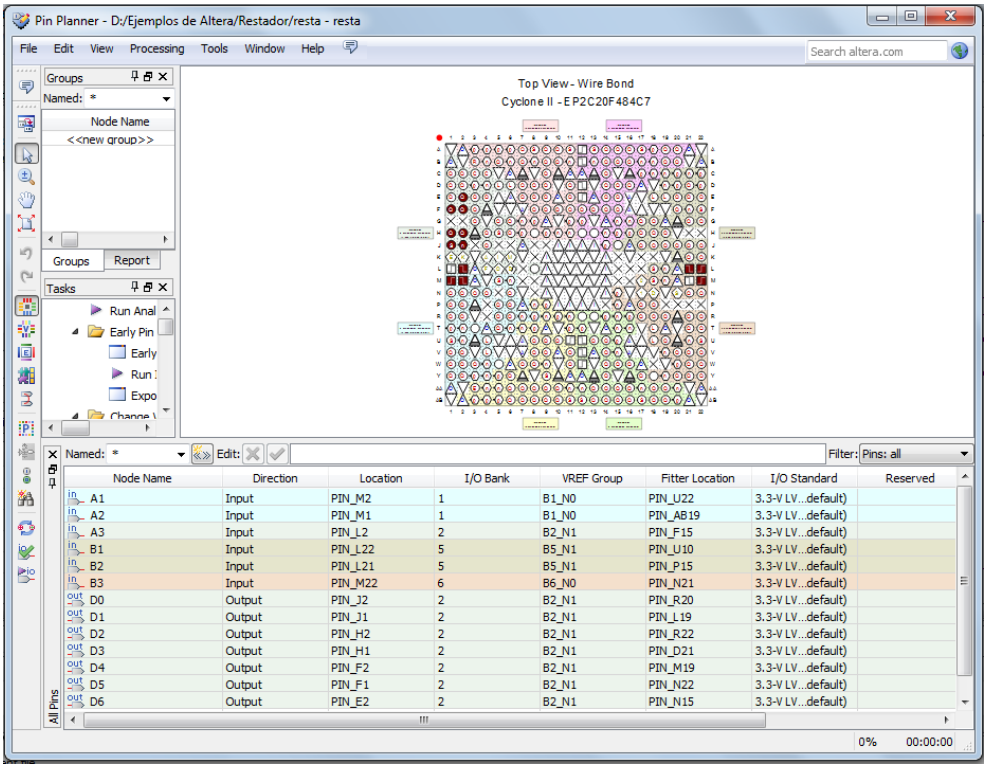


Figura 3. 15: Configuración del Pin Planner de la aplicación práctica #2.
Elaborado por: Las Autoras.

3.3. Aplicación Práctica #3: Circuito contador de 0 hasta 9 en VHDL.

En esta sección se realizará un contador de 0 a 9 mediante programación en VHDL. Primero debemos llamar a la librería <<ieee.std_logic_arith.all>>, para después declarar el entity. Dentro de la entity, declaramos la variable del reloj el cual funcionará manualmente con una señal de pulso dado por el usuario, en el caso de Reloj_in. Así mismo una variable de reloj de salida para que funcione dentro del programa como la señal de reloj interno, en el caso de Reloj_out (véase figura 3.16).

También declaramos la variable “w” este será de tipo vector con 7 segmentos para representar el numero en el display, así por ejemplo tenemos el número 1 en decimal y para representarlo en el display tenemos que apagar y encender ciertos segmentos, asignamos 1001111 a “w” donde “0” enciende el segmento correspondiente y “1” lo tiene apagado.

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3   USE ieee.std_logic_arith.all;
4
5   ENTITY contador IS
6   PORT(
7   Reloj_in: IN STD_LOGIC;
8   w: OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
9   Reloj_out: INOUT STD_LOGIC );
10  END contador;
11
```

Figura 3. 16: Declaración de librerías y Entity del programa.
Elaborado por: Las Autoras.

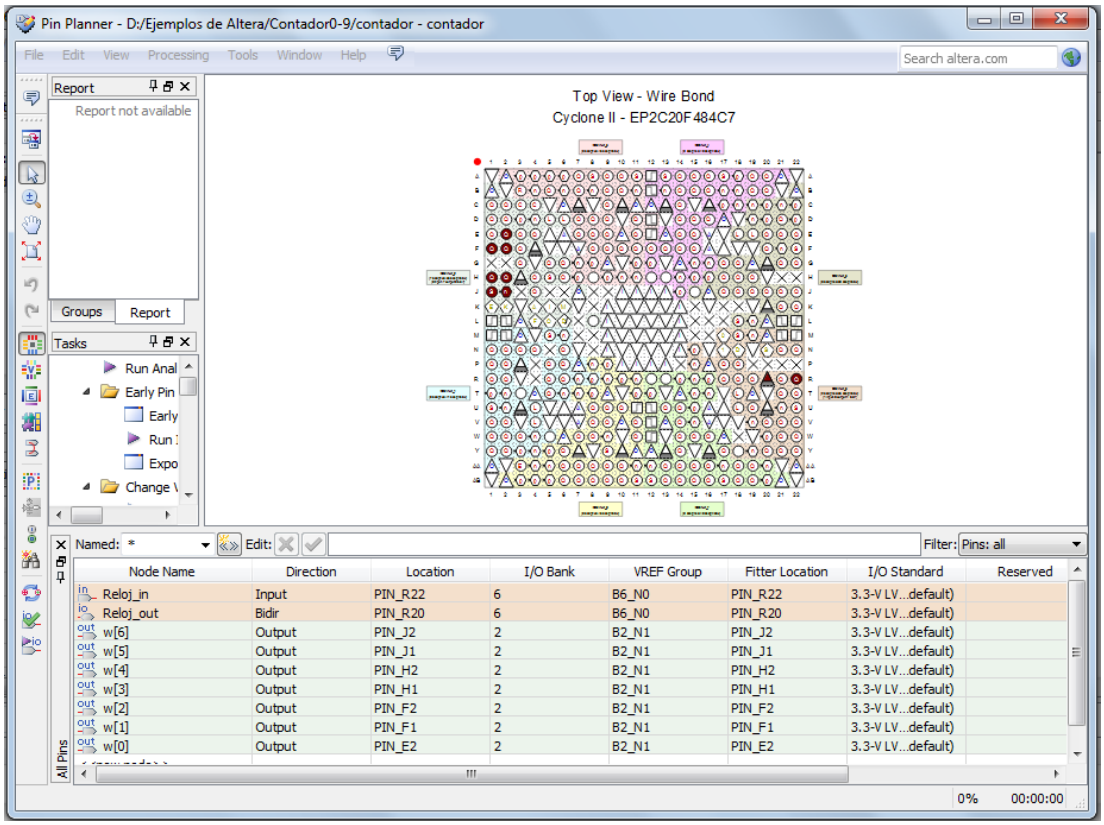
En la arquitectura del programa principal (véase figura 3.17), el usuario da un pulso de reloj por medio de un botón asignado en la tarjeta DE1, cada pulso aumenta en 1 el valor de la variable “con” y es evaluada si ha llegado a su valor máximo que en nuestro caso será el valor de 9, caso contrario sigue sumando ascendentemente hasta llegar al valor límite.

Cada pulso nos ayuda a sumar de uno en uno el valor de una variable la cual será evaluada aparte para que nos asigne un valor a una variable vector y cada cadena de 1 y 0.

```
12  ARCHITECTURE arquit OF contador IS
13
14  SIGNAL con : INTEGER RANGE 0 TO 10;
15
16  BEGIN
17     conteo:PROCESS (Reloj_in,Reloj_out) IS
18        BEGIN
19           IF (Reloj_in = '1' AND Reloj_in'EVENT) THEN
20                          con <= con + 1;
21                       if (con = 9) then
22                            con <= 0;
23                            IF (Reloj_out = '0') THEN
24                                Reloj_out <= '1';
25                            ELSE
26                                Reloj_out <= '0';
27                            end if;
28                       END IF;
29              END IF;
30     END PROCESS;
31
32  with con select
33        w <=    "1001111" when 1,
34                "0010010" when 2,
35                "0000110" when 3,
36                "1001100" when 4,
37                "0100100" when 5,
38                "1100000" when 6,
39                "0001111" when 7,
40                "0000000" when 8,
41                "0001100" when 9,
42                "0000001" when others;
43
44  END arquit;
```

Figura 3. 17: Arquitectura del programa para el contador.
Elaborado por: Las Autoras.

Terminamos la arquitectura del programa y procedemos a realizar la configuración del Pin Planner de la tarjeta altera (véase figura 3.18). Asignamos el reloj a un pulsador y cada segmento de la variable “w” se le dará un segmento del display de 7 segmentos. La localización de cada pin podemos encontrarlo en el manual de la tarjeta.



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved
Reloj_in	Input	PIN_R22	6	B6_N0	PIN_R22	3.3-V LV...default)	
Reloj_out	Bidir	PIN_R20	6	B6_N0	PIN_R20	3.3-V LV...default)	
w[6]	Output	PIN_J2	2	B2_N1	PIN_J2	3.3-V LV...default)	
w[5]	Output	PIN_J1	2	B2_N1	PIN_J1	3.3-V LV...default)	
w[4]	Output	PIN_H2	2	B2_N1	PIN_H2	3.3-V LV...default)	
w[3]	Output	PIN_H1	2	B2_N1	PIN_H1	3.3-V LV...default)	
w[2]	Output	PIN_F2	2	B2_N1	PIN_F2	3.3-V LV...default)	
w[1]	Output	PIN_F1	2	B2_N1	PIN_F1	3.3-V LV...default)	
w[0]	Output	PIN_E2	2	B2_N1	PIN_E2	3.3-V LV...default)	

Figura 3. 18: Configuración del programa en el Pin Planner.
Elaborado por: Las Autoras.

3.4. Aplicación Práctica #4: Reloj automático

En la presente aplicación práctica, explicaremos el diseño de un reloj. Este debe contar segundos y minutos automáticamente sin necesidad de algún pulsador o señal de reloj manual. Este programa explica la manera de utilizar el contador descrito en la sección anterior, pero de manera automática, a cada segundo aumenta un valor y así poder mostrarlo en el display.

Inicialmente declaramos la librería “ieee” y las variables dentro de “entity” tal como se muestra en la figura 3.19.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_arith.all;
4   entity reloj is port(
5   clk: in std_logic;
6   w,z,y,x: out std_logic_vector (6 downto 0);
7   reloj_1, reloj_2, reloj_3, reloj_4: inout std_logic
8   );
9   end reloj;
10
```

Figura 3. 19: Declaración de librerías y el Entity del programa.
Elaborado por: Las Autoras.

Después de realizar la declaración de sus respectivas variables, empezamos a describir la arquitectura del programa. Las variables que se encuentran dentro del arquitectura: cuente1, cuente2, cuente3, cuente4, son de tipo “integer” para que cada vez que se llegue al valor determinado (dependiendo de la frecuencia de reloj interno a trabajar) asignado y aumentado de uno en uno, las variables serán evaluadas a continuación para ser mostrado en el display (véase figura 3.20).

```
11  architecture arqJZ of reloj is
12
13  signal cona, conb, conc, cond: integer range 0 to 10;
14  signal cuente1, cuente2, cuente3, cuente4: integer range 0 to 1625000000;
15
```

Figura 3. 20: Inicio de la arquitectura del programa.
Elaborado por: Las Autoras.

Dentro de la arquitectura describimos los procesos que se encargan de contar los segundos, y los minutos, separando por unidad y decenas en ambos casos. Total tendremos tres procesos para realizar esta función, que serán llamados conteo1, conteo2, conteo3 (véase las figuras 3.21, 3.22, 3.23).

```
16  begin
17      conteo1: process (clk, cuente1) is
18      begin
19          if ( clk 'event and clk = '1') then
20                cuente1 <= cuente1 + 1;
21                   if cuente1 = 27000000 then
22                      cuente1 <=0;
23                      cona <= cona + 1;
24                      if (reloj_1 = '0') then
25                         reloj_1 <= '1';
26                      else
27                         reloj_1 <= '0';
28                      end if;
29                   end if;
30                   if (cona=10) then
31                      cona <=0;
32                   end if;
33          end if;
34      end process;
```

Figura 3. 21: Primero proceso en contar segundos.
Elaborado por: Las Autoras.

```
37  conteo2: process (clk, cuente2) is
38      begin
39          if ( clk 'event and clk = '1') then
40                cuente2 <= cuente2 + 1;
41                   if cuente2 = 270000000 then
42                      cuente2 <=0;
43                      conb <= conb + 1;
44                      if (reloj_2 = '0') then
45                         reloj_2 <= '1';
46                      else
47                         reloj_2 <= '0';
48                      end if;
49                   end if;
50                   if (conb=6) then
51                      conb <=0;
52                   end if;
53          end if;
54      end process;
```

Figura 3. 22: Segundo proceso para contar decimas de segundos.
Elaborado por: Las Autoras.

```
56  conteo3: process (clk, cuente3, cuente4) is
57      begin
58          if ( clk 'event and clk = '1') then
59                cuente3 <= cuente3 + 1;
60                   if cuente3 = 1620000000 then
61                      cuente3 <=0;
62                      conc <= conc + 1;
63                      if (reloj_3 = '0') then
64                         reloj_3 <= '1';
65                      else
66                         reloj_3 <= '0';
67                      end if;
68                   end if;
69                   if (conc=10) then
70                      conc <=0;
71                      if (reloj_4 = '0') then
72                         reloj_4 <= '1';
73                      else
74                         reloj_4 <= '0';
75                      end if;
76                      cuente4 <= cuente4 + 1;
77                      cuente4 <=0;
78                      cond <= cond + 1;
79                   end if;
80          end if;
81      end process;
```

Figura 3. 23: Tercer proceso para contar minutos.
Elaborado por: Las Autoras.

Después de la descripción de los procesos que presentamos en las figuras anteriores, sin terminar aún la arquitectura del programa pasamos a evaluar los resultados de las variables cona, conb, conc, cond, para asignar a los vectores w, x, y, z asignando cada uno a un display diferente para poder visualizar el resultado de nuestro programa VHDL (véase en la figura 3.24 y 3.25).

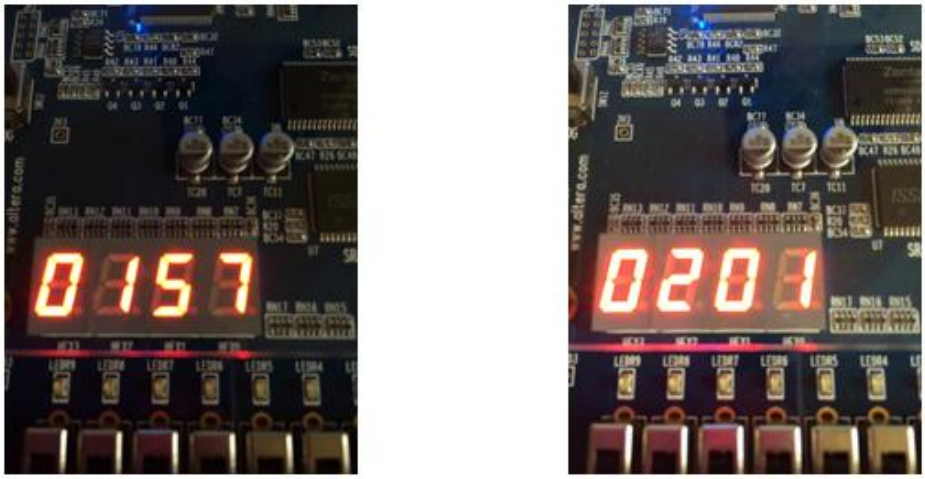
```
82  with cona select
83     w <= "1001111" when 1,
84          "0010010" when 2,
85          "0000110" when 3,
86          "1001100" when 4,
87          "0100100" when 5,
88          "1100000" when 6,
89          "0001111" when 7,
90          "0000000" when 8,
91          "0001100" when 9,
92          "0000001" when others;
93
94  with conb select
95     z <= "1001111" when 1,
96          "0010010" when 2,
97          "0000110" when 3,
98          "1001100" when 4,
99          "0100100" when 5,
100         "1100000" when 6,
101         "0000001" when others;
102
```

Figura 3. 24: Evaluación y asignación de variables.
Elaborado por: Las Autoras.

```
102
103 with conc select
104    y <= "1001111" when 1,
105         "0010010" when 2,
106         "0000110" when 3,
107         "1001100" when 4,
108         "0100100" when 5,
109         "1100000" when 6,
110         "0001111" when 7,
111         "0000000" when 8,
112         "0001100" when 9,
113         "0000001" when others;
114
115 with cond select
116    x <= "1001111" when 1,
117         "0010010" when 2,
118         "0000110" when 3,
119         "1001100" when 4,
120         "0100100" when 5,
121         "1100000" when 6,
122         "0000001" when others;
123
124
125 end arqJZ;
```

Figura 3. 25: Evaluación y asignación de variables.
Elaborado por: Las Autoras.

Obviamente el reloj no cuenta con un display más para asignar horas, ni tampoco una función para poder calibrar la hora y minutos antes de empezar pero esto a medida que el usuario se interese en la programación VHDL podrá agregarlo a este código de programación. En las siguientes imágenes presentaremos el funcionamiento del problema ya llevado a la tarjeta Altera (véase figura 3.26 (a) y (b)).



(a) (b)

Figura 3. 26: Funcionamiento del programa en la tarjeta altera.
Elaborado por: Las Autoras.

En la configuración del PIN PLANNER (véase la figura 3.28) observaremos que la variable “clk” es asignada al PIN_D12, este si revisamos el manual (ver figura 3.27) nos detalla que:

Signal Name	FPGA Pin No.	Description
CLOCK_27	PIN_D12, PIN_E12	27 MHz clock input
CLOCK_50	PIN_L1	50 MHz clock input
CLOCK_24	PIN_A12, PIN_B12	24 MHz clock input from USB Blaster
EXT_CLOCK	PIN_M21	External (SMA) clock input

Figura 3. 27: Velocidades de reloj de la tarjeta altera.
Elaborado por: Las Autoras.

He aquí la explicación del porque las variables cuente 1, al terminar de contar 27000000 toman el valor de 1, eso para los segundos. Pero en el caso de los números del siguiente nivel como décimas, minutos y decimas de minutos van a variar dependiendo, por ejemplo: Si 27000000 es 1 segundo, para 10 segundos el conteo será hasta 270000000, y para 1 minuto seria 1620000000 y así mismo para las decimas de minutos.

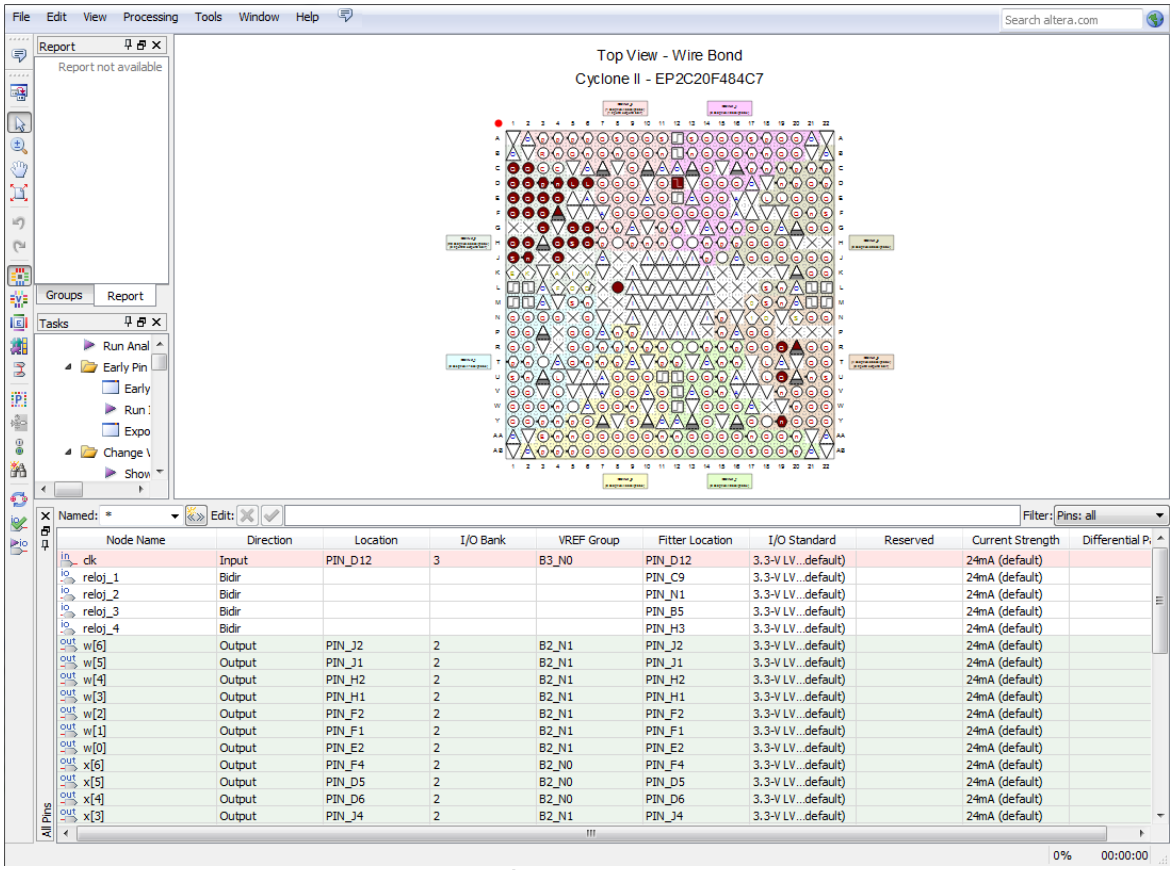


Figura 3. 28: Configuración del circuito en el Pin Planner.
Elaborado por: Las Autoras.

CAPÍTULO 4: CONCLUSIONES Y RECOMENDACIONES.

4.1. Conclusiones.

Se ha presentado inicialmente una breve introducción de una arquitectura FPGA tradicional, y el flujo de software relacionado para programar diseños de hardware en una FPGA. También se describieron varias aproximaciones empleadas para reducir algunas desventajas de FPGAs y ASICs, con o sin comprometer a sus principales ventajas.

Se ha comprobado que la aplicación didáctica de la tarjeta Altera es posible ya que se puede simular un circuito digital utilizando el programa de simulación Quartus II y llevarlo a la practica con la tarjeta FPGA Altera, con esto evitamos comprar diferentes elementos electrónicos siendo más económico y a la vez eficiente.

Esta plataforma no solo es posible utilizarla entre programadores, sino también con personas que son diseñadores de circuitos electrónicos digitales, ya que tienen varias opciones para la creación de un proyecto, y así abarcar a una gran cantidad de usuarios que pueden utilizar el FPGA Altera.

4.2. Recomendaciones.

La tarjeta DE1 de Altera permite abarcar todo el programa de estudios de los sistemas digitales 1 y 2, así como laboratorio de digitales; para lo cual

es necesario que se enseñe a programar en lenguaje VHDL en la asignatura de Sistemas Digitales 1.

Realizar investigaciones a través de la tarjeta DE1 de Altera para propuestas de trabajos de investigación básica (pregrado) y avanzada (posgrado).

El laboratorio de electrónica tiene disponibilidad para 18 computadoras, para lo cual es necesario que el cupo de la asignatura Laboratorio de Digitales sea máximo 18 estudiantes.

REFERENCIAS BIBLIOGRÁFICAS

Acosta, L. E., Duque, J., Granados, J., & Fiallo, S. (2011). *Metodología de Diseño para Sistemas Complejos en FPGA.* 9no. LACCEI, Colombia.

Boemo S., E. (2005). *Estado del Arte de la Tecnología FPGA.* Proyecto Mejora de la Eficiencia y de la Competitividad de la Comunidad Argentina. Instituto Nacional de Tecnología Industrial (INTI), Buenos Aires, Argentina.

Brengi, D. Tropea, S., & Borgna, J. (2007). *Tarjeta de diseño abierto para desarrollo y educación.* Mejora de la Eficiencia y de la Competitividad de la Comunidad Argentina. Instituto Nacional de Tecnología Industrial (INTI), Buenos Aires, Argentina.

Brown, S., & Vranesic, Z. (2007). *Fundamentos de Lógica Digital con Diseño en VHDL*. 2da Edición. McGraw-Hill, México.

Camargo, C., Cortés, J., & Jiménez, A. (2012). *Implementación de sistemas digitales complejos utilizando sistemas embebidos.* INGENIUM, 13(25), 5-15.

Castillo, A., Vázquez, J., Ortegón, J., & Rodríguez, C. (2008). *Prácticas de laboratorio para estudiantes de ingeniería con FPGA.* IEEE Latin America Transactions, 6(2), 130-136.

De Pablo, S., Cáceres, S., Cebrián, J. A., & Sanz, F. (2010). *Diseño de Circuitos Digitales a Nivel de Registro empleando Diagramas ASM++.* Información tecnológica, 21(2), 91-102.

Gómez, M, Goy, C., & Herrera, M. (2013). *Design, Implementation and Evaluation of FPGA Embedded Digital Systems Course at the University Level.* IEEE Latin America Transactions, Volume 11, No1.

Grout, I. (2011). *Digital Systems Design with FPGAs and CPLDs.* Newnes. ISBN 9780080558509.

Herrera L., J. C. (2013). *Implementación de un controlador difuso en un FPGA, utilizando lenguajes de descripción de hardware*.

Ling, A., Singh, D., & Brown, S. (2005). *FPGA Technology Mapping: A Study of Optimality*. 7ma Conferencia Internacional sobre Teoría y Aplicaciones de Satisfacibilidad.

Machado, F., Borromeo, S., & Rodríguez-Sánchez, M. C. (2011). *Diseño de sistemas digitales con VHDL.*

Melo, H., & Pérez, A. (2010). *Aplicaciones de herramientas de diseño y simulación en alto nivel para implementaciones en FPGAs.* Primer Congreso de Microelectrónica Aplicada. Buenos Aires, Argentina.

Muñoz, A. R., Bataller-Mompeán, M., & Guerrero-Martínez, J. (2008). *Aprendizaje por Proyectos: Una Aproximación Docente al Diseño Digital Basado en VHDL.* IEEE-RITA, 3(2), 87-95.

Prieto, J., Ramos, O., & Delgado, A. (2007). *Diseño de un Gene Digital en FPGA Y Matlab con Aplicaciones en Robótica móvil.* XIII Taller Iberchip, 14-16.

Pérez, A. Gutiérrez, F., Cavallero, R., & Contreras, J. (2010). *Desarrollo de Sistemas embebidos en FPGAs. Diseño e incorporación de periféricos.* Primer Congreso de Microelectrónica Aplicada. Buenos Aires, Argentina.

Rondón, J., & Sandoval, C. (2010). *Diseño de un co-laboratorio remoto basado en programación modular de dispositivos VHDL aplicado a telecomunicaciones.* Revista de la Facultad de Ingeniería Universidad Central de Venezuela, 25(2), 7-12.

Schiavon, M., & Crepaldo, D. (2010). *Formación de Ingenieros de Diseño de Circuitos/Sistemas Electrónicos.* Primer Congreso de Microelectrónica Aplicada. Buenos Aires, Argentina

Shiva, S. (2013). *Computer Organization, Design, and Architecture.* 5ta Edition. CRC Press. ISBN 9781466585546.

Tropea, S., Brengi, D., & Borgna, J. (2006). *FPGA Libre: Herramientas de Software Libre para diseño con FPGAs.* Mejora de la Eficiencia y de la Competitividad de la Comunidad Argentina. Instituto Nacional de Tecnología Industrial (INTI), Buenos Aires, Argentina.

Zambrano G., F. A., (20014). *Demodulación PSK y FSK: Implementación en FPGAs.* Repositorio digital de la Universidad Católica de Santiago de Guayaquil.