UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

Diseño y Desarrollo de una Tarjeta Entrenadora y Probadora del CPLD 7C373i

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Ingeniero en Computación

P R E S E N T A:

Rodrigo López Rojas

ASESORA DE TESIS:
M. en I. Arcelia Bernal Díaz

México 2013

“Hay una fuerza motriz más poderosa que el vapor, la electricidad y la energía atómica: la voluntad.”

-ALBERT EINSTEIN-

Dedicatoria

A la memoria de mi abuela Ofelia,

a quien mucho le hubiera gustado ver llegar este momento.

Agradecimientos

A Dios por prestarme la vida suficiente y el entendimiento de hacer las cosas de una manera correcta, por brindarme la serenidad de levantarme con calma y estrategia cuando suelo caer y quien con su luz me motivó e inspiró a seguir de pie y hacía adelante

A mis padres Dagoberto López y Lourdes Rojas los cuales fueron y seguirán siendo, maestros de gran parte de mi vida y me brindaron el arma más poderosa de todas, una carrera como profesionista, aun y a pesar de que la vida siga sin estar físicamente a su lado, siempre me quedará en el alma el recuerdo del tesón para sacar a sus hijos adelante que espero heredar.

A la Universidad Nacional Autónoma de México y a mi asesora la M. en I. Arcelia Bernal quien me sugirió este proyecto y le dio seguimiento y retroalimentación hasta su culminación. Gracias por creer en mí en un momento que ni yo mismo lo hacía.

A mí por darme cuenta que el empeño es algo que a veces se esconde pero se encuentra en las ganas de hacer las cosas bien y por entender que una tesis profesional no solo es un trabajo escrito, más bien es un esfuerzo que significa compromiso y disciplina en la vida.

A mis hermanas Marysol y Karen con quienes he compartido gran parte de mi vida y sé que puedo contar con ellas siempre. A las personas que en el momento en que había perdido la brújula y la fe me dieron las palabras de aliento adecuadas para seguir adelante sin mirar de nuevo hacia atrás.

A mi abuela Delfia quien tuvo la paciencia de educarme como madre durante muchos años y a mi familia en general; abuelos, tíos, primos y amigos a quienes aprecio tanto y me toleran a pesar de que no debe ser fácil. Infinitas Gracias.

Y finalmente un agradecimiento especial a Ara; quien con su sonrisa, ayuda y objetividad me explicó sin palabras que podía contar con ella siempre.

ÍNDICE

LISTA DE TABLAS

LISTA DE FIGURAS

INTRODUCCIÓN

En la actualidad en el área de la electrónica existen circuitos integrados que soportan la programación con lógica digital para hacer una tarea específica, a estos componentes electrónicos se les denominó PLDs (Programmable Logic Device, Dispositivos Lógicos Programables) los cuales facilitan las prácticas, ya que a comparación de las compuertas lógicas y otros chips que tienen una función fija y predeterminada para poder realizar su trabajo, éstos dispositivos tienen una función indefinida, entonces el usuario antes de integrarlos a cualquier circuito debe de programar el tipo de entradas y/o salidas que él mismo requiere para la realización de una tarea determinada, esto resulta muy útil y hace de estos circuitos integrados, dispositivos con una gran funcionalidad. Ahora existen PLDs complejos o CPLDs (*Complex Programmable Logic Device*, Dispositivo Lógico Programable Complejo) que son componentes con mayor escala de integración con los cuales se puede trabajar trayendo consigo claras ventajas como su uso en la realización de sistemas que necesiten más entradas o salidas en un mismo dispositivo, utilizando un menor espacio y reduciendo costos puesto que un CPLD equivale a varios PLDs sencillos.

Los CPLDs tienen la ventaja de que se les puede modelar un diseño desde una computadora, el cual será introducido a estos dispositivos para poder crear un circuito que resuelva las necesidades que se planteen desde el inicio de un proyecto. La lógica programable que es introducida a estos dispositivos se diseña y simula en un lenguaje creado para el modelado de circuitos electrónicos denominado VHDL (Hardware Description Lenguaje, Lenguaje de Descripción de Hardware), este lenguaje también será estudiado en este trabajo para poder entender su estructura, sintaxis e inclusive se darán algunos ejemplos y códigos completos, entre otras características propias del software, para que el lector interesado entienda y pueda practicar los términos aprendidos.

Al buscar conjuntar las ventajas que ofrece el poder introducir un código a un dispositivo lógico programable, se propone crear una tarjeta en la cual se pueda contener una etapa para que dicho dispositivo pueda ser programado, reprogramado y probado en repetidas ocasiones con diseños diferentes utilizando la lógica programable de VHDL. En ocasiones los equipos disponibles para la programación de CPLDs que se venden en el mercado tienen un alto costo y no son accesibles para los estudiantes, además para realizar una práctica se sumaría el costo de una tarjeta de prueba y los dispositivos que se requieran para exhibir un circuito funcionando. Así que se busca crear una tarjeta integrada, que conjunte estas dos etapas en una sola fase, para ahorrar tiempo y a la vez economizar gastos, puesto que se propone también que la tarjeta tenga una etapa de pruebas integrada a la misma placa, para poder tener una comprobación física de que el dispositivo está funcionando correctamente.

Lo que se pretende con este trabajo es que las personas que interactúen con estos dispositivos, puedan conocer este tipo de componentes desde el inicio. Además que el alumno se introduzca en la programación del modelado de circuitos en el lenguaje VHDL para poder realizar desde el inicio un proyecto funcional en un plano físico, entendiendo también la programación propia de los dispositivos lógicos programables y controlando la etapa de pruebas de la tarjeta entrenadora al utilizarla.

El objetivo general; diseñar y construir una tarjeta grabadora, entrenadora y probadora del CPLD 7C373i que contenga prácticas para las materias dedicadas al hardware de la carrera de Ingeniería en Computación de la Facultad de Estudios Superiores Aragón.

El texto está dividido en tres capítulos, de los cuales se da una breve introducción a continuación.

El primer capítulo explica los antecedentes históricos de los dispositivos lógicos programables y describe la arquitectura del CPLD necesaria para comprender su funcionamiento, además de las particularidades que estos chips tienen, también se

exponen más a detalle las características del CPLD CY7C373i-66JC con el cual se trabajará.

En el segundo capítulo se expone la estructura del software que se utilizará para programar el CPLD de manera más detallada, así como las características específicas del lenguaje de programación y su ambiente de trabajo; también en este capítulo se fomentará la creatividad del alumno o lector para programar y probar sus propios diseños.

En el tercer capítulo se presentará la tarjeta grabadora y probadora del CPLD 7C373i-66JC y se explicará su funcionamiento a detalle, y está dividido en el diseño y la construcción de cada una de las etapas funcionales que contiene la tarjeta entrenadora y probadora.

Para finalizar se muestran también pruebas y resultados del trabajo así como las conclusiones finales, el texto cuenta además con un Anexo, el cual es un compilado de códigos que vienen explicados de manera breve, cada uno de ellos tiene el objetivo de que el alumno aprenda conceptos nuevos, al mismo tiempo que se introduce al lenguaje VHDL. En el Anexo A se describen códigos completos que fueron creados, probados y simulados en VHDL, para que el alumno pueda reproducirlos sin temor a cometer errores y así comprobar la etapa de pruebas de la tarjeta entrenadora y probadora.

La principal aportación que tendrá esta tarjeta entrenadora y probadora es la de introducir a los alumnos con el CPLD 7C373i al diseño de circuitos digitales tanto en software como en hardware, ya que con esta tarjeta se puede tener una opción más para resolver algún problema. Se puede comenzar desde una práctica sencilla y programando de manera progresiva, se puede lograr la elaboración y diseño de un circuito complejo ayudando así a algunas de las materias relacionadas.

Capítulo I

El CPLD a Programar y sus Particularidades

En este capítulo se describirá un poco de historia de la lógica programable, la importancia del desarrollo de los circuitos integrados para la ingeniería y la evolución de los dispositivos lógicos programables, desde el primer PLD (Programmable Logic Device ó Dispositivo Lógico Programable) hasta llegar al CPLD (Complex Programmable Logic Device ó Dispositivo Lógico Programable Complejo) que se ocupará para la realización de este trabajo, se explicará también las características de este último y se describirán las definiciones de algunos términos a los lectores que no estén familiarizados con la materia. El texto se enfocará en su mayoría en dicho dispositivo, y se dará principal interés al CPLD 7C373i-66j con el cual se trabajará explicando sus características, ventajas, desventajas y motivos por los que es utilizado.

1.1 Breve historia de los dispositivos lógicos programables

Es complicado hacer una historia breve teniendo en mente los avances y el paso a paso de la electrónica hasta llegar a la lógica programable actual. Para los jóvenes que han crecido en generaciones recientes es difícil de entender las características de un televisor de bulbos, y a las personas hemos podido ver como ha evolucionado la tecnología es verdaderamente increíble pensar que las historias de ciencia ficción hoy puedan ser una realidad en los equipos móviles de sonido, en dispositivos de telefonía celular, computadoras de última generación, estaciones de juegos interactivos y enormes televisiones con pantalla plana o simplemente portátiles.

Todo esto ha sido posible gracias a que poco a poco la nanoelectrónica[1] se ha desarrollado en casi cualquier componente electrónico. Los circuitos integrados o chips, como generalmente se les conoce, han tenido una serie de modificaciones

[1] Referente al uso de la nanotecnología en componentes electrónicos, generalmente transistores. Usado normalmente para definir la tecnología de menos de 100 nm (Nanómetros) de tamaño.

para no solo ahorrar espacio, sino también para obtener muchas otras ventajas, por ejemplo, reducir la longitud de las interconexiones, por consecuencia se agiliza la velocidad en el mismo circuito y también se ahorra el consumo de potencia, además la confiabilidad del chip va en aumento, entre otras mejorías.

1.1.1 Definición y antecedentes de los Circuitos Integrados

Un circuito integrado (abreviado **CI**) es una pequeña pastilla de silicio que contiene componentes eléctricos. Los diversos componentes están interconectados dentro de la pastilla para formar un circuito electrónico, sus conexiones están soldadas a terminales externas que conforman el mismo chip.[2] Los circuitos integrados se clasifican por escalas de integración de acuerdo a su complejidad, las escalas de integración se definen dependiendo el número de compuertas lógicas que puedan encapsularse dentro del chip. En seguida se enumeran de menor a mayor complejidad:

- **SSI** (Small Scale Integration ó Short Scale Integration). Sus siglas en inglés significan la escala de integración más pequeña, y comprende los circuitos integrados compuestos por menos de 12 compuertas, como ejemplo están las compuertas primitivas (compuertas lógicas AND, OR y NOT las cuales son básicas para obtener cualquier función) y flip-flops.[3]
- **MSI** (Medium Scale Integration). Por sus siglas en inglés mediana escala de integración en este nivel se encuentran los circuitos integrados cuyo número de compuertas oscila entre los 12 y 100 elementos, como ejemplo están los codificadores, contadores y multiplexores.[4]

[2] M. Morris Mano. Lógica Digital y Diseño de Computadores. pp 31

[3] Flip-Flop: Circuito lógico encargado de almacenar la información de la memoria durante un tiempo indefinido. Según mastermagazine en español.
http://www.mastermagazine.info/termino/5022.php

[4] Multiplexor: es un circuito con 2 a la *n* líneas de entrada y una línea de salida; también contiene otras *n* líneas de entrada de selección, para determinar cuál de las entradas de datos se conectará con la única salida.

- **LSI** (Large Escale Integration). Sus siglas en ingles se traducen como circuitos de alta escala de integración, y lo comprenden chips que contienen de 100 hasta las 1,000 compuertas, se pueden encontrar en memorias, unidades aritmética/lógica (ALUs) y microprocesadores pequeños.
- **VLSI** (Very Large Escale Integration): Significan circuitos integrados de muy alta escala de integración, en esta jerarquía se encuentran los chips que contienen de 1,000 a 10,000 compuertas, en esta categoría se incluyen microprocesadores, FPGAs (Field Programmable Gate Array, Arreglos de Compuertas Programables en Campo) y CPLDs (Complex Programmable Logic Device, Dispositivos Lógicos Programables Complejos) los cuales se expondrán más adelante.

Los circuitos integrados han sido siempre indispensables para industrias importantes como lo son las áreas: militar, médica, aeroespacial, electrodoméstica, por citar algunas, de hecho, gran parte del desarrollo tecnológico surge gracias a la necesidad creciente que estas empresas tienen de nuevos métodos afines con su propia modernización.

1.1.2 Dispositivos lógicos programables y un poco de historia

Los PLDs (Programmable Logic Devices, Dispositivos Lógicos Programables) se pueden definir como componentes electrónicos empleados para la fabricación de circuitos digitales programables. A diferencia de los circuitos como las compuertas básicas que ya tienen una función preestablecida el PLD tiene una tarea indefinida al momento de fabricarse, por tanto antes de que este dispositivo pueda ser usado en un circuito, este debe ser programado.[5]

Los dispositivos lógicos programables fueron introducidos aproximadamente a mediados de la década de 1970, la propuesta era construir circuitos lógicos que pudieran ser programables, esto significaba que estuvieran constituidos de

[5] Definición de PLD. Alegsa; Diccionario de informática. http://www.alegsa.com.ar/Dic/pld.php

arreglos con combinaciones de compuertas AND y OR, recordando que con estas compuertas básicas podemos tener cualquier función a consecuencia de una operación booleana, y estos arreglos programables contenidos dentro del los dispositivos encontraban el objetivo de convertir un resultado y devolverlo a la salida de alguna de sus terminales. Esta idea proponía que al contrario de los dispositivos preprogramados u otros que poseen un hardware fijo, los PLDs permitieran también la modificación a nivel hardware. Con esto se tendría un dispositivo flexible ya que podría ser moldeable tanto en software como en hardware para un fin específico.

Los PLDs están creados de dos arreglos programables, uno de compuertas lógicas AND y otro de compuertas lógicas OR, por la misma razón que ya se ha explicado con anterioridad, mencionando que de estas compuertas se puede obtener cualquier resultado. A estos arreglos se les denomina matriz o arreglo programable, estos son una red de conductores distribuidos en filas y columnas, las cuales cuentan con un fusible en cada punto de intersección, la programación de dicho arreglo consiste en fundir o apagar los fusibles en los puntos de intersección para eliminar las variables que no serán utilizadas.

Al estar trabajando con fusibles es importante saber que una vez que estos se funden no pueden volver a programarse,[6] es decir, que la función que se requiera programar dentro del PLD, quedará grabada para siempre en este dispositivo sin poder modificarse en un futuro.

En la figura 1.1 se muestra un arreglo de compuertas AND no programado y otro ya programado, los fusibles que quedan intactos son los que conectan la variable seleccionada o los complementos de la variable, es decir, dicha variable negada con la entrada de la compuerta o compuertas que serán utilizadas para crear nuestro resultado.

[6] Maxines David, Jessica Alcalá, VHDL: El arte de programar sistemas digitales. México, DF. 2005 pp. 4 y 5

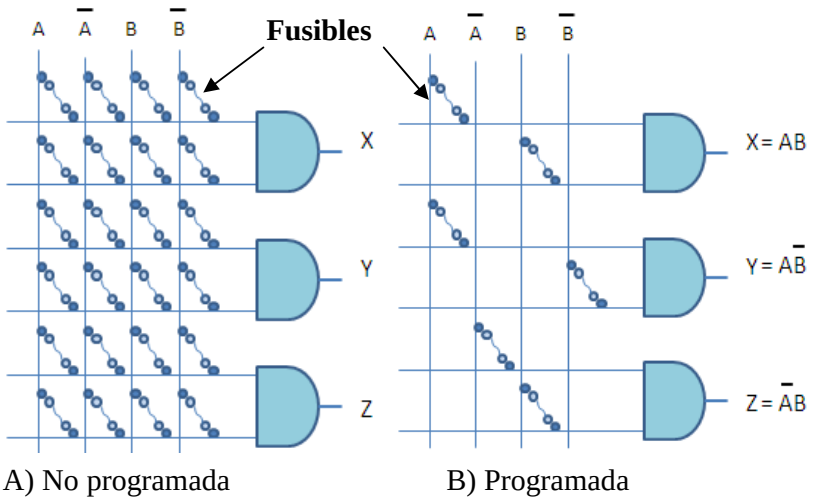


Figura 1.1 Arquitectura de una matriz de compuertas AND *con fusibles*

En la figura 1.1 se muestra claramente como los fusibles a los que se les dio la instrucción de que no fueran utilizados por medio de programación, se fundieron y ya no aparecen en el arreglo programado, haciendo clara la instrucción de multiplicar las variables que sí se conectan. Las matrices de las compuertas OR son exactamente iguales, solo que el arreglo estará conectado al final a sumadores lógicos, la desigualdad obvia es que la operación es diferente. En la figura 1.2 se muestra un ejemplo de un arreglo de compuertas OR, de igual manera que la imagen anterior se observa una matriz nueva y una ya programada.

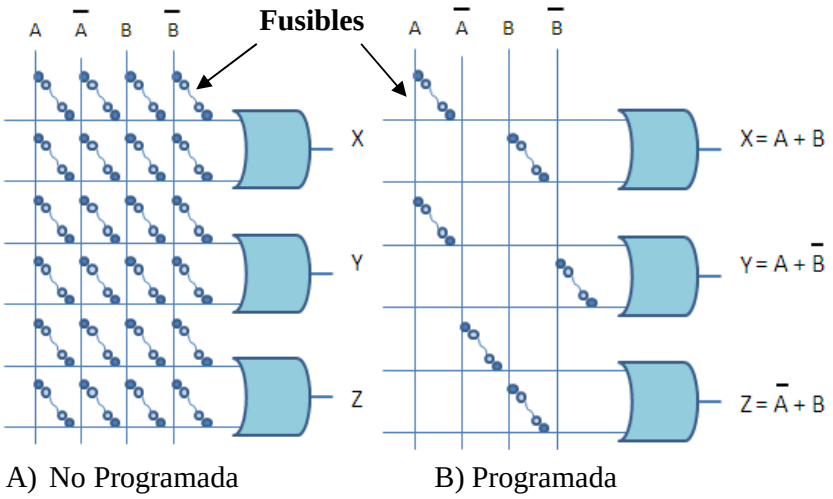


Figura 1.2 Arquitectura de una matriz de compuertas OR *con fusibles*

Uno de los primeros dispositivos lógicos que utilizaron este tipo de tecnología compuesta de dos arreglos programables fueron los PLA (*Programmable Logic Array*, Arreglo Lógico Programable), seguidos de estos, se intentaron mejorar con unos chips casi con las mismas características denominados PAL (*Programmable Array Logic*, Lógica de Arreglos Programables), algún tiempo después se crearon otros circuitos renovados nombrados GAL (*Generic Logic Array*, Arreglo Lógico Genérico) con la capacidad de programarse más de una vez, y finalmente con una estructura similar a la de múltiples GAL interconectadas dentro de un dispositivo hallamos al CPLD (*Complex Programmable Logic Device*, Dispositivo Lógico Programable Complejo). A continuación se señalan algunos de las características de cada uno de los circuitos integrados y sus mejoras a través del tiempo.

1.1.2.1 PLA (Arreglo Lógico Programable)

Las primeras empresas que comenzaron a fabricar PLDs crearon un dispositivo que tuviera una matriz OR programable y aunado a eso también una matriz AND programable, este circuito tomó el nombre de PLA por sus siglas *Programmable Logic Array*. En la figura 1.3 se muestra un diagrama de flujo que toman los datos dentro de dicho dispositivo.

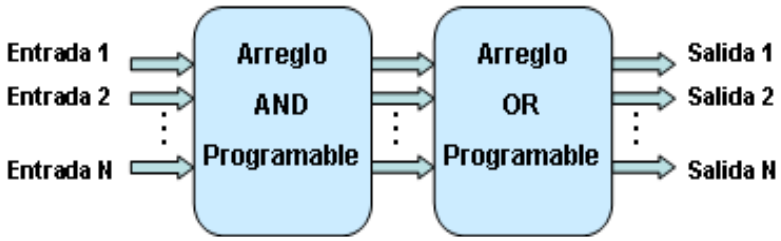


Figura 1.3 Arquitectura interna del dispositivo PLA

Por tener ambas matrices programables, esta arquitectura resultó ampliamente flexible, pero debido a su arquitectura generaba un retraso grande de la entrada con respecto a la salida, lo que hizo que estos nuevos dispositivos fueran reemplazados rápidamente por una nueva tecnología.

1.1.2.2 PAL (Lógica de Arreglos Programables)

Algunos años después de que se dieran a conocer los PLA se modificó su arquitectura para dar paso al los dispositivos PAL para superar algunas limitaciones del PLA, se determinó dejar como fija una de las dos matrices programables, ya que la implementación de fusibles adicionales que resultaban de dos arreglos programables y de la complejidad del circuito eran la causante de un retraso o *delay.*

Esta arquitectura se encuentra formada por los arreglos AND programables y arreglos OR fijos, resultó con menos retraso que la tecnología anterior, aunque no tenía toda la flexibilidad que la estructura PLA era funcional y menos tardado. En la figura 1.4 se muestra el diagrama de un dispositivo PAL, se puede notar su similitud con la tecnología anterior con la clara diferencia en uno de los arreglos.

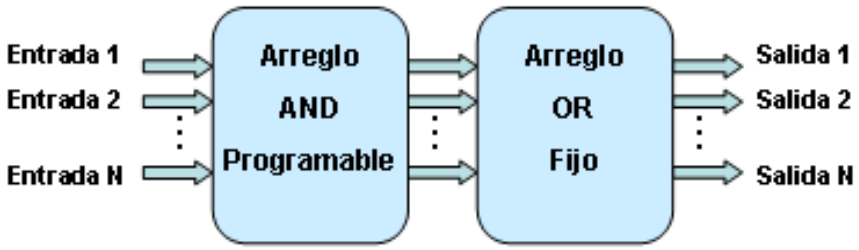


Figura 1.4 Arquitectura interna del dispositivo PAL

Lo que terminó con esta tecnología no fue un cambio en la arquitectura, si no el encontrar la manera de poder volver a programar el dispositivo, después de ser programado con anterioridad.

1.1.2.3 GAL (Arreglo Lógico Genérico)

Un desarrollo más reciente son los dispositivos GAL, esta arquitectura aun se fabrica en chips independientes. Son muy similares a las PAL, ya que se conforma con una matriz AND programable y OR fija. La principal diferencia entre uno y otro se encuentra en que las PAL usan fusibles, por tanto no pueden ser reprogramables, como ya se mencionó anteriormente. Los dispositivos GAL

se pueden programar una y otra vez, ya que utilizan tecnología diferente y renovada dejando atrás la técnica de fundir fusibles.

Este hecho en el proceso evolutivo de los dispositivos lógicos programables es considerado un gran avance, ya que siendo posible que los dispositivos fueran reprogramables se podría ocupar un mismo dispositivo para ejemplificar varias funciones, hacer las modificaciones pertinentes para perfeccionar una operación o modificar por completo una función con el mismo chip.

La tecnología que se implementó para poder reprogramar los dispositivos se denomina CMOS (*Complementary Metal Oxide Semiconductor*, Estructuras Semiconductor-Oxido-Metal Complementarias). El término CMOS se puede definir como: Un tipo de tecnología de semiconductores que permite controlar el paso de la corriente a través de sus terminales. Este tipo de semiconductor se caracteriza por utilizar dos transistores[7] conjuntos, uno de polaridad negativa para arrojar un valor binario bajo o cero lógico y otro de polaridad positiva que arroja un valor binario alto o uno lógico, esto provoca poder conectar o desconectar justo como los fusibles pero sin que este elemento deje de funcionar. Además dado que sólo un tipo de transistor está activo en un tiempo determinado, requiere de menos voltaje. Su bajo consumo de energía, lo hace atractivo para operar en máquinas que requiere un mínimo consumo eléctrico.[8]

Otra de las características importantes de los circuitos CMOS es que son regenerativos, es decir, una señal disminuida que llegue a una compuerta lógica CMOS se verá restaurada a su valor lógico inicial 0 ó 1, siempre y cuando aún esté dentro de los márgenes de ruido que el circuito pueda tolerar. Gracias a esta

[7] Transistor: Dispositivo electrónico de material semiconductor (germanio, silicio) capaz de controlar una corriente eléctrica, amplificándola o substituyéndola. Según el glosario de robótica. http://robots-argentina.com.ar/glosario.htm

[8] Definición de CMOS. Alegsa; Diccionario de informática. http://www.alegsa.com.ar/Dic/cmos.php

ventaja, los circuitos CMOS son robustos frente a ruido o degradación de señales.[9]

Esta tecnología dentro de los dispositivos lógicos programables conectará efectivamente a través del arreglo de compuertas AND, justo como lo hacían los fusibles, pero ahora las conexiones se harán sin que alguno de los semiconductores deje de funcionar, incluso dentro del arreglo programable uno de ellos puede programarse para que arroje un cero y posteriormente reprogramarlo para que arroje un uno lógico y el paso de corriente no debe tener inconveniente alguno. A este método se le nombró EECMOS (Electronically Erasable CMOS, tecnología CMOS Borrable Electrónicamente), haciendo referencia a que electrónicamente este elemento se puede reconfigurar de diferente manera. Aunque el proceso es exactamente el mismo que los dispositivos anteriores, un arreglo que nos dará una función específica.

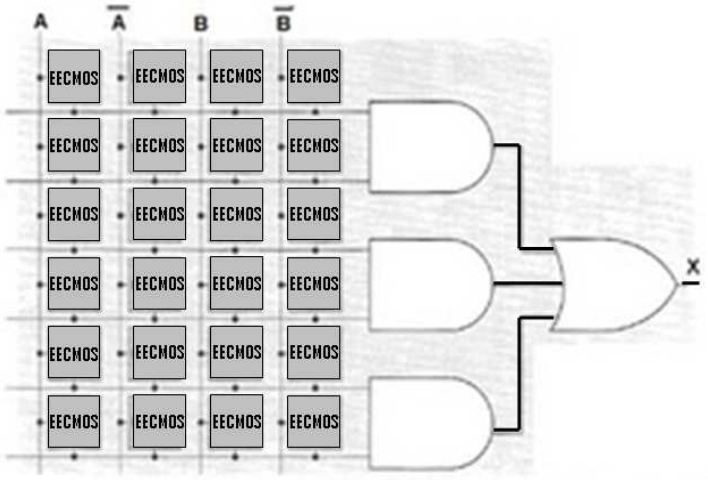


Figura 1.5 Estructura interna de una GAL no programada

En la figura 1.5 se muestra parte de la estructura interna de una GAL de manera gráfica, cabe mencionar que en esta instancia el dispositivo aun no esta programado. Cada fila está conectada a la entrada de una puerta AND, y cada columna a una variable de entrada o a su complemento (ya se mencionó que es la misma variable pero negada). Mediante la programación se activa o desactiva

[9] Ventajas de la tecnología CMOS. Wikipedia. La enciclopedia libre http://es.wikipedia.org/wiki/CMOS

cada celda EECMOS, y se puede aplicar cualquier combinación de variables de entrada, a una puerta AND para generar cualquier operación que se desee.

Las celdas activadas conectan las variables deseadas con las apropiadas entradas de las puertas AND. Estas celdas están desactivadas cuando una variable o su complemento no se utilizan en un determinado producto. La salida final de la puerta OR es una suma de productos.[10] En la figura 1.6 se muestra un arreglo de compuertas AND dentro de un dispositivo GAL ya programado. Una celda activa conecta de forma efectiva su correspondiente fila y columna, y a su vez una celda desactivada desconecta la fila y columna específica, una celda EECMOS típica puede mantener el estado en que se ha programado durante 20 años o más.[11] El objetivo de la activación y desactivación de celdas es permitir el resultado de cualquier producto de términos requerido.

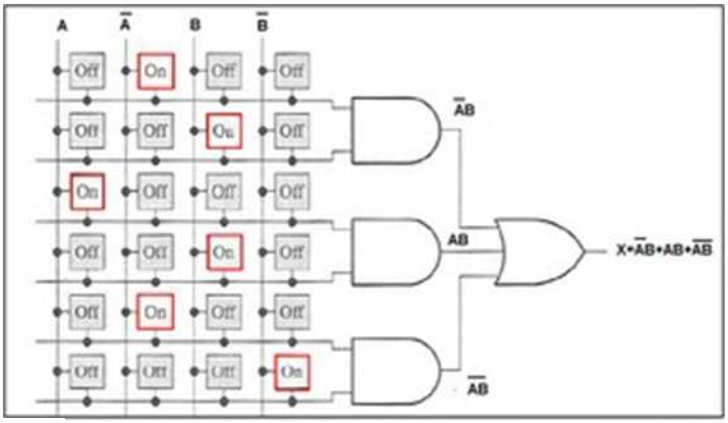


Figura 1.6 Arquitectura interna de una GAL programada

Uno de los dispositivos de esta familia (GAL) es el GAL22V10, la cual es muy importante ya que parte de su arquitectura se retomará en el siguiente tema. El dispositivo cuenta con 22 terminales totales de entrada/salida, 12 de ellas son entradas y 10 pueden configurarse como salidas o entradas dependiendo las

[10] López Carreto Juan Manuel, Carlos Genaro Olivas y Gonzalo Isaac Duchén Sánchez Teoría y práctica de diseño digital con lógica programable. México, 2009.pp. 50

[11] Lógica programada. Wikipedia. La enciclopedia libre. http://es.wikipedia.org/wiki/Lógica_programada

especificaciones que desee el programador. A continuación se muestra y explica su estructura:

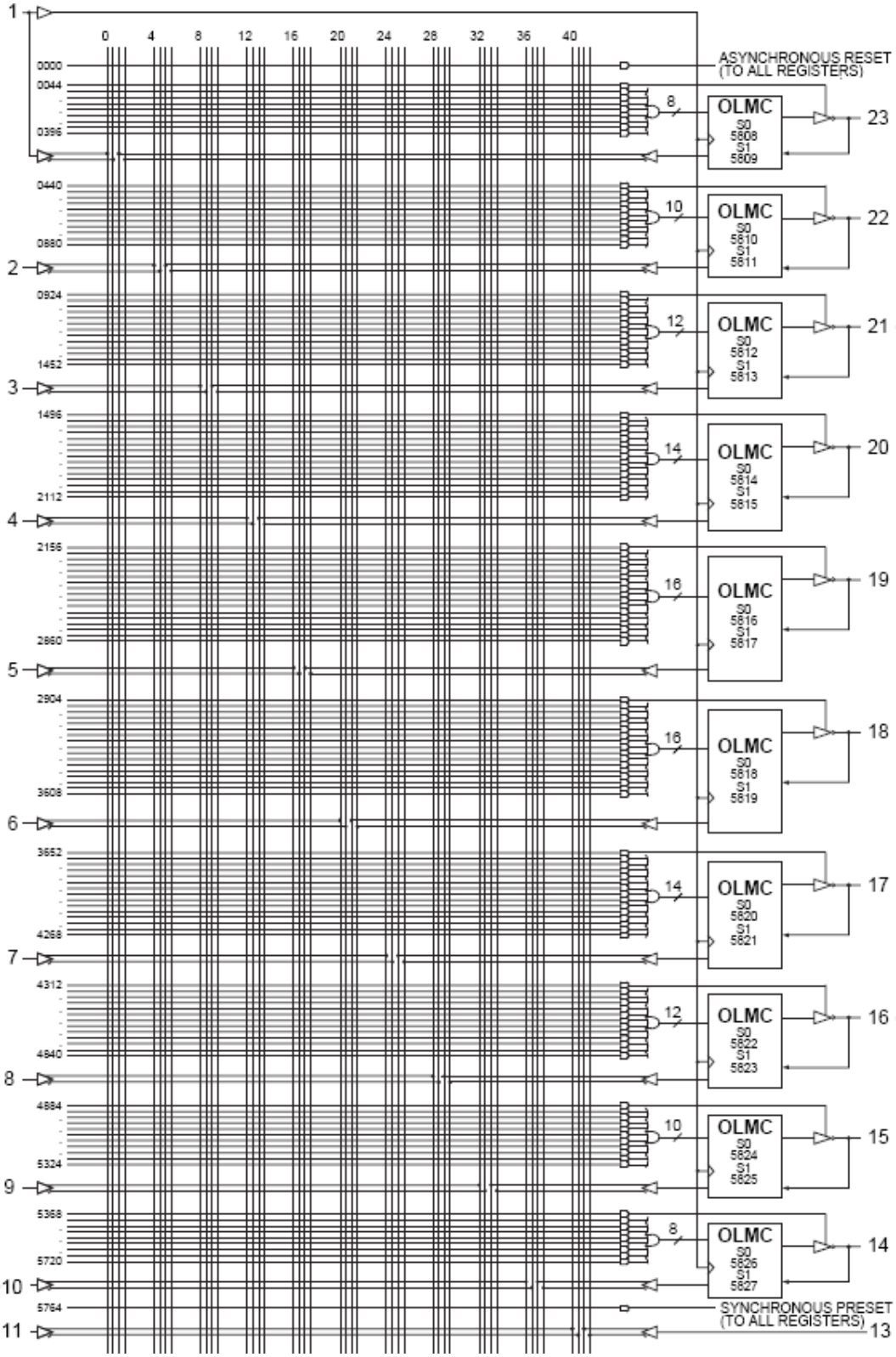


Figura 1.7 Arquitectura interna del GAL 22V10

En la figura[12] 1.7, se puede apreciar la estructura interna del GAL22V10, este chip cuenta con 22 líneas de entrada y sus complementos, lo que da un total de 44 líneas de entrada, las entradas son las líneas verticales y el encuentro de dichas entradas con las líneas horizontales, las cuales son las terminales de las compuertas, representan cada una de las intersecciones que gráficamente se han mostrado en la figuras anteriores.

Si se observa la imagen con detalle, se muestran múltiples compuertas AND al final de la línea esto es el arreglo de compuertas lógicas programables, estas llevan a una compuerta OR para hacer la suma de producto consiguiente, pero la salida de esta compuerta se introduce a un elemento llamado OLMC (*Output Logic Macrocell*, Macrocelda Lógica de Salida), del cual no se ha dado información, este elemento es un bloque que tiene una función importante para la evolución de la lógica programable ya que los siguientes dispositivos lo conservaron y redujeron para hacer chips con mayor escala de integración.

Para hacer clara la explicación, en la figura[13] 1.8 se muestra una OLMC con un acercamiento a detalle, en la imagen se expone como están compuestas las macroceldas por dentro y los elementos que contienen.

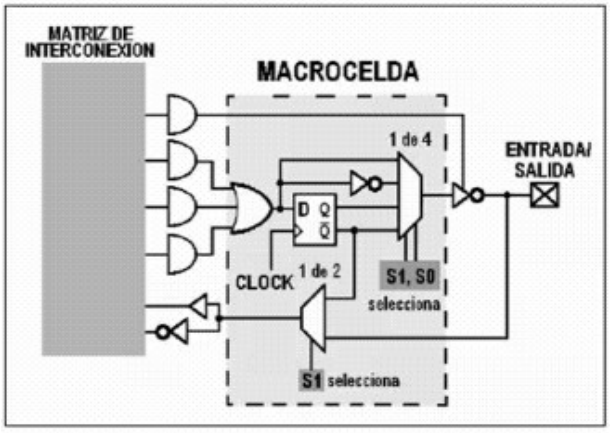


Figura 1.8 Ejemplo de una macrocelda para una GAL22V10

[12] Fuente de la imagen: Manual de usuario, fabricante Latice, GAL22V10, Pág. 5. http://www.alldatasheet.com

[13] Fuente de la imagen. Dispositivos Lógicos Programables (parte 1): http://iindustrial.obolog.com/dispositivos-logicos-programables-parte-1-209085

Una macrocelda está formada diversos elementos electrónicos, las entradas son provenientes de una matriz o arreglo programable con varias compuertas AND, estas desembocan a una compuerta OR, dicha compuerta suma los productos, el resultado de esta se puede llevar a dos lugares, por una parte el valor resultante puede ir de manera directa a un multiplexor que lo enrutará a una terminal de salida y por otra parte a un flip-flop tipo D, recordemos que un flip-flop es un circuito encargado de almacenar la información de la memoria durante un espacio de tiempo, y el tipo D se caracteriza por contener un bit (1 ó 0) y devolverlo en el momento que exista un pulso de reloj a dicho flip-flop, se le llama tipo D por la palabra *"Delay"* que quiere decir retraso,[14] el cual se da al momento de esperar el pulso para recuperar el dato, a continuación la salida de este último dispositivo se conecta también al multiplexor que contiene cuatro líneas de entrada, recordemos también que el multiplexor es un dispositivo que tiene varias líneas de entrada, una de las cuales será seleccionada para ser conectada a una única línea de salida. Las macroceldas se pueden programar de dos maneras, dependiendo el modo en que sea elegida por el programador para generar una salida.

- Modo combinacional: Es la resultante de las operaciones a través de la salida de la compuerta OR, el multiplexor seleccionará el resultado inmediato de dicha puerta lógica.
- Modo secuencial: La salida del multiplexor seleccionará la derivación del flip-flop, es decir, será el mismo valor pero éste lo contendrá un momento para después arrojarlo en el instante que reciba un pulso de reloj.

Las terminales del circuito también se pueden configurar para ser entradas, o bien pueden retroalimentar a alguna variable internamente, por esto es que existen líneas que apuntan hacia la matriz programable, es decir, hacia adentro de

14 Generalmente conocido como retardo de propagación, es el intervalo requerido para que se produzca un cambio en la salida una vez que se ha aplicado una señal a la entrada. Thomas L. Floyd. Fundamentos de Sistemas Digitales. pp. 502

la arquitectura, estas líneas tienen la tarea de ingresar datos cuando así lo requiera el programador.

Las macroceldas se configuran internamente de forma automática, mediante la programación de un conjunto de celdas que están separadas de las celdas de la matriz lógica que son las que el programador si podrá configurar.

Hasta aquí los dispositivos que se han mencionado PLA, PAL y GAL están formados por arreglos o matrices que pueden ser fijos o programables, todos estos chips son conocidos en conjunto como SPLDs (Simples Programmable Logic Device, Dispositivos Lógicos Programables Simples), pero esto es solo una recapitulación de antecedentes. El siguiente paso en la evolución de la lógica programable fue hacer posible el encapsulamiento de varios SPLDs en un mismo dispositivo lógico programable. Así es como surgen los primeros CPLDs (Complex Programmable Logic Device, Dispositivos Lógicos Programables Complejos) en seguida se describen su definición y características.

1.2 ¿Qué es un CPLD?

Actualmente hay PLDs con alto nivel de integración, dispositivos más sofisticados como los son los CPLDs (Complex Programmable Logic Device) éstos entraron al mercado marcando claras ventajas, se caracterizan por la reducción de espacio, además de incrementar la velocidad y las frecuencias de operación, a los programadores les permiten realizar cambios de diseño a los programas sin afectar la lógica, agregando o quitando hardware sin gastar mucho tiempo. En otras palabras un CPLD se fundamenta como un arreglo de múltiples SPLDs en forma de bloques encapsulados en un mismo chip.

Los CPLD se encuentran estructurados mediante bloques lógicos configurables, dichos bloques son muy similares a los dispositivos GAL22V10 y se comunican entre si utilizando una matriz que los conecta denominada PIM (Programmable Interconect Matrix, Matriz de Interconexión Programable), esta matriz que también es parte de un gran bloque, está dedicado a canalizar las

señales correctas de los bloques lógicos y los bloques de entrada o salida del dispositivo sobre las redes apropiadas.

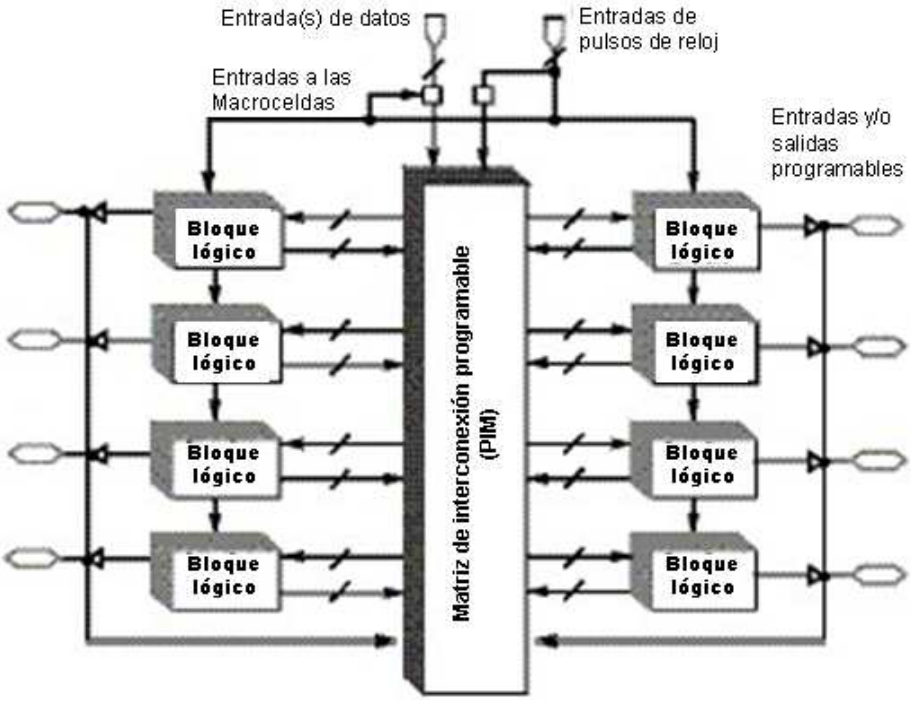


Figura 1.9 Diagrama básico de la arquitectura de un CPLD

En la figura 1.9 se pueden apreciar los elementos más importantes en las que se divide un CPLD, hacia el exterior se aprecian las terminales que pueden ser configuradas como entrada y/o salida, dependiendo como lo decida el programador, seguido por los bloques lógicos, los cuales serán los encargados de llevar la parte lógica del programa, es decir, trabajar de manera eficaz para que el resultado de las operaciones que sean programadas sean exactamente lo que se debe recibir de manera correcta y finalmente al centro de la imagen se encuentra la matriz de interconexión programable, la cual tiene la tarea de conectar efectivamente los bloques con los puertos y viceversa. En seguida se explicarán las funciones y características de cada bloque.

1.2.1 Matriz de interconexión programable

La matriz de interconexión programable permite unir cualquier bloque funcional o bloque de entrada/salida con cualquier otro bloque, es decir, por medio de ella, las terminales de entrada o salida se pueden conectar a las entradas

del bloque lógico, o bien, las salidas del bloque lógico a las entradas de otro bloque lógico. Existen dos configuraciones usadas con mayor frecuencia en los CPLDs, estas son:

- Interconexión de la Matriz programable mediante bloques.
- Interconexión de la Matriz programable mediante multiplexores

La interconexión mediante bloques: Se basa en un arreglo de filas y columnas con una celda programable de conexión en cada intersección. Al igual que en las GAL esta celda puede ser activada para conectar o desconectar la correspondiente fila y columna. Esta configuración permite una total interconexión entre las entradas y salidas del dispositivo con los bloques lógicos.[15]

La interconexión mediante multiplexores: Es colocando un multiplexor por cada entrada al bloque lógico. Las rutas de interconexión programables son conectadas a las entradas de un número de multiplexores por cada bloque lógico. Las líneas de selección de estos multiplexores son programadas para permitir que sea seleccionada únicamente una vía de la matriz de interconexión por cada multiplexor la cual se encamina hacia el bloque lógico, para que cualquier combinación de señales de la matriz de interconexión pueda ser enlazada hacia cualquier bloque lógico.[16]

1.2.2 Bloques Lógicos Programables

Un bloque lógico es un grupo de celdas que van a generar las funciones que el programador quiera reflejar, es muy similar a un PLD sencillo encapsulado en una unidad. Cada uno de éstos posee tres pequeños componentes que en su conjunto hacen funcionar a dicho bloque lógico, estos son; arreglo de productos

[15] Definición CPLD. Wikipedia. La enciclopedia libre. http://es.wikipedia.org/wiki/CPLD

[16] Ibidem.

de términos, esquema de distribución de términos y macroceldas, a continuación se explican más a detalle:

- Arreglo de producto de términos: Esta parte del bloque identifica aproximadamente que cantidad de términos corresponden a cada macrocelda y también el número máximo de productos que puede manejar dentro de un bloque lógico, aparte de realizar los productos efectuados en una matriz de compuertas AND, es decir, es muy similar al arreglo de compuertas AND en un dispositivo GAL.
- Esquema de distribución de términos: Esta parte del mecanismo es utilizado para distribuir los productos a las macroceldas mediante un arreglo programable de compuertas OR, lo que significa que crea las sumas de los productos provenientes del arreglo AND para finalmente enrutarlos a la última etapa.
- Macroceldas: Son las encargadas de dirigir las señales, estas deben dar salida al resultado de los arreglos anteriores o pueden introducir un dato si es que el diseño del programa lo llegara a requerir, son muy similares a las incorporadas en los dispositivos GAL22V10.

En la figura 1.10 se puede apreciar el flujo de los datos por todos los componentes que contiene un bloque lógico, como se ha descrito anteriormente es muy parecido a un dispositivo SPLD comprimido en un solo bloque.

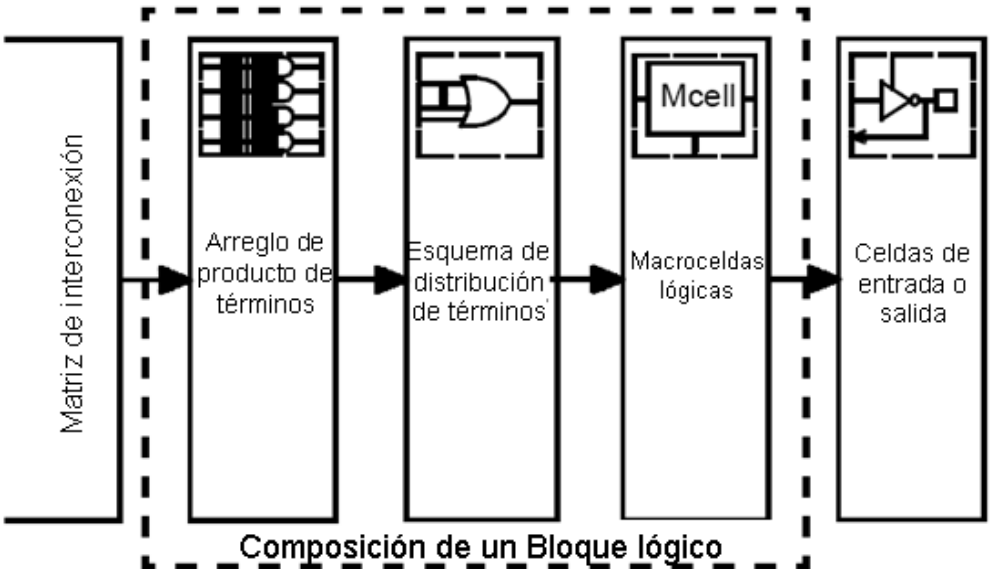


Figura 1.10 Diagrama interno de un bloque lógico programable

Dependiendo de la marca y la familia un CPLD puede tener varias macroceldas por bloque lógico, la cantidad que contenga de ellas es muy importante, ya que mientras mayor sea la suma de estas, las funciones que se puedan implementar podrán ser de mayor grado de dificultad.

El tamaño del bloque lógico también es una medida de la capacidad del CPLD, ya que de esto depende el número de terminales que se pueden manipular en un solo bloque y la complejidad de las operaciones que puedan ser realizadas dentro del mismo.

1.2.3 Bloques de Entrada/Salida

Finalmente los CPLDs contienen bloques de entrada/salida, este bloque conecta directamente toda la arquitectura anteriormente citada con el entorno exterior. La función de una celda de entrada/salida es permitir el paso de una señal hacia adentro o hacia afuera del dispositivo según se haya configurado en la programación, en ocasiones las celdas de entrada/salida se consideran parte del bloque lógico aunque la mayoría de los fabricantes consideran que son externas.[17]

1.3 El CPLD a programar (7C373i-66JC)

El tema anterior mostró claras ventajas de los CPLDs sobre los SPLDs pero también mencionó de forma estructurada como es la arquitectura de estos dispositivos, esto es relevante ya que a continuación se retomarán muchas de las características antes citadas pero ahora comparándolas en un CPLD en específico, que es sobre el cual se trabajará a lo largo de este texto.

En este tema se hará énfasis en el CPLD CY7C373i-66JC, y a través de lo que resta de este capítulo se expondrán sus generalidades y características, así como algunos diagramas que muestren su arquitectura interna y forma física.

[17] Maxines David, Op.Cit. pp. 15

1.3.1 ¿Por qué este dispositivo en específico?

La idea inicial surge al estudiar las ventajas del CPLD CY7C373i-66JC buscando crear una tarjeta de circuitos integrados la cual programe y pruebe el dispositivo y así poder explotar estas ventajas. Dentro de la tarjeta se podrá grabar, borrar y reprogramar el CPLD, pero también se tiene como objetivo probar el diseño del programador en un entorno físico ya que la tarjeta contendrá elementos que apoyen a hacer visibles las respuestas de ciertas terminales, para tener una comprobación del funcionamiento del CPLD, también se contempla que esta misma tarjeta pudiera poseer ciertas terminales externas para poder conectar en ella algunos dispositivos periféricos como son: motores a pasos, motores de corriente directa, visualizadores, pantallas de cristal líquido, o algunas etapas de sensado como: sensores de movimiento, de fotones, de luz infrarroja, de temperatura, por dar algunos ejemplos, que el usuario requiera utilizar.

Con esta tarjeta habría amplias posibilidades de crear diseños personalizados, el límite para poder hacer un diseño complejo o profesional solo lo pondría la creatividad del programador.

El número de terminales que contiene el dispositivo y el tamaño de este también fueron esenciales para concretar el diseño de la tarjeta ya que para hacer algunas prácticas complejas se requiere ocupar diversas terminales y el CPLD a programar contiene 84 en total. Aún excluyendo la terminales que se utilizarán para polarizarlo,[18] dejará libres la mayoría de estas, las cuales se podrán manipular con un lenguaje de programación, y por las dimensiones del chip la tarjeta tendrá un tamaño aproximado de 10 x 15 cm. que la harán útil y práctica. Estas ventajas fueron comparadas con algunas características del dispositivo de la misma familia CY7C372i el cual es más compacto y solo tiene 32 posibles

[18] Polarizar: Suministrar una tensión fija a una parte de un aparato electrónico, en este caso conectar el dispositivo al voltaje y a tierra en las terminales que se requiera. Según WorldReference el diccionario de la lengua española.
http://www.wordreference.com/definicion/polarizar

terminales de entradas/salidas, estas diferencias hacen del CPLD CY7C373i una mejor opción para trabajar con él.

Esto traería muchas ventajas al ámbito universitario ya que con una misma tarjeta se podría enseñar, ejemplificar y desarrollar proyectos completos utilizando la lógica digital programable.

1.3.2 Aspecto físico

Físicamente el dispositivo esta implementado en una tecnología de encapsulado llamada PLCC (*Plastic Leaded Chip Carrier*) la cual permite crear circuitos integrados implementados en muy poco espacio, ya que por norma la separación entre las terminales del dispositivo es de 1.27 milímetros ó 0.05 pulgadas, en este caso el encapsulado tiene la forma cuadrada con el mismo número de terminales en cada lado, los cuales tienen una dimensión aproximadamente de 30 milímetros y el ancho oscila entre 0.05 centímetros

En la figura 1.11 se puede apreciar el dispositivo desde arriba y de uno de sus lados (todos los lados de éste son iguales y tienen las mismas dimensiones), aunque los valores de la imagen están en pulgadas con las cantidades en centímetros que ya se mencionaron anteriormente se tiene una referencia.

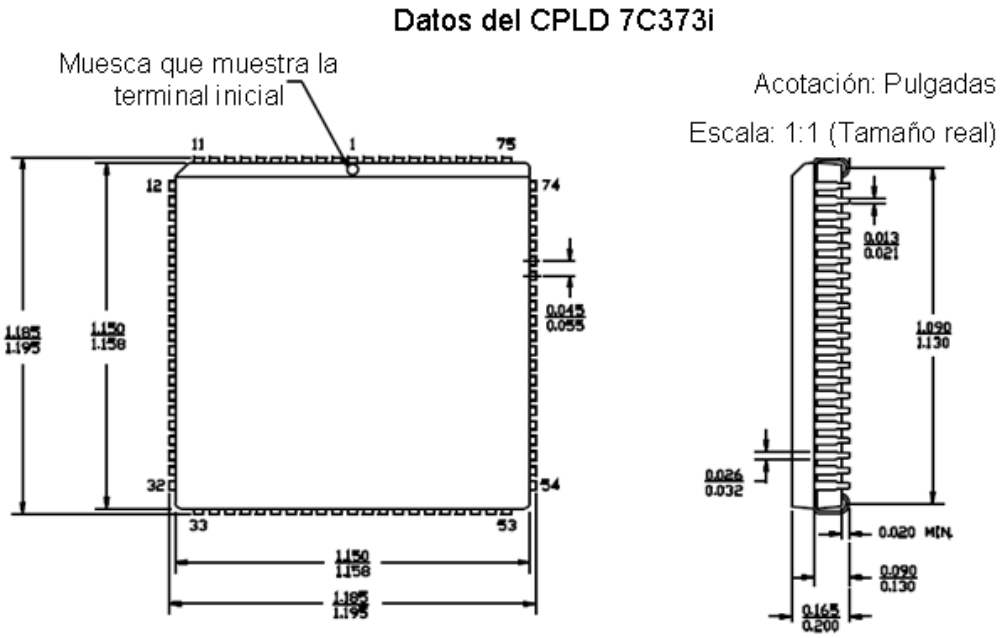


Figura 1.11 Descripción de la apariencia física del CY7C373i-66JC

En este tipo de encapsulados se tiene la ventaja de poderlo utilizar en un circuito de la manera clásica SMT (*Surface Mount Technology*), es decir en un montaje superficial o también tener un enchufe en un zócalo soldado a la superficie del circuito que conectaría el dispositivo a la placa base[19], a esta especificación se le conoce como PCI (*Peripheral Component Interconnect*). Esta causa trae consigo diversos beneficios, como son: hacer de manera sencilla el remplazo de la pieza, quitar de manera fácil el dispositivo para reprogramarlo o por cualquier otro motivo, en caso de que se necesario

En las configuraciones PLCC los chips encapsulados pueden ser cuadrados o rectangulares, por lo general tienen forma cuadrada con el mismo número de pines en cada lado, aunque existen variaciones de forma rectangular con más pines en los lados más largos, pero se caracterizan por tener siempre el mismo espacio de separación entre una terminal y otra. El nombre normalmente indica el número de pines a continuación de las siglas. En el caso de este CPLD se tendría una configuración PLCC84, es decir 21 terminales en cada uno de sus 4 lados.

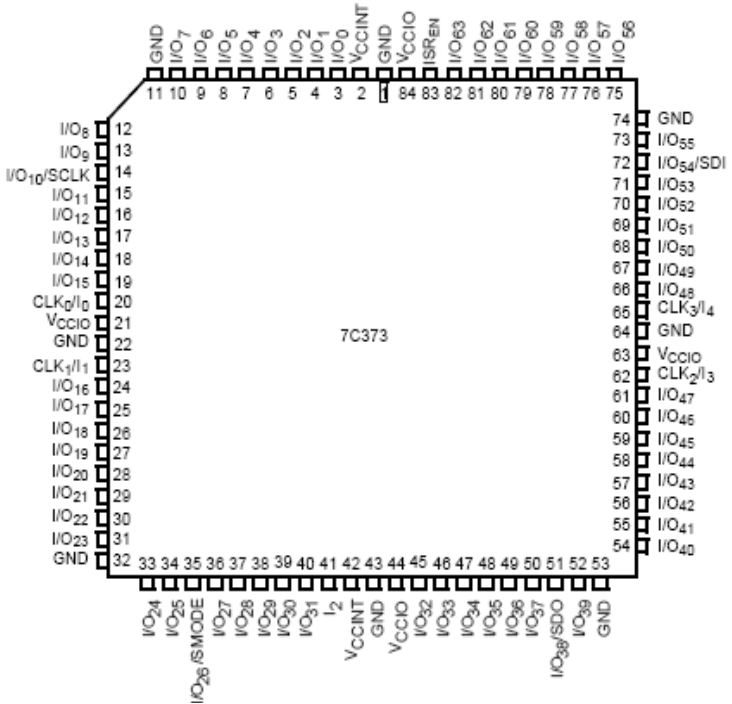


Figura 1.12 Configuración de terminales según se describe en el manual de usuario

[19] Definición PLCC. Wikipedia, la enciclopedia libre. http://es.wikipedia.org/wiki/PLCC

En la figura 1.12 se puede observar el nombre de cada una de sus terminales, como ya se mencionó algunas de estas serán conectadas a la corriente haciendo trabajar al chip, otras tienen funciones únicas específicas y algunas otras pueden tener hasta dos funciones distintas, en este caso la configuración que se utilice de una de las dos maneras será elegida dependiendo la operación que convenga al programador.

1.3.3 Funciones y configuración de las terminales

Como se pude apreciar en la figura 1.12 la terminal inicial se encuentra ubicada al centro de la parte superior del dispositivo, cada una de estas terminales tiene una función específica. Inicialmente y como cualquier dispositivo debe tener sus correspondientes terminales para conectarse hacia tierra y hacia voltaje. En cada uno de los lados del chip cuenta con 2 terminales a tierra, lo que representa 8 en total y 6 terminales que deben conectarse a voltaje o Vcc. En la figura 1.12 se muestra que hay dos grupos de terminales que deben ser alimentadas por Vcc; el primero de ellos se identifica con el nombre de VccInt, la alimentación de estas terminales se requiere para las operaciones internas del CPLD y el otro grupo llamado $V_{CCI/O}$ se ocupa de accionar las terminales de salida para habilitar los dispositivos externos que se requieran. Las terminales de VccInt se deben de conectar siempre con una alimentación constante de 5V, mientras que las de $V_{CCI/O}$ pueden conectarse a una fuente de alimentación de 3.3V ó de 5V dependiendo lo que se requiera a la salida, pero si es indispensable para evitar un mal funcionamiento que todo el grupo de terminales $V_{CCI/O}$ sea conectado a un mismo voltaje[20] de entrada.

El 7C373i es muy basto en recursos, ya que aparte de ser rápido, contiene diversas terminales de entradas y/o salidas de pulsos posibles, este dispositivo contiene 64 de estas terminales lo que proporciona gran versatilidad y a cada macrocelda le corresponde un grupo de 16 de ellas, también contiene 1 terminal que es específicamente señal de entrada, además contiene 4 entradas que tienen

[20] Datos tomados del manual de usuario CPLD CY7C373i http://www.cypress.com

la opción de ocuparse para recibir pulsos de hasta cuatro relojes diferentes cuando así lo decida el usuario para así agilizar o automatizar de manera secuencial algunas prácticas.

Además el CPLD a utilizar también cuenta con pines especializados en recibir la lógica del programa mientras se encuentra conectado en un sistema, gracias a una tecnología especial llamada ISR (*In-System Reprogrammable*) la cual nos permite hacerlo.

La tecnología ISR es la capacidad que tienen algunos microcontroladores, dispositivos lógicos programables como este, y otros chips electrónicos para ser programado mientras este está instalado en un sistema, en lugar de requerir que el chip sea programado antes de conectarlo en un sistema. La principal ventaja de esta característica es que permite a los desarrolladores de los dispositivos electrónicos integrar la programación y la prueba en una sola fase de producción, en lugar de necesitar una etapa de programación por separado antes de montar el sistema. Esto puede permitir que los usuarios puedan programar los chips y crear con esto un sistema con las características específicas que se requieran en lugar de comprar dispositivos preprogramados desde un fabricante o distribuidor,[21] además de que es posible aplicar cambios de código o el diseño en cualquier momento que se necesite y volver a cargar un diseño con una lógica de programación diferente de manera rápida.

Generalmente los chips que utilizan tecnología ISR tienen un circuito interno para comparar una tensión de programación necesaria diferente a la tensión del suministro normal, y comunicarse con el programador a través de un protocolo en serie. Para habilitar la interfaz ISR debe estar activada la terminal de voltaje de programación llamado ISREN, esta terminal tiene que ser alimentada con un energía de aproximadamente 12 Volts ya que si no cuenta con ella las terminales de programación se encontraran deshabilitadas para recibir la lógica del

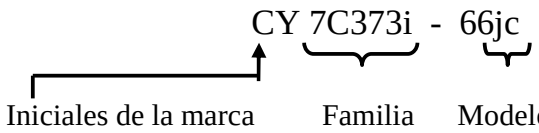
[21] Definición ISP. Wikipedia, la enciclopedia libre. http://en.wikipedia.org/wiki/In-system_programming

programa y dichas terminales solo podrán funcionar como entrada o salida de pulsos.[22] La mayoría de los dispositivos lógicos programables para poder introducir la lógica de programa utilizan un protocolo llamado JTAG (*Joint Test Action Group*) para ISR.

La interfaz JTAG es un protocolo especial de comunicaciones entre el dispositivo y el programador, de cuatro o cinco terminales agregadas a un chip. En el CPLD a utilizar, maneja cuatro terminales que tienen la función de captar estas señales para programar y/o reprogramar el dispositivo son aquellas que tienen los nombres: SCLK, SMODE, SDO y SDI respectivamente,[23] estas se explicarán de mejor manera más adelante.

1.3.4 Descripción funcional / Arquitectura

El CPLD CYC7C373i proviene de una familia, que cuenta con características propias de este conjunto de dispositivos.



Como todos los miembros de esta familia gozan del beneficio de almacenamiento en la tecnología de memoria tipo *flash*, la cual permite la lectura/escritura de múltiples posiciones de memoria en una misma operación.

En la figura 1.13 se muestra que la estructura del CPLD 7C373i contiene todos los componentes de un CPLD mencionados anteriormente. La matriz de interconexión programable (PIM), los bloques lógicos y las terminales de entrada/salida, también un pin específico de entrada y cuatro especializados para recibir pulsos de reloj, si es que se requiriera.

[22] Manual de usuario: Designing with cypress In-System Reprogrammable (ISR) CPLDs for PC cable programming. Dentro de Enables the 4-pins interface pp. 1 http://www.alldatasheet.com

[23] Definicion JTAG. Wikipedia, la enciclopedia libre. http://es.wikipedia.org/wiki/JTAG

La matriz de interconexión programable dentro de la arquitectura se encarga de conectar los 4 bloques lógicos de una manera extremadamente rápida, flexible y confiable.

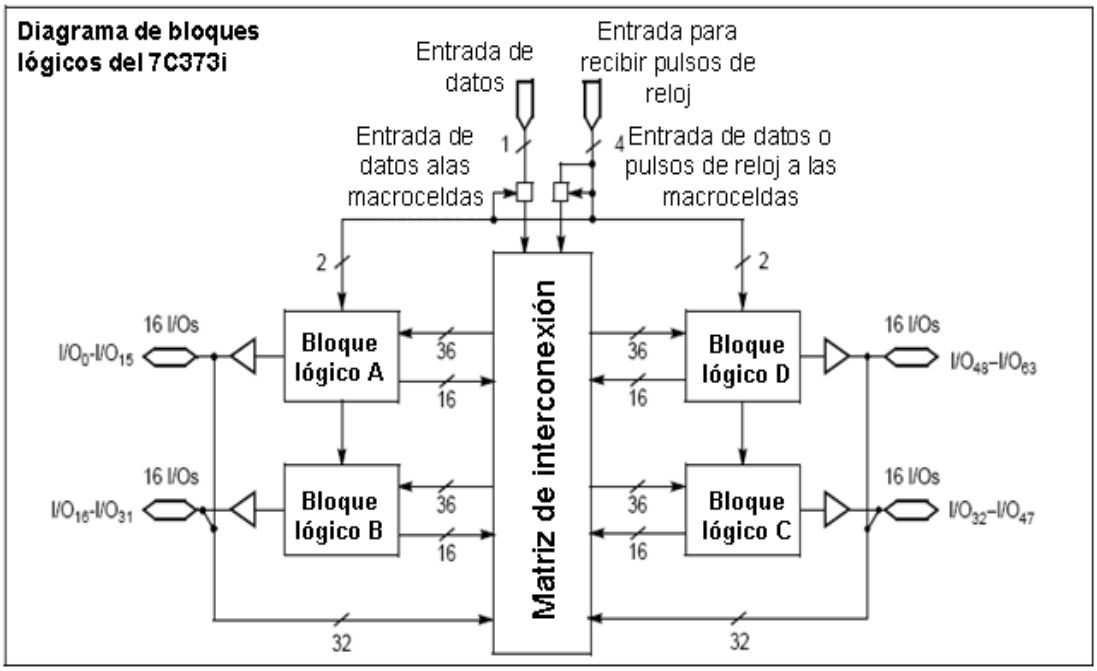


Figura 1.13 Diagrama de bloques lógicos del CPLD 7C373i-66JC

Todos los dispositivos traen consigo un retardo, lo ideal sería que ninguno lo tuviera, pero esto es imposible ya que debe pasar una cantidad de tiempo (aunque sea mínima) para que los datos que entran se procesen y fluyan a una terminal de salida. El CY7C373i cuenta con excelentes estrategias de contrarrestar los efectos de retardo, inicialmente la matriz de interconexión trae consigo un retardo corto y uniforme al interconectar bloques. Además contiene un modelo de tiempos en donde no hay retardos ocultos como lo son; los efectos de fan-out[24], retardos de interconexión, o retardos de expansión, sin importar el número de los recursos usados o el tipo de uso que se le dé al dispositivo.

[24] Fan-Out ó "factor de carga máxima de salida": Es el número máximo de entradas lógicas que una salida puede manejar confiablemente. Si se excede, no se puede garantizar su buen funcionamiento.
Ronald J. Tocci, Neal S. Widmer. Sistemas digitales: principios y aplicaciones
ED. Prentice Hall. México 2003. pp 415

1.3.4.1 Bloques lógicos

En los dispositivos CPLD el común denominador para el rápido funcionamiento se ve reflejado en el número de macroceldas que pueda contener un bloque lógico y en la efectividad que tengan cada una de estas, en este CPLD en específico se cuenta con 4 bloques lógicos, cada uno de estos bloques incluye 16 macroceldas, lo que da un total de 64 en todo el dispositivo, además están constituidos por un arreglo de productos de términos y un esquema inteligente de distribución de términos.

Cada bloque lógico contiene tres componentes en su interior, como ya se ha descrito en temas anteriores estos son; un arreglo de productos de términos, un esquema de distribución de términos y las macroceldas lógicas de salida. En la figura 1.14 se muestra el diagrama interno de uno de los bloques lógicos del CPLD 7C373i a continuación se explicarán las especificaciones de cada uno de los elementos dentro del este bloque lógico.

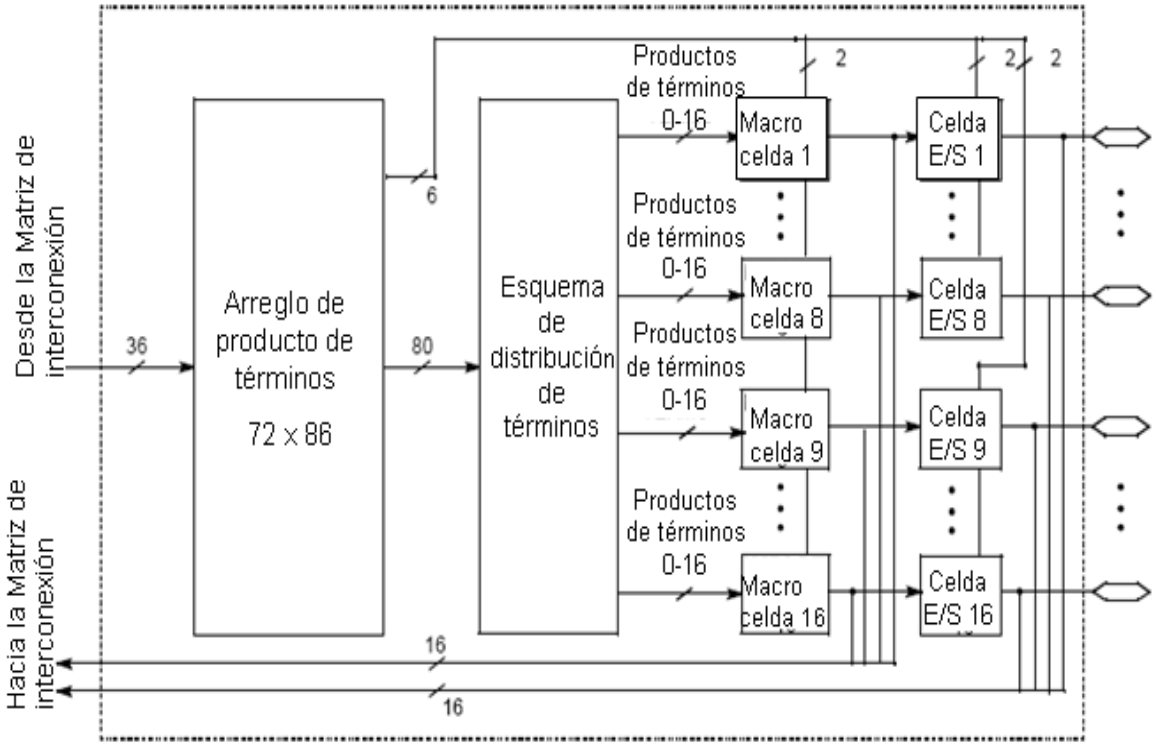


Figura 1.14 Diagrama interno de un Bloque Lógico del CPLD 7C373i-66JC

a) Arreglo de productos de términos

Esta parte identifica aproximadamente la cantidad de términos que puede manejar un bloque lógico, aparte de realizar los productos efectuados en una matriz programable de compuertas AND

El arreglo de producto de términos en los bloques lógicos de este dispositivo incluye 36 entradas desde la matriz de interconexión, y este dispositivo puede proveer hasta 86 salidas, que desembocarían al esquema de distribución de términos. Las 36 entradas desde la matriz de interconexión programable llegan al arreglo de productos en ambas polaridades (positiva y negativa) teniendo un total de 72 señales, haciendo que el arreglo total sea de un tamaño de 72 x 86, esta es una capacidad considerable en una matriz, su función radica en que operaciones complejas sean implementadas rápidamente a través del dispositivo.

b) Esquema de distribución de términos

El esquema de distribución de términos es un recurso configurable que introduce las respuestas del arreglo de productos de términos a las macroceldas que se requieran. Los términos de productos pueden ser asignados de 1 a 16 macroceldas, inclusive pueden ser asignadas a macroceldas de otros bloques lógicos (a esto se le denomina *product term steering* o manejo del producto de término), además estos términos se pueden compartir entre macroceldas si la lógica así lo requiere, esto significa que las variables que son comunes para más de una salida se puedan ejecutar con un solo término de producto.

c) Macroceldas lógicas de salida

Cada una de las macroceldas de este CPLD recibe a su entrada, una suma de entre 0 a 16 términos del esquema de distribución de términos. La macrocelda incluye un control de la polaridad sobre el término sumado de la entrada y dos relojes para accionar el registro, también ofrece una trayectoria de vuelta a la

matriz de interconexión para poder introducir datos si la terminal fuera configurado como entrada

1.3.5 Ventajas y desventajas del dispositivo

Los PLDs complejos tienen múltiples ventajas, y este dispositivo en específico tiene características propias interesantes, a lo largo de este capítulo se han expuesto algunas de las ventajas que tiene el dispositivo, pero de manera concreta se enumera el siguiente resumen:

- Dispositivo reprogramable gracias a su tecnología eléctricamente borrable.
- Gran capacidad debido a sus 64 pines de entrada/salida.
- 5 terminales específicas para recibir señales de entrada incluyendo 4 que pueden recibir señales de reloj.
- Contiene 4 bloques lógicos con 16 macroceldas cada uno, dando un total de 64 macroceldas repartidas.
- Programable con sistema ISR, conteniendo tecnología flash e interfase JTAG.
- Maneja de buena manera los retardos y no los esconde.
- Es un dispositivo que trabaja a altas velocidades.
- Guarda o mantiene las funciones de las terminales confiablemente después de ser programado.
- Para operar se puede trabajar en dos diferentes tipos de voltaje en las terminales de salida.
- Totalmente compatible con tecnología PCI.
- Disponible en un práctico encapsulado PLCC de 84 terminales.

Estas características ya se fueron explicando con la intención de que el lector pueda darse una idea al tener un concentrado de todas estas propiedades, pero como en cualquier dispositivo además todos sus beneficios, también se tienen

que citar sus defectos para poner en claro todas las especificaciones. Algunas de las desventajas que puede aportar este dispositivo son:

- A pesar de que el tamaño de los CPLDs han ido disminuyendo, existen variados dispositivos que son encapsulados en un espacio menor al 7C373i, en la electrónica el tamaño de un dispositivo puede ser vital para un circuito, ya que el usuario preferirá los circuitos de menor tamaño por múltiples cuestiones, entre ellas está la comodidad, el diseño, la practicidad de no lidiar con tantas terminales, el costo, por citar algunas.
- Teniendo el dispositivo tantas terminales de entrada/salida, la tarjeta entrenadora que este chip puede ocupar se ve envuelta en el mismo problema de tamaño, teniendo en cuenta todas las líneas a conectar y que en dicha tarjeta se tienen que agregar algunas otras etapas para que el chip pueda funcionar.
- El CPLD a programar es un chip versátil y completo por tanto habrá que saber utilizarlo y aprovecharlo, de nada servirá tener tantas terminales de entrada/salida si el dispositivo solo se utiliza para que se ocupen las terminales mínimas.
- El proceso de saber que con este dispositivo se puede hacer desde lo más sencillo[25] hasta lo más complejo, puede hacer que el usuario se familiarice desde el inicio con este chip, esto trae consigo la responsabilidad de intentar cambiar los procedimientos de los usuarios para utilizarlo en lugar de múltiples chips GAL, todo cambio trae consigo una etapa de incertidumbre.
- Al ser un elemento de mayor tamaño, completo, con mejor flexibilidad y para operaciones más complejas, el costo obviamente será relativamente mayor que los dispositivos PLDs simples.

25 Este CPLD no es viable para hacer cosas muy simples ya que se desperdiciaría su capacidad, con sencillo me refiero; a los usuarios que inician a adentrarse en el entorno de la lógica programable y tengan que hacer sus primeras practicas, las podrán crear haciendo uso de la tarjeta para después poder implementar diseños más aptos para dicho dispositivo

Conociendo la arquitectura general de este tipo de dispositivos para su programación y la del CPLD específico que se quiere manipular y su evolución cronológica, desde mediados de la década de 1970, en el segundo capítulo se hace un análisis sobre la forma de programación y se dividirá en subtemas con ejemplos para su mejor estudio.

Capítulo II

VHDL como Herramienta de Trabajo y sus Características

En este capítulo se expondrá la parte referente al software que será necesario para grabar la lógica programable en el CPLD ya que lo conocemos, se explica también paso a paso, cómo se trabaja y el entorno de funcionamiento, con el objetivo de que el lector se oriente al momento de crear su propia lógica programable para hacer funcionar cualquier idea que a éste le venga a la mente. También se abordan los temas de forma clara con la finalidad de que el alumno pueda hacer sus primeros programas de una manera rápida, así como el lector interesado o un usuario más experimentado, pueda hacer diseños más elaborados a los aquí expuestos.

2.1 Introducción a VHDL

Tal como lo indican sus siglas **VHDL** es la abreviatura de **HDL** en combinación con **VHSIC**, donde **HDL** es a su vez el acrónimo de (*Hardware Description Language*, Lenguaje de Descripción de Hardware) y **VHSIC** es el acrónimo de (*Very High Speed Integrated Circuit*, Circuito Integrado de Muy Alta Velocidad).[26] Es un lenguaje enfocado a la descripción o modelado de circuitos electrónicos digitales, a veces es incorrectamente tratado como un lenguaje de programación, ya que su objetivo consiste en describir, analizar y evaluar el comportamiento de un sistema electrónico digital.

VHDL es un lenguaje de alto nivel ya que permite la integración de sistemas digitales sencillos, complejos o ambos en un dispositivo lógico programable, sea de baja capacidad o de gran capacidad de integración. También los circuitos descritos en VHDL pueden hacer una representación utilizando una pantalla de

[26] Definición de VHDL. The free Dictionary.
http://encyclopedia2.thefreedictionary.com/VHDL

simulación para reproducir el funcionamiento de un circuito.[27] Además utilizando los recursos adecuadas de programación, es decir, con el respectivo hardware y los protocolos de comunicación, se puede implementar dicho circuito en un dispositivo lógico programable.

El conocimiento del lenguaje estandarizado VHDL se ha convertido en algo imprescindible para todos los estudiantes, diseñadores e ingenieros que están de alguna manera ligados al desarrollo de sistemas electrónicos digitales, pero se debe tener en cuenta que el lenguaje VHDL en este texto, es solamente un medio, no debe convertirse en el fin, por esto no se debe entender que éste lenguaje de descripción de hardware no tenga interés por sí mismo, solo que trabajaremos en conjunto con más elementos, como el simulador y el software de grabación, probando el funcionamiento del CPLD 7C373i físicamente en la tarjeta entrenadora y probadora.

2.1.1 Breve historia de VHDL

Conforme pasaba el tiempo la necesidad de integrar un mayor número de dispositivos en un solo circuito integrado iba acrecentándose, entonces se idearon y diseñaron nuevas herramientas que auxiliarían al ingeniero a integrar sistemas de mayor complejidad. En la década de los cincuentas comenzaron a aparecer los primeros lenguajes de descripción de hardware (HDL) como una opción de diseño para el desarrollo de sistemas más personalizados. Después de eso, durante los años sesenta estos lenguajes comenzaron lentamente a tener mayor aceptación entre la mayoría de los usuarios, pero alcanzaron un mayor auge en los años setenta donde algunas empresas comenzaron a perfeccionarlos y se desarrollaron algunos de ellos como IDL de IBM, TI-HDL de Texas Instruments, ZEUS de General Electric por citar algunos, todos los anteriores enfocados al área industrial, la desventaja de éstos era que no estaban disponibles fuera de la empresa que los manejaba, y aunque algunos lenguajes más básicos salieron de

[27] Sánchez-Elez Marcos, Molina María del Carmen. Introducción para la programación en VHDL. Universidad Computense de Madrid. 2006. pp. 3

sus empresas para reforzar el ámbito universitario carecían de soporte y mantenimiento adecuados que permitieran su estabilidad.[28]

El software VHDL surge siguiendo con el la evolución de lenguajes *Hardware Description Lenguage,* en la década de los ochentas. El Departamento de Defensa de Estados Unidos lo requiere debido a la crisis del ciclo de vida del hardware, es decir, el costo de reproducir material electrónico cuando las tecnologías se volvían obsoletas, alcanzaban el punto de crisis por que el funcionamiento no estaba debidamente documentado y los dispositivos con que se contaba usaban lenguajes y simuladores incompatibles.[29]

El proceso de estandarización fue buscado desde un inicio ya que muchas áreas serían favorecidas, tanto el Departamento de Defensa, como las entidades de desarrollo e industria del país, beneficiando también al ámbito educacional. El primer borrador fue creado en agosto de 1985 diseñado por *Intermetrics*, *IBM* y *Texas Instruments*. En diciembre de 1987 fue aprobado como estándar del IEEE, posteriormente una vez como estándar VHDL pasó a revisión durante cinco años antes de ser entregado a la industria, en 1993, fue revisado y registrado como norma IEEE Std1076-1993.[30]

Después de que VHDL se convirtió en un estándar, lo cual es una ventaja poderosa, se complementó además con otras características que lo hacen particularmente atractivo; es un lenguaje independiente de la tecnología, no emparejado a un simulador o software específico y no requiere una metodología precisa de diseño, además VHDL permite implementar nuevas tecnologías en diseños existentes.

---

28 Maxines David, Jessica Alcalá, VHDL: El arte de programar sistemas digitales. México, DF. 2005. pp. 25

29 VHDL: Una breve historia de VHDL. Blog: http://vhdl-blog.blogspot.com/2008/01/una-breve-historia-de-vhdl_14.html

30 Rico Rafael, Salvador Marcos. Apuntes de VHDL. Servicios de publicaciones UAH. España 2008. pp. 1

2.1.2 VHDL en la actualidad

Hoy en día es uno de los lenguajes más populares, ya que como se mencionó anteriormente, los ingenieros y desarrolladores de sistemas electrónicos digitales encontraron en este software un gran complemento, y con la cual poder afrontar y solucionar algunos obstáculos comunes que permitieran concluir algún sistema.

El lenguaje VHDL sirve para dar respuesta a numerosos problemas planteados en el desarrollo y documentación de hardware y sistemas digitales. La documentación requerida para describir un sistema electrónico puede ocupar gran cantidad de páginas y suele ser muy costoso reemplazar la información contenida cuando la tecnología o las especificaciones cambian o evolucionan. Un lenguaje de descripción adecuado resuelve el problema ya que la "documentación" es ejecutable,[31] e incluso puede ser comprobable gracias a su simulación, antes de completarlo físicamente.

En la actualidad es uno de los lenguajes de descripción más populares a nivel industrial, por sus propiedades tales como; ser capaz de soportar el proceso de diseño de sistemas electrónicos complejos, además de reducir los recursos tecnológicos requeridos y reducir el tiempo al diseñar.

Una ventaja de VHDL es que hace de manera fácil la descripción, el modelado y la síntesis de circuitos digitales y sistemas complejos. Además permite abordar un problema lógico a nivel funcional, es decir, evalúa un problema sólo conociendo las entradas y salidas, lo cual facilita la descripción de soluciones alternativas antes de iniciar su diseño detalladamente.

A razón de sus beneficios y su popularidad, la actividad que se ha generado en torno a este software es muy intensa. En muchos países se han creado grupos de trabajo alrededor de este lenguaje, incluso se realizan periódicamente conferencias, reuniones y hasta congresos anuales.

31 Ibidem

Algunas otras empresas dedicadas a la minielectrónica se han ido adaptando poco a poco al lenguaje, incluso algunas zonas que son potencias en este terreno como Japón tienen gran aceptación, a pesar de que cuentan con un lenguaje estándar propio.[32]

2.2 Nociones básicas y elementos fundamentales en VHDL

Hasta ahora se ha expuesto la evolución del lenguaje y algunas de las ventajas que éste contiene, pero al ser un software se compondrá de algunas reglas, a las cuales se les puede llamar características del lenguaje.

Para introducir un poco, se pueden describir algunas de las reglas generales, no habrá que memorizarlas ya que al utilizarlas constantemente, estarán muy presentes y cada que se realice un nuevo programa, el lector podrá familiarizarse con ellas. Las mayúsculas y minúsculas pueden ser usadas indistintamente en VHDL, existen varias palabras reservadas que el programador no puede usar para nombrar algún tipo de objeto o módulo de diseño. Ya que el programa ignora los espacios y saltos de línea es importante seguir estrictamente la sintaxis, los comentarios se hacen mediante dos guiones medios seguidos, siendo comentario todo lo que se encuentre después de los guiones hasta el final de la línea, estos consejos son bastante prácticos para tener nociones antes de empezar a programar de cualquier manera más adelante se muestra la manera correcta de hacer uso de la sintaxis.

2.2.1 Identificadores

Como se ha redactado en el párrafo anterior, dentro de un código habrá ocasiones en donde se podrá nombrar (con algunas restricciones) algún tipo de proceso, objeto, módulo, etcétera. Al tener casi por completo la flexibilidad de dar un nombre, nos podremos ayudar de éste ya que puede facilitar la descripción del diseño, esta designación recibe el nombre de identificador.

[32] Maxines David. Op.Cit. VHDL en la actualidad. pp.28

Los identificadores en VHDL son nombres o etiquetas, compuestos por una secuencia de una o varias letras, se permiten mayúsculas y minúsculas indistintamente, números o guiones bajos y no tienen una restricción en cuanto a su longitud.

Todos los identificadores deben cumplir con ciertas especificaciones para que el momento de compilar[33] el programa lo hagan sin errores, las restricciones de dichas etiquetas son:

- El primer carácter de un identificador debe ser una letra.
- El último carácter de una etiqueta no puede ser un guión bajo
- Dos guiones bajos seguidos no son permitidos
- Los caracteres especiales no son utilizados
- No se permite nombrar un identificador con palabras reservadas para el lenguaje

INCORRECTO	CORRECTO
4Suma	Suma4
S4bits_	S4_bits
Suma__22	Suma_2_2
Bit#8	Bit_8
Signal	Signal_A

Tabla 2.1 Ejemplos para la escritura de *identificadores*

Algunos ejemplos de las formas en que se puede escribir correcta e incorrectamente las etiquetas se muestran explícitamente en la tabla 2.1, ahí se exhibe la manera ortográfica de cada una de las condiciones que se explicaron anteriormente.

Nota: Las letras mayúsculas y minúsculas son consideradas iguales al hacer uso de un identificador, así que; Suma4, SUMA4 y suma4 para un código en VHDL serán completamente idénticas.

[33] Compilación: Proceso de traducción de un código fuente (escrito en un lenguaje de programación de alto nivel) a código objeto (lenguaje máquina) para que pueda ser ejecutado por la computadora. La aplicación o la herramienta encargada de la traducción se llama compilador. Según: Diccionario de la microcomputación

2.2.2 Tipos de objetos

Un objeto en VHDL es un elemento que retiene un valor en específico. Todos los objetos pueden pertenecer a una de tres clases: constantes, señales o variables.[34]

a) Constantes.

Son objetos que mantienen siempre un valor fijo durante la ejecución del programa. La característica de este objeto estriba en que el valor inicial que se le asigna al objeto no puede ser modificado a lo largo del código.

Sintaxis y ejemplo

Las constantes requieren de un identificador, un tipo de dato y una expresión que indique el valor específico que se le asigna en su sintaxis, como este objeto permite identificar el valor que se le asigna desde el inicio, por ende el lector puede vincular fácilmente su trabajo en el código.

La declaración de una constante debe ser de la siguiente forma:

```
constant identificador [, identificador…]: tipo := valor;
```

Ejemplo:

```
constant digital: integer := 10;
```

En el ejemplo anterior se define una constante llamada “digital” que tomará un valor entero de 10, lo que vemos entre corchetes significa que opcionalmente se pueden asignar más constantes, agregando diferentes identificadores, si es que así se requiere.

Cabe mencionar que las constantes solo son reconocidas en la unidad de diseño que se asignen,[35] es decir, al momento de programar una constante, si

[34] HDL Reference Manual. Cypress Semiconductor. California, Estados Unidos. 2002. pp. 5.

ésta es definida dentro de una unidad de diseño, por ejemplo una entidad, este objeto solo se podrá ocupar dentro de dicha entidad y no tendrá funcionamiento alguno fuera de ella, o fuera de la unidad de diseño en que se establezca, más adelante se explicarán cuales son dichas unidades.

b) Variables

Una variable tiene la tarea de poder cambiar un valor asignado continuamente dentro del código, generalmente se utilizan para manejar datos que no pueden tener un valor fijo. Un objeto de la clase variable puede también conservar un solo valor de un tipo determinado en cualquier momento del código, aunque no sería lógico utilizarlo para esta tarea en específico, ya que para esa labor se contemplaría un objeto de tipo constante. Una variable, puede adquirir diversos valores a lo largo del diseño del programa, es decir, aunque contiene un tipo de valor único, este puede irse modificando, los valores se declaran a una variable por medio de una sentencia de asignación, puede ser desde que se crea o posteriormente.

Sintaxis y ejemplo

Las variables requieren también de un identificador, un tipo de dato y de ser necesario una a expresión que indique el valor que se le asigna al ser declarado en su sintaxis inicial, seguramente dicho valor inicial cambiará al menos una vez, aunque podría hacerlo constantemente.

La declaración de una variable debe ser de la siguiente forma:

```
variable identificador [, identificador…]: tipo [:= valor];
```

Ejemplo:

```
variable contador_binario : bit;
```

[35] Álvarez Marquina Agustín. Tecnología de computadores. Descripción de VHDL. http://www.ele.uva.es/~sduenas/ASIC/VHDL2A.pdf

En el ejemplo anterior se define una variable llamada “contador_binario” que tomará un valor de tipo bit y podrá ir cambiando indistintamente, recordando que lo que se encuentra entre corchetes es opcional, se pueden asignar un valor inicial si fuera necesario, e inclusive otras variables, agregando más identificadores diferentes.

Las variables generalmente se utilizan como índices, apuntadores o contadores principalmente en instrucciones de bucle o procesos de iteración.

c) Señales

Un objeto de tipo señal también puede cambiar el valor que le han asignado constantemente. Las señales son muy similares a una variable, pero tienen una diferencia importante: las señales pueden retener o pasar valores lógicos, mientras que las variables no pueden hacerlo,[36] es decir, las señales pueden representar elementos de memoria o conexiones y las variables no tienen referente directo alguno con el hardware.

Sintaxis y ejemplo

Las sintaxis de las señales son casi idénticas a la de las variables, con las diferencias que ya se mencionaron anteriormente.

La declaración de una señal debe ser de la siguiente forma:

```
signal identificador [, identificador…]: tipo [:= valor];
```

Ejemplo:

```
signal signal_A, signal_B: std_logic;
```

En el ejemplo anterior se crean dos señales llamadas “signal_A” y “signal_B” y el tipo de dato que utiliza es el std _logic.

La mayoría de los objetos en VHDL, sean constantes, variables o señales deben ser declarados antes de que estos puedan ser utilizados. Aunque existen

[36] Sánchez-Elez. OP.Cit. pp 4

algunas excepciones, por ejemplo; los puertos de una entidad son implícitamente declarados como señales en el momento de la declaración, ya que estos representan conexiones físicas.

2.2.3 Tipos de dato

Los tipos son pautas de datos que el diseñador asigna a los objetos que contiene algún valor, dependiendo la tarea que realice ese objeto será su tipo.

En VHDL se tiene una inmensa gama de tipos, a continuación se muestran los tipos predefinidos más utilizados por el programa, incluyendo con los que se trabajará en este texto[37]:

TIPO BIT: En este tipo los valores solo pueden tomar el valor de 1 y 0 lógicos, la sintaxis de este tipo es de la siguiente manera:

objeto identificador [, identificador…]: bit [:= valor];

Ejemplo:

```
signal binario: bit := '0';
```

En el ejemplo anterior se muestra la manera de como es que se debe asignar una señal de tipo bit, que en este caso se nombra "binario" y se le asigna un valor de inicio '0', es muy importante introducir el valor de tipo bit entre comillas sencillas

TIPO BOOLEAN: Solo puede ocupar los valores de verdadero o falso en un dato. Su correcta sintaxis es:

objeto identificador[, identificador …]: boolean [:= valor];

Ejemplo:

```
signal positivo: boolean:= true;
```

En el ejemplo anterior tenemos una señal de tipo booleana, la cual se llama "positivo" e inicializamos su valor para que comience en verdadero

[37] HDL Reference Manual. Op. Cit. pp 7

TIPO INTEGER: Puede tomar un valor entero sobre 32 bits, es decir de 2,147,483,648 hasta -2,147,483,647. En su sintaxis un entero puede estar limitado en un rango para evitar dicha codificación de 32 bits[38].

objeto identificador: integer range <valor inicial> to <valor final>;

Ejemplo:

variable DiasMes: integer range 1 to 31; ó constante cinco: integer:=5;

Ahora se muestra la sintaxis de dos objetos de tipo entero, uno de los cuales es delimitada entre un rango establecido, éste tomará un valor entre 1 y 31, aunque recordemos que delimitar una expresión de este tipo es opcional, el otro es una constante que siempre tendrá el valor entero de 5.

TIPO CHARACTER: El tipo character puede adquirir los valores que consisten en los 128 caracteres del código ASCII. Los caracteres de ASCII que son imposibles de ver, son representados por dos o tres caracteres, que los identifican respectivamente.

objeto identificador: character [:= valor];

Ejemplo:

variable pesos: character := '$';

En su sintaxis se observa que este tipo puede contener cualquier tipo de caracter, si es que este puede ser visible, por regla el caracter que le sea asignado deberá estar entre comillas simples (‘’).

TIPO STRING: Es un tipo de dato que consiste en un conjunto de elementos de tipo carácter, o sea un arreglo de caracteres.

objeto identificador[es]: string [:= valor];

Ejemplo:

constante saludo: string(1 to 11) :="Hola mundo!";

[38] Jan Van der Spiegel. VHDL Tutorial. University of Pennsylvania. Department of Electrical and Systems Engineering.
http://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html#_Toc526061352

Como se puede observar en el texto anterior, se describe un clásico ejemplo de programación, se asignará el conocido "Hola mundo" a una constante que se llamará "saludo", lo que es importante resaltar es que por ser un conjunto de elementos, éstos deben de ser asignados entre comillas dobles o inglesas ("").

TIPO BIT_VECTOR: Es un grupo de bits, llamado comúnmente bus, cualquiera de ellos tiene las características del tipo bit, el bus puede ser del tamaño que se requiera. Para manipular los buses estos se pueden ordenar de manera ascendente o descendente según convenga.

objeto identificador [es]: bit_vector (magnitud) [:= valor];

Ejemplo:

signal vector_A, vector_B: bit_vector (0 to 7);

En el ejemplo antes citado, se expone una asignación de dos vectores llamados "vector_A" y "vector_B", ambos tienen una magnitud de ocho bits (teniendo en cuenta el cero) en el siguiente tema se explicará de forma más detallada como diseñar objetos con vectores y sus características.

TIPO STD_LOGIC: Este tipo de dato es una extensión del tipo bit, pero tiene la ventaja de ser más práctico por que puede tomar valores diferentes a los preestablecidos que son; 1 y 0 lógicos, los que se utilizan con más frecuencia son los valores de "alta impedancia" representado por la letra 'Z' y el valor de "no importa" representado por el signo "-"

objeto identificador[es]: std_logic [:= valor];

Ejemplo:

variable salida, resultado: std_logic;

Como se muestra en la sintaxis y el ejemplo anterior se crean dos variables que llevan los nombres de "resultado" y "salida" y en seguida se declaran con el tipo std_logic; aunque puede tomar distintos valores generalmente

este tipo de dato trabajará con los valores alto y bajo como un ejemplar de tipo bit.

Cabe aclarar que este tipo de dato se encuentra en un paquete (STD_LOGIC_1164) que a su vez está adentro de una biblioteca llamada IEEE, para utilizar este tipo de dato debemos declarar la utilización de dicho paquete[39] antes de poder asignárselo a un objeto

TIPO STD_LOGIC_VECTOR: El tipo de dato std_logic_vector es simplemente un vector con un arreglo de elementos del tipo std_logic. Su uso es muy similar al tipo del bit_vector. La diferencia principal entre un bit_vector y el std_logic_vector se encuentra en que cada uno de los elementos del arreglo serán de tipo std_logic.

```
objeto identificador[es]: std_logic_vector (magnitud) [:= valor];
```

Ejemplo:

```
variable vec: std_logic_vector (0 to 4):= "00000";
```

La manera correcta de escribir la sintaxis se muestra en el ejemplo anterior creando un vector llamado "vec" el cual contiene un bus de cinco bits, en este caso se inicializó con un valor de cero, también habrá que tomar en cuenta cuando se tiene un conjunto de elementos estos irán entre comillas dobles.

De igual manera que el tipo anterior es necesario declarar que se va a utilizar la biblioteca correspondiente que los contiene

2.2.4 Diseño de objetos mediante vectores

Se ha citado ya la sintaxis para poder asignar un tipo de dato vector en un objeto, pero estos datos tienen características propias interesantes y se expondrán de manera más explícita algunas de sus cualidades para poder efectuar con más eficiencia el resultado de un proyecto.

[39] Véase Apartado 2.3.3 Bibliotecas, paquetes y su uso

Como un vector es una colección de elementos, se debe decidir que cantidad de elementos contendrá este, esto se conoce como **magnitud**, y cada uno de estos elementos tendrá su lugar específico en este arreglo, los valores del vector siguen un orden. Existen dos tipos de orden que se pueden utilizar en VHDL, estos son; ascendentes y descendentes. El utilizar uno u otro dependerá en gran medida de la forma en que el programador quiera tener referencia en cuanto al bit menos significativo y el de mayor valor dentro del arreglo. A manera de ilustrarlo a continuación se muestran dos vectores agrupados de diferente manera:

Ascendente	Descendente
Vector_X = (X0, X1, X2, X3)	Vector_Y = (Y3, Y2, Y1, Y0)

Para personalizar el orden y el tamaño de la magnitud se debe poner entre paréntesis el numero de valor inicial (el cual siempre será cero), el valor final de la cantidad de elementos requeridos y utilizar las palabras reservadas ***"downto"*** o *"**to"*** respectivamente entre ellas, esto último determinara si el orden del vector es ascendente o descendente. A continuación se describen unos ejemplos para que el lector pueda entender su correcta sintaxis:

Correcta declaración de vectores

```
signal A: bit_vector (0 to 7);          signal C: std_logic_vector (0 to 5);

signal B: bit_vector (7 downto 0);      signal D: std_logic_vector (11 downto 0);
```

Cualquiera de estos cuatro vectores esta correctamente escrito, se puede notar que el orden varía y que la cantidad de elementos que tiene cada uno es diferente. Se recomienda utilizar con más frecuencia la manera decreciente, ya que esta es la forma en que está acomodada los numeración binaria, pero el utilizar una u otra dependerá en la forma en que al diseñador le acomode.

Para asignar un valor al bus de algún vector que ya se ha creado, se hará a través de comillas dobles debido a que es un arreglo, a menos que se refiera específicamente a un elemento de dicho arreglo.

Correcta asignación de vectores

A <= B"01100101"	**D <="000000001111"**
B <= X"A8"	**D <= O"0017"**
C <= O"37"	**D <= X"00F"**

Como se puede leer en las asignaciones anteriores, puede ser acertado hacerlo de diferentes maneras, el prefijo "X" denota un valor hexadecimal, el prefijo de "O" denota un valor octal, el prefijo "B" denota un valor binario. Si no hay prefijo incluido, se asume que el bus es binario. Los valores hexadecimales y octales deben ser utilizados solamente si puede coincidir con el tamaño del vector.[40]

Los vectores son herramientas que nos ayudaran en gran medida a programar ya que en ocasiones es más sencillo trabajar con un grupo de bits que hacerlo por separado en cada elemento. En resumen las características de mayor importancia son las siguientes.

- Para evitar errores, el orden en el cual se utiliza el vector debe ser el mismo en el cual esta asignado.
- No es necesario utilizar el vector en su totalidad.
- Los vectores que se requieran convertir a escala octal deben de ser de una magnitud binaria correspondiente a un múltiplo de tres, a su vez para un vector que se quiera convertir en hexadecimal la magnitud binaria de este debe corresponder a un múltiplo de cuatro.
- Se recomienda que la magnitud de los vectores que se someta a cualquiera tipo de operación entre ellos debe ser igual.

Con respecto al último punto, los vectores como algunos otros tipos de dato pueden ser comparados o pasados por alguna operación aritmética o booleana. Conoceremos estos tipos de operadores, así como sus características en el tema siguiente, este apartado será de mucha importancia en el futuro de este texto ya que en la práctica los operadores son usados con mucha regularidad.

[40] HDL Reference Manual. Op. Cit. pp 9

2.2.5 Tipos de operadores

El lenguaje VHDL proporciona operadores comúnmente usados para construir expresiones o para calcular valores. VHDL también cuenta con operadores de asignación o de asociación los cuales se explicarán a continuación.

Los tipos de operadores se dividirán en tres grupos, para poder dar un mejor seguimiento de sus funciones, estos son:

- Operadores lógicos
- Operadores relacionales
- Operadores aritméticos

Además de estos, existen los operadores de asignación que transfieren valores a partir de un objeto a otro y los operadores de la asociación que asocian tipos de objetos entre si.

2.2.5.1 Operadores lógicos

Los operadores lógicos son los siguientes; AND, OR, NAND, NOR, XOR, XNOR y NOT, estos operadores servirán para describir funciones booleanas, las operaciones que se efectúen entre ellos, con excepción de NOT, deben realizarse con datos que tengan la misma longitud de bits

Los operadores lógicos realizan sus funciones en cualquier clase de objetos pero solo en algunos tipos de dato como los son los de tipo bit (std_logic, bit), los de tipo vector (std_logic_vector, bit_vector) y los de tipo booleano (boolean).[41]

A continuación un ejemplo de una función booleana utilizando operadores lógicos tal y como se expresaría en VHDL:

Ecuación	Ejemplificada en VHDL
$Q = (w + x) \cdot (y + z)$	Q<=((w or x)and(y or z))**;**

[41]López Carreto Juan Manuel, et al. Teoría y práctica de diseño digital con lógica programable. Operadores lógicos en VHDL. pp.71

Como una sentencia de operación considerablemente grande puede escribirse en una misma línea, los operadores lógicos en una expresión deben ser delimitados por paréntesis para garantizar el análisis y la evaluación de dicha línea sin error alguno.

2.2.5.2 Operadores relacionales

Los operadores relacionales son usados para probar la igualdad, desigualdad o magnitud de una expresión. Los operandos de cada operación relacional deben ser del mismo tipo, y aunque el número de bits comparados puede ser diferente se recomienda solo hacer comparaciones con vectores que contengan la misma longitud para evitar confusiones.

En seguida se expone la tabla 2.2 que muestra los signos de los operadores relacionales y el significado de cada uno de ellos.

Operador	Significado
=	Igual que
/=	Desigual o diferente
<	Menor que
<=	Menor o igual
>	Mayor que
>=	Mayor o igual

Tabla 2.2 Operadores relacionales

Estos comparadores operan sobre todo en tipos de dato como los de tipo bit (std_logic, bit), de tipo vector (std_logic_vector, bit_vector), de tipo booleano (boolean) y los enteros (integer).[42]

El resultado de cada operación relacional es de tipo booleano, es decir, será cierto cuando la condición se cumpla y falso cuando no.

[42] Ibidem pp.72

2.2.5.3 Operadores aritméticos

Como su nombre lo indica estos operadores permiten realizar cálculos de tipo aritmético. Generalmente son utilizadas para diseñar algún dispositivo sumador, restador o contadores, ya sea incrementando o decrementando valor en un dato.

A continuación se puede observar en la tabla 2.3 los operadores aritméticos y una breve descripción de su funcionamiento.

Operador	Descripción
+	Suma
-	Resta
/	División
*	Multiplicación
**	Potencia
Mod	Modulo de la división
Rem	Resto de la división
Abs	Valor absoluto

Tabla 2.3 Operadores aritméticos

Estos operadores se utilizan con objetos de tipo entero (Integer) pero igualmente pueden ser soportados sobre algún vector (bit_vector, std_logic_vector) si es que fuera necesario, para esto nos tendríamos que apoyar en un paquete[43] especializado que nos permita hacer uso de ellos, esto se expondrá más adelante.

En VHDL los operadores "+" y "-" realizan suma y resta respectivamente, a estos operadores se les conoce como Operadores de adición, los operandos "*" y "/" son utilizados para multiplicar y dividir respectivamente, también se le unen las palabras reservadas rem y mod, estos dos operadores devuelven el residuo

43 Véase el apartado 2.3.3 Bibliotecas, paquetes y su uso para más información

cuando un operando es dividido por otro y solo aceptan valores de tipo entero. Si los dos datos son positivos *mod* y *rem* producen el mismo resultado, pero con valores negativos *mod* da el signo del denominador y *rem* da el signo del numerador,[44] a estos cuatro operadores anteriores se les conoce como Operadores de multiplicación cabe mencionar que el resultado será del mismo tipo de dato que los operandos.

Por último tenemos los dos operadores restantes que son "abs" y "**", la sentencia "abs" esta definida para objetos de tipo entero y es utilizada para devolver el valor absoluto del operando, El operador "**" incremente un número a una potencia al cuadrado.

2.2.5.4 Otros operadores

El lenguaje VHDL también cuenta con un operador que concatena valores y se representa con el símbolo "&", se puede considerar parte de los operadores de adición, ya que aunque su función propiamente no es sumar, si agrega un valor a otro.

Este operador puede unir dos vectores y el resultado del concatenamiento es un arreglo unidimensional cuya longitud es la suma de las longitudes de los operandos, y cuyos elementos de inicio consisten en los elementos del operando izquierdo seguido por los elementos del operando derecho, es decir, el orden que se tomará será de izquierda a derecha.[45]

Además tiene dos operadores más que son representados por los signos siguientes: ":=" y "<=" A estos dos operadores se les conoce como Operadores

44 Fundamentos de programación, Operadores aritméticos: http://www.iuma.ulpgc.es/users/jmiranda/docencia/libro_ada/libro_ada_html/node27.htm#foot810

45 El lenguaje VHDL 93. Programación y ejemplos. http://www.angelfire.com/al4/vhdl93/vhdl93.htm

de asignación, el primero es usado para asignar las señales y el segundo para asignar variables.

En el tema tipos de objetos se explicó que a cualquier objeto se le podría asignar de inicio con un valor, pero tanto las señales como las variables pueden cambiar de valores cuantas veces sean necesario, esto es posible gracias a los operadores de asignación. La asignación de una variable solo puede ocurrir adentro de un proceso, mientras que la asignación de una señal puede tener lugar en cualquier parte dentro de la arquitectura,[46] estos módulos de diseño se explicaran en seguida.

2.3 Organización y arquitectura de VHDL

El software VHDL tiene sus propias unidades de diseño y esas unidades se rigen por normas, es decir, la colocación de un código debe tener coherencia para la interpretación del mismo programa, todas estas normas se verán a detalle tanto en sintaxis como también en la manera de estructurarlas. Así mismo este software tiene diversos estilos de programación para moldear un sistema, estos tipos dependen en gran medida de la creatividad y visión del programador y también se abordarán de manera breve.

2.3.1 Unidades básicas de diseño

El software VHDL fue diseñado a base de los principios de la programación estructurada, su arreglo general está formada por módulos o unidades de diseño, cada uno de ellos cuenta con un conjunto de declaraciones específicas para poder ser funcionales, también cada uno de ellos sirve para estructurar y complementar un sistema digital y complementarse entre sí, ya que en la mayoría de los casos un módulo se completa con algún otro. A continuación se citan los elementos básicos con los cuales se diseña.

46 Véase la sección 2.3.1 Unidades básicas de diseño

Existen unidades básicas para el diseño en VHDL [47] que pueden ser analizadas individualmente para una mejor investigación y entendimiento del lector, estas son:

- Declaración de la entidad – Entity declaration: Definen la representación externa del modelo.
- Configuración – Configuration: Definen la relación entre entidades y arquitectura.
- Arquitectura – Architecture: Define la estructura interna del modelo.
- Paquete – Package: Colección de elementos que pueden ser utilizados en otros diseños. Para su estudio se divide en dos: cuerpo del paquete y declaración del paquete.
- Biblioteca – Library: Almacena las unidades de diseño resultantes de la compilación

En el desarrollo de los programas VHDL se pueden utilizar o no tres de los cinco módulos, pero dos de ellos; entidad y arquitectura son indispensables para la estructura de un programa, a la pareja entidad/arquitectura se le llama **modelo** y es una dupla a la cual le lector se tiene que acostumbrar ya que como se ha descrito son imprescindibles para un código funcional.

Los módulos; declaraciones de entidad, declaración del paquete y configuración se consideran unidades de diseño primarias, mientras que la arquitectura y el cuerpo del paquete son unidades de diseño secundarias, ya que dependen de una entidad primaria que se debe de analizar antes que ella.[48]

La esencia de este programa es definir la interfaz de una entidad de hardware mientras se dejan invisibles sus detalles internos. Muchos diseñadores conciben

47 El lenguaje VHDL. Unidades básicas de diseño. Departamento de electrónica (DEPECA) España: http://depeca/repositorio/asignaturas/30791/T4_02_unidades_diseno.pdf

48 Maxines David. Op.Cit. VHDL: su organización y arquitectura pp.37

la entidad es como la funda de la arquitectura dejando invisible la lógica que está contenida dentro de ella.

2.3.2 Declaración de la entidad

En VHDL el bloque conocido como entidad se declara en primer lugar. Una entidad indica las señales que entran y salen del circuito, además especifica cuantas son, el tamaño, si son de entrada o salida y el tipo, en otras palabras, declara la relación del circuito con el mundo exterior detallando las terminales.

Para este programa cualquier diseño lógico por muy simple que éste sea será una entidad, es decir, hasta el dispositivo más sencillo como una compuerta básica será reconocida como una entidad, ya que sin importar lo complejo que sea su comportamiento éste tendrá señales de entrada, las cuales esperará por algún acceso y señales de salida que serán devueltos al exterior dependiendo el resultado del proceso que contenga el código, cabe aclarar que se puede tener múltiples salidas.

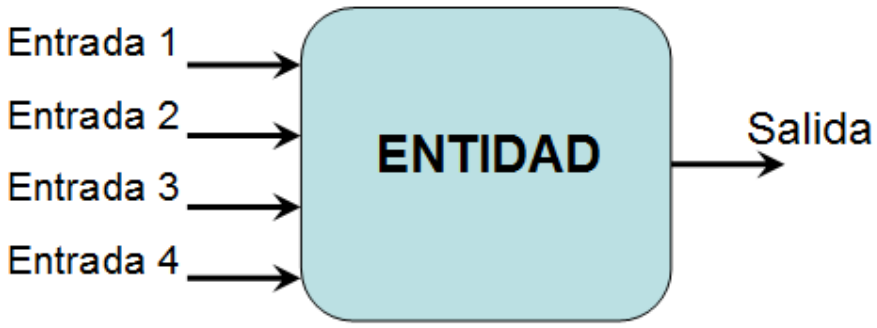


Figura 2.1 Entidad con cuatro terminales de entrada y una terminal de salida

Una entidad es el bloque más elemental para VHDL, en la figura 2.1 claramente se pueden apreciar dicho módulo con cuatro entradas y una salida, el proceso que se sigue para devolver una salida no la sabemos, ya que la lógica que se sigue para elaborar la salida no consta de la entidad, la prioridad de este modulo son asignar las entradas y salidas y su correcto funcionamiento, es decir, que éstas sean congruentes.

Para entender de forma sencilla esta unidad de diseño la podemos ver como el equivalente al esquemático de lo que queremos representar físicamente, dicho esquemático describe las conexiones de un componente al resto del proyecto.[49]

En la entidad las señales de entrada/salida se conocen como **puertos**, los cuales son similares a las terminales de cualquier circuito. Todos los puertos que sean declarados deben estar bien detallados, y sintácticamente deben contener un identificador, un modo y un tipo de dato.

2.3.2.1 Modos

Un modo permite definir la dirección de los datos que serán transferidos a través de un puerto. Existen cuatro valores de modos diferentes, estos son: entrada (in), salida (out), entrada/salida (inout) y retroalimentación o buffer.

- Modo entrada: Con este modo se representan las señales de entrada de la entidad, esta es unidireccional y solo permite el flujo hacia adentro de la entidad.
- Modo salida: Indica que las señales irán de adentro hacia fuera, como el modo anterior este también es unidireccional.
- Modo entrada/salida: Esta terminal como su nombre lo indica será de entrada/salida, permite declarar un puerto de forma bidireccional permitiendo la retroalimentación de señales dentro y fuera de la entidad.
- Modo buffer: Este modo se utiliza con señales que además de salir de la entidad, permite hacer retroalimentaciones internas en ella. Pareciera muy similar al modo anterior, pero su diferencia se encuentra en que el puerto que sea declarado buffer será unidireccional.

[49] Introducción a VHDL. Declaración de entidades.
http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/rubio_s_d/capitulo3.pdf

Cuando se omite el modo de una señal en la declaración de la entidad se sobreentiende que es de entrada[50], aunque en nuestro caso esto sería un olvido ya que no se recomienda que los puertos se dejen sin modo.

2.3.2.2 Tipos de dato

Los tipos de datos predefinidos son los que se vieron en la sección 2.2.3 en este texto se trabajará con algunos de ellos, pero los que se ocuparán con mayor frecuencia son los que tengan características de tipo bit, ya que a través de este tipo de señales se reflejarán en los puertos los pulsos correspondientes, estos son:

- Bolean
- Bit
- Bit_Vector
- Std_Logic
- Std_Logicd Vector.

Recordemos que el objetivo de esto es poder enviar y recibir pulsos de entrada o de salida con un fin específico en el CPLD que se ha elegido (7C373i), con estos pulsos, que serán la base de cualquier diseño físico, se trabajará para hacer funcionar la tarjeta probadora.

2.3.2.3 Diseño de puertos mediante vectores

El software VHDL hace una excepción que consiste en que dentro de la entidad se le asignen señales a los puertos, incluso sin que estos objetos se hayan creado con anterioridad y hasta cierto punto esto es lógico ya que forzosamente debe contener mínimamente alguna señal, ya que sin entradas o salidas, el dispositivo que representa una entidad no serviría de nada.

Dentro de la entidad, los puertos de entrada o salida pueden trasportar una cantidad de bits al interior o al exterior según corresponda. Obviamente en

50 Definición VHDL: Modos. Wikipedia. La enciclopedia libre: http://es.wikipedia.org/wiki/VHDL

algunos momentos será conveniente, que en lugar de bits independientes, se manipule un arreglo de tipo vector para ahorrar tiempo y espacio.

En seguida se muestra la arquitectura de como de debe componer una entidad, y la sintaxis de todo lo antes expuesto para dar más claridad al texto.

2.3.2.4 Estructura y sintaxis de la entidad

A pesar de que la descripción que se ha hecho aquí es extensa, al contrario de lo que se pueda suponer la estructura de una entidad es muy simple, pero aunque la arquitectura sea relativamente sencilla no se debe pasar por alto que tiene reglas muy sensibles y las cuales se deben respetar para su buen funcionamiento.

La entidad debe llevar un identificador para nombrarla, se recomienda que ese identificador haga referencia al contenido del código, también es ampliamente recomendado que el archivo en VHDL pueda llamarse de la misma manera para ayudar al programador a relacionar el archivo o la entidad con el contenido.

Las señales que se tienen que definir para asignarlas a las terminales del circuito lo harán con la palabra reservada ***"port"***, siendo todo lo que haya entre paréntesis en seguida de dicha palabra; todas las señales a definir en las terminales correspondientes del dispositivo, terminando la asignación con punto y coma. La sintaxis general de una entidad se muestra a continuación:

Declaración de una entidad

```
ENTITY <nombre_de_la_entidad> IS
PORT (asignación de terminales);
END [nombre_ de_la_entidad];
```

Por ejemplo supongamos que tenemos una entidad a la que llamaremos "operaciones" la cual debe tener cuatro variables de entrada, entre las cuales dos de ellas serán vectores de cuatro bits y una única salida de tipo boleana, donde recibiremos el resultado de las operaciones hechas entre nuestras variables de entrada.

Figura 2.2 Entidad llamada operaciones con cuatro entradas y una salida

La entidad mostrada en la figura 2.2 expone de manera gráfica las terminales de entrada y salida utilizadas por el dispositivo, el código estaría representado de la siguiente manera dentro de VHDL.

```
--Declaración de la entidad en VHDL
ENTITY operaciones IS
PORT (Vector_A, Vector_B: in bit_vector (3 downto 0);
     Operando_A, Operando_B: in bit;
     Resultado: out boolean);
END operaciones;
```

La pantalla anterior muestra el código correspondiente a la entidad de la figura 2.2. Como se puede apreciar la primer línea es un comentario, la segunda es la declaración de la entidad utilizando un identificador para nombrarla, en seguida declaramos las señales que habrá en los puertos, estas son cinco, cuatro de ellas son de entrada, y la última es de salida, nótese que cuando más de un objeto asignado tiene la mismas características que otros se puede asignar en forma de lista anteponiendo comas entre ellas, este es el caso de las señales llamadas “Vectores” y las nombradas “Operandos”. Finalizando en la línea que contiene la palabra reservada **end** y el nombre de la entidad aunque poner este último identificador también es opcional.

2.3.3 Bibliotecas, paquetes y su uso

Una parte muy importante en el lenguaje VHDL es el uso de bibliotecas y paquetes, ya que estos pueden ayudar a un diseño complejo facilitando las cosas, una biblioteca es una colección de elementos previamente analizados (compilados y elaborados) como pueden ser; paquetes, componentes, entidades, arquitecturas, etcétera, que puedan referenciar a otros diseños de VHDL.

Si toda la información sobre una descripción de diseño tuviera que aparecer en un mismo archivo, muchos archivos de VHDL serían enormes e incómodos, y la reutilización de la información sería complicada. Afortunadamente, VHDL permite que el usuario comparta la información entre los archivos por medio de dos estructuras; bibliotecas y paquetes.[51]

A las bibliotecas también se les conoce como librerías comúnmente, aunque de manera incorrecta, esto sucede por el anglicismo de donde provienen, la palabra "library". Una biblioteca como su nombre lo indica es un lugar donde se tiene almacenada un gran cúmulo de información, así mismo en VHDL tendremos que acceder a ella para poder hacer uso de las herramientas que se albergan dentro como los paquetes. Para hacer el contenido de una biblioteca accesible para un programa en VHDL, se le tiene que mandar a llamar con la palabra reservada ***"library"***. La sintaxis para acceder a una biblioteca se muestra a continuación:

LIBRARY nombre de la librería [, nombre de la librería …];

Ejemplo

LIBRARY ieee, my_pkg;

En el ejemplo anterior, esta sentencia hace accesible los contenidos de las dos bibliotecas llamadas "ieee" y "my_pkg". En VHDL generalmente se trabajará con dos bibliotecas que contienen paquetes que nos son de utilidad, estos son:

- Biblioteca ieee: Contiene paquetes para la manipulación del tipo de dato std_logic, también contiene algunos paquetes aritméticos, entre otras cosas.
- Biblioteca work: Los resultados de analizar la descripción del código se ponen por defecto en esta biblioteca para su uso posterior, por eso este almacén de trabajo no necesita la sentencia **library** ya que de cualquier manera siempre está presente al desarrollar algún diseño. En

[51] HDL Reference Manual. Op. Cit. pp. 79

esta biblioteca, se almacenan unidades de diseño una vez compiladas, en este lugar también se contienen paquetes aritméticos.[52]

Ahora bien, si es necesario utilizar un paquete de alguna biblioteca, este también debe de ser llamado por una sentencia para poder hacer uso de él. Un paquete es una unidad de diseño que contiene algoritmos preestablecidos.

Para poder acceder a los paquetes necesariamente deber mandar llamar antes la biblioteca que lo contiene, en seguida el paquete por medio de la palabra reservada **use**, seguida del nombre de la biblioteca, el nombre del paquete y módulo del paquete, todo esto separado por puntos,[53] la sintaxis se muestra a continuación:

USE Nombre_de_la_biblioteca.Nombre_del_paquete.Módulo_del_paquete;

Ejemplo:

USE Project_lib.special_pkg.comp1;

La sintaxis anterior muestra cómo se accede al paquete llamado "special_pkg" que a su vez esta dentro de la biblioteca "Project_lib". Ahora supongamos que se quiere utilizar un paquete de la biblioteca ieee y otro de la biblioteca work, Los paquetes que se utilizarán con más frecuencia en la práctica servirán para ejemplificar la sintaxis correcta.

```
--Declaración de bibliotecas y paquetes
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.std_arith.all;
```

Como se ha mencionado anteriormente para poder acceder a un paquete primero se define la biblioteca que lo contiene, después se llama al paquete. En este caso hay dos paquetes de bibliotecas diferentes, se accede a el primero con la palabra **library**, no se hace lo mismo con la biblioteca **work** por que esta ya está

[52] El lenguaje VHDL. Unidades básicas de diseño. Departamento de Electrónica. Universidad de Alcalá. España. pp. 116

[53] Ing. Diego Barragán Guerrero. VHLD: Su organización y arquitectura. http://es.scribd.com/doc/20215819/VHDL-su-organizacion-y-arquitectura

presente, a continuación se declara el permiso para utilizar los dos paquetes "std_logic_1164" y "std_arith" con la sentencia **use**, la palabra reservada **all** se usa para acceder a todos los componentes que contenga ese paquete que puedan ser de utilidad, voy a tratar de explicar brevemente lo que hacen estos paquetes para que se tenga una idea concreta del porqué se utilizan con regularidad:

- Std_logic_1164: Define los tipos de datos std_logic, std_logic_vector que suelen emplearse con mucha frecuencia en VHDL
- Std_arith: Contiene funciones y operadores aritméticos, aparte define las operaciones matemáticas soportadas para los tipos de datos vector.

Hasta ahora se han descrito los módulos de diseño que dan forma de la estructura de nuestro proyecto con el mundo exterior, es decir el encabezado del código, pero falta una parte vital para cualquier diseño y esta es la lógica del programa, dicha lógica se lleva a cabo en un módulo diferente a los anteriores y el cual es imprescindible para cualquier programa en VHDL, en el tema que sigue se expondrá de este módulo llamado arquitectura y de las estructuras que puede contener para su pleno funcionamiento.

2.3.4 Declaración de la arquitectura

En este apartado se explicará la parte final de un código, es decir, se expone la manera de complementar un programa para que este cumpla con la finalidad que se desea.

La arquitectura es el módulo de diseño donde se escribe la lógica del código, y se describe el funcionamiento de una entidad, en otras palabras, la arquitectura describe lo que se debe hacer con los puertos de entrada para que los puertos de salida puedan tener funcionamiento y obtener de ella un resultado. Si la entidad se viera como una caja negra, la arquitectura es el conjunto de detalles que se encuentra dentro de la caja.[54]

[54] Rubio Sánchez D. Modelado y Simulación de un Conmutador utilizando VHDL. Implementación en VHDL. pp 19

Dentro de la arquitectura existen varias estructuras básicas que se pueden usar, en este tema se abordarán estas estructuras mostrando la correcta sintaxis para que el lector tenga las nociones suficientes al crear su propia lógica programable, pudiendo solucionar el problema no solo de una, sino de diferentes maneras contando con distintos recursos.

El software VHDL diseña códigos para describir un circuito, para esto cuenta con diferentes estilos de programación, el lector puede hacer uso del que más convenga al momento de encontrarse con un nuevo proyecto o bien combinarlos si es que esto le puede resultar más eficiente.

De forma general los estilos de diseño que se utilizan en VHDL son los siguientes:[55]

DESCRIPCIÓN FUNCIONAL: Su principal característica es que describe el circuito de forma secuencial, esta forma de diseño es la que más se parece a los lenguajes de programación estructurada,[56] las sentencias secuenciales deben de ir dentro de un proceso. Dichos procesos y declaraciones secuenciales permiten modelar una función con rapidez.

DESCRIPCIÓN POR FLUJO DE DATOS: En este estilo se describe como la información es transmitida de una señal a otra y como se transmite de la entrada a la salida sin seguir una estructura en cadena. En este tipo de descripción se permite definir el flujo que tomarán los datos entre módulos encargados de realizar operaciones, sin el uso de asignaciones secuenciales. En este estilo no se utilizan procesos.

55 Capítulo 3. Implementación en VHDL. Estilos de programación http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/rubio_s_d/capitulo3.pdf

56 Programación estructurada: Consiste en que las rutinas de control de un programa se encuentran de forma tal que es posible leer la secuencia del mismo desde su inicio hasta su fin en forma continua, sin tener que saltar de un lugar a otro del programa. Fuente de la información: www.monografias.com

DESCRIPCIÓN ESTRUCTURAL: Esta descripción utiliza entidades descritas y compiladas previamente. Se declaran los componentes que se utilizan y después, mediante los nombres que los identifican, se realizan las conexiones entre las compuertas, es decir, el diseño se crea con instancias de componentes previamente hechos, estas instancias forman el diseño, al conectar los puertos de estas instancias con las señales internas del código o con las terminales del circuito.

ESTILO MIXTO: Esta es la combinación de dos o más estilos anteriormente citados.

La manera en que se pueda elegir entre uno de los anteriores estilos de diseño depende de cual le acomode al programador, o de qué manera abordamos el diseño para solucionarlo de manera eficaz.

2.3.4.1 Estructura y sintaxis de la arquitectura.

El cuerpo de la arquitectura especifica cómo funciona el circuito, cómo es implementado y así mismo cómo se ejecuta, pero esta también tiene sus normas de sintaxis a seguir. Anteriormente se explicó que la arquitectura es una estructura secundaria y depende completamente de una entidad que se analizó antes que ella, así que el declararla se tiene que dar a conocer por fuerza la entidad a la que esta sujeta.

A continuación se muestra la manera correcta de escribir la sintaxis de una arquitectura, la mayoría de las palabras reservadas se pondrán en mayúsculas para su identificación, aunque recordemos que no es necesario.

Declaración de una arquitectura

```
ARCHITECTURE <nombre_de_la_arquitectura> OF <NOMBRE_DE_LA_ENTIDAD> IS
[Declaración de objetos]
BEGIN
[Proceso]
--Zona de declaraciones
END [nombre_ de_la_arquitectura];
```

La estructura y sintaxis de los módulos en VHDL son relativamente sencillos ya que las estructuras son simples, lo que es más elaborado es la lógica de programación, la cual va dentro de la zona de declaraciones de la arquitectura.

Para explicar mejor lo que se describe en el párrafo anterior, se expone un ejemplo referenciandonos en la imagen que se presenta en la figura 2.3, como se puede observar se tiene una arquitectura que requiere dos compuertas lógicas con dos entradas cada una las cuales desembocaran en dos salidas, una para cada compuerta.

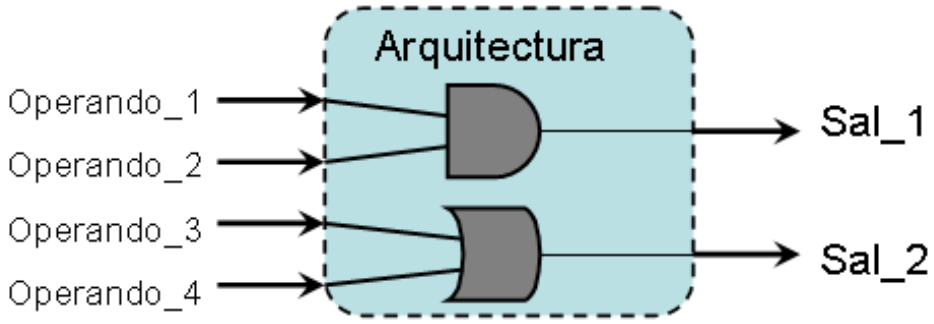


Figura 2.3 Arquitectura compuesta de dos compuertas lógicas

La forma más sencilla para abordar este problema lo podemos generar con operadores lógicos los cuales se han visto en el apartado 2.4.1. Ahora se verá el complemento de código que consta de la arquitectura que resuelve las operaciones lógicas.

```
--Declaración de la arquitectura en VHDL

ARCHITECTURE funciones OF compuertas IS
BEGIN
     Sal_1<=(Operando_1 and Operando_2);
     Sal_2<=(Operando_3 or Operando_4);
END funciones;
```

En este caso la arquitectura tiene el nombre de “funciones” y se entiende que antes de ella se declaró una entidad llamada “compuertas” la cual es el complemento de esa la arquitectura, en dicha entidad se debieron declarar seis señales, cuatro de ellas de entrada y dos de salida. La lógica de la arquitectura comienza con la palabra reservada “begin” asignando a la primer salida el

resultado de una multiplicación booleana con los operandos iniciales y a la salida dos el resultante de una compuerta OR entre los operandos tres y cuatro.

El ejemplo anterior es simple para la comprensión del lector novato, pero existen arquitecturas muy completas y más elaboradas, para tener la noción se debe conocer algunas estructuras de control con las cuales se puede tener más gama de manejo del lenguaje. El siguiente tema contiene dichas estructuras, su sintaxis y algunos ejemplos

2.3.4.2 Estructuras básicas de control.

El software VHDL contiene algunas estructuras de control, las cuales se mencionarán en este apartado. En los siguientes párrafos se estudiarán las estructuras lógicas de decisión e iteración, también se expondrán una serie de ejemplos para ver su correcto uso y funcionamiento.

Las estructuras básicas y algunas de las más clásicas que existen en el mundo de la programación se describen en el estilo funcional, esta es la forma en que se puede escribir un código de la manera secuencial, pero recordemos que las descripciones funcionales se deben de apoyar de la declaración de un proceso.

Un proceso es la construcción principal en el comportamiento funcional, las declaraciones de proceso deben aparecer dentro del cuerpo de una arquitectura y comienza con la palabra reservada "**process**". El contenido del proceso incluye declaraciones secuenciales, estas declaraciones se utilizan para definir las salidas del proceso con respecto de sus entradas. La declaración del proceso puede contener asignaciones de señales para especificar las salidas al hardware, estas se deben declarar previamente para poder ser manipuladas dentro del proceso, aunque las declaraciones secuenciales a menudo no tienen ninguna correspondencia directa con terminales del dispositivo a programar.[57]

[57] Definición de proceso: Green Mountain. Computing systems, Inc. http://www.gmvhdl.com/process.htm

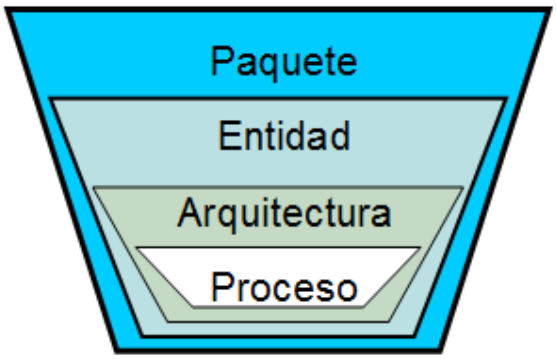


Figura 2.4 Jerarquías de los módulos de diseño

Como se muestra en la figura 2.4 los objetos declarados en un paquete están disponibles para todos los módulos en VHDL que quieran usar ese paquete, los objetos declarados en una entidad están disponibles para todas las arquitecturas de esa entidad, los objetos declarados en una arquitectura están disponibles para todas las sentencias de esa arquitectura y los objetos declarados en un proceso solo están disponibles dentro de ese proceso.[58]

La sintaxis para una declarar un proceso es la siguiente:

Declaración de un proceso

```
[nombre_del_proceso:] PROCESS (señales a utilizar)
[Declaración de constantes o variables]
BEGIN
        --Zona de declaraciones
END PROCESS [nombre_del_proceso];
```

El nombre del proceso es opcional, el ponerlo solo ayudará a conocer que tarea hace dicha estructura, en seguida de la declaración del proceso se escribe entre paréntesis las señales de entrada que se ocuparán al interior de él, esta lista de señales es muy sensible y las que no se mencionen en ella y operen dentro del

[58] El lenguaje VHDL: Unidades básicas de diseño. Op Cit.
Visibilidad de un objeto. pp 148

proceso no solo no se tomarán en cuenta, sino que también provocarán un error al momento de compilar el código.

```
ARCHITECTURE igualdad OF comparador IS
BEGIN
     Cont_salida: PROCESS(cont, C_out)
     BEGIN
          IF (cont='1') THEN
               C_out<='1';
          ELSE
               C_out<='0';
          END IF;
     END PROCESS Cont_salida;
END igualdad;
```

En este fragmento de código es notoria otra condición que es opcional pero nos puede resultar muy útil; la sangría del lado izquierdo cada vez que entramos a un nuevo módulo nos ayudará a relacionarlo con el final de dicho módulo ya que estarán alineadas las palabras de inicio y fin de cualquier estructura.

Los procesos son especialmente usados para declarar tareas de carácter funcional, y son necesarios para estructuras de control secuenciales como las que veremos a continuación.

A) IF-THEN-ELSE

Una de las estructuras más básicas y además una de las de mayor uso, es también una de las más populares en el ámbito de la programación estructurada, se trata de la declaración Si-Entonces-De lo contrario. Esta estructura decide entre dos posibles opciones; como su nombre lo indica si la decisión es correcta entonces tomará la primera opción, de lo contrario no tendrá alternativa y tomará la segunda.

Esta es una de las declaraciones secuenciales, ya que lleva un orden desde el inicio de la estructura, sirve para seleccionar una condición o condiciones basadas en que el resultado de una decisión sea falso o verdadero. Su sintaxis es de la siguiente forma.

Declaración de una estructura IF-THEN-ELSE

```
IF <condición> THEN
        Realiza operación 1
ELSE
        Realiza operación 2
END IF;
```

La labor de esta estructura es muy simple, la misma arquitectura va guiando el funcionamiento, a continuación se describe un ejemplo en un fragmento de código para hacerlo más explícito.

```
-- Declaración de una estructura IF-THEN-ELSE

IF (Entrada_1 = Entrada_2) THEN
     Igual<='1';
ELSE
     Igual<='0';
END IF;
```

En el código anterior se muestra una condición con dos variables (Entrada_1 y Entrada_2) las cuales deben ser iguales para que el objeto llamado "Igual" pueda tomar el valor de uno, de lo contrario tomará el valor de cero. Esta estructura de decisión debe ir dentro de un proceso.

B) ELSIF

Cuando se necesita una estructura condicionada del tipo IF pero se tienen más de dos condiciones a evaluar, VHDL cuenta con una estructura de control que cumple con estos requisitos y su sintaxis es muy parecida a la descripción anterior.

En la arquitectura ELSIF cada condición es una expresión boleana. Hasta que se encuentre una condición verdadera la operación seguida de esta condición será realizada, de lo contrario buscará en orden sentencia a sentencia hasta encontrar dicha condición.

Declaración de una estructura ELSIF

```
IF <condición> THEN
        Realiza operación 1
ELSIF <condición> THEN
        Realiza operación 2
ELSE
        Realiza condición n
END IF;
```

Esta estructura puede tener tantas condiciones como se desee, pero lo más conveniente es que sean pocas ya que para operaciones de múltiples decisiones el lenguaje tiene otras estructuras especializadas para esos casos las cuales gastarán menos recursos al momento de operar.

```
-- Declaración de una estructura ELSIF

IF (Entrada_1 > Entrada_2) THEN
     Signal_A<='1';
ELSIF (Entrada_1 < Entrada_2) THEN
     Signal_C<='1';
ELSE
     Signal_B<='1'
END IF;
```

En este fragmento de código se puede observar que hay tres condiciones asignadas a diferentes señales, las cuales nos indicarán el tipo de comparación que existe entre las dos variables de entrada, si la primera es mayor a la segunda la primer señal (llamada A) se encenderá, en caso contrario la última señal (llamada C) tomará el valor de uno, si no sucede nada de lo anterior, entonces las entradas serían iguales, la señal llamada B será la indicada para prender.

Esta estructura es frecuentemente usada cuando se necesita una arquitectura de decisiones rápida para más de dos condiciones, al ser secuencial también debe estar declarada dentro de un proceso.

C) WHEN-ELSE

Esta declaración se utiliza para dar valores a una señal mediante una condición determinada y ejecutada dentro de ella. Puede tener tantas condiciones a evaluar como sea, aunque esta arquitectura esta diseñada como un seleccionador de señales, es decir, asignará un valor a un resultado o resultados en específico cuando se cumpla una condición y descalificara a las demás opciones.

Aunque podría parecer en estructura muy similar al anterior, la diferencia clara de éste consiste en que la condición final descalificará a la mayoría de las opciones. Su sintaxis es la siguiente:

Declaración de una estructura WHEN-ELSE

```
Signal_X <= "valor" WHEN (condición) ELSE
            ["valor" WHEN (condición) ELSE]
            "valor"
```

Con un ejemplo se dará mejor claridad a lo citado anteriormente, a continuación se muestra la tabla 2.4 que corresponde a las salidas de una compuerta OR exclusiva, como se expone, dicha tabla contiene solo dos variables de entrada, en este caso Entrada A y Entrada B, esta estructura se puede utilizar con funciones más elaboradas pero utilizaremos este ejemplo por ser sencillo y práctico.

Entrada A	Entrada B	Salida A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 2.4 Tabla de verdad para la compuerta OR exclusiva (XOR)

El código de solución para esta compuerta utilizando la estructura WHEN-ELSE sería la siguiente.

```
-- Declaración de una estructura WHEN-ELSE

 Salida_XOR <= '1' WHEN (a='0' and b='1') ELSE
               '1' WHEN (a='1' and b='0') ELSE
               '0';
```

En esta ocasión el ejemplo podría resultar ilógico ya que como ya se aprendió en el apartado 2.2.4.1 las compuertas se pueden representar fácilmente con un operador lógico, pero el objetivo es mostrar la arquitectura en este fragmento de código solo para mostrar su funcionamiento.

D) WITH-SELECT-WHEN

Esta estructura es muy similar a la anterior pero tiene características propias, funciona principalmente para asignar un valor a una señal teniendo como referencia el valor de otra señal previamente seleccionada.

La sintaxis de la arquitectura se muestra a continuación:

Declaración de una estructura WITH-SELECT-WHEN

```
WITH Signal_X SELECT
        Signal_Y <= "valor_asignado" WHEN "valor_seleccionado"
                    ["valor_asignado" WHEN "valor_seleccionado"]
                    "valor_asignado" WHEN OTHERS;
```

La arquitectura pareciera un poco más compleja que las otras, pero no es tan complicada, su función consta de seleccionar una señal (Signal_X) y de acuerdo al valor que contenga esa señal en ese momento se le asignará un valor a otra señal (Signal_Y).

Utilizaremos ahora la tabla de verdad de la OR exclusiva negada para dar un ejemplo justo como lo hicimos anteriormente con la intención de que la arquitectura sea fácilmente comprendida por los lectores.

Entrada A0	Entrada A1	Salida $\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

Tabla 2.5 Tabla de verdad para la compuerta OR exclusiva negada (XNOR)

Teniendo en cuenta que ambas entradas fueran parte de un arreglo de dos bits, el código de solución para esta compuerta utilizando la estructura WITH-SELECT-WHEN sería la siguiente.

```
-- Declaración de una estructura WITH-SELECT-WHEN

 WITH Vector_A SELECT
          F <= '1' WHEN "00",
               '1' WHEN "11",
               '0' WHEN OTHERS;
```

Ahora se puede observar claramente qué variable está siendo asignada con respecto al valor de otra variable, la señal F tomará el valor de uno cuando ambos elementos en la señal "Vector_A" sean iguales y valdrá cero cuando sean

diferentes. La palabra reservada “others” incluye cualquier combinación que no esté contemplada y pudiera presentarse

E) CASE-IS

Esta estructura ejecuta una de varias declaraciones basándose en el valor de una expresión sencilla. La expresión debe evaluar a un número entero, o un arreglo de tipo bit_vector.[59] La arquitectura CASE evalúa la expresión y compara el valor con cada una de sus opciones. Cuando la expresión que corresponde a la opción concuerda, se ejecutarán las declaraciones siguientes. Esta estructura contiene dos restricciones lógicas, las cuales son:

- No deben repetirse sus opciones y cada opción debe tener una sola sentencia.
- La opción “when others” es opcional en la sintaxis, pero si esta no está presente, todos los posibles valores de la variable deben de ser cubiertos por las opciones.

La sintaxis de esta estructura es como se muestra en seguida:

Declaración de una estructura CASE-IS-WHEN

```
CASE Signal_X IS
        WHEN <opción> => (expresión de asignación);
        WHEN <opción> => (expresión de asignación);
        [WHEN OTHERS => (expresión de asignación);]
END CASE;
```

Supongamos que necesitamos un código que funcione para evaluar calificaciones, donde solo aprobarán alumnos con más de 71 puntos, así cada decena arrojará una señal dependiendo la cantidad de la calificación.

```
-- Declaración de una estructura CASE-IS-WHEN

 CASE Calif IS
         WHEN 71 to 80 => C <= ‘1’;
         WHEN 81 to 90 => B <= ‘1’;
         WHEN 91 to 100=> A <= ‘1’;
         WHEN OTHERS => F <= ‘1’;
 END CASE;
```

[59] Jan Van der Spiegel. VHDL Tutorial. Op. Cit.

En este caso la variable "Calif" contendrá un número entero que será evaluado y dependiendo el valor de la misma las demás variables se encenderán. Esta estructura necesita de un proceso para su funcionamiento.

F) LOOP

Una estructura de tipo LOOP se utiliza para ejecutar en varias veces una secuencia de sentencias, esta se repetirá hasta que encuentre una condición determinada, inclusive podría no llevar sentencia de salida y seguir iterando si es que el proyecto así lo requiere, por lo antes mencionado es que existen algunas variantes de esta estructura, la sintaxis para la arquitectura LOOP es la siguiente:

Declaración de una estructura LOOP

```
[Etiqueta_del_Loop:] [WHILE <condicion> | FOR <condicion>] LOOP
        --Zona de declaraciones
        [NEXT [Etiqueta] WHEN <condicion>];
        [EXIT [Etiqueta] WHEN <condicion>];
END LOOP [Etiqueta_del_Loop];
```

Las etiquetas son opcionales pero son útiles para saber de qué se trata la estructura y cuando son necesarias las sentencias NEXT y EXIT con el uso de su etiqueta se ayuda a su identificación. Estas declaraciones solamente se pueden utilizar dentro de una estructura loop.[60]

- NEXT: Esta declaración salta la ejecución a la siguiente iteración dentro de una estructura loop, obliga a la evaluación de un nuevo índice y procede a ejecutarlo.
- EXIT: Esta sentencia salta el resto de las declaraciones terminando la estructura completamente y continúa ejecutando la instrucción siguiente después de la estructura.

[60] Las definiciones y ejemplos son basados de: VHDL: Sintaxis y ejemplos http://usuarios.multimania.es/israelsu/unimayab/Arquitectura%20de%20Computadoras/Arq%20computadoras_06.pdf

- WHEN: Esta palabra es opcional, pero ayuda para ejecutar cualquiera de las dos sentencias (next y exit) cuando la condición que se esté evaluando sea verdadera.

Las variantes que concede esta estructura son muy parecidas aunque obviamente tienen características propias, a continuación se exponen ejemplos de las tres formas de la estructura para que se pueda tener mejor conocimiento de ellas.

```
-- Ej 1 LOOP Básico
i:=0;
Contador_8: LOOP
BUS(i)<='0';
i:=i+1;
EXIT WHEN i='8';
END LOOP Contador_8;
```

```
-- Ej 2 FOR-LOOP
FOR i IN 0 TO 7 LOOP
  NEXT WHEN i='6';
  BUS(i)<='0';
 END LOOP;
```

```
-- Ej 3 WHILE-LOOP
i:=0;
WHILE i<8 LOOP
  BUS(i)<='0';
  i:=i+1;
END LOOP;
```

El ejemplo sostiene la misma tarea para las tres variantes de estructura, se tiene una señal en forma de vector de ocho bits llamada "BUS" a la cual cada uno de sus elementos se irá asignando el valor de cero por cada ciclo.

- La estructura de LOOP básica es un esquema de iteración continuo, en el primer ejemplo se puede ver que a la arquitectura le pusimos la etiqueta de "Contador_8" y que esta tiene un fin cuando se cumpla la condición de la sentencia EXIT.
- La variante FOR_LOOP generalmente utiliza un rango de la iteración de un número entero que determine las vueltas del ciclo, de igual manera se repetirá ocho veces, recordemos que para asignar un rango a un número entero en VHDL se utilizan las palabras reservadas ***"downto"*** y ***"to"*** respectivamente. En este ejemplo también vemos que se encuentra la sentencia NEXT que obliga a la estructura a saltar al siguiente índice cuando la variable **i** tiene el valor de seis
- La estructura WHILE-LOOP evalúa una condición boleana que se da al inicio de la iteración. Mientras la condición sea verdadera, el ciclo se repite, de lo contrario la estructura obliga a hacer un salto y la ejecución se detiene

Las estructuras de control que aquí se han presentado pueden tener similitudes entre ellas pero si es posible identificar sus características y tener la visión de dónde es más apropiado usar cada una de ellas, serán de gran utilidad, finalmente esa es la terea de un buen diseñador.

2.4 Entidad y arquitectura en un modelo.

Ahora retomaremos algo de los temas anteriores y ya que se tiene la noción de trabajar con los paquetes, la entidad y la arquitectura, recordando que estos dos últimos módulos son imprescindibles y el conjunto de estas se llama modelo, se puede hacer un ejemplo más completo conjuntando enteramente todas las estructuras que ya se conocen.

La idea de juntar los fragmentos de códigos que aquí se han presentado es ejemplificar de inicio a fin un programa en VHDL, en el siguiente tema se describirá la manera en que se divide la lógica de programación y se mencionarán las características que tiene cada uno de estos tipos.

2.4.1 Instrucciones concurrentes y secuenciales

En el diseño lógico de VHDL existen dos maneras de abordar la lógica estructural, estas son:

- Lógica Combinacional: Para diseñar un circuito con este tipo de instrucción el valor de la salida dependerá completamente de la operación que desarrolle el programa con las entradas correspondientes y el resultado se dará en seguida de que se determine dicha operación.
- Lógica Secuencial: Este tipo de diseño está formado por un circuito combinacional, pero a la salida de este se habrá de almacenar en una memoria que guardará el resultado de una manera temporal, hasta que un pulso de reloj active la instrucción al dispositivo de arrojar el resultado.

La opción de elegir entre uno u otro dependerá de lo que el diseñador requiera

2.4.1.1 Modelo en instrucciones concurrentes

A continuación se muestra un ejemplo de lógica combinacional conjuntando un modelo completo, el cual muestra un multiplexor de cuatro entradas a elegir con dos líneas de selección:

```
-- Descripción de un multiplexor 2 x 4

library ieee;
use ieee.std_logic_1164;
ENTITY Multiplexor IS
     PORT (a, b, c, d: in std_logic_vector (3 downto 0);
          seleccion: in std_logic_vector (1 downto 0);
          salida: out std_logic_vector (3 downto 0);
END Multiplexor;
ARCHITECTURE Arq_mux OF Multiplexor IS
     WITH seleccion SELECT
     salida <= a WHEN “00”,
               b WHEN “01”,
               c WHEN “10”,
               d WHEN OTHERS;
END Arq_Mux;
```

Como se observa en el ejemplo anterior se tienen una señal de entrada llamada “selección” la cual es un arreglo de dos bits, ésta seleccionará el valor de una de la señales de entrada y lo filtrará para que tome el valor de la salida.

2.4.1.2 Modelo en instrucciones secuénciales

Estos modelos son constantemente ocupados para que un dispositivo funcione con un pulso que retroalimente el circuito y pueda automatizarlo, en VHDL hay una señal que está reservada para este tipo de casos, y para su correcto funcionamiento se tiene que declarar en la entidad como entrada y con el nombre de ***“clk”***. Las operaciones de un programa con lógica secuencial síncrona se basaran en la frecuencia de los pulsos dados en esta señal.

Atributos de señales

Las señales se pueden apoyar en atributos predefinidos que el lenguaje VHDL tiene para ellas, la señal de “clk” constantemente utiliza estos atributos para

automatizar su funcionamiento en un código. Algunos de estos atributos se muestran en la tabla 2.6

Atributo	Función
Signal_X'event	Devuelve un valor boleano "verdadero" si ocurre un cambio de valor en la señal, de lo contrario devuelve un valor "falso"
Signal_X'active	Regresa el valor boleano de "verdadero" si ha habido una asignación en la señal, de lo contrario devuelve un valor "falso"
Signal_X'transaction	Devuelve una señal del tipo bit que cambiará de señal (de 0 a 1 ó de 1 a 0) cada vez que hay una asignación en la señal.
Signal_X'last_event	Regresa el intervalo de tiempo desde que el último evento en la señal ocurrió
Signal_X'last_value	Da el valor de la señal antes de que el acontecimiento pasado ocurriera en la señal
Signal_X'stable**(T)**	Devuelve un valor boleano "verdadero" si ningún acontecimiento ha ocurrido en la señal durante el intervalo T, de lo contrario devuelve un "falso". El intervalo de tiempo T es opcional, si no se pone tendrá un valor de cero

Tabla 2.6 Sintaxis y funcionamiento de los atributos de las señales

Al consultar bibliografía para hacer este trabajo me encontré frecuentemente con la sentencia:

```
If (CLOCK'event and CLOCK='1') then...
```

Sin duda algunos lectores la reconocerán e incluso algunos hemos usado esta sentencia sin que el autor nos explicara que era un atributo de señal y que existen algunas otras que podemos utilizar con otros fines, esta expresión comprueba si ha llegado un flanco de subida de reloj. Por ejemplo para descubrir cuánto ha pasado el tiempo desde el flanco de reloj pasado, uno puede utilizar el atributo siguiente:[61]

```
CLOCK'last_event...
```

61 La tabla de atributos y los ejemplos fueron basados de. VHDL Tutorial: http://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html#_Toc526061352

A continuación se muestra un ejemplo de lógica secuencial conjuntando un modelo completo, el cual muestra un flip-flop tipo D, este dispositivo contiene dos entradas, una de ellas es un pulso de reloj, y una salida que tomará el valor de la entrada cuando el flanco de subida del reloj se presente.

```
-- Descripcion de un flip-flop tipo D

library ieee;
use ieee.std_logic_1164;
ENTITY FFtipoD IS
     PORT (D, Clk: in std_logic;
          Salida_Q: out std_logic;
END FFtipoD;
ARCHITECTURE Arq_ff OF FFtipoD IS
     Flanco_reloj: PROCESS (Clk)
     BEGIN
          IF (clk'event and clk='1') THEN
          Salida_Q <= D;
          END IF;
     END PROCESS Flanco_reloj;
END Arq_ff;
```

Como se muestra, la implementación del modelo para crear la lógica de un flip-flop tipo D está hecha con un proceso que contiene una condición de reloj, cada vez que el reloj sostenga un flanco de subida el valor que contenga la señal de entrada D se le asignará a la señal de salida.

Declaraciones de espera

Las condiciones de flanco de reloj también pueden ser controladas mediante otras declaraciones, existe una sentencia muy funcional la cual se llama declaración de espera; La declaración de la espera parará un proceso hasta que ocurra un acontecimiento declarado en la sentencia de espera.[62] Las sintaxis de estas sentencias son las siguientes.

- **Wait until** <condición>**;**
- **Wait for** <expresión de tiempo>**;**
- **Wait on** <signal>**;**
- **Wait;**

62 Ibidem.

La expresión más utilizada es la sentencia “wait until”, ya que esta depende de una condición. La condición en esta declaración debe ser verdadera para que el proceso reasuma sus funciones. Algunos ejemplos se muestran en seguida.

```
WAIT UNTIL CLK='1';
WAIT UNTIL CLK='0';
WAIT UNTIL CLK'event  and  CLK='1';
WAIT UNTIL not CLK'stable and CLK='1';
```

En el primer ejemplo la sentencia esperará hasta que ocurra un flanco de reloj positivo, mientras que el segundo ejemplo, la expresión esperará hasta que llegue un flanco de reloj negativo. Los dos ejemplos siguientes tienen el mismo funcionamiento, estarán a la espera de un flanco de reloj positivo para activarse realizar las sentencias siguientes.

Asignación de señales a terminales del CPLD

Por último, dos atributos que darán gran flexibilidad y serán de utilidad para la tarjeta, son dos instrucciones de lenguaje que deben ser implementadas desde el código dentro de la entidad. Estos atributos imponen la asignación de terminales físicas del dispositivo a las señales creadas en la entidad, ambas instrucciones serán de gran utilidad para reservar o asignar terminales del CPLD cuando así se requiera, su sintaxis, características y ejemplos se muestra a continuación.

El primero de ellos es un atributo llamado “pin_avoid”, con esta instrucción el código no podrá asignar ninguna señal de manera segura a el número de terminal específico que el programador determine. Su sintaxis es la siguente.

Asignación de terminales físicas a señales con “pin_avoid”

```
ENTITY nombre_de_la_entidad IS
        --Declaración de señales o puertos en la entidad
END nombre_de_la_entidad;
ATTRIBUTE pin_avoid OF nombre_de_la_entidad: ENTITY  IS
        "#_terminal  [#_terminal  #_terminal …]";
```

Generalmente es utilizado para excluir alguna terminal que el software pueda tomar como entrada/salida pero físicamente no está habilitada para usarse.

```
--Asignación de terminales utilizando pin_avoid

ENTITY Compuertas IS
     PORT(A, B, C, D: in std_logic;
          F, G: out std_logic);
END Compuertas;
      ATTRIBUTE pin_avoid OF Compuertas: ENTITY IS;
        "14 35 51 54 83";
```

En el fragmento de código anterior se muestra un ejemplo de la asignación de este tipo de atributo, en este caso específico el software tiene la instrucción de no asignar ninguna terminal de entrada o salida a las terminales 14, 35, 51, 54 y 83 ya que estas terminales son las encargadas de ingresar la lógica programable al CPLD 7C373i.

Existe otro atributo que asigna terminal a terminal las señales creadas en la entidad, así el usuario tendrá entera disposición de asignar las señales determinadas a las terminales físicas que se requiera usar en el CPLD en que se está programando. La sintaxis correcta de este tipo de atributo se describe a continuación.

Asignación de terminales físicas a señales con "pin_numbers"

```
ENTITY nombre_de_la_entidad IS
            --Declaración de señales en la entidad
END nombre_de_la_entidad;
ATTRIBUTE pin_numbers OF nombre_de_la_entidad: ENTITY IS
            "Signal_X:#_terminal  Signal_Y:#_terminal  Signal_Z:#_terminal";
```

El siguiente fragmento de código muestra un ejemplo de cómo es utilizado este atributo para seleccionar las señales creadas y asignarlas a algunas terminales específicas elegidas en el CPLD a programar.

```
--Asignación de terminales en una entidad

ENTITY Compuertas IS
     PORT(A, B, C, D: in std_logic;
          F, G, H, I: out std_logic);
END Compuertas;
     ATTRIBUTE pin_numbers OF Compuertas: ENTITY IS;
       "A:2 B:3 C:4 D:5" &
       "F:22 G:23 H:75 I:76";
```

Estos tipos de atributos por lo general no son utilizados ya que VHDL asigna automáticamente las señales a terminales predeterminadas del dispositivo

a programar, pero en este caso puede ser de mucha utilidad ya que en ocasiones se trabajará con terminales específicas, recordando que la tarjeta que se propone crear contiene una etapa de prueba integrada.

El lenguaje VHDL es un estándar pero el software fue retomado por las compañías desarrolladoras y estas le han agregado sus propias características, en este caso el dispositivo con el cual se trabajará (el CPLD CY7C373i-66j) es de la compañía Cypress Semiconductor y el software que es manejado en esta compañía se llama Warp, en el siguiente tema se pueden observar algunas características.

2.5 Ambiente de trabajo

En este tema se tratarán las generalidades del software proporcionado por Cypress Semiconductor denominado “Warp”, este es un editor de VHDL para el diseño de circuitos lógicos el cual es muy flexible y funcional, ofrece una interfaz grafica llamada Galaxy, la cual es amigable para que el usuario interactúe con él, también permite el reconocimiento de diferentes familias de dispositivos lógicos programables.

El software también contiene un simulador que describirá todas las terminales con respecto a sus pulsos emitidos y pude hacer un simulacro de su funcionamiento antes de que el proyecto sea grabado físicamente en el hardware.

Los temas siguientes se desglosarán en cuatro puntos que son los que describen el proceso general para implementar un diseño:

1. Escribir el código
2. Compilar el código
3. Simular el funcionamiento del diseño
4. Grabar el CPLD

Dentro de la página de Internet de Cypress Semiconductor podremos encontrar este software para instalarlo dentro de nuestro equipo de cómputo.

2.5.1 Escribir el código

En la interfaz grafica Galaxy las entidades virtuales de trabajo se denominan proyectos, en dichos proyectos se albergarán los códigos que se generen. Para iniciar un proyecto seleccionaremos la opción Project en la barra de menús, dentro de esta buscaremos opción de New y se asignara la ruta donde se guardara dicho proyecto como se muestra en la figura 2.5, generalmente al inicio Warp genera un proyecto propio para comenzar a trabajar.

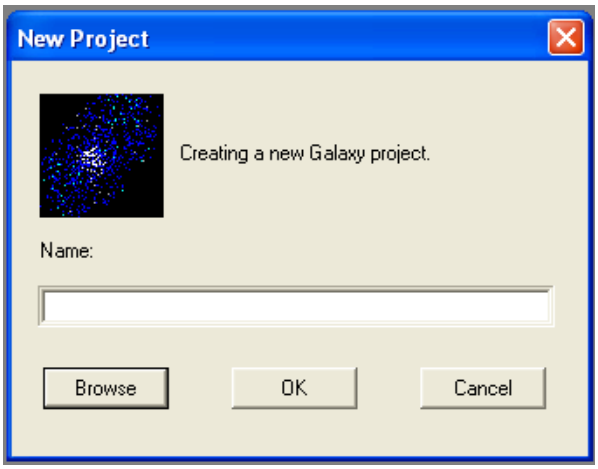


Figura 2.5 Creación de un nuevo proyecto

Si ya se tiene un proyecto creado por el usuario o por el software se tendrá acceso a la ventana de edición del mismo, esta se muestra en la figura 2.6, al inicio daremos clic en el botón New de la sección Edit para abrir una ventana que edite código, en caso de que ya tengamos algunos códigos hechos, estos se podrán visualizar dentro del cuadro blanco a la izquierda de la ventana.

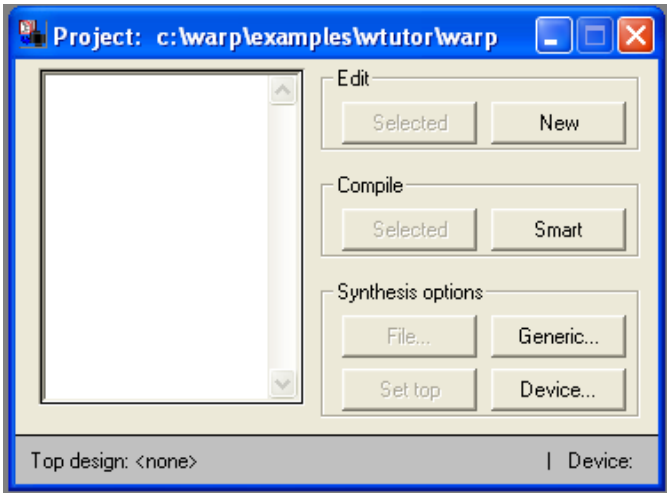


Figura 2.6 Ventana de edición de un proyecto

Al momento de seleccionar un nuevo editor se presentará ante el usuario una ventana blanca que será el área de trabajo, en ella se plasmará el código VHDL precedido de una numeración automática a cada línea.

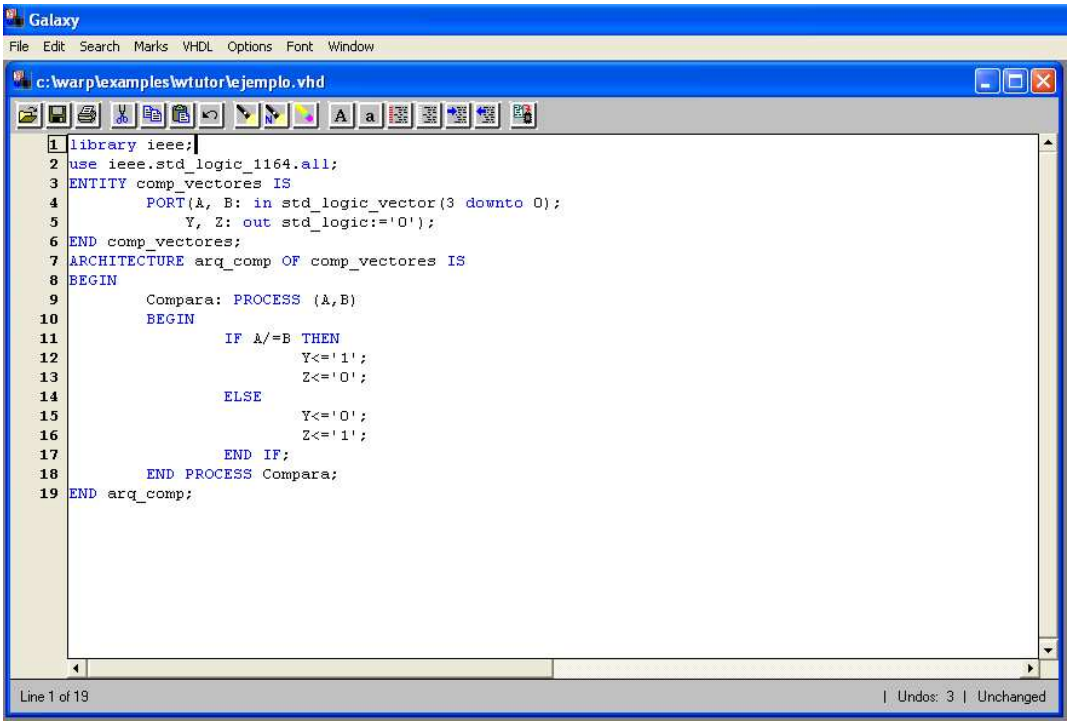


Figura 2.7 Área de trabajo y edición de código

Como se muestra en la figura 2.7 en el área de descripción del código se describe un código con el cual ya se había trabajado, para compilar el programa se puede hacer desde la barra de menús, en la opción VHDL y seguidamente la elección Compile o bien desde la ventana de edición de proyecto, agregando el código a la ventana desde el menú Files y posteriormente la opción Add y después pulsando el botón Selected de la sección Compile. En seguida debe aparecer una ventana emergente que indique si existiera algún error en el código, de ser así debe mencionar en qué línea se encuentra este, de lo contrario el código compiló sin error y en la ventana se debe leer la leyenda "*WARP done*."

2.5.2 Compilar el código

La función de un compilador lógico es procesar y sintetizar el diseño generando lo que se conoce como un mapa de fusibles, este es representado por

un archivo JEDEC el cual es reconocido por el grabador de dispositivos lógicos programables. Para poder generar un archivo de tipo JEDEC el código debe de estar correctamente descrito, ya que si el compilador detecta un error de semántica o sintaxis impedirá su creación.

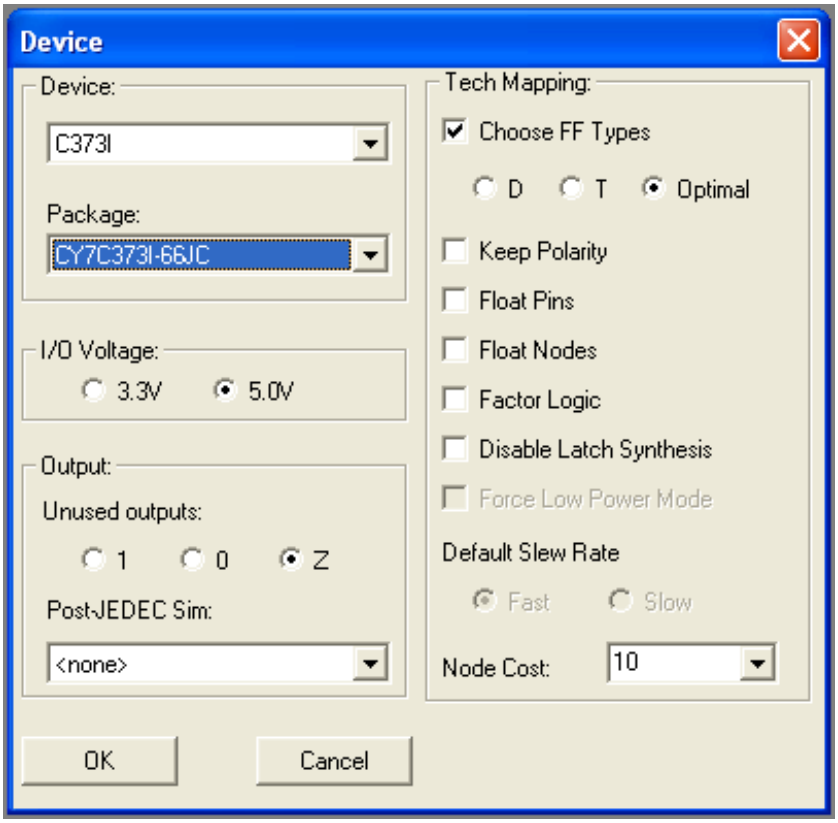


Figura 2.8 Ventana de asignación de dispositivo

Dentro de la ventana de edición del proyecto se encuentra un botón que dice Device, al pulsarlo nos lleva a la ventana mostrada en la figura 2.8, la cual contiene todas las especificaciones del dispositivo en el cual se grabará la información del mapa de fusibles. En nuestro caso se utilizará un CPLD de la familia C373I. En la opción Package se exhibe una lista de los dispositivos disponibles de la familia seleccionada, en ella se encuentra el tipo de encapsulado CY7C373I-66JC. Por último verificamos que la sección de voltaje de Entrada/Salida se encuentre en 5 Volts que es con lo cual funciona el dispositivo elegido.

2.5.2.1 Archivo de reporte.

Una vez que se han establecido todos los parámetros anteriores y que se ha compilado sin error alguno, esta acción trae consigo la generación de un archivo que documenta los detalles del código.

El "*Report file*" es un archivo con extensión **.rpt** el cual presenta información importante del diseño. Para analizar este archivo se elige Info en el menú principal y se selecciona la opción Report file, el archivo que se presenta es muy extenso, pero en él se pueden leer datos básicos como nombre y fecha de creación además contiene el reporte correspondiente a la distribución de las terminales físicas.

2.5.3 Simular el funcionamiento del diseño

El software Warp contiene también un simulador llamado Nova, la cual es una pantalla que permite verificar el comportamiento de un diseño mostrando los pulsos que corresponden a cada terminal en un momento de tiempo específico.

La manera de acceder al simulador Nova es desde la barra de menús principal dando clic en Tools y en seguida seleccionando la opción Nova o bien desde el botón de Inicio buscando el simulador en la lista de programas, cabe mencionar que ningún código se podrá simular si no está previamente compilado.

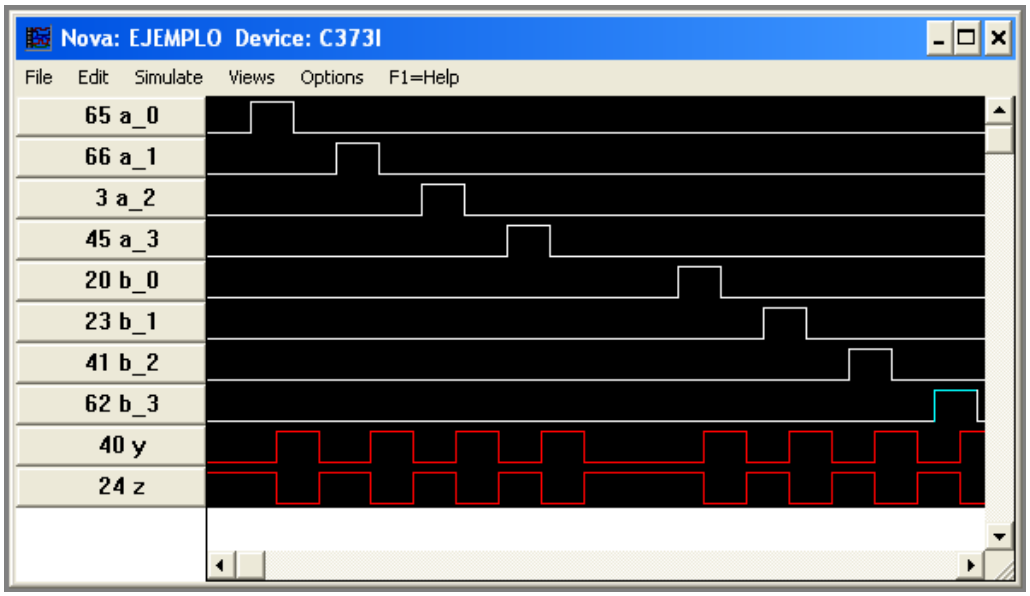


Figura 2.9 Simulador Nova dando valores a archivo ejemplo

Como se puede apreciar en la figura 2.9 el simulador es una pantalla que refleja las señales en un intervalo de tiempo determinado, mencionando además las terminales que les corresponden, ubicadas del lado izquierdo de la pantalla, a las señales de entrada se les puede modificar el valor para observar su comportamiento que se tendrá con respecto a la salida, en este caso el programa ejemplo tiene dos vectores de entrada y dos variables de salida llamadas "Y" y "Z" las cuales están en constante cambio dependiendo el valor de las entradas.

Para asignar valores a las señales de entrada se debe seleccionar la terminal, cuando esto suceda, ésta tomara un color azul y después dentro del menú "*Edit*", se encuentran los diferentes valores que se puede asignar a una señal de entrada.

2.5.4 Grabar el CPLD

Después de simular el proyecto uno de los aspectos más importantes es poder grabarlo en un dispositivo físico. En este caso el dispositivo CPLD seleccionado puede ser grabado, borrado o reprogramado, tiene estas ventajas por una característica llamada *"In System Reprogrammable"* o ISR por sus siglas.

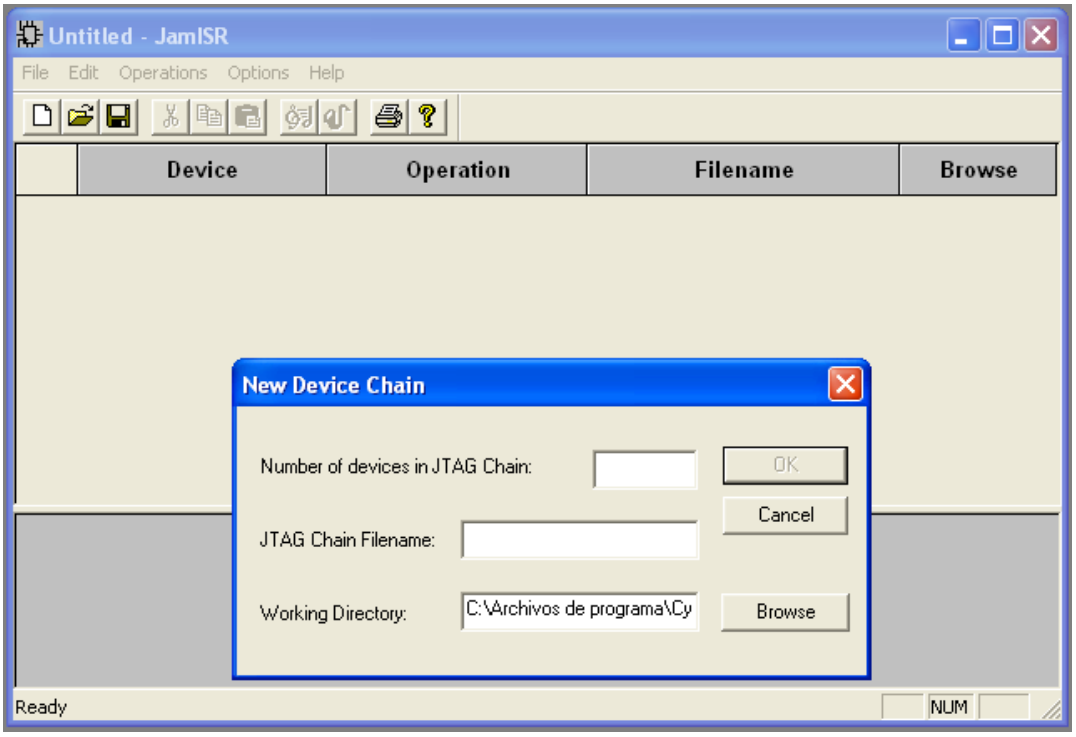


Figura 2.10 Plataforma de programación ISR

La empresa Cypress Semiconductor también cuenta con una plataforma que permite detectar, programar, borrar y reprogramar el circuito, este software se muestra en la figura 2.10 y funciona con un protocolo especial que comunique a la computadora con el dispositivo cumpliendo las reglas de una interfaz de comunicación.

El dispositivo que cumpla con la ventaja de tener este protocolo de comunicación tiene también terminales específicas por las cuales pasará la información de la lógica programable, grabando el mapa de fusibles creado previamente en el dispositivo físico.

Finalmente en el siguiente capítulo se expondrá la creación de una tarjeta que cuente con los beneficios de cumplir con este protocolo, pero también que pueda trabajar de manera secuencial y que contenga una etapa de pruebas para que se puedan comprobar físicamente algunas de las prácticas diseñadas en el lenguaje VHDL.

Nota: Las imágenes presentadas anteriormente son tomadas de las ventanas del software descrito instalado en el equipo.

Capítulo III

Diseño y Desarrollo de la Tarjeta Entrenadora y sus Etapas

Al inicio de este proyecto se planteó llegar al fin concreto de tener una tarjeta entrenadora de CPLDs, en este capítulo se expondrá el diseño de la tarjeta y posteriormente el desarrollo de sus etapas para poder tenerla físicamente. Con ella será más fácil mostrar diversas pruebas y prácticas variadas con la finalidad de que el alumno de Ingeniería en Computación pueda, por medio de códigos creados por ellos mismos en el lenguaje VHDL, tener en un entorno físico un resultado de pulsos en una tarjeta que pueda ser capaz de probarlo, soportar el proceso de grabar, borrar el dispositivo y volver a cargar un programa en el mismo dispositivo sin desprenderlo de la tarjeta, además se pueda tener acceso a todas las terminales de entrada o salida que éste contenga.

Se puede proponer entonces que con esta tarjeta el alumno se podrá familiarizar de una manera sencilla con el lenguaje de modelado y simulación de circuitos virtuales, además logrará el estudio de un dispositivo como lo es el CPLD, también el reconocimiento de los componentes electrónicos y las etapas de la tarjeta, así como su funcionamiento en conjunto dará mucho empuje a los laboratorios de las materias: análisis de circuitos eléctricos, dispositivos eléctricos, diseño lógico, por citar algunas. Ya que de una manera paulatina pero amigable el alumno será introducido en un trabajo práctico con dispositivos reales.

Este capítulo se separará en las unidades funcionales que contiene la tarjeta para la mayor comprensión del lector, a su vez estos bloques se dividen en dos partes, las cuales serán diseño y construcción; el diseño contempla la parte teórica del sistema dando una explicación breve de cada elemento que conforme la tarjeta, la construcción tratará de exponer como están compuestas físicamente, verificando su funcionamiento para lograr que el lector al final integre en un todo cada una de las partes expuestas.

3.1 ¿Qué es una tarjeta de circuitos?

Una tarjeta de circuitos es una pieza fundamental de la tecnología de hoy en día. Es la base para el montaje de un circuito específico y consiste en una placa compuesta de varios elementos sobre la cual el diseñador conectará los dispositivos electrónicos para la elaboración de una tarea determinada.

Una tarjeta de circuito impreso o PCB (del inglés *printed circuit board*), es una placa constituida en su mayor parte de un material no conductor sobre el cual se trazan caminos o pistas de un elemento conductor. La tarjeta de circuito impreso es una superficie que puede ser de plástico o fibra de vidrio utilizada para sostener elementos electrónicos y diseñada para interconectar eléctricamente dichos elementos sobre la misma tarjeta por medio de caminos generalmente de cobre.[63]

Cabe mencionar que antes que la tarjeta tenga estos caminos o pistas, ésta solo se conforma de una lamina completa de cobre sobre la superficie y dichas pistas se crean atacando esta superficie con un acido o corrosivo, generalmente cloruro férrico o ácido clorhídrico mezclado con agua, los cuales permiten quitar el excedente de cobre que no se ocupará en la superficie y dejar en ella solo los caminos de cobre pertinentes. Esto se realiza con un diseño previamente delineado para que los componentes electrónicos no tengan problema alguno en interconectarse de manera efectiva.

La producción de las tarjetas de circuitos y el montaje de los componentes puede programarse para ser automatizada. Esto permite que en ambientes de producción en masa, esta técnica sea más económica y confiable que otras alternativas de montaje,[64] además de ahorrar tiempo al no hacer la conexión eléctrica de manera manual.

---

63 Definición de PBC según planeta electrónico. Sabemos y construimos electrónica http://www.planetaelectronico.com/cursillo/tema4/tema4.1.html

64 Definición y Ventajas de los circuitos impresos, Wikipedia. La enciclopedia libre. http://es.wikipedia.org/wiki/Circuito_impreso

3.2 Diseño y desarrollo de la tarjeta entrenadora de CPLDs.

En el siguiente capítulo se expone el diseño y la construcción de la tarjeta entrenadora y probadora de CPLDs 7C373i-66JC, esta tarjeta tendrá múltiples funciones, ya que podrá reprogramar y probar la lógica de programación de manera secuencial y/o combinacional con el pulso a elegir de dos relojes diferentes para los programas que lo requieran, cabe mencionar que los circuitos para crear los pulsos de reloj también están contemplados para quedar soldados dentro de la misma tarjeta. Probablemente cuando el lector acabe de examinar las características de esta tarjeta tendrá algunas dudas de cómo lograr todo en un mismo espacio, pero como ya se ha mencionado el capítulo se separará en los diversos bloques con que cuenta la tarjeta y el lector se irá familiarizando a cada página con ella, para una mayor comprensión se agregarán algunos diagramas e imágenes que harán más fácil entender la información.

3.2.1 Características de la tarjeta entrenadora.

Algunas de las características más importantes en esta tarjeta son las siguientes:

- Unidad de potencia regulada y con salida externa de la tarjeta de 5 y 12 Volts respectivamente.
- Tecnología de programación ISR.[65]
- Capacidad de borrar o reprogramar el CPLD sin ser retirado.
- Montaje superficial de dispositivos más sensibles con opción a removerlos y reemplazarlos fácilmente.
- La etapa de pulsos de reloj esta compuesta de dos circuitos astables[66] con pulsos de reloj variables.

65 ISR: Capacidad de algunos dispositivos lógicos programables, microcontroladores, y otros dispositivos para ser programados o reprogramados mientras este está instalado en un sistema. Según Wikipedia. La enciclopedia libre.

- La tarjeta contiene dos bloques de pruebas, uno de ellos integrado en la misma tarjeta y otro externo con el cual se pueden manipular algunos otros dispositivos fuera de la tarjeta.
- Las etapas de prueba de la tarjeta pueden programarse para trabajar de manera combinacional o secuencial según convenga al usuario.
- El CPLD a programar puede manipular dispositivos contando con más de 60 terminales de entrada/salida para hacerlo.

3.2.2 Bloques funcionales de la tarjeta.

Para la mejor comprensión y estudio de las unidades funcionales, éstas van a ser separadas de acuerdo a su ocupación dentro de la tarjeta. A continuación se muestra una breve explicación de cada uno de los bloques y como es que están conformados.

- Etapa de potencia: Esta etapa es una de las más importantes para la tarjeta entrenadora ya que su función es primordial, su tarea es alimentar de una adecuada energía eléctrica no solo al CPLD sino a todas las otras etapas dentro de la tarjeta y los dispositivos que conforman dichas etapas.
- Etapa de pulsos de reloj: Consta de dos circuitos independientes que emiten pulsos de reloj de manera constante, pero a su vez tiene como opción ajustar el intervalo de tiempo de dichos pulsos. Van conectados directamente al CPLD para trabajar de manera conjunta cuando así convenga al usuario.
- Etapa de programación: La etapa de programación es también una parte importante dentro de la tarjeta, consta de un puerto el cual recibirá los datos necesarios para poder programar y/o reprogramar el CPLD sin tener que retirarlo de la placa donde se encuentra conectado.

[66] Astable: Dispositivo que genera pulsos de manera constante y alternadamente cambiando de un estado alto a uno bajo sin intervención externa. Según forosdeelectronica.com La comunidad internacional de electrónicos.

- Etapa de pruebas: Esta etapa será la encargada de direccionar todas las terminales de entrada y/o salida que contenga el CPLD, es la única que se dividirá en dos para su estudio. Etapa de pruebas interna; esta fase tiene la cualidad de contar con diversas entradas que pueden ser controladas por el usuario y algunas salidas conectadas a elementos que puedan comprobar la funcionalidad del dispositivo dentro de la tarjeta. Etapa de pruebas externa; esta etapa puede conectar de manera directa las terminales con dispositivos periféricos externos

3.2.2.1 Diseño de la etapa de potencia

La etapa de potencia es determinante para no solo alimentar al CPLD con el cual vamos a trabajar, sino también para suministrar energía a toda la tarjeta en su conjunto. Las demás etapas también dependerán de ella, es por eso que se hace referencia en que es una de las más importantes.

Se plantea que en esta etapa sea posible suministrar energía de forma segura para los dispositivos que se encuentran dentro de la tarjeta de circuitos. En el primer capítulo, (en el tema Función y configuración de terminales) se explicó que el CPLD funcionaba con una energía constante de 5 volts, es por esta razón que el diseño debe contar con un limitador o regulador de voltaje que entregue la alimentación correcta a todas las etapas de la tarjeta.

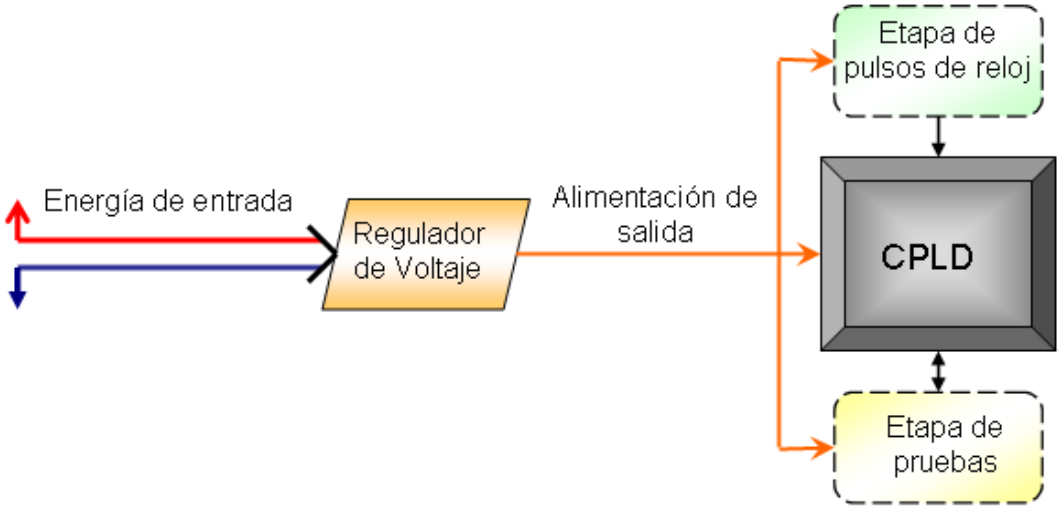


Figura 3.1 Diagrama a bloques de la etapa de potencia.

En la figura 3.1 se expone el funcionamiento básico de la etapa de potencia, que es el siguiente: la energía de entrada es captada y encausada hacia el regulador de voltaje el cual convertirá el suministro de energía eléctrica en una línea de alimentación que conduce el voltaje correcto hacía el CPLD y las distintas etapas.

3.2.2.2 Construcción de la etapa de potencia

Con base en la anterior, lo contenido a continuación es la manera en que está constituida la estructura de la etapa de potencia, además se describirán brevemente los elementos que la conforman así como la función que éstos tienen dentro de la arquitectura.

La etapa de potencia de esta tarjeta, está constituida por un puerto hembra de un convertidor de voltaje externo el cual entrega 12 volts de corriente directa a la entrada de la tarjeta de circuitos. Independientemente de las variaciones de voltaje que pueda entregar esta línea de alimentación, lo que se necesita es obtener una energía regulada de 5 volts constantes a la salida de una fuente de voltaje.

- Regulador de voltaje

Para disminuir un voltaje superior a uno inferior, se contempla un regulador de voltaje para que alimente la tarjeta. El dispositivo LM7805 es un limitador de voltaje que a su vez pretende eliminar el ruido de manera eficiente, donde los últimos dos dígitos de este dispositivo representan el número que corresponde a la tensión que devuelve a la salida[67], este regulador tiene tres terminales, una para recibir una señal de entrada, otra que es conectada a tierra y finalmente una última que entrega un voltaje regulado de salida, éste será el componente principal para suministrar 5 volts a la tarjeta.

[67] Regulador de tensión a 5 volts. tuelectronica.es Electrónica básica. http://www.tuelectronica.es/esquemas/alimentacion/regulador-de-tension-a-5v.html

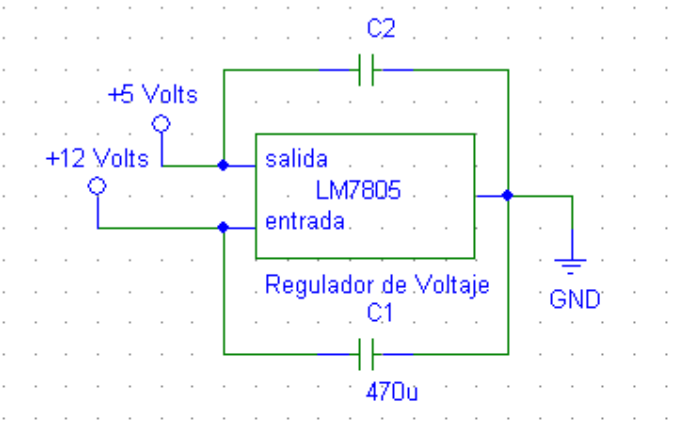


Figura 3.2 Regulador de voltaje a 5 volts con el dispositivo LM7805

La ficha técnica del regulador indica que para su mejor funcionamiento se debe colocar un capacitor en la terminal de entrada del dispositivo si este se encuentra cerca del suministro de potencia eléctrica, eso servirá para crear un buen filtrado de voltaje y eliminar ruido, también se puede colocar opcionalmente un capacitor cerámico en la terminal de salida, tal como se muestra en la figura 3.2, éste no es necesario para estabilizar, pero puede ayudar a transitar el voltaje,[68] aunque en este caso no será necesario.

- Etapa de potencia en su conjunto

La etapa de potencia esta pensada para alimentar toda la tarjeta e incluso algunos dispositivos periféricos externos de manera segura, es por esto que contiene puentes eléctricos, los cuales se enlazan desde la entrada de alimentación y la salida del regulador de voltaje directamente a terminales que se encuentran en un extremo de la tarjeta con opción a poder conectar algún otro dispositivo electrónico. En la figura 3.3 se muestra el limitador conectado siendo las terminales 1, 2 y 3 las salidas externas respectivamente.

---

[68] Manual de usuario del regulador de voltaje LM7805: LM78XX Series 3-Terminal Positive Regulators. http://www.national.com

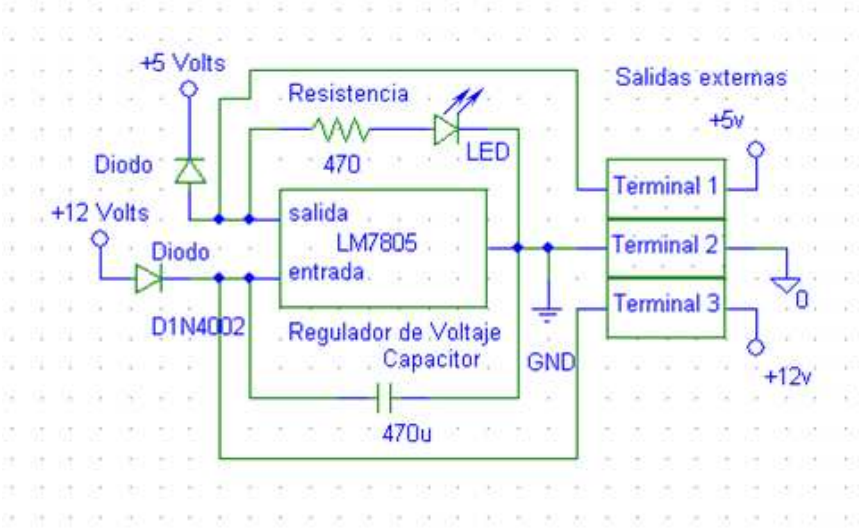


Figura 3.3 Circuito equivalente a la etapa de potencia

En la figura 3.3 se muestra que el regulador de voltaje está conectado justo como ya se expuso anteriormente ya que así lo exige su diagrama técnico, también se observa que en la entrada y la salida de dicho dispositivo se colocan diodos como medida de seguridad para todos los elementos eléctricos de la tarjeta entrenadora y probadora de CPLDs.

- Diodos

Un diodo es componente electrónico de dos terminales que permite el paso de la corriente eléctrica a través de un solo sentido,[69] es decir, en este circuito los diodos están colocados de manera tal que solo funcionarán si la fuente de alimentación está correctamente conectada, en caso contrario no conduce la corriente eléctrica comportándose como un circuito abierto, con la finalidad de no estropear los dispositivos más sensibles en la tarjeta.

Los diodos generalmente hechos de silicio tienen una tensión de umbral de 0.7 Volts, por encima de esta diferencia de potencial el umbral se encontrará roto y dejará pasar la corriente oponiendo muy poca resistencia eléctrica, de lo contrario, mientras el umbral no sea rebasado el diodo no permitirá fluir la

[69] Definición según Wikipedia. La enciclopedia libre
http://es.wikipedia.org/wiki/Diodo

corriente,[70] esta función es similar a comportarse como una compuerta para dejar pasar la tensión y dar con certeza un 1 o un 0 lógico.

Por último, para comprobar en que momento la tarjeta esta habilitada para poder trabajar, se coloca un diodo emisor de luz, también conocido como "LED" por su acrónimo en ingles: *Ligth-Emitting Diode,* el cual es un dispositivo que emite luz cuando se polariza de forma correcta y circula por este una corriente eléctrica, por lo tanto si la luz esta encendida representará que el circuito esta alimentado, en caso contrario la tarjeta estará inhabilitada para trabajar. En el diagrama se observa que este LED va acompañado de una resistencia externa que se debe agregar para limitar la corriente en este tipo de diodo, ya que los diodos por si solos oponen muy poca resistencia eléctrica.[71]

3.2.2.3 Diseño de la etapa de pulsos de reloj.

La etapa de pulsos de reloj pretende ser una herramienta de mucha utilidad para automatizar y/o acelerar las prácticas que así lo requieran, esta etapa se conectará directamente con el CPLD por sus entradas configuradas especialmente para recibir dichos pulsos de reloj

Recordemos que el CPLD con el que estamos trabajando tiene cuatro terminales que pueden configurarse para recibir pulsos de reloj, de este modo se debe seleccionar alguna de estas entradas para poder trabajar con ella de manera secuencial. En el primer capítulo (dentro del tema de Dispositivos Lógicos Programables) se expuso que el modo secuencial permite arrojar el resultado de una terminal de salida siempre y cuando intervenga un pulso de reloj en ella, de esta manera se puede desarrollar un mecanismo que proporcione un intervalo de tiempo constante que convenga al usuario, para que el CPLD despida las salidas correspondientes de manera automática.

70 Los diodos y sus aplicaciones. Monografias.com Tesis, Documentos, Publicaciones y Recursos Educativos: http://www.monografias.com/trabajos-pdf/diodos-aplicaciones/diodos-aplicaciones.pdf

71 M. Morris Mano. Lógica digital y diseño de computadores. pp. 585

Para la creación de los pulsos de reloj se contempla un temporizador, el cual es un dispositivo que puede generar pulsos de manera constante y sea capaz de cambiar de un estado a otro sin intervención externa, es decir, al ser alimentado comenzara su ciclo permaneciendo en un estado por cierto tiempo y luego cambiando al otro estado y permaneciendo en éste una cantidad de tiempo similar al estado anterior, a estos circuitos se les conoce como multivibradores astables o también conocidos como osciladores de carrera libre.[72]

Al tener el CPLD más de una entrada que pueda configurarse para poder recibir los pulsos de reloj, se contempla que la tarjeta no contenga solo un temporizador, y que ellos trabajen de manera independiente para que se puedan aprovechar las terminales del CPLD ya mencionadas. El funcionamiento de esta etapa se ejemplifica de manera resumida en la figura 3.4. La etapa de pulsos de reloj debe ser alimentada por la etapa de potencia, y se planea que pueda contener dos temporizadores funcionando de forma autónoma, los pulsos que se generen en esta etapa serán encausados a las terminales del CPLD que puedan recibirlas para así poder hacer uso de ellos.

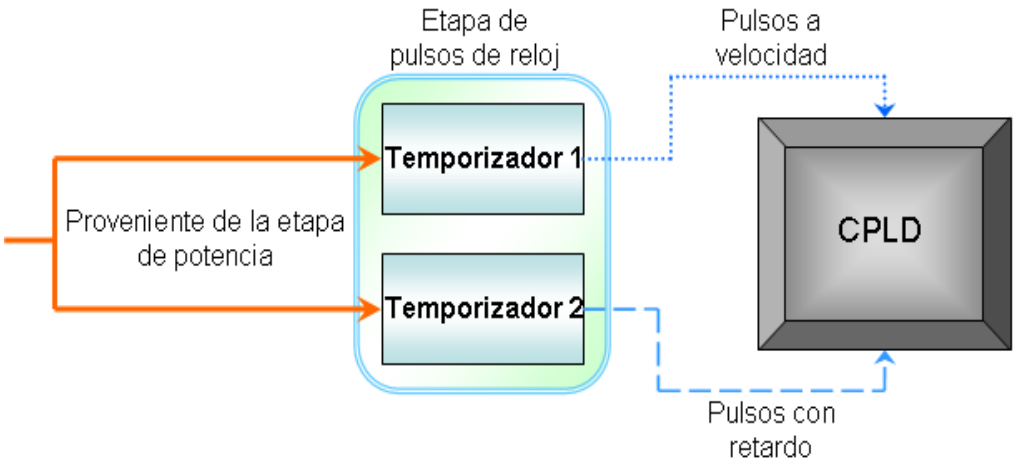


Figura 3.4 Diagrama a bloques de la etapa de pulsos de reloj

[72] Definición multivibrador astable según forosdeelectronica.com La comunidad internacional de electrónicos. http://www.forosdeelectronica.com/f27/diferencias-entre-biestable-astable-monoestable-4437/

Finalmente se plantea que como la etapa de pulsos de reloj contiene dos temporizadores independientes, para un mejor aprovechamiento, los pulsos serán desiguales para cada uno de ellos, además se busca que éstos puedan de alguna manera ser manipulados para entregar intervalos de velocidades diferentes, ya que con esto el rango de pulsos será más amplio.

3.2.2.4 Construcción de la etapa de pulsos de reloj.

Con base a la planeación que se expuso en el diseño de la etapa, en seguida se describe a detalle la manera en que ésta está constituida, además se exponen los diagramas gráficos de los elementos más importantes de la etapa y la función que realizan dentro de ella.

La etapa de potencia será la encargada de suministrar energía a toda la tarjeta, la etapa de pulsos de reloj dependerá de esta energía y comenzará su ciclo una vez que sea alimentada. El elemento primordial para generar pulsos de reloj será el temporizador, el cual es el creador de una señal constante que detonará un pulso alto y un pulso bajo alternadamente y de manera repetida, esta condición es la que le da nombre a la etapa.

- Temporizador

Para el conteo de tiempo la tarjeta contiene el circuito integrado LM555, este será el dispositivo detonador de pulsos de reloj, dicho circuito integrado tiene diferentes maneras de configurarse, pero como ya lo mencionamos, en este caso se busca que al ser alimentado sea capaz de cambiar de un estado a otro sin ser manipulado de manera externa. El LM555 es un dispositivo altamente confiable ya que puede generar periodos de oscilación muy estables,[73] este circuito integrado contiene 8 terminales en total, 2 de éstas serán para conectarse a la etapa de potencia, además una terminal de salida de la cual obtendremos los pulsos resultantes y terminales de funcionamiento, dentro de ellas hay una

[73] Manual de usuario del contador de tiempo LM555/LM555C Timer. National Semiconductor. http://www.national.com

llamada “reset” la cual tiene la cualidad reiniciar la operación si es que el usuario así lo decide, pero en este caso no se requiere que esto suceda y la conectaremos directamente a los 5 Volts para evitar que el dispositivo se reinicie por error y nos envíe una salida a nivel bajo sin alternarse.

La figura 3.5 muestra la manera en que la ficha técnica del dispositivo LM555 asigna para configurar el temporizador y que este opere de forma astable como un oscilador de carrera libre, la etapa de potencia es representada por una fuente llamada “EP” que alimenta el circuito. El valor de dos resistencias externas y un capacitor mostrados en dicha configuración es lo que determinará la duración de los pulsos, es decir, el tiempo de los periodos dependerá de los elementos Ra, Rb y C3 específicamente.[74]

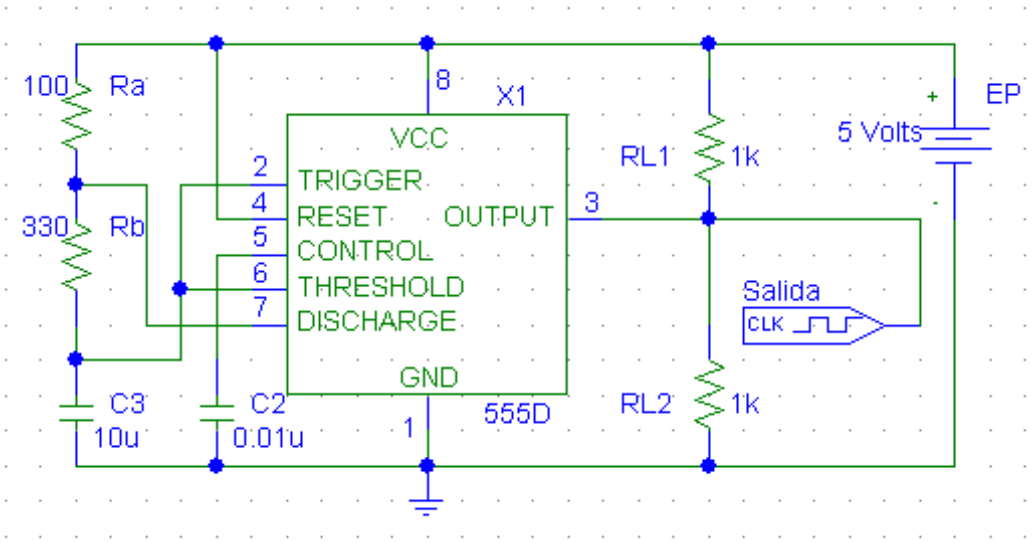


Figura 3.5 Configuración como modo astable del temporizador 555

Cuando la señal de salida permanece por un periodo de tiempo en un nivel elevado a este intervalo de tiempo se le denomina t_1 y será dado aproximadamente por la siguiente fórmula matemática:

$$t_1 = 0.693(Ra + Rb) C3 \text{ [segundos]}$$

[74] Definición de temporizador según Wikipedia. La enciclopedia libre. http://es.wikipedia.org/wiki/Circuito_integrado_555

Por el contrario, cuando la señal de salida permanece por un periodo de tiempo en un nivel bajo a este intervalo de tiempo se le denomina t_2 y este será dado aproximadamente por la siguiente fórmula matemática:

$$t_2 = 0.693 (Rb) C3 \text{ [segundos]}$$

Finalmente la sumatoria de ambos intervalos darán la duración total del periodo, a esta duración se le denominará T y será la constante que se repetirá una y otra vez mientras el dispositivo se encuentre funcionando, su fórmula es:

$$T = 0.693 (Ra + 2Rb) C3 = t_1 + t_2$$

Estas fórmulas serán de gran ayuda para determinar que periodo de tiempo es el que puede ser de mayor utilidad para el desarrollo de alguna práctica.

- Etapa de pulsos de reloj en su conjunto.

En la etapa de pulsos de reloj uno de los dispositivos más importantes es el temporizador, pero también existen otros componentes que hacen que la etapa se complemente, además se debe recordar que la etapa esta conformada por dos circuitos disparadores de pulsos, los cuales son independientes uno del otro, también al igual que la etapa anterior habrá algún elemento que sirva para comprobar que los temporizadores permanezcan funcionando.

Como ya se explicó anteriormente algunos elementos dentro de la tarjeta están diseñados para desarrollar sus funciones desde una plataforma que los soporte y los conecte a la misma tarjeta, con esto se pretende que estos dispositivos puedan ser reemplazados de manera más sencilla si por algún motivo dejaran de funcionar o se requiere removerlos. Los dispositivos LM555 están colocados de esta manera ya que si alguno falla podría ser removido con facilidad, cabe mencionar que lo único que se remplazaría sería dicha pieza por que el circuito seguiría funcionando de la misma manera y a la misma velocidad ya que así esta configurado dentro de nuestra tarjeta.

La figura 3.6 muestra de manera gráfica como es que esta conformada la etapa de pulsos de reloj que consta dentro de la tarjeta, se pueden apreciar que es muy parecido a la figura 3.5 solo que este circuito contiene dos dispositivos y cada uno de ellos tiene a su vez un LED conectado a la salida de ambos dispositivos para verificar su funcionamiento, cuando el resultado que sea arrojado corresponda a un pulso alto el LED emitirá una luz, por el contrario cuando esta se apague corresponderá a un pulso bajo, por lo tanto, en la tarjeta tendremos dos luces parpadeando constantemente a tiempos diferentes, con esto se tiene la certeza de que los temporizadores esta funcionando, asimismo de cuando el valor del pulso es alto y de cuando se convierte en bajo y finalmente también es visible la velocidad a la que transcurre el periodo, esa misma señal será la que se conecte al CPLD por sus entradas establecidas.

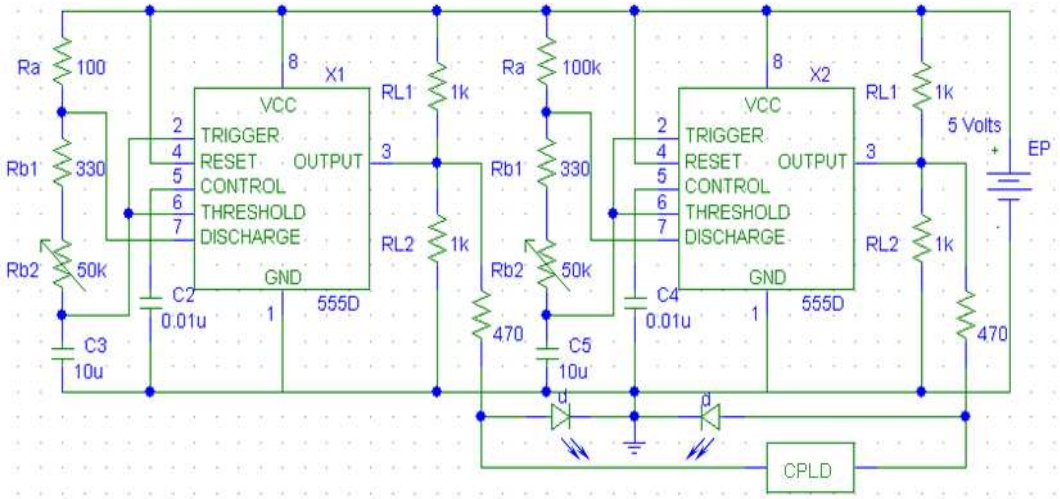


Figura 3.6 Dos circuitos astables, equivalentes a la etapa de pulsos de reloj.

Existe una diferencia también entre ambos contadores de tiempo, uno de ellos tendrá una combinación diferente en el valor de las resistencias eléctricas, recordando que esto hace que los pulsos de ambos circuitos sean desiguales, por lo tanto uno disparará de una manera más rápida que el otro. Además se puede apreciar otra diferencia con respecto a la figura 3.5, ya que Rb es una suma de dos resistencias colocadas en serie, una de ellas es Rb2, y como se muestra en la imagen ésta es una resistencia variable, la cual es comúnmente llamada potenciómetro.

- Potenciómetro

Un potenciómetro es un componente eléctrico similar a una resistencia, con la característica de que opone una resistencia eléctrica de forma variable.[75] Dicha resistencia se puede modificar manualmente con un control gradual que aumenta o disminuye, dependiendo el caso, la diferencia de potencial que fluye entre sus terminales.

Si se observan las fórmulas matemáticas expuestas anteriormente para conocer los periodos de tiempo de los cuales constan los pulsos de reloj que arrojan nuestros circuitos, encontraremos que ambas formulas (tanto para t_1, como para t_2) se encuentran afectadas por Rb, así que teniendo una resistencia variable en este punto se puede modificar por completo la duración del periodo, pero también la duración de los tiempos en que los pulsos se encuentren en un nivel alto y/o en un nivel bajo. En la figura 3.7 se muestra la manera en que un disparador de pulsos cambia aumentando o disminuyendo la resistencia eléctrica con el potenciómetro que se encuentra dentro de la tarjeta. Se puede apreciar que un mismo circuito puede detonar pulsos de una manera completamente distinta si las resistencias dentro de ese arreglo son modificadas.

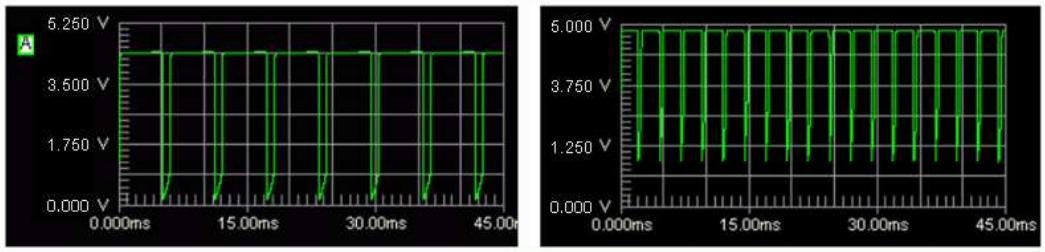


Figura 3.7 Pulsos de un circuito astable modificando su juego de resistencias

En conclusión, dentro de la tarjeta se tiene un rango de pulsos de reloj muy amplio para elegir ya que no solo se puede hacer uso de uno de los dos circuitos astables independientes, sino que también se puede controlar de manera manual algún periodo de tiempo adecuado con el cual se pueda trabajar, ya sea de un contador de tiempo o del otro.

[75] Definición de potenciómetro según Scribd. La biblioteca en línea más grande del mundo. http://es.scribd.com/doc/14802723/potenciometros

3.2.2.5 Diseño de la etapa de programación

Esta etapa pretende dar una ventaja al usuario al buscar programar, borrar o reprogramar el CPLD sin ser retirado de la misma tarjeta, explotando una cualidad dentro de este dispositivo, una tecnología denominada ISR (*In-System Reprogrammable*) la cual nos permite hacerlo.

La tecnología ISR es una característica que algunos de los dispositivos lógicos programables y otros chips contienen para ser programados y/o reprogramados mientras este está instalado en un circuito, en lugar de que el dispositivo sea programado antes de ser conectado en el sistema. Esta definición se explicó dentro del primer capítulo (en el tema 1.3.3 Funciones y Configuración de las Terminales), la principal ventaja de esta característica es que ya no es necesario depender de una etapa de programación antes de montar el dispositivo en el sistema, con esto, ahora se procura integrar la programación y prueba en una sola fase dentro de la misma tarjeta entrenadora. Lo que hace posible aplicar cambios en el diseño del programa que se encuentra dentro del CPLD o bien crear un código completamente nuevo y cargarlo rápidamente, además debido a la alta velocidad de direccionamiento de este dispositivo en ocasiones permite a los usuarios cambiar la lógica existente en los diseños de programación mientras simultáneamente asigna las terminales de salida,[76] es decir, el proceso de programación se carga a gran velocidad y también tiene la ventaja de no mover el chip de su lugar para así comprobar la lógica que se ha insertado en él de inmediato con la etapa de pruebas.

Generalmente los chips que utilizan tecnología ISR tienen un circuito interno para comparar una tensión necesaria para programarlo que es diferente a la tensión del suministro normal, y así habilitar las funciones de las terminales por las cuales se recibirá la lógica programable. Para habilitar la interfaz ISR debe estar activada la terminal de voltaje de programación llamada ISR$_{EN}$, esta

[76] Manual de usuario CPLD CY7C373i. Descripción Funcional. pp 1. http://www.cypress.com

terminal tiene que ser alimentada con un energía de entre 6 a 12 Volts para hacerlo, ya que si no cuenta con ella las terminales de programación se encontrarán deshabilitadas para recibir la lógica del programa, en este caso cuando en el dispositivo la terminal ISREN se encuentre conectada a una tensión de 0 a 5 Volts las terminales por las cuales se recibe la programación solo funcionarán como entrada y salidas de pulsos,[77] recordando que dichas terminales comparten estas dos funciones.

La mayoría de los dispositivos lógicos programables para poder introducir la lógica de programa utilizan un protocolo llamado JTAG (*Joint Test Action Group*) para ISR. La interfaz JTAG es un protocolo especial de comunicaciones entre el dispositivo y el programador, de cuatro o cinco terminales agregadas a un chip. El CPLD a programar, esta configurado para utilizar cuatro terminales que tienen la función de captar estas señales para programar el dispositivo y son aquellas que tienen los nombres: SCLK, SMODE, SDO y SDI respectivamente, además de la terminal ISREN que tiene que estar activada y que ya se explicó con anterioridad.

Cada una de las terminales las cuales son encargadas de implementar la lógica programable al CPLD tienen una función determinada para poder lograr el objetivo de programar y/o reprogramar el dispositivo, estas cuatro terminales (SCLK, SMODE, SDO y SDI) permiten la interacción con el chip creando una puerta de entrada a la que se le denomina TAP[78] (*Test Access Port*, Puerto de Acceso de Pruebas). A continuación se describen cada una de las terminales que conforman este puerto de manera particular y su función dentro del protocolo de programación.

[77] Manual de usuario: Designing with cypress In-System Reprogrammable (ISR) CPLDs for PC cable programming. Dentro de Enables the 4-pins interface pp. 1 http://www.alldatasheet.com

[78] Definición de TAP, según thefreelibrary. Articles and Books. http://www.thefreelibrary.com/Limited+access+tools+to+improve+test+coverage%3a+smaller+test+pads+are+...-a0210847588

Debido a que el protocolo contiene solamente una línea de datos disponible, esta interfaz es necesariamente en serie.[79]

- SDI (*Serial Data Input*, Datos Seriales de Entrada). Durante la programación, esta terminal proporciona los datos de entrada al dispositivo.
- SDO (*Serial Data Output*, Datos Seriales de Salida). Durante la programación, esta terminal es la que conduce los datos de salida del dispositivo.
- SCLK (*Serial Clok*, Terminal de Pulsos de reloj). Esta terminal es la entrada de la señal de pulsos de reloj interna, un dato es cargado por TDI mientras el flanco de reloj es ascendente y otro dato es retirado a través de TDO en el flanco de bajada, esto sucede repetidamente por cada pulso de reloj de la señal SCLK hasta terminar el proceso de programación.
- SMODE (*Select Mode*, Controlador de Modo). Cada flanco de subida de la terminal de pulsos de reloj, el Controlador de Modo también es verificado, dependiendo del estado de SMODE se permite controlar el acceso de las entradas del puerto TAP a los registros correspondientes.

Todas las terminales de la interfaz funcionan en su conjunto con el objetivo de programar el dispositivo, introduciendo en él un código previamente diseñado en una computadora. En la figura 3.8 expone de manera gráfica este proceso.

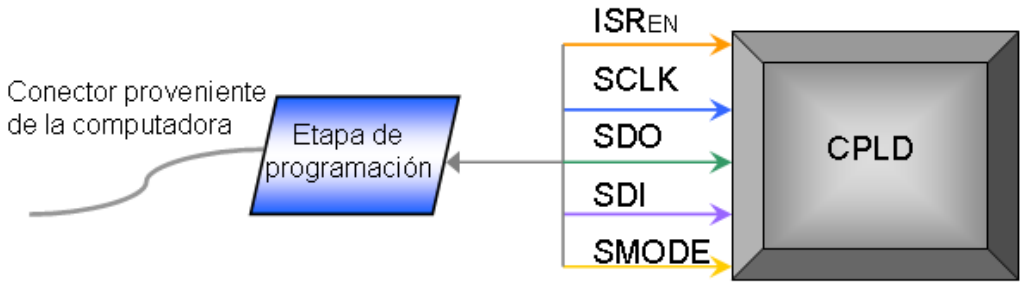


Figura 3.8 Diagrama a bloques de le etapa de programación

[79] Manual de usuario: Designing with FLASH370i for PC Cable Programming. Dentro de ISR Programming Cable pp. 2

En la figura 3.8 también se muestra que la etapa de programación será entonces un puerto que conectará de manera independiente cada uno de los elementos de la interfaz por diversos caminos, encaminándolos a las terminales correspondientes del CPLD.

3.2.2.6 Construcción de la etapa de programación

Cuando se tiene el conocimiento teórico en esta etapa, la construcción se vuelve sencilla puesto que lo que se requiere es conectar las terminales apropiadas al puerto de manera correcta, justo como se expuso anteriormente. A continuación se explicarán los elementos que conforman la etapa de programación así como su función dentro de la misma.

En el tema anterior se expusieron las terminales que serán utilizadas para introducir un código al CPLD por medio de un lenguaje de programación, para que el dispositivo pueda realizar una tarea específica que el programador haya desarrollado. Ahora se expondrá de forma sencilla la manera en que queda conformado el puerto para introducir lógica programable a la tarjeta eficazmente.

- Puerto de programación para la tarjeta entrenadora

El puerto que contiene la tarjeta esta diseñado para un cable plano de 10 conectores acomodados en 2 líneas y 5 conectores para cada una, este puerto a su vez cuenta con pasadores de seguridad para que el cable no se desconecte ni se mueva por error al momento de estar programando. La lógica programable con que se cargará el CPLD será introducida a la tarjeta por un conector hembra vinculando de manera efectiva todas las terminales de programación que se requieren. En la figura 3.9 se muestra la configuración del conector entrante.

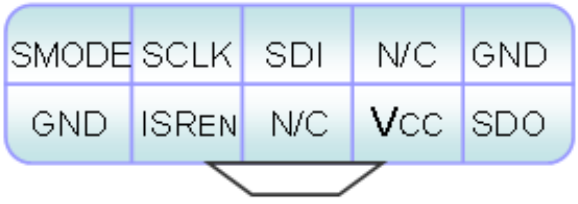


Figura 3.9 Configuración de conector visto desde abajo

Como se puede observar en la imagen anterior el puerto de entrada a la tarjeta va a recibir las cinco terminales configuradas para programar el dispositivo las cuales son ISREN, SCLK, SDO, SDI y SMODE, también hay otros tres conectores dedicados a proporcionar energía adecuada para que el CPLD funcione de manera correcta, dos del ellas son terminales a tierra llamadas GND, la otra denominada Vcc es una terminal la cual arrojará un voltaje de alimentación de 5 Volts aproximadamente, por último la imagen contiene dos terminales llamadas N/C las cuales significan que en estas dos posiciones no habrá una línea de salida o entrada hacia la tarjeta.

- Etapa de programación en su conjunto

Ahora que las características del puerto están dadas la etapa consistirá en enlazar de manera correcta los conectores del puerto con las terminales específicas del CPLD que se encargan de cargar la lógica programable y las que alimentan al dispositivo para que este esté funcionando y así poder programarlo.

En la figura 3.10 se demuestran las conexiones que van desde el puerto de la etapa hacia las terminales correspondientes del CPLD y también se contempla un LED que sirve para comprobar que las líneas de alimentación estén funcionando.

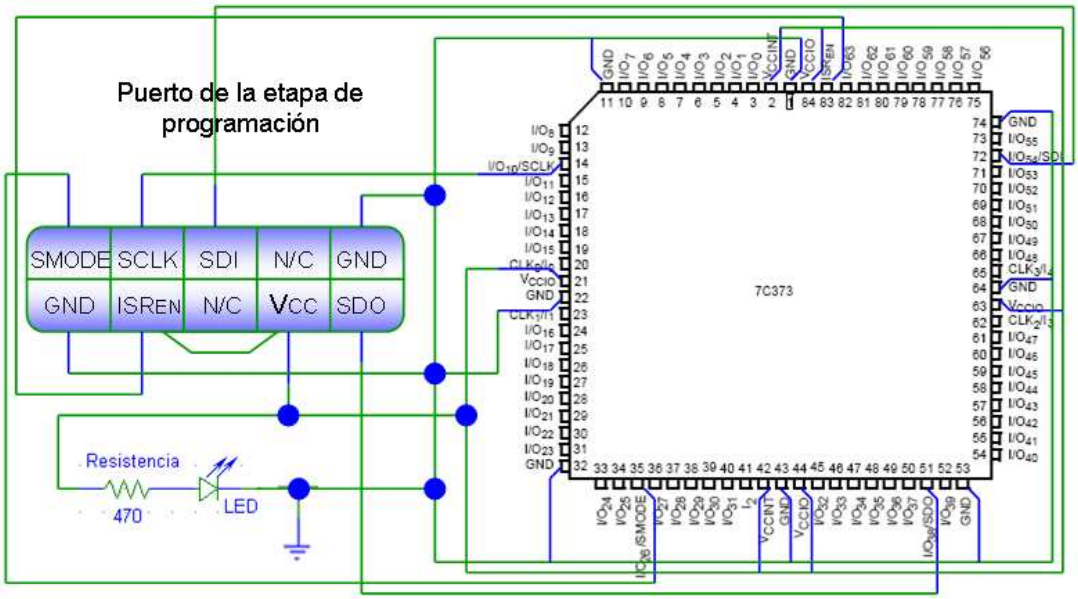


Figura 3.10 Circuito equivalente a la etapa de programación

Como se muestra en la imagen 3.10 los conectores de la etapa de programación van conectadas directamente a las terminales configuradas para recibir la lógica de programación, como se ha explicado estas son; ISREN, SDI, SDO, SMODE y SCLK los cuales tienen el número de terminal 83, 54, 51, 35 y 14 correspondientemente, entendiendo también que el dispositivo esta visto desde arriba y que la terminal inicial se encuentra en el centro de la parte superior, al mismo tiempo los conectores que se encargan de dar energía al chip se encuentran conectadas a las terminales de Voltaje (2, 21, 42, 44, 63 y 84) y a tierra (1, 11, 22, 32, 43, 53, 64 y 74) respectivamente para poder alimentarlo, restando las terminales que no están utilizadas en esta etapa quedan 65 ellas y estarán conectadas a otras etapas como en la de pulsos de reloj, las más de 60 terminales sobrantes serán utilizadas para realizar la etapa de pruebas que se explica a continuación.

3.2.2.7 Diseño de la etapa de pruebas

Lo que se requiere en esta etapa final es que el CPLD le sea funcional al usuario para la realización de una tarea en particular que él mismo haya diseñado, después de que el dispositivo es alimentado, programado y tiene la opción de funcionar de forma secuencial, restaría poder comprobar que el chip realmente funciona exactamente de la manera en que se requiere.

El CPLD tiene una buena capacidad debido a las 64 terminales de entrada/salida que contiene y una terminal especial que esta configurada para solo recibir datos de entrada, la etapa de pruebas tendrá la tarea de ingresar los pulsos de entrada y a su vez buscar que a la salida, los datos recuperados sean correctamente encaminados a las terminales por las cuales se diseñó que fueran expulsados.

Esta es la única etapa que será separada en dos partes para poder explicar de mejor manera como es que está conformada, se dividirán en; etapa de pruebas interna y etapa de pruebas externa. En seguida se exponen las características de cada una de ellas.

- Etapa de pruebas interna

Como se expuso en el tema anterior, gracias a la ventaja de programar y reprogramar el chip en la misma tarjeta sin ser retirado de la etapa de pruebas, ésta está pensada para ser una herramienta que intente comprobar que tanto las entradas, como los pulsos emitidos por el CPLD sean los correctos, teniendo para esto, dispositivos dentro de la tarjeta que ayuden a confirmarlo.

Para esta parte de la tarjeta habrá elementos que busquen dar mayor funcionalidad a la etapa como lo son, poder controlar y manipular los pulsos de entrada y comprobar que los pulsos de salida sean visibles, se contempla que con dispositivos específicos estos elementos se contengan dentro de la tarjeta para que el usuario le sea sencillo y rápido utilizarlos.

- Etapa de pruebas externa.

La diferencia fundamental de la etapa de pruebas interna con esta; es que en esta etapa no habrá elementos que nos ayuden a comprobar los pulsos dentro de la tarjeta, esta etapa será una conexión directa de todas aquellas terminales de entradas o salidas de las cuales podemos hacer uso, pero para ello se deben utilizar dispositivos periféricos externos, como pueden ser; motores, visualizadores, sensores infrarrojos, de temperatura, etcétera.

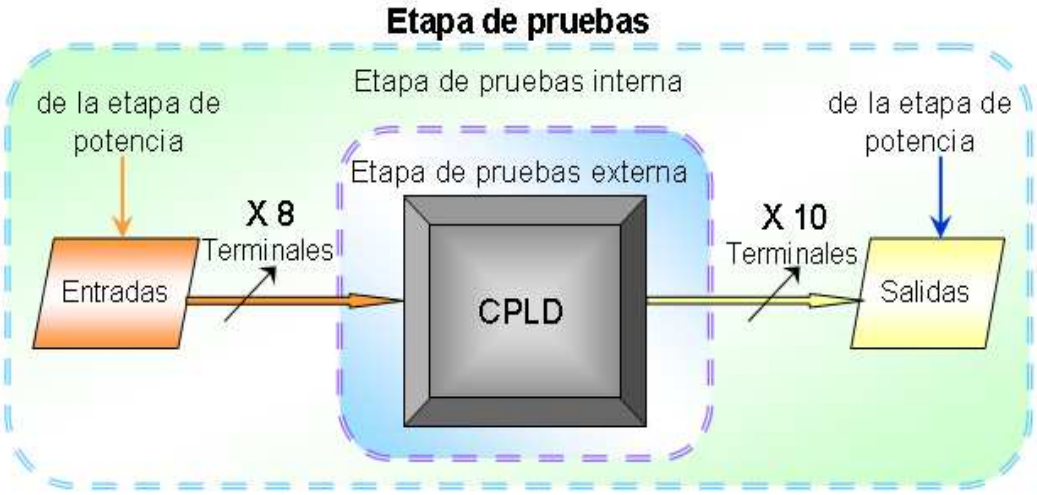


Figura 3.11 Diagrama a bloques de la etapa de pruebas

La figura 3.11 muestra el diagrama a bloques de la etapa de pruebas. Comprendiendo las diferencias de estas etapas se puede apreciar que la de pruebas externas se encuentra alrededor del CPLD ya que las conexiones de las terminales de entradas y/o salidas serán tomadas de manera directa inmediatamente. Por su parte la etapa de pruebas internas contendrá ocho terminales de entrada que pueden ser controladas por el usuario y diez terminales de salida funcionando.

3.2.2.8 Construcción de la etapa de pruebas interna

Con base a la planeación que se expuso en el diseño de la etapa, en seguida se describe la manera en que esta etapa está constituida, además se exponen los diagramas gráficos de los elementos más importantes de la etapa y la función que estos realizan dentro de ella.

Como anteriormente se ha explicado, la etapa contiene algunos dispositivos para poder controlar o manipular los pulsos de entrada y también comprobar que los pulsos de salida sean visibles, para que estos elementos no consuman energía cuando no se estén utilizando se les conectará previamente un interruptor de tensión, que es un aparato ideado para cortar el paso a un circuito eléctrico[80], tanto en el caso de las entradas como en el caso de las salidas este elemento servirá como un seleccionador para saber cuando estos dispositivos estarán habilitados para funcionar y así evitar desperdiciar recursos.

- Interruptores múltiples

La parte de la tarjeta con las terminales de entrada hacia la etapa de pruebas interna está conformada por ocho entradas que apuntan al CPLD, estas serán manipuladas por varios interruptores que se encontrarán abiertos y podrán cerrar el circuito para que los pulsos de entrada encuentren su destino. La figura 3.12 muestra el circuito que contiene un arreglo de interruptores en paralelo para conectar los pulsos de entrada al chip.

[80] Matilde Navarro Castaño, et al. Diccionario enciclopédico Quillet. Séptimo tomo. pp 206

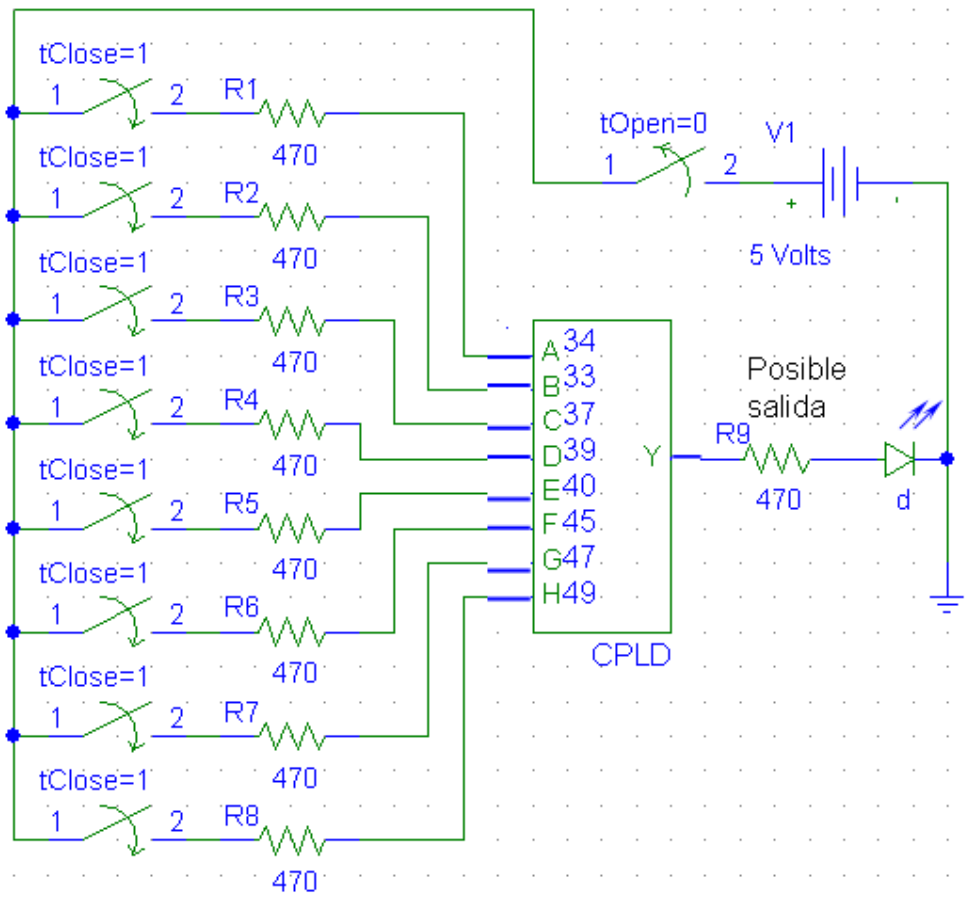


Figura 3.12 Circuito equivalente a las entradas de la etapa de pruebas interna

A este arreglo de múltiples interruptores generalmente se le conoce como DIP Switch (*Dual In-Line Package Switch*, Interruptor en un Paquete de Doble Línea) ya que son interruptores encapsulados en un circuito rectangular[81] que se utiliza comúnmente para personalizar el comportamiento de un circuito. Las terminales que se configuraron como entradas se encuentran señaladas en la imagen.

- Visualizador de 7 segmentos

Por otro lado algunas de las salidas provenientes del CPLD tienen como opción ser conectadas a un visualizador de 7 segmentos, o también llamado *display*, que es un dispositivo compuesto de siete líneas que pueden encender o

[81] Definición de DIP Switch, según electónica2000. Diccionario de electrónica ingles-español. http://www.electronica2000.com/dic_elec/d.htm

apagar individualmente para poder formar la representación de números o letras. La figura 3.13 muestra la manera en que el circuito esta conectado para que las salidas de las terminales lleguen a ser visualizadas.

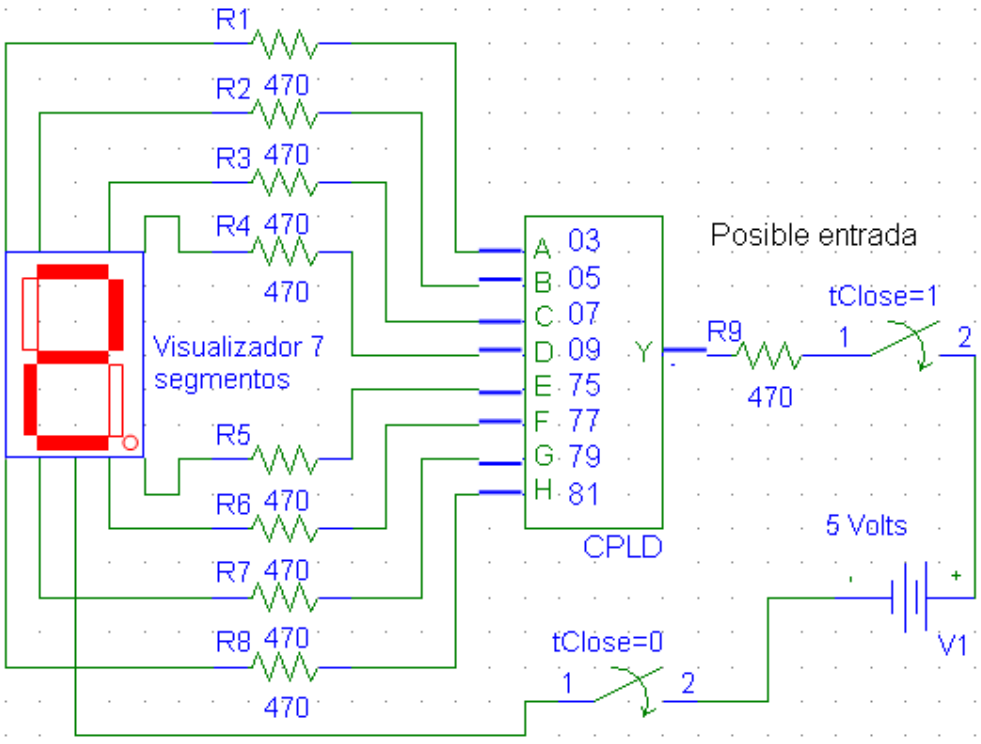


Figura 3.13 Circuito equivalente a las salidas hacia el Visualizador de 7 segmentos de la etapa de pruebas interna

El funcionamiento de este tipo de dispositivos consiste en que las terminales de todas las luces que se conectan a tierra están unidos internamente entre sí, a estas terminales se les denomina cátodos, los cátodos de todos los segmentos que integran el dispositivo salen por una terminal en común que debe estar conectada a tierra para poder funcionar, por eso recibe el nombre de cátodo común. El encendido de cada segmento individual se realiza aplicando potencia en la terminal correspondiente a través de una resistencia que limite el paso de la corriente.[82] Este dispositivo también irá previamente conectado por el cátodo

[82] Definición de Display 7 segmentos. Según Wikipedia. La enciclopedia libre. http://es.wikipedia.org/wiki/Visualizador_de_siete_segmentos

común a un interruptor que seleccionará si las salidas pueden ser visualizadas por medio de este dispositivo.

- Arreglo de leds

Esta parte de la tarjeta opera de igual manera que el dispositivo anterior con las terminales de salida, cada uno de los cátodos de los LEDs es soldado a una misma línea de cobre, creando un propio cátodo común, la diferencia existe en que aquí se pueden manejar 10 salidas y se tiene la ventaja de que los LED vienen en varias presentaciones, así que se les puede colocar de diferentes colores a manera de simular un semáforo, ya que esta es una práctica recurrente, también se busca que la luz que irradien sea intensa para poder trabajar con estas salidas aunque en el lugar de trabajo haya luz de sol, la cual en otros casos impediría su observación. La figura 3.14 señala el circuito resultante de esta salida de la etapa de pruebas.

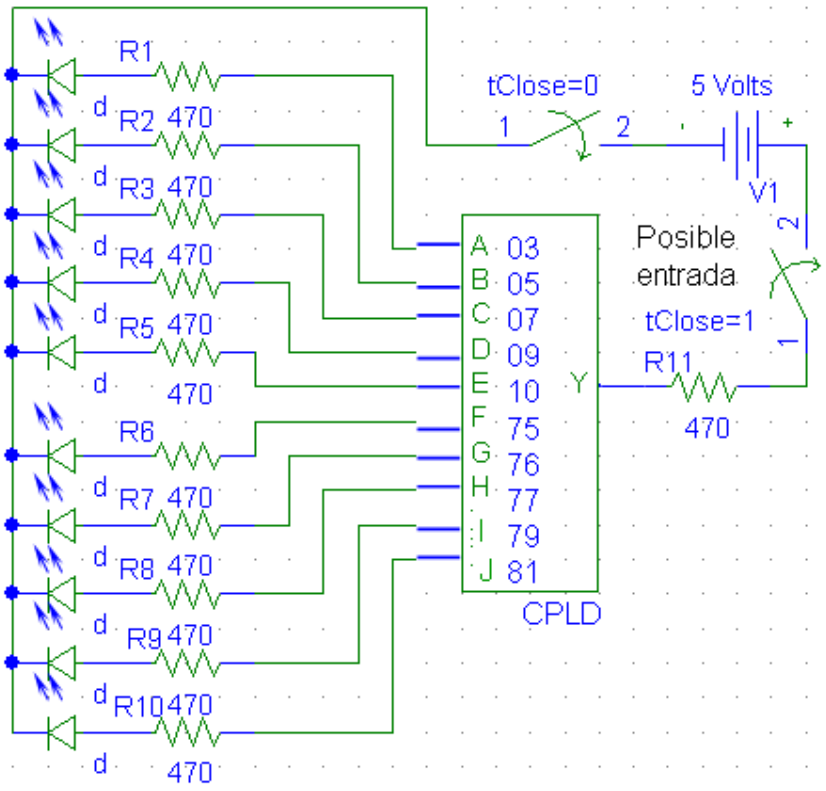


Figura 3.14 Circuito equivalente a las salidas hacia los LEDs de la etapa de pruebas interna

Todas las terminales que puedan ser posibles salidas están conectadas también a la etapa de pruebas externa, incluso las terminales que están conectadas al arreglo de LEDs o el Visualizador de 7 segmentos, pero se tiene la ventaja de que con los interruptores de tensión que se colocaron al inicio de cada circuito se puede seleccionar que parte de la etapa es la que el usuario decide hacer funcionar.

3.2.2.9 Construcción de la etapa de pruebas externa

Ya que la etapa de pruebas externo no tiene ningún elemento eléctrico o electrónico que intervenga entre las terminales de entrada/salida y los puertos que contiene la tarjeta, no existe un diagrama que muestra un circuito como tal, pero sí se mostrará el acomodo que tienen los puertos y las terminales. En la figura 3.15 se expone la manera en que están ordenados los puertos y las terminales con referencia al número de terminal que se les asignó.

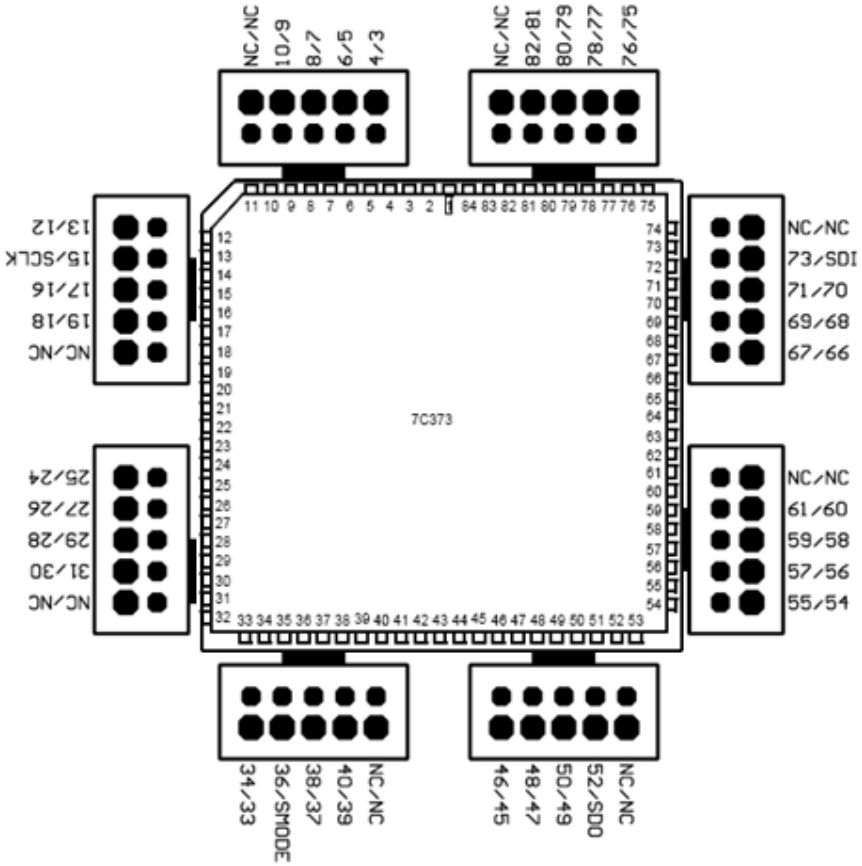


Figura 3.15 Ocho puertos conectados a la plataforma que contiene el CPLD

Las salidas o entradas correspondientes de las terminales son encausadas a ocho puertos que están diseñados para recibir un cable plano hembra de 10 conectores, acomodados en 2 líneas y 5 conectores para cada una, con el cual se podrá conectar algún otro dispositivo periférico externo a la tarjeta si así lo desea el usuario.

A pesar de que en el capítulo anterior se ha explicado la forma correcta de configurar las terminales de entrada/salida, es necesario resaltar que solo es posible ocupar aquellas que funcionen como terminales de entrada o salida y no están conectadas a alguna otra etapa o sirvan para alimentar al dispositivo.

3.2.3 Funcionamiento de las etapas en su conjunto

Todas las etapas anteriormente señaladas están contenidas dentro de la tarjeta, cada una de ellas tiene una tarea particular a realizar, pero también cabe mencionar que todas reunidas en un mismo plano hacen que la tarjeta sea más funcional, porque las etapas pueden estar trabajando de manera individual o bien pueden estar trabajando juntas, pues una puede depender de otra.

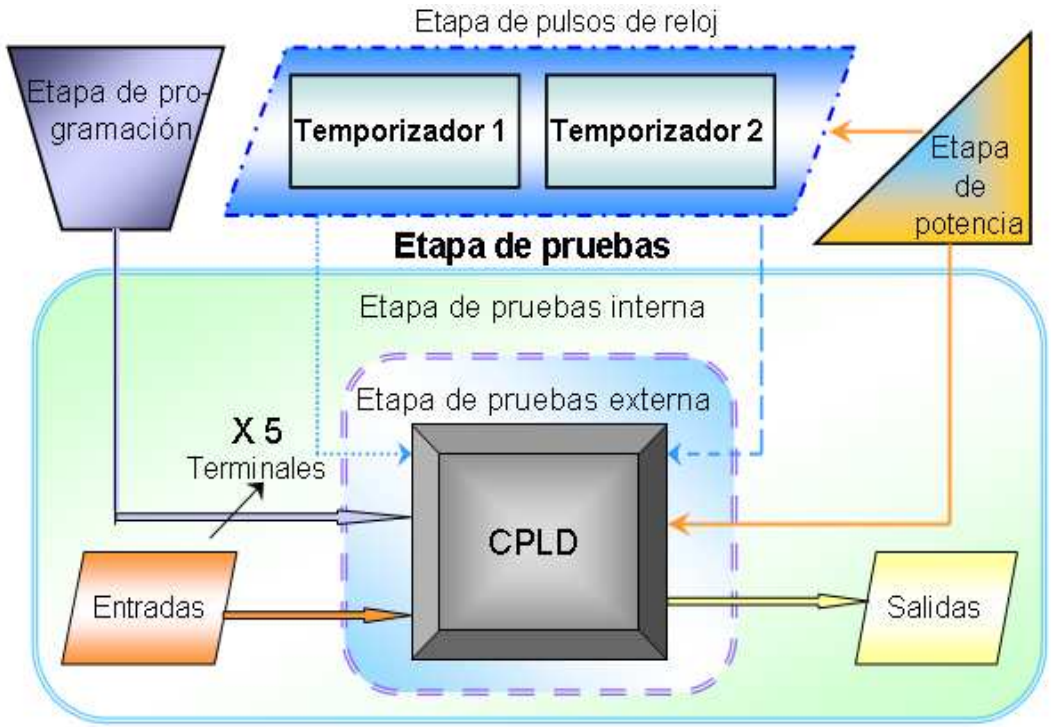


Figura 3.16 Diagrama de bloques lógicos en su conjunto.

En la figura 3.16 se muestra un diagrama a bloques de las etapas trabajando en su conjunto, teniendo todas las fases juntas se puede tener más opciones de trabajo con la misma tarjeta, con el objetivo de dar una mejor comprensión al lector, la imagen muestra también algunas líneas en forma de flecha que representan la comunicación entre etapas o también entre elementos electrónicos contenidos dentro de la tarjeta.

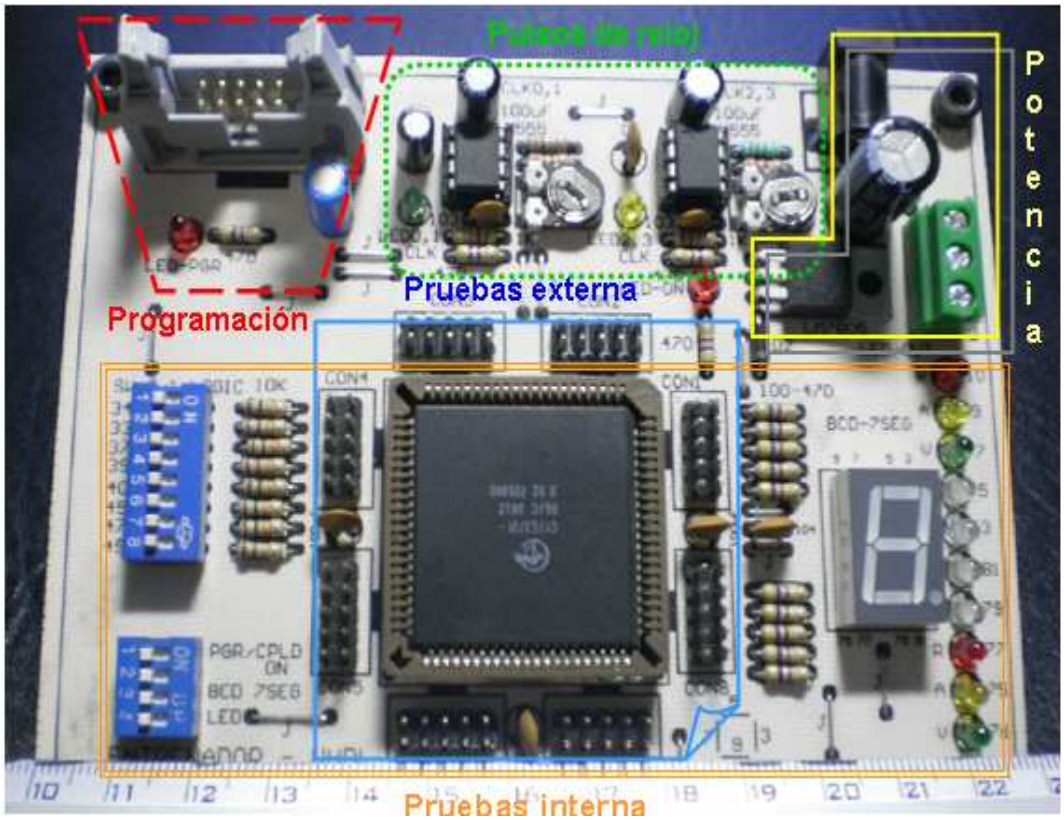


Figura 3.17 Imagen de la tarjeta entrenadora resaltando sus etapas

La figura 3.17 es una imagen real de la tarjeta entrenadora y probadora de CPLDs, dentro de la imagen se encierran las etapas funcionales que esta contiene, es similar a la imagen anterior, pero esta expone los dispositivos reales, además contiene una regla en la parte inferior de la imagen, con la cual se puede comparar sus dimensiones.

3.3 Ventajas y desventajas de la tarjeta

Para finalizar este capítulo se enumeran una serie de ventajas y desventajas que presenta esta tarjeta, algunas de ellas seguramente ya se habrán señalado con anterioridad, cuando se explicaron las etapas correspondientes.

- Unidad de potencia regulada y con salida externa de la tarjeta de 5 y 12 Volts respectivamente para controlar otros dispositivos si es que se necesita.
- La etapa de pulsos de reloj esta compuesta de dos circuitos astables con pulsos de reloj cambiantes que pueden ser controlados por el usuario con una resistencia variable.
- Capacidad de programar, borrar o reprogramar el CPLD sin ser retirado de la tarjeta.
- Montaje superficial en temporizadores y CPLD considerados los dispositivos más sensibles con opción a removerlos si es que así se requiere o para reemplazarlos fácilmente.
- La tarjeta contiene dos bloques de pruebas, uno de ellos integrado en la misma tarjeta y otro externo con el cual se pueden manipular algunos otros dispositivos fuera de la tarjeta.
- Las etapas de prueba de la tarjeta pueden programarse para trabajar de manera combinacional o secuencial según convenga al usuario.
- La etapa de pruebas interna contiene elementos prácticos para controlar entradas y visualizar salidas de forma sencilla.
- Mejor aprovechamiento de la tecnología ISR que el CPLD contiene
- Cuenta con ocho puertos especiales para manipular dispositivos periféricos contando con más de 60 terminales de entrada y/o salida para hacerlo.
- Tarjeta útil y de gran practicidad por su tamaño de 12.9 cm X 10.2 cm de dimensión

Finalmente existen algunas características que son desfavorables a esta tarjeta, en seguida se enumeran las desventajas que pueden existir.

- Debido a que el regulador de tensión interno solo puede trabajar cuando se le introduce un rango de voltaje pequeño (hasta 18 Volts) a la entrada, forzosamente se tendrá que alimentar a la tarjeta con un regulador externo para que esta pueda funcionar.
- La etapa de pruebas interna tiene pocas terminales de entrada y salida, contiene solo ocho entradas para controlar y solo hasta diez salidas para poder visualizar el resultado debido a la búsqueda de una tarjeta que fuera compacta y sus dimensiones no aumentarán.
- Las terminales del protocolo de programación que contienen una doble función no podrán ser utilizados como terminales configurables para entrada o salida ya que se conectan únicamente y de manera directa a la etapa de programación
- Para poder utilizar la etapa de pruebas externa y visualizar los resultados físicamente es forzoso conseguir y adaptar un dispositivo periférico externo a la tarjeta.
- Como cualquier tarjeta de circuitos, esta tarjeta es propensa a daños por descuidos de los usuarios o por uso inapropiado, las terminales de muchos elementos electrónicos se pueden desoldar ya que son delicadas y pequeñas, se sugiere tratar la tarjeta con delicadeza y conseguir algún estuche para poder transportarla.

Pruebas y Resultados

Después de explicar las características del dispositivo, la sintaxis del software con el cual se va a programar y las ventajas de la tarjeta entrenadora, así como sus alcances y limitaciones, en las siguientes líneas se describen algunas de las resultantes que arroja dicha tarjeta al ser probada.

La tarjeta entrenadora y probadora del CPLD 7C373i-66JC ofrece una etapa de potencia regulada y segura, también una unidad de pulsos de reloj que se compone de dos circuitos astables independientes que disparan pulsos de reloj diferentes y que pueden ser manipulados por el usuario utilizando un control gradual, puesto en una resistencia variable, situado en cada uno de los disparadores, aprovechando así poder encontrar un lapso de tiempo que sea adecuado en alguno de los circuitos para poder trabajar con él, además varias piezas están montadas a unas placas que se encuentran conectadas a la tarjeta entrenadora, por lo tanto dichas piezas no están soldadas a la tarjeta y pueden ser reemplazadas con facilidad en cualquier momento que se requiera.

Al ser conectada la tarjeta, debe encender el LED de comprobación de la etapa de potencia y también los LEDs que visualizan los flancos que disparan los pulsos en los dos circuitos astables, que se encuentran ubicados dentro de la etapa de pulsos de reloj, si esto no llegara a suceder la tarjeta tendrá algún problema de alimentación externo, o bien, dentro de ella, la cual impide la llegada de energía a las diferentes fases de la tarjeta. Si no enciende alguna de estas luces pero otras sí lo hacen, probablemente uno de los chips que conforman algún circuito astable no esté funcionando o podría ser también que el LED que dejó de encender simplemente esté fundido

Una vez que algún programa es cargado al CPLD, este puede ser probado en la tarjeta. La tarjeta está diseñada para que se pueda programar y ser probada en seguida con su etapa de pruebas, ya sea manipulando los elementos electrónicos que se contienen soldados a la misma placa o bien utilizando algunos otros

dispositivos periféricos externos a ella, como son: motores de corriente directa, motores a pasos, conversores analógico digital o digital analógico, pantallas de cristal líquido, visualizadores de LEDs, sensores de movimiento, de fotones, infrarrojos, de presión, de temperatura, por citar algunos dispositivos, los cuales pueden ser fácilmente adaptados y conectados a la tarjeta entrenadora.

Para el diseño y desarrollo de los dispositivos electrónicos y su buen funcionamiento, la lógica combinacional y la lógica secuencial se utilizan regularmente, el CPLD 7C373i-66JC tiene la ventaja de soportar lógica programable de estas dos maneras. Recordando que la lógica combinacional corresponde a aquel esquema donde el resultado de las terminales de salida del circuito están en función del valor inmediato de las operaciones realizadas con las terminales de entrada, sin que intervenga algún estado previo, mientras que el modo secuencial a comparación con la lógica anterior consiste en que el valor de la salida no solo involucra a las entradas, sino también depende de un estado de memoria interna y las terminales de salida solo podrán arrojar el resultado cuando intervenga un pulso de reloj que será introducido por alguna terminal especial que la pueda recibir.

La etapa de pruebas está pensada para que el lector pueda diseñar y crear proyectos prácticos, esta etapa está compuesta por algunos elementos adheridos a la tarjeta con los cuales se pueden desarrollar códigos para ambos tipos de lógica programable y la tarjeta entrenadora tiene integrada la opción de funcionar con ocho entradas diferentes y hasta diez posibles salidas visibles para trabajar de manera combinacional, secuencial o ambas. A su vez, la intención de que la tarjeta contenga una amplia gama de pulsos de reloj alternando un pulso alto y un pulso bajo repetidamente sin manipulación externa consiste en que con esta condición se puedan desarrollar prácticas de manera secuencial, en las cuales el programador pueda encontrar un lapso de tiempo que le convenga para automatizar una práctica que así lo requiera. También la etapa de pruebas contiene además ocho puertos que conectan directamente con las diversas terminales del CPLD que puedan ser configuradas como entradas o salidas de

datos, en ellas es posible conectar algunos dispositivos exteriores para resolver las necesidades que surjan en prácticas que requieran hacer funcionar otro tipo de mecanismos, o bien, poder crear ejemplos que contengan más de ocho entradas o diez posibles salidas.

Este trabajo también contiene una explicación útil y breve de los tipos de estructuras que se utilizan para programar, mostrando códigos completos que puedan ser introducidos al CPLD, estas prácticas a desarrollar se sugieren en el Anexo A de este texto, con ellas el alumno puede apoyarse para programar sin error lo que ahí se describe y buscar con estos fundamentos crear programas más elaborados con su propia creatividad, entendiendo que en algunas materias de la carrera de Ingeniería en Computación las prácticas contempladas pueden ser variadas.

Cabe mencionar que cada una de las prácticas sugeridas en el Anexo A, se han programado, grabado, simulado y probado con éxito; y han sido elaboradas pensando en mostrar las diversa estructuras de control que maneja el lenguaje de programación VHDL, además con ellas se exponen temas importantes sobre algunos dispositivos electrónicos que se pueden diseñar con el lenguaje de programación y representar con este CPLD como lo son: compuertas, codificadores, multiplexores, decodificadores, demultiplexores, contadores y flip-flops, por mencionar algunos. A su vez las prácticas sugeridas fueron ideadas teniendo en cuenta el aprovechamiento de la etapa de pruebas interna de esta tarjeta entrenadora y probadora.

Conclusiones

Se diseñó y construyó una tarjeta de circuitos integrados, programadora y probadora, la cual nos permite trabajar con un CPLD (Complex Programmable Logic Device, Dispositivos Lógico Programable Complejo) de la familia y modelo 7C373i-66JC, por las ventajas que este dispositivo ofrece, gracias a una de las características de este dispositivo, se implementó en la misma tarjeta entrenadora una interfaz en la etapa de programación con la cual este chip puede ser manipulado, programado, borrado y reprogramado sin necesidad de ser retirado de la tarjeta, el dispositivo tampoco necesita una fase previa de programación para ser montado en la placa. Inclusive este CPLD puede permitir que su lógica programable sea fijada para no reprogramarlo más, una vez que así lo decida el usuario y tener así una función específica permanentemente si es que las instrucciones que contiene dicho dispositivo necesitaran ya no ser cambiadas en el futuro y con esto tener a salvo el diseño de la lógica programable dentro del dispositivo.

El CPLD debe ser programado con un lenguaje de programación llamado VHDL (Hardware Description Lenguaje, Lenguaje de Descripción de Hardware), en este trabajo también se expusieron las estructuras, sintaxis y ejemplos, entre otras características propias del software. Este será el programa indicado con el cual le será asignada la lógica programable al dispositivo y las terminales específicas con las cuales se requiere trabajar. Con este texto los alumnos estudiantes de Ingeniería en Computación de la Facultad de Estudios Superiores Aragón podrán aprender a programar, o bien, si es que ya lo hacen, pueden comprender algunos conceptos nuevos.

Se describieron a detalle las etapas de las cuales se constituye la tarjeta para dar una idea clara de la manera en que está conformada, explicando también la función de la cual se ocupa cada una de ellas y haciendo de este texto un manual de usuario teórico y práctico que muestra las ventajas y desventajas de la tarjeta entrenadora y probadora del CPLD 7C373i-66JC.

La tarjeta trabaja de manera correcta y cumple efectivamente los programas creados con lógica combinacional y con lógica secuencial, para la cual contiene amplias opciones de lapsos de pulsos de reloj para poder trabajar. De tal manera que se logra conjuntar en una sola tarjeta entrenadora un circuito que soporta hacer diferentes funciones al CPLD como lo son: ser borrado, asignado, programado tolerando diferentes lógicas, grabado y en seguida probado, ahorrando así el tiempo que se llegaba a demandar entre un proceso y otro.

Las materias que lo decidan podrán apoyarse en esta tarjeta entrenadora para programar en VHDL y ejemplificar proyectos propios o sugeridos. Esta tarjeta esta diseñada para que algunas materias que manejan el área de hardware (como son: Análisis de Circuitos eléctricos, Dispositivos electrónicos, Diseño Lógico, Diseño de sistemas digitales o Microprocesadores y Microcontroladores) puedan utilizarla en algún laboratorio o bien creando y comprobando prácticas en clases teóricas. Con estos fundamentos, en las materias optativas que se encuentren en la recta final de la carrera y que pertenezcan a esta área, los alumnos podrán asimilar de mejor manera los nuevos conceptos tanto teóricos como prácticos que surjan.

Definitivamente esta tarjeta será una opción más para aquellos alumnos que se encuentren dentro de la población de Ingeniería en Computación a los cuales les interese el área de hardware, con esta podrán resolver desde prácticas sencillas hasta proyectos complejos utilizando el lenguaje de programación VHDL, desarrollando nuevos códigos con diseños propios y aprendiendo cómo se conforma la tarjeta entrenadora y probadora para darle un excelente uso y poder aprovechar al máximo su potencial.

Finalmente al diseñar y construir la tarjeta entrenadora y probadora, describiendo el lenguaje con el cual se trabaja y exponiendo las prácticas que se han desarrollado especialmente para trabajar con las especificaciones de la tarjeta y del CPLD correspondientes, los objetivos descritos al inicio de este trabajo se han logrado.

BIBLIOGRAFÍA.

Referencias bibliográficas.

[1] Almanzar Vázquez María Guadalupe (2004). *Guía para la elaboración de trabajos de investigación documental*. México: Editorial independiente.

[2] Cypress Semiconductor (2002). *HDL Reference Manual*. California, Estados Unidos: Cypress Corporation.

[3] Cypress Semiconductor (2001). *Manual de Usuario del CPLD 7C373i*. California, Estados Unidos: Cypress Corporation.

[4] Cypress Semiconductor (2001). *Designing with cypress In-System Reprogrammable CPLDs for PC cable programming.* California, Estados Unidos: Cypress Corporation.

[5] Cypress Semiconductor (2002). *Designing with FLASH-370i for PC cable programming.* California, Estados Unidos: Cypress Corporation.

[6] López Cano George (1999). *Diccionario de la microcomputación 1*. Colombia: Ediciones Euromexico.

[7] López Carreto Juan Manuel, Carlos Genaro Olivas y Gonzalo Isaac Duchén Sánchez (2009). *Teoría y práctica de diseño digital con lógica programable.* México: Editorial Limusa.

[8] Matilde Navarro Castaño, Lourdes Fernández Zetina, Francisco López Reyes, ET AL (1982). Diccionario Enciclopédico Quillet. (Séptimo tomo. Pág. 412). México: Editorial Cumbre, S. A.

[9] Maxines David, Jessica Alcalá (2005). *VHDL: El arte de programar sistemas digitales*. México, DF: Compañía Editorial Continental (CECSA)

[10] Morris Mano (1982). *Lógica Digital y Diseño de Computadores* (Segunda Edición). México: ED. Prentice Hall.

[11] Pardo Carpio Fernando (2004). *VHDL: Lenguaje para síntesis y modelado de circuitos*. México: Alfa-Omega.

[12] Rico Rafael, Salvador Marcos (2008). *Apuntes de VHDL.* España: Servicios de publicaciones UAH

[13] Ronald J. Tocci, Neal S. Widmer (2003). *Sistemas digitales: principios y aplicaciones*. México: ED. Prentice Hall.

[14] Rubio Sánchez D (2004). *Modelado y Simulación de un Conmutador utilizando VHDL*. Tesis de licenciatura. Ingeniería en Electrónica y Computadoras. Universidad de las Américas Puebla. México.

[15] Sánchez-Elez Marcos, Molina María del Carmen (2006). *Introducción para la programación en VHDL.* España: Universidad Computense de Madrid.

[16] Schmelkes Corina (1998). *Manual para la presentación de anteproyectos e informes de investigación (Tesis).* México: Oxford University Press.

[17] Thomas L. Floyd (2000). *Fundamentos de Sistemas Digitales* (Séptima edición). España: ED. Prentice Hall.

Referencias de páginas web.

[1] Definición Nanoelectrónica. Wikipedia. La enciclopedia libre
http://es.wikipedia.org/wiki/Nanoelectr%C3%B3nica

[2] Glosario Mastermagazine en español.
http://www.mastermagazine.info/termino/5022.php

[3] Definición de PLD. Alegsa; Diccionario de informática.
http://www.alegsa.com.ar/Dic/pld.php

[4] Electrónica Fácil. Escalas de integración
http://www.electronicafacil.net/tutoriales/ESCALAS-INTEGRACION-CIRCUITOS-LOGICOS-SSI-MSI-LSI.php

[5] Glosario de robótica. http://robots-argentina.com.ar/glosario.htm

[6] Definición de CMOS. Alegsa; Diccionario de informática.
http://www.alegsa.com.ar/Dic/cmos.php

[7] Ventajas de la tecnología CMOS. Wikipedia. La enciclopedia libre
http://es.wikipedia.org/wiki/CMOS

[8] Definición de Lógica programada. Wikipedia. La enciclopedia libre.
http://es.wikipedia.org/wiki/Lógica_programada

[9] Electrónica integrada. Definiciones y antecedentes
http://electronicaintegradaunexpo.blogspot.com/2008/02/gal-y-vhdl.html

[10] Dispositivos Lógicos Programables (parte 1):
http://iindustrial.obolog.com/dispositivos-logicos-programables-parte-1-209085

[11] Definición CPLD. Wikipedia. La enciclopedia libre.
http://es.wikipedia.org/wiki/CPLD

[12] WorldReference el diccionario de la lengua española:
http://www.wordreference.com/definicion/polarizar

[13] Definición PLCC. Wikipedia. La enciclopedia libre:
http://es.wikipedia.org/wiki/Plastic_leaded_chip_carrier

[14] Definición JTAG. La enciclopedia libre. http://es.wikipedia.org/wiki/JTAG

[15] Mci Electronics's Blog. Historia-PLDs
http://mcielectronics.wordpress.com/2010/03/14/un-poco-de-historia-dispositivos-logicos-programables-plds/

[16] Introducción a VHDL y dispositivos FPGA. Estructura y ejemplos.
http://usuarios.multimania.es/israelsu/unimayab/Arquitectura%20de%20Computadoras/Arq%20computadoras_06.pdf

[17] Andrés Iborra, Juan Suarez Díaz. Diseño de sistemas combinacionales con VHDL.
http://www.dte.upct.es/personal/andres_iborra/docencia/elec_ind/pdfs/VHDL%20Combinacional.pdf

[18] Definición de VHDL. The free Dictionary.
http://encyclopedia2.thefreedictionary.com/VHDL.

[19] Álvarez Marquina Agustín. Tecnología de computadores. Descripción de VHDL.
http://www.ele.uva.es/~sduenas/ASIC/VHDL2A.pdf

[20] Ing. Diego Barragán Guerrero. VHLD: Su organización y arquitectura.
http://es.scribd.com/doc/20215819/VHDL-su-organizacion-y-arquitectura

[21] VHDL. Sistemas Digitales:
http://www.utm.mx/~jvasquez/3_Sintaxis%20de%20VHDL.pdf

[22] Definición de Programación estructurada. Monografías.com "Estamos construyendo algo grande entre todos"
http://www.monografias.com/trabajos/progestructu/progestructu.shtml

[23] VHDL: Una breve historia de VHDL.
Blog: http://vhdl-blog.blogspot.com/2008/01/una-breve-historia-de-vhdl_14.html

[24] El lenguaje VHDL. Unidades básicas de diseño. Departamento de Electrónica. Universidad de Alcalá. España. Págs. 149.
htp://193.146.57.132/depeca/repositorio/asignaturas/30791/T4_02_unidades_diseno.pdf

[25] Jan Van der Spiegel. VHDL Tutorial. University of Pennsylvania. Department of Electrical and Systems Engineering.
http://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html#_Toc526061352

[26] Definición de proceso: Green Mountain. Computing systems, inc.
http://www.gmvhdl.com/process.htm

[27] Fundamentos de programación, Operadores aritméticos:
http://www.iuma.ulpgc.es/users/jmiranda/docencia/libro_ada/libro_ada_html/node27.htm#foot810

[28] El lenguaje VHDL 93. Programación y ejemplos.
http://www.angelfire.com/al4/vhdl93/vhdl93.htm

[29] Objetos de VHDL:
http://www.fceia.unr.edu.ar/eca1/files/LDD/Tipo_datos%20V_2006.pdf

[30] Green Mountain. Computing systems, inc.
http://www.gmvhdl.com/VHDL.html

[31] VHDL Sintaxys reference
http://webdocs.cs.ualberta.ca/~amaral/courses/329/labs/VHDL_Reference.html

[32] Definición de Tarjeta de circuitos impreso.
Planeta electrónico. Sabemos y construimos electrónica.
http://www.planetaelectronico.com/cursillo/tema4/tema4.1.html

[33] Definición de los circuitos impresos. Wikipedia. La enciclopedia libre. http://es.wikipedia.org/wiki/Circuito_impreso

[34] Historia de la placa de circuito impreso. Articles en español, categoría: Computadoras y Hardware. http://bot-info.net/?p=151895

[35] María Riesco García y Alicia Beisner Muñoz (2009). *Fundamentos de diseño de circuitos impresos*. [Diapositiva 4] España. Presentación: Introducción y tipos de circuitos impresos. 20 Diapositivas. http://usuarios.multimania.es/josegpedrique/MANUALES/circuitos%20impresos.ppt#261

[36] Definición ISR: Wikipedia. La enciclopedia libre. http://en.wikipedia.org/wiki/In-system_programming

[37] Regulador de tensión a 5 volts. tuelectronica.es Electrónica básica. http://www.tuelectronica.es/esquemas/alimentacion/regulador-de-tension-a-5v.html

[38] Manual de usuario del regulador de voltaje LM7805: LM78XX Series 3-Terminal Positive Reguladors. http://www.national.com

[39] Definición multivibrador astable según forosdeelectrónica.com La comunidad internacional de electrónicos. http://www.forosdeelectronica.com/f27/diferencias-entre-biestable-astable-monoestable-4437/

[40] Definición de temporizador según Wikipedia. La enciclopedia libre. http://es.wikipedia.org/wiki/Circuito_integrado_555

[41] Definición de TAP, según thefreelibrary. Articles and Books. http://www.thefreelibrary.com/Limited+access+tools+to+improve+test+coverage%3a+smaller+test+pads+are+...-a0210847588

[42] Definición de DIP Switch, según electónica2000. Diccionario de electrónica ingles-español. http://www.electronica2000.com/dic_elec/d.htm

[43] Definición de Display 7 segmentos. Según Wikipedia. La enciclopedia libre. http://es.wikipedia.org/wiki/Visualizador_de_siete_segmentos

[44] Todos los manuales de usuario. http://www.alldatasheet.com

[45] Cypress Corporation. http://www.cypress.com

Anexo A

Prácticas sugeridas

En el siguiente texto se muestran algunos códigos que van desde proyectos simples hasta algunos más elaborados, aquí se describirán ejemplos de los diseños implementados en VHDL para que los alumnos novatos en la materia puedan transcribirlos y programar sin error sus primeras prácticas, todos los códigos que a continuación se describen ya han sido programados y probados anticipadamente en la tarjeta entrenadora y probadora del CPLD 7C373i, para evitar errores o confusiones. Cada uno de los ejemplos que aquí se exponen contiene un objetivo, una descripción en la cual se detalla con qué tipo de estructura se está trabajando o como lograremos el objetivo que se plantea y el código fuente numerado a cada línea para poder dar una mejor explicación de él, entre otros aspectos.

Al inicio los códigos serán solo la declaración de alguna unidad de diseño, esto con el fin de que se comprenda la manera correcta de desarrollar una entidad o una arquitectura entes de comenzar un código completo. Después de esto se implementará el diseño de códigos completos utilizando las estructuras de control que contiene el software y que se han redactado previamente en el segundo capítulo de este trabajo, para que el lector elija cuales de estas estructuras le serán de utilidad para un proyecto propio a futuro.

De igual manera para el diseño de prácticas diversas, la lógica combinacional y la lógica secuencial se utilizan regularmente, este texto será dividido en ambos tipos de lógica. Recordando que la lógica combinacional corresponde a aquel esquema donde el resultado de las terminales de salida del circuito están en función del valor inmediato de las operaciones realizadas con las terminales de entrada, sin que intervenga algún estado previo, mientras que el modo secuencial a comparación con la lógica anterior consiste en que el valor de la salida no solo

involucra a las entradas, sino también depende de un estado de memoria interna y las terminales de salida solo podrán arrojar el resultado cuando intervenga un pulso de reloj que será introducido por alguna terminal especial que la reciba.

Las prácticas sugeridas contienen diseños con ambos tipos de lógica dando ejemplos concretos y funcionales para que el alumno las pueda diferenciar y encontrar en estas prácticas el diseño que se acomode a sus propias necesidades para así trabajar en proyectos propios.

Con el objetivo de que el lector tenga una opción más para solucionar los problemas abordados en clases teóricas o laboratorios del área de hardware de la carrera de Ingeniería en Computación se redactan las siguientes prácticas para su estudio y desarrollo.

Declaración de unidades de diseño

El software VHDL fue diseñado a base de los principios de la programación estructurada, su arreglo general está formada por módulos o unidades de diseño, cada uno de ellos cuenta con un conjunto de declaraciones específicas para poder ser funcionales, también cada uno de ellos sirve para estructurar un sistema digital y complementarse entre sí, ya que en la mayoría de los casos un módulo se completa con algún otro. En el desarrollo de los programas VHDL entidad y arquitectura son indispensables para la estructura de un programa, a la pareja entidad/arquitectura se le llama **modelo**.

Declaración de Entidad

En VHDL el bloque conocido como entidad se declara en primer lugar. Una entidad indica las señales que entran y salen del circuito, además especifica cuantas son, el tamaño, si son de entrada o salida y el tipo, en otras palabras, declara la relación del circuito con el mundo exterior detallando las terminales.

Por ejemplo, supongamos que se necesita crear un programa el cual contenga 8 entradas y 4 salidas de tipo bit. La declaración de la entidad quedaría de la siguiente forma:

```
1. --Declaración de la entidad en VHDL
2. ENTITY operaciones IS
3. PORT (a, b, c, d, e, f, g, h: in bit;
4.       i, j, k, l: out bit);
5. END operaciones;
```

La línea 1 inicia con dos guiones, los cuales indican que el texto a la derecha es un comentario y no se tomará en cuenta en el código del programa. Las palabras que se encuentran en mayúsculas o en negritas son palabras reservadas. En la línea 2 se asigna nombre a la entidad, en este caso llamada operaciones. Los puertos de entrada y salida se declaran en las líneas 3 y 4 respectivamente, como se muestra se crean 8 señales de entrada y 4 de salida. Por último, en la línea 5 termina la declaración de la entidad.

Declaración de arquitectura

La arquitectura es el módulo de diseño donde se escribe la lógica del código, y se describe el funcionamiento de una entidad, en otras palabras, la arquitectura organiza lo que se debe hacer con los puertos de entrada para que los puertos de salida puedan tener funcionamiento y obtener de ella un resultado. Dentro de la arquitectura existen varias estructuras básicas que se pueden usar, en este tema se abordarán estas estructuras mostrando la correcta sintaxis para que el lector tenga las nociones suficientes al crear su propia lógica programable, para solucionar los problemas de diferentes maneras contando con distintos recursos.

Siguiendo con el ejemplo de la entidad anterior se propone crear una arquitectura que contenga las 8 señales de entrada y 4 de salida para hacer un código que haga funcionar 4 compuertas diferentes con operaciones booleanas.

```
1. --Declaración de la arquitectura en VHDL
2. ARCHITECTURE compuertas OF operaciones IS
3. BEGIN
4.      i<=(a and b);
5.      j<=(c or d);
6.      k<=not(e and f);
7.      l<=not(g or h);
8. END compuertas;
```

Para crear una arquitectura es necesario nombrarla con un identificador y escribir a que entidad pertenece, tal y como se muestra en la línea 2. En las líneas 3 y 8 la arquitectura inicia y termina respectivamente y en las líneas 4, 5, 6 y 7 se encuentra el cuerpo de la arquitectura. En este caso se asignan los resultados de 4 compuertas diferentes a las variables de salida –i, j, k y l- y dichos resultados están en función de todas las señales que se asignaron previamente en la entidad.

Declaración de modelo mediante vectores

A la pareja que conforma entidad y arquitectura se le conoce como modelo y esta dupla es necesaria para que un desarrollo a programar en VHDL pueda funcionar, estas dos unidades de diseño son básicas e indispensables. Mientras que una de ellas especifica las señales que entran y salen del circuito, la otra describe la lógica con que se conforma, así es como una complemente a la otra.

Por otra parte, los vectores son herramientas que nos ayudaran en gran medida a programar ya que en ocasiones es más sencillo trabajar con un grupo de bits que hacerlo por separado en cada elemento. Es importante para evitar confusiones, que la magnitud de los vectores que se sometan a cualquiera tipo de operación entre ellos pueda ser igual.

Recordando que para el uso de los tipos de dato “std_logic_vector” antes de declarar la entidad se tienen que mandar llamar la biblioteca y los paquetes que harán accesible el contenido de estos tipos de dato.

```
1. --Declaración de una entidad con vectores
2. library ieee;
3. use ieee.std_logic_1164.all;
4. ENTITY comparador IS
5. PORT (Vector_A, Vector_B: in std_logic_vector (3 downto 0);
6.         C_out: out std_logic);
7. END comparador;
```

Como se muestra en la pantalla anterior, en las líneas 2 y 3 se establecen la biblioteca y los paquetes a utilizar. En la línea 4 se declara la entidad con el nombre “comparador” y en la líneas 5 y 6 respectivamente se definen las señales de entrada que consta de dos vectores y una salida de tipo bit.

Después del breve repaso que se ha mostrado en las páginas anteriores, a continuación se sugieren algunas prácticas que pueden ser de utilidad al alumno para comenzar a programar sin error, o bien para dar una idea concreta para resolver algún problema similar a los aquí expuestos.

Práctica I. Función Booleana.

Objetivo: Se propone crear un código que pueda solucionar una función booleana mediante operadores lógicos en la cual intervienen cuatro compuertas y está dada por el valor de tres señales de tipo bit de entrada y una señal de salida de tipo bit. Dicha función se muestra en la figura 4.1

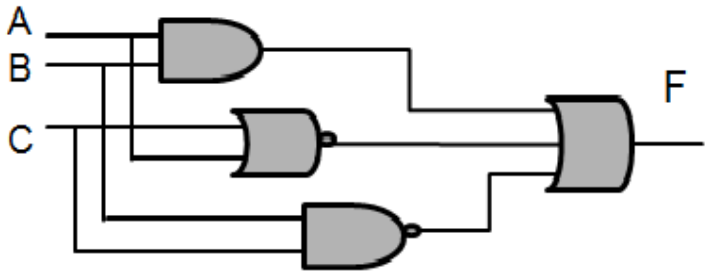


Figura 4.1 Circuito lógico de 4 compuertas

Código:

```
1. library ieee;
2. use ieee.std_logic_1164.all;
3. ENTITY funcion IS
4. PORT (A, B, C: in std_logic;
5.         F: out std_logic);
6. END funcion;

7. ARCHITECTURE compuertas OF funcion IS
8. BEGIN
9.         F<= ((A and B) or (not(A or C)) or (not(B and C)));
10.END compuertas;
```

Descripción: En las líneas 1 y 2 se utilizan una biblioteca y un paquete para tener acceso de manera correcta a los beneficios del tipo de dato “std_logic”. En la línea 3 se declara la entidad “función” con tres señales de entrada y una de salida y en la línea 7 se declara la arquitectura “compuertas”. Finalmente en la línea número 9 se describe la operación lógica para crear el resultado y asignarla a la señal de salida llamada “F”.

Resultado: Al combinar los valores de las señales de entrada en una tabla de verdad se puede comprobar los correctos resultados de la señal de salida.

Practica II. Comparador con la estructura IF-THEN-ELSE

Objetivo: Se propone hacer un comparador de dos vectores de una magnitud idéntica de 5 bits, el cual encienda una señal de salida si es que los vectores comparados son iguales con la estructura IF-THEN-ELSE.

Código:

```
1. library ieee;
2. use ieee.std_logic_1164.all;
3. ENTITY comparador IS
4. PORT (Vector_A, Vector_B: in std_logic_vector (4 downto 0);
5.         C_out: out std_logic);
6. END comparador;

7. ARCHITECTURE igualdad OF comparador IS
8. BEGIN
9.    --El nombre del proceso es opcional
10.   Comp_salida: PROCESS(Vector_A, Vector_B, C_out)
11.   BEGIN
12.         IF (Vector_A=Vector_B) THEN
13.               C_out<='1';
14.         ELSE
15.               C_out<='0';
16.         END IF;
17.   END PROCESS Comp_salida;
18.END igualdad;
```

Descripción: Siguiendo con el ejemplo de declarar una entidad con vectores se crea una arquitectura que compara la igualdad entre los dos vectores de entrada. En la línea número 9 se muestra un comentario que menciona que el nombre del siguiente proceso es opcional, el ponerlo o no dependerá del estilo del programador, es recomendable ponerle una etiqueta a los procesos para intentar definir que tarea realiza dentro del código. Esto será muy útil en un futuro cuando el programador tenga que desarrollar códigos más complejos que involucren decenas de líneas. La estructura IF-THEN-ELSE se puede observar de las líneas 12 a la 16.

Resultado: En esta práctica solo existe una señal de salida que es utilizada para confirmar la igualdad de los vectores, cuando ambos son exactamente idénticos la señal llamada “C_out” se encenderá, de lo contrario permanecerá apagada.

Práctica III. Comparador con la estructura ELSIF

Objetivo: Basados en la práctica anterior se propone crear un comparador que contenga más de una señal de salida, la primera de ellas será llamada “A” y se encenderá si el “Vector A” es más grande que el “Vector B”, si sucediera lo contrario una señal de salida llamada “B” será la que se encienda. Por último si los vectores fueran iguales se tendrá que encender una señal llamada “C”.

Código:

```
1. library ieee;
2. use ieee.std_logic_1164.all;
3. ENTITY comparador IS
4. PORT (Vector_A, Vector_B: in std_logic_vector (4 downto 0);
5.         A, B, C: out std_logic);
6. END comparador;

7. ARCHITECTURE igualdad OF comparador IS
8. BEGIN
9.    Comp_salida: PROCESS(Vector_A, Vector_B, A, B, C)
10.   BEGIN
11.   A<='0';
12.   B<='0';
13.   C<='0';
14.         IF (Vector_A>Vector_B) THEN
15.               A<='1';
16.         ELSIF (Vector_A=Vector_B) THEN
17.               C<='1';
18.         ELSE
19.               B<='1';
20.         END IF;
21.   END PROCESS Comp_salida;
22.END igualdad;
```

Descripción: El código anterior muestra gran semejanza con la práctica pasada, con la diferencia evidente de que en esta ocasión se utiliza la estructura ELSIF, ya que tenemos más de una opción a elegir a la salida. Para evitar errores con las señales de salida, éstas se inicializan en apagadas al comenzar el proceso como se muestra en las líneas 11 al 13.

Resultado: A la salida del circuito se tendrán tres señales diferentes pero solo una de ellas podrá ser encendida dependiendo el resultado de la comparación de los vectores, las otras permanecerán apagadas hasta que el resultado de la comparación cambie.

Práctica IV. Resolver una tabla de verdad con la estructura WHEN-ELSE

Objetivo: Se propone crear un modelo para comprobar el funcionamiento de un circuito cuya tabla de verdad es la siguiente:

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Tabla 4.1 Tabla de verdad 3 entradas 1 salida

Código:

```
1. library ieee;
2. use ieee.std_logic_1164.all;
3. ENTITY funcion IS
4. PORT (A, B, C: in std_logic;
5.         F: out std_logic);
6. END funcion;

7. ARCHITECTURE tabla OF funcion IS
8. BEGIN
9.         F<= '1' WHEN (A='0' and B='0' and C='0') ELSE
10.             '1' WHEN (A='0' and B='0' and C='1') ELSE
11.             '1' WHEN (A='0' and B='1' and C='1') ELSE
12.             '1' WHEN (A='1' and B='1' and C='1') ELSE
13.              '0';
14.END tabla;
```

Descripción: Las declaraciones WHEN-ELSE se utilizan para asignar valores a una señal determinada. En este caso el valor de la señal de salida "F" tomará el valor de "1" lógico cuando las condiciones de las otras tres señales independientes de entrada se cumplan, esta estructura está descrita de la línea 9 a la 12, de lo contrario la función arrojará un "0" lógico (línea 13).

Resultado: Con el código anterior se puede comprobar la correcta función de salida de la tabla de verdad en todas las combinaciones.

Práctica V. Creación de un codificador de decimal a binario

Objetivo: Se propone crear un codificador el cual es un circuito que presenta en la salida el código binario, correspondiente a lo introducido a la entrada. En este caso con un vector de 8 entradas que representarán números decimales y un vector de cuatro salidas que representarán números binarios.

Código:

```
1. library ieee;
2. use ieee.std_logic_1164.all;
3. ENTITY codif IS
4. PORT (Vector_A: in std_logic_vector(7 downto 0);
5.         Vector_B: out std_logic_vector(3 downto 0));
6. END codif;

7. ARCHITECTURE seleccion OF codif IS
8. BEGIN
9.         Vector_B<= “0001” WHEN (Vector_A=”00000001”) ELSE
10.              “0010” WHEN (Vector_A=”00000010”) ELSE
11.              “0011” WHEN (Vector_A=”00000100”) ELSE
12.              “0100” WHEN (Vector_A=”00001000”) ELSE
13.              “0101” WHEN (Vector_A=”00010000”) ELSE
14.              “0110” WHEN (Vector_A=”00100000”) ELSE
15.              “0111” WHEN (Vector_A=”01000000”) ELSE
16.              “1000” WHEN (Vector_A=”10000000”) ELSE
17.               “0000”;
18.END seleccion;
```

Descripción: La pantalla anterior muestra de las líneas 9 a 17 cómo el código de la arquitectura asigna a la variable de salida, en este caso llamada “Vector_B”, la numeración en binario del 1 al 8 siempre y cuando se active uno de los valores correspondientes en un vector de 8 bits de entrada que representa números decimales del uno al ocho. Cuando le sea introducido un valor diferente, una combinación de valores o ningún valor, el vector de salida permanecerá apagado para todos estos casos.

Resultado: Creando dos vectores diferentes, uno la entrada y otro a la salida se puede ejemplificar y recrear un codificador de decimal a binario.

Práctica VI. Creación de un multiplexor usando WITH-SELECT-WHEN

Objetivo: Se propone diseñar un multiplexor de 4 X 2, recordando que un multiplexor es un circuito con varias entradas y una única salida de datos, éstos dispositivos tienen entradas de control que pueden seleccionar una de las entradas de datos, para permitir su transmisión de la entrada seleccionada hacia la salida del circuito. Las entradas a seleccionar tienen que ser vectores de una magnitud de 4 bits.

Código:

```
1. library ieee;
2. use ieee.std_logic_1164.all;
3. ENTITY mux IS
4. PORT (A, B, C, D: in std_logic_vector(3 downto 0);
5.         S: in std_logic_vector(1 downto 0);
6.         F: out std_logic_vector(3 downto 0));
7. END mux;

8. ARCHITECTURE arqmux OF mux IS
9. BEGIN
10.        WITH S SELECT
11.        F<= A WHEN “00”,
12.             B WHEN “01”,
13.             C WHEN “10”,
14.             D WHEN OTHERS;
15.END arqmux;
```

Descripción: El funcionamiento de un multiplexor es el prototipo perfecto para ejemplificar la estructura WITH-SELECT-WHEN, ya que es una estructura que asigna una salida en función de alguna otra señal que se esté evaluando en ese momento. Como se puede observar la asignación de la estructura inicia en la línea 10 y termina en la línea 14, con la instrucción de asignar un vector de entrada dependiendo lo que el valga vector de selección llamado “S” y asignando el valor de “D” para cualquier otra combinación que pudiera presentarse.

Resultado: Se tiene que para cualquier combinación probable de las 2 líneas de selección, el multiplexor tomara el valor de alguno de los vectores entrantes y lo asignará a la única salida del circuito.

Práctica VII. Resolver una tabla de verdad con WITH-SELECT-WHEN

Objetivo: Se propone crear un modelo con la estructura WITH-SELECT-WHEN para comprobar el funcionamiento de un circuito cuya tabla de verdad es la siguiente:

Z_0	Z_1	Z_2	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Tabla 4.2 Tabla de verdad 1 vector de entrada y 1 función de salida

Código: Considerando que la entrada "Z" es un vector de 3 bits y que F es la función de salida

```
1. library ieee;
2. use ieee.std_logic_1164.all;
3. ENTITY vectores IS
4. PORT (Z: in std_logic_vector(0 to 2);
5.         F: out std_logic);
6. END vectores;

7. ARCHITECTURE resultado OF vectores IS
8. BEGIN
9.         WITH Z SELECT
10.        F<= '1' WHEN "010",
11.             '1' WHEN "100",
12.             '1' WHEN "101",
13.             '1' WHEN "110",
14.             '0' WHEN OTHERS;
15.END resultado;
```

Descripción: La arquitectura de esta estructura permite trabajar de mejor manera con vectores, asignando a una señal de salida un valor diferente dependiendo la condición de otra señal que está siendo evaluada.

Resultado: La comprobación de la tabla de verdad es la correcta para todas las combinaciones del vector de entrada.

Práctica VIII. Creación de un decodificador de binario a siete segmentos

Objetivo: Se propone crear con la estructura CASE-IS un decodificador que convierta un vector de entrada que contenga un código binario, en un vector que contenga 7 líneas de salida, para con ellos manipular un visualizador de 7 segmentos de cátodo común, en el cual se activarán las letras iniciales del alfabeto latino.

Código:

```
1. library ieee;
2. use ieee.std_logic_1164.all;
3. ENTITY deco IS
4. PORT (Vector_Bin: in std_logic_vector (2 downto 0);
5.         Vector7seg: out std_logic_vector (6 downto 0));
6. END deco;

7. ARCHITECTURE abc OF deco IS
8. BEGIN
9.    Asigna: PROCESS (Vector_Bin, Vector7seg)
10.   BEGIN
11.         CASE (Vector_Bin) IS
12.              WHEN “001” =>Vector7seg<= ”1110111”;
13.              WHEN “010” =>Vector7seg<= ”0011111”;
14.              WHEN “011” =>Vector7seg<= ”1001110”;
15.              WHEN “100” =>Vector7seg<= ”0111101”;
16.              WHEN “101” =>Vector7seg<= ”1001111”;
17.              WHEN “110” =>Vector7seg<= ”1000111”;
18.              WHEN “111” =>Vector7seg<= ”1111011”;
19.              WHEN OTHERS =>Vector7seg<= ”0000000”;
20.         END CASE;
21.   END PROCESS Asigna;
18.END abc;
```

Descripción: En las líneas de 3 a 6 se tiene declarada una entidad con dos vectores, uno de entrada y otro de salida. La arquitectura está compuesta por una estructura CASE-IS que debe ir adentro de un proceso el cual se crea en la línea 9. La estructura de control y asignación de valores al vector de salida “Vector7seg” se muestra en las líneas 11 a la 19, las cuales activan los segmentos del visualizador dependiendo el número binario que se encuentre activo en ese momento, si el vector binario vale cero, el vector de 7 segmentos se apagará.

Resultado: La pantalla muestra un contador binario el cual se puede automatizar y dependiendo la magnitud del vector binario, éste puede seguir sumando para poder crear más letras. En este caso sólo se llega hasta la letra “G”.

Prácticas secuenciales.

Para poder automatizar las prácticas haremos uso de un pulso de reloj constante el cual nos pueda ayudar a sumar, restar, contar, etcétera.

Práctica IX. Contador y restador automático con "*reset*"

Objetivo: Se propone crear un contador automático por medio de pulsos de reloj el cual contendrá dos botones de entrada, uno de ellos sumará o restará la cantidad del contador dependiendo si está activo o no, el otro será un botón de "*reset*" el cual mientras se encuentre encendido el contador permanecerá en cero.

Código:

```
1. library ieee;
2. use ieee.std_logic_1164.all;
3. use work.std_arith.all;
4. ENTITY contador IS
5. PORT (clk: in std_logic;
6.         suma, reset: in std_logic;
7.         conta: inout std_logic_vector(3 downto 0):=”0000”);
8. END contador;
9. ARCHITECTURE arqconta OF contador IS
10. BEGIN
11.   PROCESS (clk, suma, reset, conta)
12.   BEGIN
13.         IF (clk’event and clk=’1’) THEN
14.               IF (reset=’1’) THEN
15.                     conta<=”0000”
16.               ELSE
17.                     IF (suma=’1’) THEN
18.                           conta<=conta+1;
19.                     ELSE
20.                           conta<=conta-1;
21.                     END IF;
22.               END IF;
23.         END IF;
24.   END PROCESS;
25.END arqconta;
```

Descripción: En el código anterior se pueden observar elementos que no se habían utilizado antes, en la línea número 3 se muestra la declaración de un paquete que contiene los operadores matemáticos que se utilizarán en la arquitectura y en la línea número 13 se escribe el atributo para activar el contador con cada pulso alto que arroje reloj.

Resultado: Se obtuvo un código funcional de una secuencia IF-THEN-ELSE contenida en otra secuencia IF-THEN-ELSE. La sangría ayuda a identificarlos.

Práctica X. Secuencia de dos motores a pasos con dos pulsos de reloj diferentes.

Objetivo: Se propone modelar un circuito el cual haga girar dos motores a pasos automáticos, activándolos con pulsos de reloj. Para cada motor se tendrá un reloj independiente con la finalidad de que ambos giren al mismo tiempo pero con diferentes intervalos de tiempo.

Código:

```
1. library ieee;
2. use ieee.std_logic_1164.all;
3. use work.std_arith.all;
4. ENTITY DosMotores IS
5. PORT (clk1, clk2: in std_logic;
6.         Mpasos1, Mpasos2: out std_logic_vector(3 downto 0));
7.    ATTRIBUTE pin_numbers OF DosMotores: ENTITY IS
8.    "clk1:20 clk2:65 Mpasos1(0):5 Mpasos1(1):3 Mpasos1(2):81
      Mpasos1(3):79";
9. END DosMotores;
10.ARCHITECTURE pasos OF DosMotores IS
11.BEGIN
12.   Motor2: PROCESS (clk2, Mpasos2)
13.   VARIABLE Contador8: std_logic_vector (0 to 2):="000";
14.   BEGIN
15.         IF (clk2'event AND clk2='1') THEN
16.         Contador8:=Contador8+1;
17.               IF (Contador8="001") THEN
18.                     Mpasos2<="0001";
19.               ELSIF (Contador8="011") THEN
20.                     Mpasos2<="0010";
21.               ELSIF (Contador8="101") THEN
22.                     Mpasos2<="0100";
23.               ELSIF (Contador8="111") THEN
24.                     Mpasos2<="1000";
25.               ELSE
26.                     Mpasos2<="0000";
27.               END IF;
28.         END IF;
29.   END PROCESS Motor2;
30.   Motor1: PROCESS (clk1, Mpasos1)
31.   VARIABLE Conta8: std_logic_vector (0 to 2):="000";
32.   BEGIN
33.         IF (clk1'event and clk1='1') THEN
34.         Conta8:=Conta8+1;
35.               IF (Conta8="001") THEN
36.                     Mpasos1(0)<='1';
37.               IF (Conta8="011") THEN
38.                     Mpasos1(1)<='1';
39.               IF (Conta8="101") THEN
40.                     Mpasos1(2)<='1';
41.               IF (Conta8="111") THEN
42.                     Mpasos1(3)<='1';
43.               ELSE
44.                     Mpasos1<="0000";
45.               END IF;
46.         END IF;
47.   END PROCESS Motor1;
48.END pasos;
```

Descripción: El código que se muestra en esta práctica es más elaborado que los demás, esto se debe a que tiene dos procesos diferentes. Al observarlo con atención el lector puede identificar cosas nuevas pero de gran importancia para prácticas futuras. En la líneas 7 y 8 se declara un atributo el cual asignará las señales creadas en los puerto de la entidad a las terminales físicas del dispositivo CPLD 7C373i, como se expone en dichas líneas, los pulsos de reloj serán introducidas por las terminales 20 y 65 respectivamente, y uno de los motores también es asignado a terminales específicas, los cuatro pasos del motor 1 fueron establecidos a las terminales 3, 5, 79 y 81 respectivamente, esto con la finalidad de mostrar la correcta asignación de un reloj o de una señal a terminales físicas, es importante describir que todas las señales pueden ser asignadas a las terminales que las puedan recibir, en este caso la asignación está hecha con base a la etapa de pruebas interna de la Tarjeta Entrenadoras y Probadora del CPLD 7C373i. El código cuenta con dos procesos para que los conteos de los dos relojes sean independientes el uno del otro, dichos procesos están escritos de la línea 12 a la 29 y de la línea 30 a la 47, en ambos casos al principio se declara un objeto de tipo variable en las líneas 13 y 31, que son utilizadas como contador sin que sea necesario declararlo en un puerto, se expone su correcta declaración e inicialización, ya que ambos contadores comienzan en cero, es muy importante notar que las declaraciones de tipo variable tienen una sintaxis diferente a las de las señales que se habían programado con anterioridad. Por último se tiene que los dos procesos hacen exactamente la misma tarea, pero el proceso llamado “Motor2” trabaja tratando las señales como un vector completo, en cambio el proceso “Motor1” trabaja encendiendo bits específicos dentro del vector, ambos funcionan de manera correcta y ejemplifican de manera diferente como solucionar una secuencia idéntica.

Resultado: Se tienen dos motores a pasos rotando hacia una misma dirección aunque ambos giran de manera distinta ya que la activación de sus pasos depende de dos disparadores de pulsos de reloj que trabajan de manera independiente a velocidades diferentes.