REPÚBLICA BOLIVARIANA DE VENEZUELA
UNIVERSIDAD VALLE DEL MOMBOY
DECANATO DE LA FACULTAD DE INGENIERIA
INGENIERIA DE COMPUTACIÓN
CARVAJAL ESTADO TRUJILLO

USO DE LAS MAQUINA DE ESTADOS FINITOS Y MAQUINA DE ESTADOS ALGORITMICA EN LOS LENGUAJES DE DESCRIPCION DE HARDWARE.

Autor:
Br.Darwin Materan
Tutor:
Dr. Iván Pérez

Carvajal, marzo de 2018

REPÚBLICA BOLIVARIANA DE VENEZUELA
UNIVERSIDAD VALLE DEL MOMBOY
DECANATO DE LA FACULTAD DE INGENIERIA
INGENIERIA DE COMPUTACIÓN
CARVAJAL ESTADO TRUJILLO

USO DE LAS MAQUINA DE ESTADOS FINITOS Y MAQUINA DE ESTADO ALGORITMICA EN LOS LENGUAJES DE DESCRIPCION DE HARDWARE

Trabajo Especial de Grado presentado como requisito parcial para optar al Título de: **INGENIERO EN COMPUTACIÓN**

Autor:
Br.Darwin Materan
Tutor:
Dr. Iván Pérez

Carvajal, marzo de 2018

DEDICATORIA

En este momento tan importante de mi vida quiero dedicar este logro alcanzado a todas aquellas personas que siempre han estado a mi lado en todo momento.

A Dios todopoderoso, por nunca hacerme perder la fe en esos momentos tan difíciles de mi vida y carrera universitaria, permitiéndome seguir adelante y cumplir esta meta tan importante para mí.

A mi Madre Nelly Viloria, por ser la mujer más especial, amorosa y guerrera de este mundo y darme todo lo que necesite en la vida enseñándome todas las cosas buenas para ser alguien de bien, esta meta tan importante es tuya también, **TE AMO!!!**

A mi Hermano Jhon Materan, aunque ya no estas físicamente con nosotros, sé que desde el cielo me cuidas y me acompañas en cada momento de mi vida, cumplí lo que tanto esperabas mío y nuestra hija está orgullosa, **TE QUIERO!!!**

A mi hija Jhonnelly Materan, eres una bendición princesa y prometo que siempre vas a contar conmigo, cumpliré la promesa que le hice a mi hermano, **TE QUIERO!!!**

A mi hijo (a), eres mi semillita y una bendición, prometo que voy hacer un papá en el que puedas confiar, con el que puedas hablar, y sobre todo darte cariño y amor. TE AMO!!!!

A mi Abuela Ana Rita, por dedicar parte de su vida para criarme y formarme, sé que estas feliz en cielo viéndome lograr esta meta, fuiste mi segunda madre, **TE ADORO!!!.**

A mi novia Yercely Quintero, por siempre apoyarme en las buenas y en las malas, confiando en que cumpliría esta meta, a pesar de las dificultades que se nos presentaron, **TE QUIERO!!!.**

A mi tía Yrma, por quererme tanto y haberme ayudo en muchos momentos difíciles de mi vida y siempre brindarme apoyo, eres como una madre para mí, **TE QUIERO!!!**

A mi tío Morillo, fuiste alguien muy especial porque siempre me quisiste como si fuese tu sobrino de sangre.

Son muchas las personas que me gustaría nombrar, la lista es grande, a todas les dedicó este éxito una meta que gracias a Dios he podido lograr

.

.

AGRADECIMIENTO

A Dios, por siempre darme fortaleza en los momentos más difíciles de me vida y nunca abandonarme, por guiarme, iluminarme y darme salud para cumplir esta meta contigo todo es posible.

A mi Madre, por siempre brindarme ese amor incondicional y enseñarme el significado de las palabras amor y valores. Todo lo bueno que tengo como persona es gracias a ti, no caben las palabras en este trabajo, para agradecerte tanto que me has dado en la vida. Te amo madre bella.

A mi Padre, por darme esos consejos que me ayudaron a tomar buenas decisiones en mi vida.

A mi novia, por haberme esperado, aguantando todas las cosas malas y brindándome apoyo incondicional durante este tramo de mi vida, gracias por ayudarme a levantar en el momento más difícil de vida, sin ti no lo hubiera hecho.

A mis tíos, José, Alberto y Luis, por haberme apoyado en algunos momentos de mi vida.

A mi tía Rosa, por transmitir siempre alegría en los momentos no tan favorables.

A mi tía Yrma, por darme amor y apoyo cuando lo necesite, gracias por ser tan especial conmigo.

A mi abuelo Silvio, por haber compartidos momentos de felicidad durante mi niñez.

A mis primos (as), por haberme animado en esos momentos donde se pierden las ganas de seguir adelante, en especial a mis primos Ronald, José Luis y Luis Ramón, son como mis hermanos!!!

A mi maestro y gran amigo Manuel Correa, por ser mi conductor y guía en este trabajo de grado, y darme consejos muy significativos para la vida, eres el ejemplo de lo que es un verdadero “Maestro”.

A mi tutor, por haberme exigido como estudiante durante mi carrera universitaria, y así prepararme para mi desarrollo profesional fuera de la casa estudio.

A la profesora Corina Salas, por siempre brindarme ayuda y resolver dificultades que se me presentaron académicamente. Gracias.

A mis amigos Anderson y Rafael, por siempre poder contar con ustedes en las buenas y en malas. Gracias.

A los integrantes de la asignatura de arquitectura del computador en el semestre A-2018 del NURR, por haberme recibido y brindarme su colaboración para que el desarrollo de esta tesis fuese completado.

A todas esas personas que me faltaron por nombrar, amigos, familiares y compañeros de estudio que me apoyaron en las buenas y en las malas durante este tramo de mi vida “GRACIAS A TODOS”.

A la universidad Valle del Momboy y a los profesores que dedican su vida para impartir sus conocimientos enseñando y formando profesionales aptos para asumir cualquier cargo profesional fuera de la institución.

A TODOS GRACIAS.

ÍNDICE GENERAL

	Pp.
DEDICATORIA	iii
AGRADECIMIENTO	v
ÍNDICE GENERAL	vii
ÍNDICE DE FIGURAS	ix
ÍNDICE DE GRÁFICOS	xi
ÍNDICE DE TABLAS	xii
RESUMEN	xiii
INTRODUCCIÓN	1
CAPÍTULOS	
I EL PROBLEMA	
Planteamiento del Problema	3
Objetivos de la investigación	4
Objetivo General	4
Objetivos Específicos	4
Justificación de la investigación	4
Delimitación de la investigación	5
II MARCO TEÓRICO REFERENCIAL	
Antecedentes de la investigación	6
Bases Teóricas	11
Lógica secuencial síncrona	11
Flip-flops	14
Disparo del flip-flop	19
Circuitos Secuenciales temporizados	19
Tabla de estados	19
Diagrama de estados	20
RTL	20
Introducción a las Máquinas de estado finitos	21
Máquinas de estados finitos (FSM)	21
Máquina de Mealy	23
Máquina de Moore	26
Introduccion a las Maquinas de estado Algorítmicas	29
Máquinas de estado algorítmicas (ASM)	31
Diagrama ASM	31
Bloque ASM	33
Operaciones de registro	34
Consideraciones de temporizado	36

Lenguajes de Descripción de hardware (HDLs) 36
Introducción al Lenguaje VHDL 37
VHDL 38
Unidades básicas de diseño 41
Tipos de datos 44
Declaración de entidades 45
Paquetes 48
Arquitecturas 49

III MARCO METODOLÓGICO
Tipo y modalidad de la Investigación 54
Diseño de la Investigación 55

IV DESARROLLO DE LA PROPUESTA
USO DE LAS MAQUINAS DE ESTADO FINITOS Y MAQUINAS ALGORITMICAS EN LOS LENGUAJES DE DESCRIPCION DE HARDWARE
Etapas de la metodología para programar en lenguajes de descripción de hardware mediante el uso de FSM y ASM 58

V CONCLUSIONES Y RECOMENDACIONES 71
CONCLUSIONES
RECOMENDACIONES 72

REFERENCIAS BIBLIOGRÁFICAS 73

ÍNDICE DE FIGURAS

FIGURA		Pp.
1	Diagrama de bloques de un Circuito Secuencial	14
2	Diagrama lógico de un flip-flop RS	16
3	Diagrama lógico de un flip-flop D	17
4	Diagrama lógico de un flip-flop JK	17
5	Diagrama del lógico d un flip-flop T	18
6	Sentencia RTL	20
7	Diagrama de flujo de Mealy	24
8	Diagrama general de Mealy	25
9	Diagrama a nivel de compuertas y flip-flop de Mealy	25
10	Diagrama de Maquina de estado de Mealy	26
11	Diagrama de flujo de Moore	27
12	Diagrama general de Moore	27
13	Diagrama a nivel de compuertas y flip-flop de Moore	28
14	Diagrama de Maquina de estado de Moore	28
15	Interacción de control y procesador de datos	31
16	Caja de estado	33
17	Caja de decisión	33
18	Caja de condición	33
19	Bloque ASM	34

20	Descripción a nivel de compuertas de una entidad	42
21	Símbolo funcional de la entidad	42
22	Diagrama a bloques representativo de la entidad	42
23	Comparador de igualdad	43
24	Modos y el curso de sus señales	44
25	Declaración de la entidad sumador	45
26	Entidad representadas por vectores	46
27	Contenido de la librería ieee y work	48
28	Descripción funcional de un comparador de igualdad de 2 bits	50
29	Arquitectura funcional de un comparador de igualdad de 2 bits	51
30	Arquitecturas por flujos	52
31	Descripción estructural de un comparador de 2 bits	53
32	FSM llevada a ASM	63
33	Código para definir componentes en VHDL	68

ÍNDICE DE GRÁFICOS

GRÁFICO		Pp.
1	Esquema del diseño de la metodología.	57
2	Metodología para programar en lenguajes de descripción de hardware (VHDL) mediante el uso de las FSM y ASM.	69
3	Esquema general de la metodología para programas en lenguajes de descripción de hardware (VHDL) mediante el uso de las FSM y ASM	70

ÍNDICE DE TABLAS

TABLA		Pp.
1	Tabla de la verdad del flip-flop RS	16
2	Tabla de la verdad del flip-flop D	17
3	Tabla de la verdad del flip-flop JK	18
4	Notación simbólica para las operaciones de registros	35
5	Especificación para la escritura de identificadores	46

REPÚBLICA BOLIVARIANA DE VENEZUELA
UNIVERSIDAD VALLE DEL MOMBOY
DECANATO DE LA FACULTAD DE INGENIERIA
INGENIERIA DE COMPUTACIÓN
VALERA ESTADO TRUJILLO

USO DE LAS MAQUINAS DE ESTADO FINITOS Y MAQUINAS DE ESTADO ALGORITMICAS EN LOS LENGUAJES DE DESCRIPCION DE HARDWARE

Autora:
Br. Darwin Materan
Tutor: Dr. Iván Pérez
Año: 2018

RESUMEN

En esta tesis se propone una metodología para el desarrollo de programas de descripción de hardware aplicando como estrategia didáctica el uso de Maquinas de estados finitos y Maquinas de estado algorítmicas como abordaje en el análisis a los problemas planteados y tratando de establecer un esquema de fases para la resolución de problemas tal como se cursó en los diseño de programas en los cursos de programación básica al establecer metodologías de programación que permitían realizar un análisis previo de los problema y luego su diseño y edición. Y así, dar soporte a los estudiantes de Arquitectura del Computador del Tercer semestre de la carrera de Ingeniería en sistemas quienes emplean los lenguajes de descripción de hardware como conocimiento transversal del curso, de ahí que se aplica un auto-aprendizaje del mismo

En primer lugar se abordan el empleo de las máquinas de estado finitos en el diseño de circuitos secuenciales y luego se traduce una máquina de estado algorítmica para así poder llegar de una manera más sencilla y didáctica a la programación en VHDL.

Por medio de Lenguajes de Descripción de hardware es posible simplificar circuitos y el uso de herramientas de análisis como FMS y ASM nos permite desarrollar circuitos secuenciales en su mínima expresión cumpliendo con las tareas encomendadas en su desarrollo.

Palabras Clave: Metodología, Hardware, Estrategia, Maquina, Estado, Finito, Algorítmica, Lenguaje, VHDL, Circuito Diseño, Programación.

INTRODUCCIÓN

En la actualidad se encuentra una gama de metodologías para el análisis y desarrollo de programas, muchos de ellos para programación modular. Cuando nos enfrentamos a programas de descripción de hardware observamos la ausencia de algoritmos que permitan realizar un bosquejo del mismo a pesar de existir en la literatura herramientas que permiten abordar los mismos.

Ante esta situación se busca elaborar una metodología completa que permita atacar el problema de hardware y describirlo a través de un programa de descripción de hardware, específicamente VHDL. Para ello se realizará una investigación conceptual y aplicada que permitirá abordar toda la documentación existente y elaborar una metodología completa que permita al diseñador de hardware documentar y dar solución al problema planteado.

Se iniciara con la investigación de máquinas de estados finitas y algorítmicas que permitirán representar gráficamente el problema y que representa uno de los puntos ausentes en las metodologías propuestas en las literaturas actuales, con ella se evaluará y editará la solución al problema en VHDL para ello, se emplearan tres casos de estudio para depurar y establecer la metodología general que permita abordar la descripción de cualquier tipo de problema de hardware que se presente.

Durante el desarrollo se documentará todo el marco teórico que dará soporte a la investigación documental y permitirá al lector entender cada uno de los procedimientos planteados en la metodología. El desarrollo de esta investigación abarca un conjunto de capítulos, enumerados de la siguiente manera:

Capítulo I: El Problema, se describe el planteamiento del problema, donde se especifican las causas y efectos que dieron lugar al mismo, así como también los objetivos tanto generales como específicos, la justificación y las delimitaciones de la investigación.

Capítulo II: Marco Teórico, el cual se encuentra sustentado a través de los antecedentes y bases teóricas. Dentro de este mismo capítulo se definen un conjunto de términos básicos los cuales facilitarán la comprensión de los tópicos aquí estudiados. Para el logro y desarrollo de este capítulo fue necesario por parte del investigador indagar acerca de las diversas teorías que hoy por hoy se encuentran en torno a la temática planteada en este trabajo especial de grado.

Capítulo III: Marco Metodológico, comprende el tipo, diseño y modalidad de la investigación, la cual se ubica en una perspectiva metodológica de investigación documental e investigación aplicada porque se construye conocimientos a partir de un trabajo ya planteado.

Capítulo IV: Desarrollo de la Propuesta. Presenta los resultados obtenidos durante cada una de las fases de la metodología correspondiente al trabajo de grado.

Capítulo V: Para finalizar en este capítulo se establecen las conclusiones y recomendaciones de la investigación. Así como también las referencias bibliográficas que sirvieron de apoyo para este trabajo.

CAPÍTULO I

EL PROBLEMA

Los Lenguajes de descripción en hardware surgieron como herramienta de diseño que auxiliaba al ingeniero para integrar un mayor número de dispositivos en un solo circuito integrado. Una de sus principales características radica en su capacidad para describir en distintos niveles de abstracción (funcional, transferencias de registros RTL y lógico o nivel de compuerta) ciertos diseños.

En la actualidad el lenguaje de descripción en hardware más utilizado a nivel industrial es VHDL y por tal motivo no se escapa de ser utilizados en asignaturas de diseño de hardware en las diferentes carreras universitarias como informática, computación, sistemas, electrónica y carreras afines.

Los estudiantes de las carreras citadas, cursan asignaturas de programación básica donde se les enseña una metodología de programación para abordar cualquier problema planteado. La fase más importante es la del análisis, al finalizar la misma el estudiante debe tener la capacidad de esbozar el problema a través de un algoritmo o diagrama de flujo, previo a la edición y codificación.

Ahora, cuando abordan la programación en lenguajes de descripción en hardware, se encuentra que las mayoría de las metodologías atacan el problema directamente considerando el diseño final del producto e implementando cada uno de los dispositivos bajo las plantillas existentes en la documentación, dejando atrás el aprendizaje adquirido en las asignaturas previas y trabajando por separado cada etapa del diseño, lo que dificulta la documentación del producto final.

Ante este planteamiento se requiere de una metodología integral para el diseño de hardware que agrupe las herramientas que logren describir el problema paso a paso, antes de ser programado. Dentro de esas herramientas se encuentran las Máquinas de estado finitos y las máquinas de estados algorítmicas que permiten modelar la dinámica de un circuito digital, al estilo de los algoritmos o diagramas de flujos en las asignaturas previas y luego si codificar el mismo en cualquier lenguaje de descripción en hardware.

OBJETIVOS DE LA INVESTIGACIÓN

Objetivo General

Proponer el uso de Máquinas de estados Finitos y Máquinas de estados algorítmicas como herramientas didácticas para el desarrollo de programas en Lenguajes de descripción de hardware.

Objetivos Específicos

1. Examinar las bases teóricas de las máquinas de estado finitos y máquinas de estados algorítmicas.
2. Plantear un algoritmo para representar un problema dado en máquinas de estado finitas.
3. Desarrollar un algoritmo para representar un problema dado en máquinas de estado algorítmicas.
4. Elaborar metodología que permita la codificación de la máquina diseñada en Lenguaje de Descripción de Hardware.
5. Validar la metodología de diseño de máquinas propuestas.

Justificación de la investigación

El manejo de lenguajes de descripción de hardware es una herramienta importante en el uso de simulación de procesadores y sistemas computacionales. El desarrollo de esta propuesta va a permitir a los

estudiantes y profesores de las asignaturas afines al diseño de circuitos secuenciales abordar los problemas de una manera más práctica, sin dejar de escapar ningún detalle en los estados presentes en el mismo.

Al igual que la enseñanza en los cursos básicos de programación donde se implementa una metodología para abordar los problemas planteados, en este trabajo se propone varias etapas importantes para la generación de una metodología de programación para el desarrollo de circuitos secuenciales a ser implementados en Lenguajes de Descripción de hardware y de esta manera evaluar su comportamiento y posibilidad de simplificarlo a la hora de decidir sintetizarlo en un circuito integrado.

Pero más importante aún, se les da una nueva herramienta didáctica y definida a los estudiantes de arquitectura de computadoras para el desarrollo de la práctica de los problemas y actividades planteadas en clase y que puedan sintetizarlos a través de circuitos lógicos programables como las tarjetas de arreglo lógico programable en el campo (FPGAs) implementado a través de los lenguajes de Descripción de Hardware de Circuitos Integrados de muy alta velocidad (VHDL).

Delimitación de la investigación

Temática: Uso de las Maquinas de estado Finitos y Maquinas Algorítmicas en los lenguajes de descripción de hardware.

Espacial: en la asignatura de arquitectura de las computadoras en la carrera de ingeniería en sistemas en la Universidad de los Andes.

Temporal: se está desarrollando durante los meses de enero del 2018 y abril de 2018.

CAPÍTULO II

MARCO TEÓRICO REFERENCIAL

En este capítulo se refleja la información necesaria para hacer uso de las máquinas de estado finito y algorítmico en los lenguajes de descripción de hardware, se hace referencia a los Antecedentes de la Investigación, las bases teóricas.

Antecedentes de la investigación

La electrónica y la computación en estos últimos años han evolucionado rápidamente, por ello es necesario actualizarse continuamente, de modo que con el uso y aplicación de Máquinas estados finitos (FSM) y Máquina de estados algorítmicas (ASM) es posible realizar diseños en menor tiempo debido a la disminución a la hora de tratar de simular un diseño directamente en un lenguaje de descripción de hardware.

En las últimas décadas el modelado de circuitos ha permitido simular los mismos para comprobar que se corresponde con la funcionalidad deseada y luego poder sintetizarlo para crear un circuito que funcione como modelo y así mantener los microchips con funcionalidades específicas. En la Universidad de Los Andes, Carrera Ingeniería en Sistemas ciclo básico se dicta la asignatura Arquitectura de Computadoras que tiene como uno de sus objetivos modelar en VHDL las arquitecturas digitales y permitir a sus estudiantes pasar de la teoría a la práctica a partir de la observación de la lenguaje de descripción de hardware ya que el estudiante debe aprender de manera transversal el uso del mismo. En este sentido se cuentan con herramientas que facilitan el aprendizaje y empleo del VHDL, algunas propuestas, que se buscan sintetizar en un solo documento por medio de una

Metodología de diseño de máquinas, entendiendo que una computadora es una máquina de dos estados.

En este sentido De Pablo, Cáceres, Cebrián, Berrocal y Sanz (2015) de la Universidad de Valladolid a través de su trabajo de investigación “Los diagramas ASM++ como herramienta aplicada en la enseñanza de la electrónica digital” exponen las ventajas del uso de máquinas de estados algorítmicas (ASM) en relación con máquinas de estados finitas (FSM) considerando que ellas detallan más que las transiciones entre sus estados agregando las operaciones que ocurren entre ellos, durante y al final de cada estado, logrando representar de forma muy directa el comportamiento del circuito, materializando ideas y probando alternativas.

Los autores explican que el uso de Lenguajes de descripción de hardware son muy sencillos de emplear, sin embargo el empleo de diagramas ASM tiene muchas ventajas en el proceso de formación en electrónica digital y en la producción de circuitos digitales y proponen una mejora del mismo al cual denomina ASM++ el cual genera como salida el código HDL.

Concluyen con la presentación de una metodología gráfica muy intuitiva y completa que facilita la representación del comportamiento de los diseños digitales a nivel de registro, lo que es especialmente útil en el proceso de enseñanza de la electrónica digital. El lenguaje gráfico propuesto es fácil de aprender y es entendido sin dificultad por estudiantes universitarios, tanto de nivel básico como de nivel avanzado, que lo emplean como parte de su metodología de diseño para producir circuitos digitales sobre FPGA.

El aporte de esta investigación está centrada en las diferencias sustanciales entre FSM y ASM, pero sustancialmente el uso práctico de las ASM y su inclusión en alguna metodología de diseño.

Lara (2002) en su trabajo de Maestría en Ingeniería Eléctrica con especialidad en Control, explica la ventaja de emplear lenguajes de descripción de hardware en los diseños de sistemas digitales, su simplificación y reutilización, específicamente en VHDL.

El objetivo del trabajo es actualizar a los estudiantes, como profesores en las carreras de Electrónica y Comunicaciones, Electrónica y Automatización en las herramientas actuales existentes en el mercado, objetivo necesario debido a la reducción del horario, de ahí la necesidad de ofrecer herramientas de diseño digital como Lenguajes de descripción de Hardware que permitan adquirir conocimientos, desarrollar habilidades, actitudes y valores; es decir, ampliar los espacios de aprendizaje fuera del salo de clases o laboratorio.

Para preparar a los estudiantes el autor propone una metodología para el diseño digital utilizando VHDl, la cual compone cuatro (4) fases, donde la segunda fase sugiere la utilización de algún tipo de diagrama de bloque con la descripción de funcionamiento de cada uno y diagramas de estados, en esta sugerencia se puede asumir que hacer referencia al uso de Máquinas de Estados Algorítmicas y Maquinas de estados Finitas respectivamente.

El autor concluye que el diseño programable presenta una serie de ventajas sobre el método de diseño tradicional; donde menciona la rapidez del diseño, las pruebas antes de la implementación (simulación), la reprogramación de los dispositivos en caso de fallas y cambios de los mismos, la reducción de costos y la vanguardia en cuestión tecnológica.

Para esta investigación aporta el abordaje de una metodología de diseño que va muy ajustada a la propuesta, donde el punto que cubre es justamente la sugerencia realizada en la fase 2 de la metodología descrita.

Por su parte Holguí, Orozco y Escobar (2011) en su trabajo de investigación titulado “Metodología para el diseño de autómatas finitos con salidas en lenguaje ladder bajo el estándar IEC 61131-3” abordan los autómatas de estados finitos como una metodología de análisis empleada en el diseño de sistemas lógicos secuenciales desde la perspectiva de la automatización, considerando lo de mayor interés los sistemas secuencias con salidas, en los cuales se encuentran las denominadas Máquinas de Mealy y Máquina de Moore.

Destacan que los autómatas finitos son mayormente empleados por la gran variedad de metodologías disponibles para su diseño, además de poder aplicar análisis sistemático con el fin de aplicar simplificación de los autómatas obtenidos llevando a reducir la implementación física, conservando la funcionalidad con una implementación de menos estados.

Realizan un breve estudio de los Autómatas finitos lo que permite definir las etapas de diseño que finalmente son empleadas en la metodología de diseño propuesta por los autores, donde concluyen que la metodología presentada permite traducir satisfactoriamente todo autómata finito, con y sin salidas para su implementación.

Esta investigación da soporte a este trabajo en cómo se abordan las etapas de diseño de los autómatas finitos en las máquinas de Moore y Mealy y como las misma son englobadas en un metodología de diseño.

Por último, Pollán, Martín y Ponce (2006) de las Escuela Universitaria de Ingeniería Técnica Industrial de Zaragoza, Departamento de Ingeniería Electrónica y Comunicaciones. Universidad de Zaragoza. España en su trabajo de investigación titulado “Máquinas algorítmicas como opción didáctica de sistemas complejos” hacen referencia a que las máquinas algorítmicas son sistemas digitales con separación entre la parte de control y la parte operativa y expresión de dicho control mediante un grafo de estados.

Implementándolas en los ejercicios o trabajos de la asignatura Electrónica Digital, considerando la posibilidad del uso de un lenguaje de descripción de hardware como VHDL.

Para la separación de la parte de control y operativa proponen un esquema separado por etapas y luego abordan ejemplo clásicos de la asignatura.

Concluyen que el objetivo de esta investigación es llamar la atención de los profesores de electrónica digital sobre la utilidad y viabilidad de las máquinas algorítmicas en la asignatura. Lo cual permite configurar procesadores con número de instrucciones limitados y autómatas programables para luego ser implementadas en FPGAs y verificar su funcionamiento.

Este trabajo de investigación permite abordar las máquinas de estados algorítmicas a nivel de grafos, lo que permite un sencillo aprendizaje del mismo para un abordaje inicial para la migración de Máquinas de estado finitos a Máquinas de estado algorítmicas durante la investigación y creación de la propuesta de investigación.

Bases Teóricas

Lógica secuencial síncrona

Los circuitos digitales que hasta ahora se han considerado han sido combinacionales, esto es, las salidas en cualquier momento dependen por completo de las entradas presentes por el tiempo. Aunque cualquier sistema digital es susceptible de tener circuitos combinacionales, la mayoría de los sistemas que se encuentran en la práctica también incluyen elementos de memoria, los cuales requieren que el sistema se describa en términos de lógica secuencial.

Un diagrama de bloques de un circuito secuencial consta de un circuito combinacional al que se conectan elementos de memoria para formar una trayectoria de retroalimentación. Los elementos de memoria son dispositivos capaces de almacenar dentro de ellos información binaria. La información binaria almacenada en los elementos de memoria en cualquier momento dado define el estado del circuito secuencial. El circuito secuencial recibe información binaria de entrada externa. Estas entradas, junto con el estado presente de los elementos de memoria determinan el valor binario en las terminales de salida. También determinan las condiciones para cambiar el estado en los elementos de memoria. El diagrama de bloque demuestra que las salidas externas en circuito secuencial son funciones no solo de las entradas externas sino también del estado presente de los elementos de memoria. El siguiente estado de los elementos de memoria también es una función de las entradas externas y del estado presente. Por tanto, un circuito secuencial está especificado por una secuencia de tiempos de entradas, salidas y estados internos.

Hay dos tipos principales de circuitos secuenciales. Su clasificación depende del temporizado de sus señales. Un circuito secuencial síncrono es un sistema cuyo comportamiento puede definirse por el comportamiento de sus señales en instantes discretos de tiempo. El comportamiento de un circuito secuencial asíncrono depende del orden en el cual cambian las señales de entrada y puede afectarse en cualquier instante del tiempo. Los elementos de memoria que por lo común se utilizan en los circuitos secuenciales asíncronos son dispositivos de retardo de tiempo. La capacidad de memoria de un dispositivo de retardo de tiempo se debe al hecho de que toma un tiempo finito para que la señal se propague a través del dispositivo. En la práctica. El retardo de propagación interno en las compuertas lógicas es de suficiente duración para producir el retardo necesario, de modo que pueden ser innecesarias unidades físicas de retardo de tiempo. En los

sistemas asíncronos de tipo de compuerta, los elementos de memoria constan de compuertas lógicas cuyos retardos de propagación constituyen la memoria requerida. Por consiguiente, un circuito secuencial asíncrono puede considerarse como un circuito combinacional con retroalimentación. Debido a la retroalimentación entre compuertas lógicas, un circuito secuencial asíncrono a veces puede llegar a ser inestable. El problema de la inestabilidad le impone muchas dificultades al diseñador.

Los sistemas lógicos secuenciales síncronos usan amplitudes fijas, como niveles de voltaje para las señales binarias. La sincronización se logra por un dispositivo temporizador llamado reloj maestro generador, el cual genere un tren periódico de pulso de reloj. Los pulsos del reloj se distribuyen a través del sistema en tal forma que los elementos de memoria están afectados solo por la llegada del pulso de sincronización. En la práctica los pulsos del reloj se aplican a las compuertas AND juntos con las señales que especifican el cambio requerido en los elementos de memoria. Las salidas de la compuerta AND pueden transmitir señales solo en los instantes que coinciden con la llegada de los pulsos del reloj. Los circuitos secuenciales síncrono que usan pulso del reloj en las entradas de los elementos de memoria se denomina circuito secuencial del reloj. Los circuitos secuenciales del reloj son de tipo que se encuentran con más frecuencia. No manifiestan problemas de inestabilidad y su temporizado se desglosa fácilmente en pasos discretos e independientes, cada uno de los cuales se considera por separado.

Los elementos de memoria que se usan en los circuitos secuenciales de reloj se llaman flip-flops. Estos circuitos son celdas binarias capaces de almacenar un bit de información. Un circuito flip-flop tienen dos salidas, una para el valor normal y otra para el valor complementario del bit almacenado en él. La información binaria puede entrar aun flip-flop en una gran variedad de formas, hecho que da lujar a diferentes tipos de flip-flops.

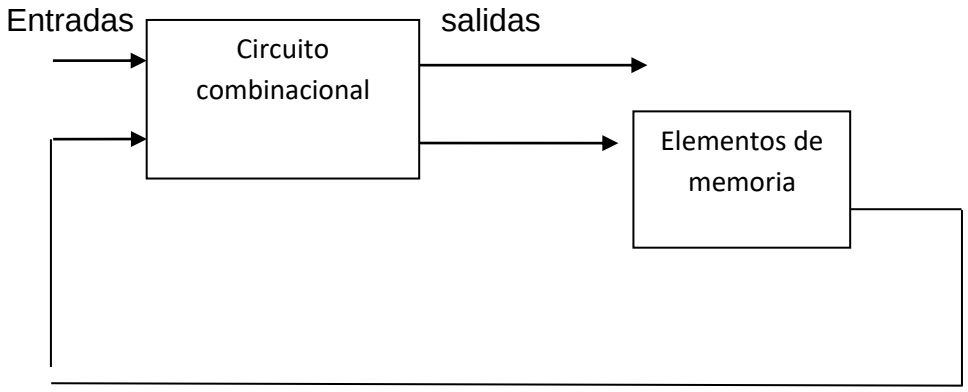


Figura 1. Diagrama de bloques de un circuito secuencial. Fuente: Mano, 1987, p. 206.

Definición de términos

Señal de Reloj**:** es en la electrónica digital una señal binaria, que sirve para coordinar las acciones de varios circuitos, en especial para la sincronización de biestables en sistemas digitales complejos. Según su aplicación, la señal de reloj se puede repetir con una frecuencia predefinida o también ser aperiódica.

Flanco: es la transición del nivel bajo al alto (flanco de subida) o del nivel alto al bajo (flanco de bajada).

Flip-Flops

El flip-flop es el nombre común que se le da a los dispositivos de dos estados (biestables), que sirven como memoria básica para las operaciones de lógica secuencial. Los Flip-flops son ampliamente usados para el almacenamiento y transferencia de datos digitales y se usan normalmente en unidades llamadas “registros”, para el almacenamiento de datos numéricos binarios.

Características

a) Asumen solamente uno de dos posibles estados de salida
b) Tienen un par de salidas que son complemento de la otra
c) Tienen una o más entradas que pueden causar que el estado del flip-flop cambie.

Los flip-flops se pueden dividir en:

a) Asíncronos: solamente tienen entradas de control. El más empleado es el biestable RS.
b) Síncronos: además de entradas de control posee una entrada de sincronismo o de reloj.

Tipos de flip-flops

Flip-flop RS con reloj

El flip-flop básico, es un circuito secuencial asíncrono. Por la adición de compuertas a las entradas del circuito básico, puede hacerse que el flip-flop responda a niveles de entrada durante la ocurrencia de un pulso de reloj. El flip-flop RS con reloj consta de un flip-flop básico NOR y dos compuertas AND. Las salidas de las dos compuertas AND permanecen en 0 en tanto que el pulso del reloj (abreviado CP, de las iniciales en inglés clock pulse) sea 0, sin importar los valores de entrada S y R. Cuando el pulso del reloj va en 1, se permite que la información de las entradas S y R alcancen al flip-flop básico. El estado de ajustes se alcanza son S=1, R=0 y CP=1. Para cambiar el estado despejado, las entradas deben ser S=0, R=1 y CP=1. Tanto con S=1 y R=1, la ocurrencia de un pulso de reloj provoca que ambas salidas momentáneamente a 0. Cuando se elimina el pulso, el estado del flip-flop es de ajuste o la de restaurar del circuito flip-flop básico permanezca en 1 durante un tiempo más prolongado antes de la transición a 0 al fin del pulso.

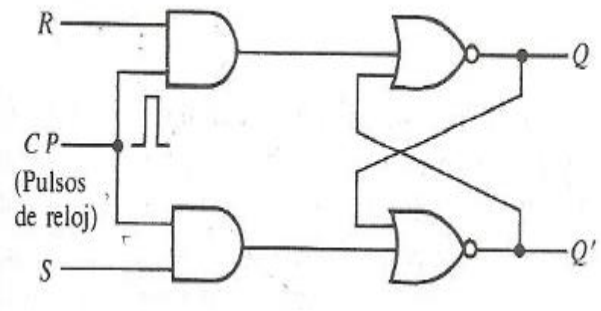


Figura 2. Diagrama lógico de un flip-flop RS. Fuente: Mano, 1987, p. 209

Tabla1.

Tabla de la verdad del flip-flop RS

Q	S	R	Q (t+1)
0	0	1	0
0	0	0	0
0	1	1	1
0	1	0	indeterminado
1	0	1	1
1	0	0	0
1	1	1	1
1	1	0	indeterminado

Fuente: Mano, 1987, p.209

Flip- flop D (Delay)

Es una modificación del flip-flop RS con reloj, es uno de los flip-flops más sencillos y su función es deja pasar lo que entra por D, a la salida Q, después de un pulso del reloj.

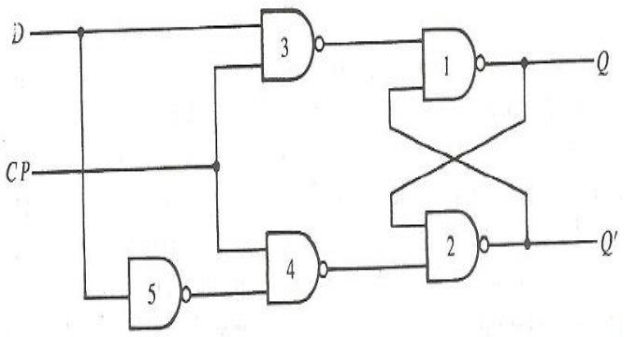


Figura 3. Diagrama lógico de un flip-flop D. Fuente: Mano, 1987, p. 2011

Tabla 2

Tabla de la verdad flip-flop D

Q	D	Q (t+1)
0	0	0
0	1	1
1	0	0
1	1	1

Flip-flop JK

Es un refinamiento del flip-flop RS ya que el estado indeterminado del tipo RS se define en el tipo JK.

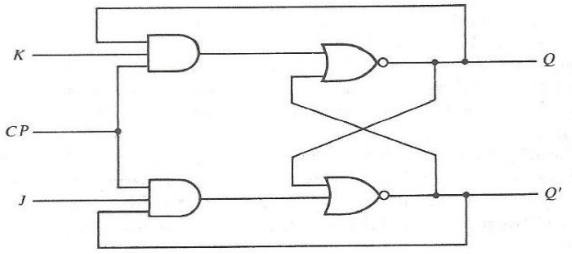


Figura 4. Diagrama lógico de un flip-flop JK. Fuente: Mano, 1987, p. 212

Tabla 3.

Tabla de la verdad del flip-flop JK

Q	J	K	Q (t+1)
0	0	1	0
0	0	0	0
0	1	1	1
0	1	0	1
1	0	1	1
1	0	0	0
1	1	1	1
1	1	0	1

Fuente: Mano, 1987, p.212

Flip-flop T

Es una versión de una sola entrada del flip-flop JK, este tipo de flip-flop se obtiene mediante un tipo JK si ambas entradas se ligan. La denominación T proviene de la capacidad del flip-flop para conmutar (de la inicial del en inglés: toggle), o cambiar de estado.

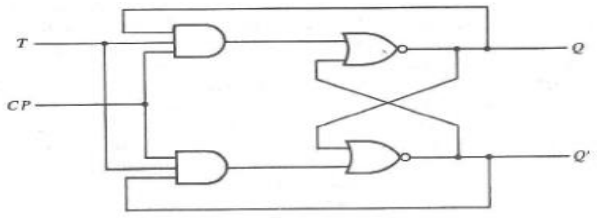


Figura 5. Diagrama lógico de un flip-flop T. Fuente: Mano, 1987, p. 213

Disparo del flip-flop

El estado de un flip-flop se cambia por una modificación momentánea en la señal de entrada. Este cambio momentáneo se denomina gatillo y la transición que provoca se dice que dispara el flip-flop. Los flip-flops asíncronos requieren una entrada de gatillo definida por un cambio de nivel de señal. Este nivel debe volver a su valor inicial antes de aplique un segundo gatillo.

Los flip-flops temporizados se disparan por pulsos. Un pulso comienza desde un valor inicial 0, pasa en forma momentánea a 1 y después de un corto tiempo, regresa a su valor inicial 0. El intervalo de tiempo desde la aplicación del pulso hasta que ocurre la transición de la salida es un factor crítico que requiere más investigación.

Circuitos secuenciales temporizados

El comportamiento de un circuito secuencial determina mediante las entradas, las salidas y los estados de sus flip-flops. Tanto las salidas como el estado siguiente son función de las entradas y el estado presente. El análisis de estos circuitos consiste en obtener una tabla o un diagrama de las secuencias de tiempo de las entradas, salidas y los estados internos. También es posible escribir expresiones booleanas que describen el comportamiento de los circuitos secuenciales. Sin embargo esas expresiones booleanas deben incluir la secuencia de tiempo necesaria ya sea de forma directa o indirecta.

Tabla de estado

La secuencia en tiempo de las entradas, salidas y estados de flip-flop puede enumerarse en una tabla de estado. Una tabla de estado consta de tres secciones etiquetadas (estado presente, estado siguiente y salida). El

estado presente indica los estados de los flip-flops antes de la ocurrencia del pulso del reloj. El estado siguiente muestra los estados de los flip-flops después de la aplicación de un pulso del reloj, y la sección de salida lista los valores las variables de salida durante el estado presente.

Diagrama de estado

La información en una tabla de estado puede representarse en forma gráfica en un diagrama de estado. En este diagrama, un estado se representa con un círculo y la transición entre estados se indica con líneas dirigidas que se conectan los círculos.

RTL (lenguaje de transferencia de registros)

Es una representación intermedia (RI) que es muy cercano al lenguaje ensamblador, ya que es usado en un compilador. Es básicamente una operación usada para transferir información de un jugar a otro.

Una sentencia RTL permite representar en texto un circuito síncrono, por ejemplo las sentencias:

R1<= R1 <<1

R2<= R1+R2

Equivalen al circuito de la siguiente figura.

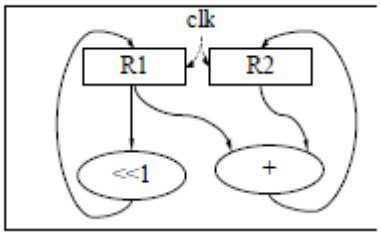


Figura 6. Grafica de sentencias RTL. Fuente: Muñoz, 2002, p. 86

En una sentencia RTL usualmente, se toman los datos de los registros, se operan en un circuito combinacional y el resultado es almacenado en otro registro cuando llega el pulso del reloj.

Introducción a las máquinas de estados finitos

Las máquinas de estado finito son una herramienta muy útil para especificar aspectos relacionados con tiempo real, dominios reactivos o autónomos, computación reactiva, protocolos, circuitos, arquitecturas de software, etc.

Una Maquina de estado Finito es una lógica matemática. Es esencialmente un programa de computadora: representa una secuencia de instrucciones a ser ejecutadas, donde cada instrucción depende del estado actual de la máquina y del actual estimulo.

Máquinas de estados finitos (FSM)

Las máquinas de estado finito, más conocidas por su acrónimo en inglés FSM (Finite State Machine), se utilizan ampliamente en el diseño de circuitos digitales (además de en otros ámbitos de la ingeniería, como la programación), para describir el comportamiento de un sistema según el valor de sus entradas y de cómo van cambiando en el tiempo. Ésta es una definición parcial pero que nos permite hacernos una primera idea intuitiva. Desde el punto de vista de las FSM, un sistema está compuesto de estados por los que va pasando el sistema, de señales de entrada que modifican esos estados y de señales de salida que pueden utilizarse para conocer el estado del sistema y actuar en consecuencia.

El diseño de una FSM involucra lo siguiente:

a) Definición de estados.
b) Definición de transición

c) Optimización/minimización.

Definición de términos

Diagrama de estado: ilustra la forma y funcionamiento de la máquina de estado. Usualmente se dibuja como un diagrama de burbujas y flechas.

Alfabeto: es un conjunto finito de símbolos se denota (∑).

Estados: un estado puede considerarse una descripción abreviada de la situación actual del sistema. A modo de sencillo ejemplo, pensemos en un dispositivo periférico que pueda tener tres modos de funcionamiento alternativos del máximo nivel: on, off, o standby (reposo).

Eventos: un evento es un mensaje de entrada a la máquina de estados. El evento puede provocar que la máquina de estados cambie de estado. La presión del botón On en un televisor en modo standby es un tipo de evento que con un poco de suerte devolverá la vida al televisor, o a estado on.

Transiciones: una transición es una ruta dirigida de un estado a otro. La transición se designa generalmente con un nombre de evento, indicando así que la transición tendrá lugar si el evento especificado sucede cuando la máquina de estados se encuentra en el estado de inicio de la transición. El resultado final es que la máquina de estado al estado buscado por la transición.

Acciones: una acción es una actividad a realizar por la máquina de estados en un punto determinado en el tiempo. Generalmente una acción realiza alguna tarea en el entorno, como la lectura o escritura de un dispositivo de E/S. las condiciones pueden asociarse a transiciones, o bien pueden especificar como acciones de entrada y salida en estados específicos.

Ramificación: El cambio del estado presente al estado siguiente.

Estado siguiente: es el estado hacia el cual la máquina de estado realiza la siguiente transición, determinada por las entradas presentes cuando el dispositivo es secuenciado por un clock.

Las máquinas de estados pueden ser:

Síncronas: necesitan de la intervención de un pulso de reloj. Si la entrada participa también en la salida se denomina máquina de Mealy, y si ni participa se denomina de Moore.

Asíncronas: no necesitan de la intervención de un pulso de reloj. Estos circuitos evolucionan cuando cambian las entradas.

Ventajas de las máquinas de estado finito

a) Son intuitivas y fáciles de entender
b) Abstraen convenientemente detalles secundarios que no son necesarios para el análisis del sistema a un alto nivel y se centran en aspectos claves del mismo.
c) Aportan un componente visual que facilita el análisis y diseño del sistema.
d) Son universalmente aplicables.
e) Su uso es común, un sistema de transición de datos y el uso de protocolos de comunicación.

Máquinas de Mealy y Moore

Máquina de Mealy: es un tipo de máquina de estados finitos que genera una salida basándose en su estado actual y una entrada. Esto significa que el diagrama de estados incluirá ambas señales de entrada y salida para cada línea de transición. Para cada máquina de Mealy hay una máquina de Moore equivalente cuyos estados son la

unión de los estados de la máquina de Mealy y el producto cartesiano de los estados de la máquina de Mealy y el alfabeto de entrada.

Definición formal:

Una máquina de Mealy es una séxtupla, (S, S0, Σ, Λ, T, G), donde:

S es un conjunto de estados finitos
S0 un estado inicial el cual es un elemente de S
Σ es el alfabeto de entrada
Λ es el alfabeto de salida
T=S x Σ → S una función de transición
G: S × Σ → Λ una función de salida

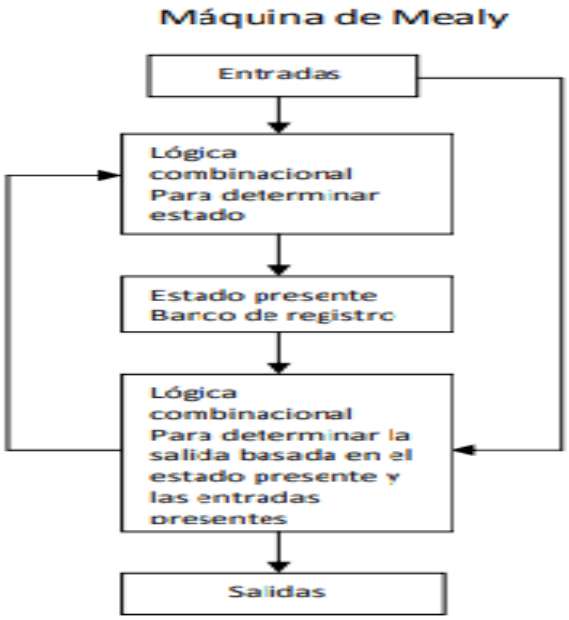


Figura 7. Diagrama de flujo de Mealy. Fuente: Cassanovas, 2014, p.05

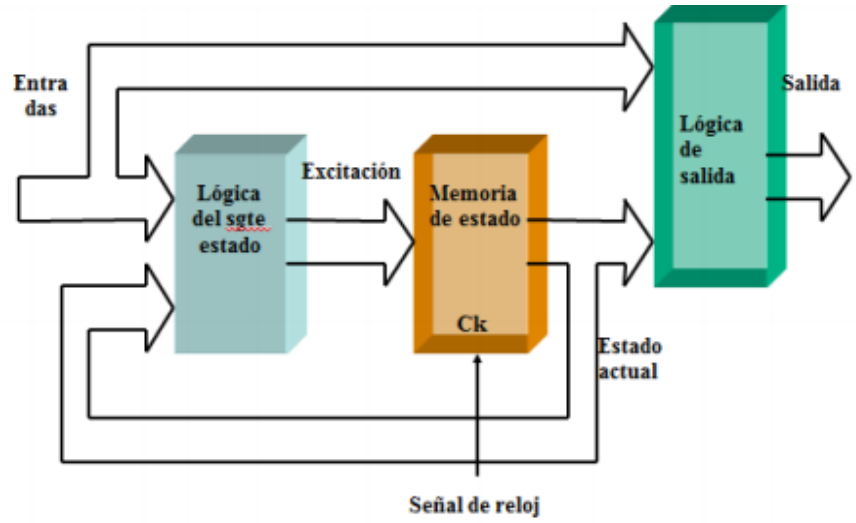


Figura 8. Diagrama general de Mealy. Fuente: Cassanovas, 2014, p.07

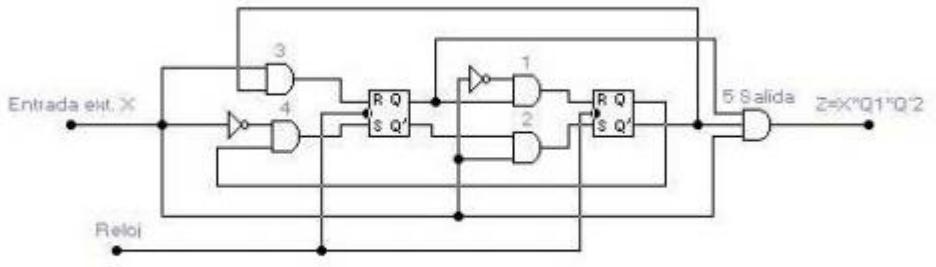


Figura 9. Diagrama a nivel de compuertas y flip-flop de Mealy. Fuente: Cassanovas, 2014, p.07.

La máquina de Mealy genera salidas basada en el estado presente y las entradas a la máquina.

De modo de ser capaz de generar diversos patrones diferentes de señales de salida para el mismo estado, dependiendo de las entradas presentes en el ciclo del reloj.

Las salidas son mostradas sobre las transiciones del momento que están determinadas de la misma manera que el estado siguiente.

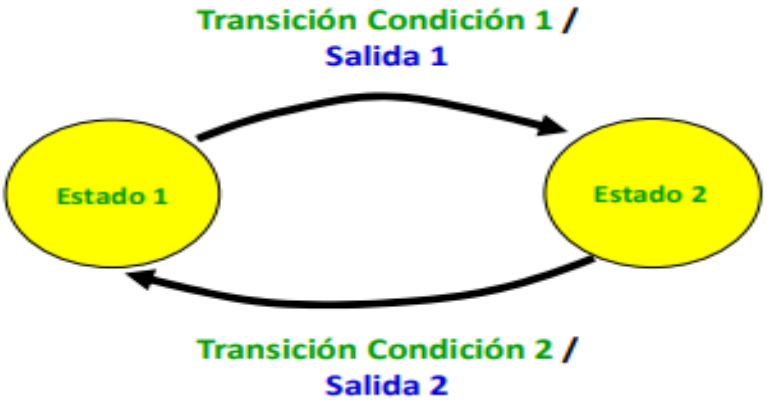


Figura 10. Diagrama de máquina de estado de Mealy. Fuente: Cassanovas, 2014, p. 08.

Máquina de Moore: Las salidas son independientes de las entradas. Las salidas se producen efectivamente desde dentro del estado de la máquina. En una máquina de Moore sus salidas dependen solo y exclusivamente del estado de la máquina.

Definición formal:

Una máquina de Moore es una séxtupla, (S, S0, Σ, Λ, T, G), donde:

S es un conjunto de estados finitos

S0 un estado inicial el cual es un elemente de S

Σ es el alfabeto de entrada

Λ es el alfabeto de salida

T=S x Σ → S una función de transición mapeando un estado y una entrada al siguiente estado.

G: S × Σ → Λ una función de salida mapeando cada estado al alfabeto salida.

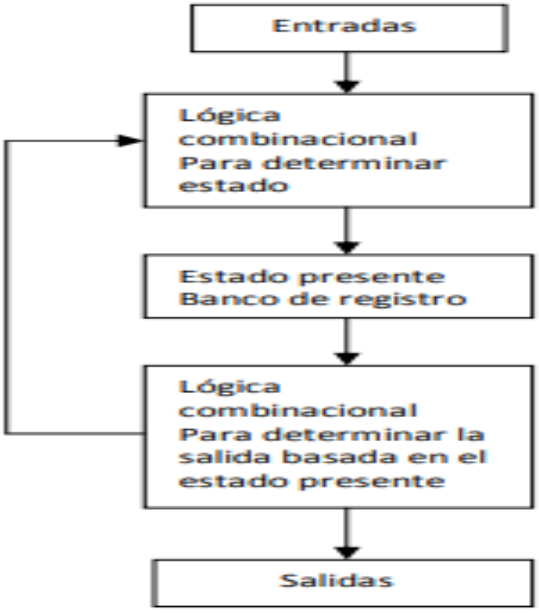


Figura 11. Diagrama de flujo de Moore. Fuente: Cassanovas, 2014, p.05.

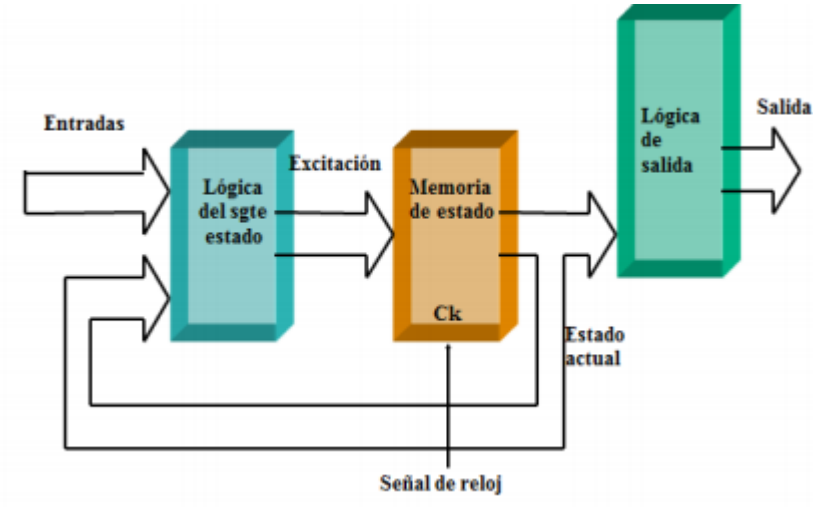


Figura 12. Diagrama general de Moore. Fuente: Cassanovas, 2014, p.06

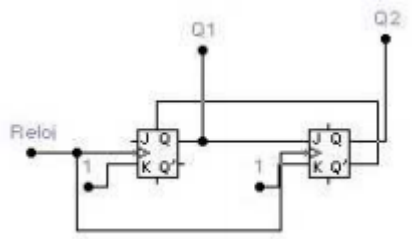


Figura 13. Diagrama a nivel de compuertas y flip-flop de Moore. Fuente: Cassanovas, 2014, p.06.

La salida en una máquina de estado de Moore se muestra dentro de la burbuja de estado, debido a que la salida se mantiene igual, mientras la maquina se mantenga en ese estado. La salida puede ser arbitrariamente compleja pero debe ser la misma cada vez que la maquina entra a esa estado.

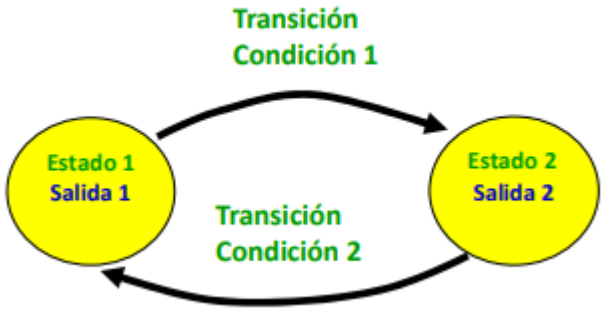


Figura 14. Diagrama de máquina de estado de Moore. Fuente: Cassanovas, 2014, p, 08.

Diferencias entre Moore y Mealy

Las FSM Mealy y Moore pueden ser funcionalmente equivalentes.

Las FSM Mealy tiene una mejor descripción y usualmente requieren de un número menor de estados y menor área de circuito.

Mealy calcula las salidas tan pronto como se produce cambio entradas.

Mealy responde un ciclo del reloj antes que el equivalente de Moore.

Las FSM Moore no posee un camino lógica combinacional entre entradas y salidas.

Introducción a las Maquinas de Estados Algorítmicos (ASM)

La información binaria almacenada en un sistema puede clasificarse ya sea como datos o control de información. Los datos son elementos discretos de información que se manipulan para realzar tareas de aritmética, lógica, corrimiento y otras tareas similares de procesamiento de datos. Estas operaciones se implantan en componentes digitales como sumadores, decodificadores, multiplexores, contadores y registro de corrimiento. La información de control proporciona señales de mano que supervisan diversas operaciones de la sección de datos con objeto de llevar a cabo tareas deseadas de procesamiento de datos. El diseño lógico de un sistema digital puede dividirse en dos partes distintas. Una parte se ocupa del diseño de los circuitos digitales que llevan a cabo las operaciones de procesamiento de datos. La otra parte se ocupa del diseño del circuito de control que supervisan las operaciones y sus secuencias.

El subsistema procesador de datos manipula los datos en los registros de acuerdo con los registros del sistema. El control lógico inicia los mandos en secuencia apropiada al procesador de datos. El control lógico las condiciones de estado del procesador de datos para servir como variables de decisión para determinar la secuencia de las señales de control.

El control lógico que genera las señales para dar la secuencia de las operaciones en el procesador de datos es un circuito secuencial cuyos estados internos dictan los mandos de control del sistema. En cualquier momento el estado de control secuencial inicia un conjunto prescrito de mandos. Dependiendo de las condiciones de estado y otras entradas externas, el circuito secuencial pasa al estado siguiente para iniciar otras operaciones. Los circuitos digitales que actúan como el control lógico

proporcionan una secuencia de tiempo de señales para iniciar las operaciones en el procesador de datos y determinar el siguiente estado del mismo subsistema de control.

La secuencia de control y tareas de procesamiento de datos de un sistema digital se especifican mediante un algoritmo en el hardware. Un algoritmo consta de un número finito de paso de procesamiento que especifican como obtener una solución a un problema. Un algoritmo de hardware es un procedimiento para implantar el problema con una pieza dada de equipo. La parte del diseño digital más creativa y más desafiante es la formulación de algoritmos de hardware para lograr objetivos requeridos.

Una forma conveniente de especificar la secuencia de los pasos de proceso y las trayectorias de decisión para un algoritmo es un diagrama de flujo. El diagrama de flujo para un algoritmo de hardware traduce la estipulación en palabras a un diagrama de información que enumera la secuencia de las operaciones junto con las condiciones necesarias para su ejecución. Un diagrama especial de flujo que ha sido desarrollado específicamente para definir algoritmos de hardware digitales se denomina diagrama de máquina de estado algorítmico (ASM). Una máquina de estado es otro término para un circuito secuencial, el cual es la estructura básica de un sistema digital.

El diagrama ASM se asemeja a un diagrama convencional de flujo pero se interpreta en formal algo diferente. Un diagrama convencional de flujo describe la secuencia de los pasos de procesamiento y las trayectorias de decisión para un algoritmo sin ocuparse de sus relaciones en el tiempo. El diagrama ASM describe la secuencia de eventos lo mismo que las relaciones de temporizado entre los estados de un contralados secuencial y los eventos que ocurren cuando pasa de u estado al siguiente. En forma específica está adaptado para especificar con precisión la secuencia de control y las

operaciones de procesamientos de datos en un sistema digital, tomando en consideración las restricciones del hardware digital.

En esta investigación se presenta una metodología de diseño que usa el diagrama ASM para hacer más fácil la programación en los lenguajes de descripción de hardware. Los diversos que componen el diagrama se definen primero y las relaciones de temporizado entre los bloques se explican entonces mediante ejercicios.

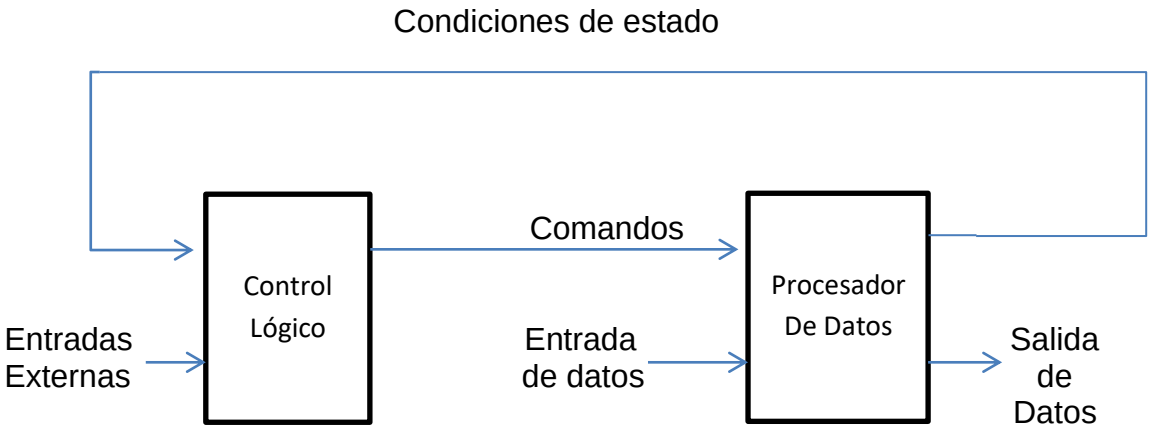


Figura 15. Interacción de control y el procesador de datos. Fuente: Mano, 1987, p.312.

Máquina de estados algorítmica (ASM)

Las máquinas de estado algorítmicas (ASM) es un método para el diseño de máquinas de estados finitos y se utiliza para representar diagramas de circuitos integrados digitales.

Diagrama o carta ASM

El diagrama ASM es un tipo especial de diagrama de flujo adecuado para describir las operaciones secuenciales en un sistema digital, es decir, es una descripción gráfica del desarrollo de instrucciones en microoperaciones.

Un diagrama ASM está compuesto por tres elementos básicos los cuales son:

Casilla o bloque de decisión: en este bloque de estado representa el “estado” de una maquina secuencial, tiene forma rectangular y debe contener la siguiente información:

Nombre del estado, este se escribe en la parte superior izquierda de la casilla y por lo general se utilizan números (0, 1, 2, 3,4….etc.) o letras (A, B, C…. etc.,).

Código de del estado (“xxxx”), se refiere al código binario asignado al estado y este se coloca en la parte superior derecha de la casilla.

Lista de salidas, estas son señales de salida asignada al estado y que solo se encuentran activos durante el tiempo que permanezca el sistema en ese estado y se coloca dentro de la casilla.

Casilla o bloque de decisión: esta casilla escribe el efecto de una entrada en el subsistema de control. Es una casilla con forma de rombo con dos o más trayectorias de salida. La condición de entrada que va a probarse está escrita dentro de la casilla. Una trayectoria de salida se toma si la condición es cierta y la otra cuando la condición es falsa. Cuando una condición de entrada es asignada a un valor binario las dos trayectorias se indican por 0 y 1.

Casilla o bloque condicional: es de forma ovalada esto las diferencias de la casilla de estado. La trayectoria de entrada a la casilla condicional debe llegar desde una de las trayectorias de salida de una casilla de decisión. Las operaciones de registro o salidas listadas dentro de una casilla condicional se generan durante un estado dado siempre que se satisfaga la condición de entrada.

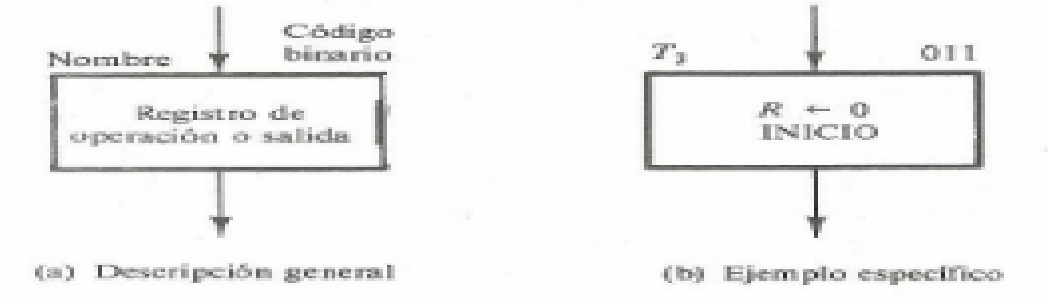


Figura 16.Caja de estado. Fuente: Mano, 1987, p.314

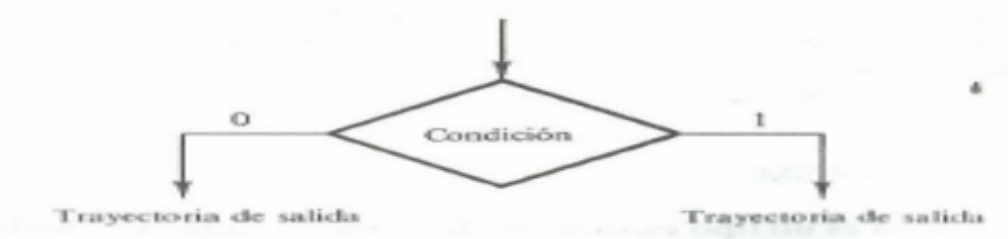


Figura 17. Cada de decisión. Fuente: Mano, 1987, p.314

Figura 18. Cada de condición. Fuente: Mano, 1987, p.315

Bloque ASM

Un bloque ASM es una estructura que consta de una casilla de estado y todas casillas de decisión y condicionales conectadas a sus trayectorias de salida. Un bloque ASM, tiene una entrada y cualquier número de trayectorias

de salidas representadas por la estructura de las casillas de decisión. Un diagrama consta de uno o más bloques interconectados.

Cada bloque en el diagrama ASM describe el estado del sistema durante el intervalo de un pulso de reloj, es decir, un bloque ASM se ejecuta en 1 ciclo de reloj.

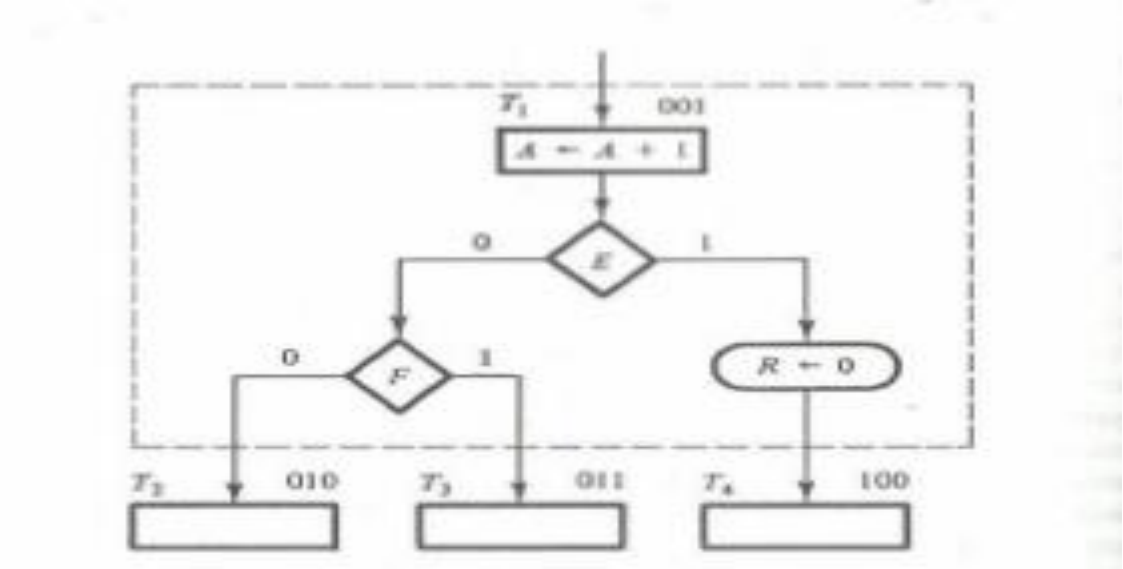


Figura 19. Bloque ASM. Fuente: Mano, 2008, p.316

Operaciones de registro

Un sistema digital con bastante frecuencia se define por los registros que contiene y las operaciones que se realizan en los datos almacenados en ellos. Un registro en su sentido más amplio incluye registros de almacenamiento, registros de corrimiento, contadores y flip-flops solos. Los ejemplos de operaciones de registro son corrimiento, incremento, adición, despeje y transferencia de datos. Algunas veces es conveniente adoptar una notación adecuada para describir las operaciones llevadas a cabo entre los registros.

Un registro se indica por una o más letras mayúsculas como A, B, C o RA. Las celdas individuales o flip-flops dentro de un registro de n-bits se

numeran en secuencia desde 1 a n o desde 0 a n-1. Un flip-flop solo se considera como un registro de 1-bit.

La transferencia de datos de un registro a otro se simboliza por una flecha dirigida que denota transferencia de contenido de un registro fuente a un registro de destino. La operación de despejar registro se simboliza por una transferencia de 0 al registro.

Un flip-flop solo puede despejarse establecerse en 1 o despejarse en 0. Para incrementar un registro en 1q, es necesario que el registro sea capaz de contar hacia arriba como en un contador binario. La operación de decremento requiere un contador descendente.

El contenido de dos registros puede agregarse mediante un circuito sumador. Algunas operaciones, como por ejemplo la operación de corrimiento, no tienen símbolo conocido. En tal caso usan las palabras “corrimiento a la derecha R” para denotar un corrimiento a la derecha del registro R.

Tabla4.

Notación simbólica para las operaciones de registro

Notación Simbólica	Descripción
a) A← B	a) Transferencia del contenido del registro B al registro A
b) R← 0	b) Despejar el registro R
c) F← 1	c) Establecer el flip-flop en 1
d) A← A + 1	d) Incrementar el registro A en 1(cuenta arriba)
e) A← A – 1	e) Disminuir el registro A en 1 (cuenta abajo)
f) A← A + B	f) Agregar el contenido del registro B al registro A

Fuente: Mano, p. 317

Consideraciones de temporizado

El temporizado de todos los registros y flip-flops en un sistema digital se controla por un reloj generador maestro. Los pulsos de reloj se aplican no solo a los registros de la subsección del procesador de datos, sino también a todos los flip-flops en el circuito lógico. Las entradas también están sincronizadas con los pulsos del reloj porque normalmente se generan como salidas de otro circuito que usa las mismas señales de reloj. Si la señal de entrada cambia en un tiempo arbitrario independientemente del reloj se le llama entrada asíncrona. Para simplificar el diseño se supone que todas las entradas están sincronizadas con el reloj y que cambian de estado como respuesta a una transición de borde de pulso del reloj. En forma similar, cualquier salida que es una función del estado presente y una entrada síncrona también estarán sincronizadas.

Lenguajes de descripción de hardware (HDLs)

Como consecuencia de la creciente necesidad de integrar un mayor número de dispositivos en un solo circuito integrado, se desarrollaron nuevas herramientas de diseño que auxilian al ingeniero a integrar sistemas de mayor complejidad. Esto permitió que en la década de los cincuenta aparecieran los lenguajes de descripción de hardware (HDL) como una opción de diseño para el desarrollo durante los años setenta, lapso en que se desarrollaron varios de ellos como IDL de IBM, TI-HDL de Texas Instrumens, ZEUS de General Electric, etc., todos orientados al área industrial, así como los lenguajes en el ámbito universitario (AHPL, DDL, CDL, ISPS, etc.) los primeros no estaban disponibles fuera de la empresa que los manejaba, mientras que los segundos carecían de soporte y mantenimiento adecuados que permitieran su utilización industrial. El desarrollo continuó y en la década de los ochenta surgieron lenguajes como VHDL, Verilog, ABEL 5.0, AHDL, etc., considerados lenguajes de descripción en hardware porque permitieron abordar un problema lógico a nivel funcional (descripción de un problema

solo conociendo las entradas y salidas), lo cual facilita la evaluación de soluciones alternativas antes de iniciar un diseño detallado.

Una de las principales características de estos lenguajes radica en su capacidad para describir en distintos niveles de abstracción (funcional, transferencia de registros RTL y lógico o nivel de compuertas) cierto diseño. Los niveles de abstracción se emplean para clasificar modelos HDL según el grado de detalle y precisión de sus descripciones.

Los niveles de abstracción descritos desde el punto de vista de simulación y síntesis del circuito pueden definirse como sigue:

a) Algoritmo: se refiere a la relación funcional entre entradas y salidas del circuito o sistema, sin hacer referencia a la realización final.
b) Transferencia de registros (RTL): consiste en la partición del sistema en bloques funcionales sin considerar a detalle la realización final de cada bloque.
c) Lógico o de compuertas: el circuito se expresa en términos de ecuaciones o lógicas o de compuertas.

Estos lenguajes son sintéticamente similares a los de programación de alto nivel por ejemplo Verilog tiene una sintaxis similar a la de C y VHDL a ADA y se diferencian de estos en que su semántica está orientada al modelado del hardware. Su capacidad para permitir distintos enfoques en el modelado de los circuitos y su independencia de la tecnología y metodología de diseño permiten extender su uso a los distintos ciclos de diseño que puedan realizarse. Por ellos, para los profesionales relacionados de alguna manera con el diseño o mantenimiento de sistemas digitales resulta hoy en día imprescindible su conocimiento.

Introduccion al lenguaje VHDL

En la actualidad, el lenguaje de descripción en hardware más utilizado a nivel industrial es VHDL (Hardware Description Language), que apareció en la década de los ochenta como un lenguaje estándar, capaz de soportar el proceso de diseño de sistemas electrónicos complejos, con propiedades para reducir el tiempo de diseño y los recursos tecnológicos requeridos. El Departamento de Defensa de Estados Unidos creó el lenguaje VHDL como parte del programa "Very High Speed Integrated Circuits" (VHSIC), a partir del cual se detectó la necesidad de contar con un medio estándar de comunicación y la documentación para analizar la gran cantidad de datos asociados para el diseño de dispositivos de escala y complejidad deseados; es decir, VHSIC debe entenderse como la rapidez en el diseño de circuitos integrados. Después de varias versiones revisadas por el gobierno de los Estados Unidos, industrias y universidades, el IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) publicó en diciembre de 1987 el estándar IEEEstd 1076-1987. Un año más tarde, surgió la necesidad de describir en VHDL todos los ASIC creados por el Departamento de Defensa, por lo que en 199.3 se adoptó el estándar adicional de VHDL IEEE1164. Hoy en día VHDL se considera como un estándar para la descripción, modelado y síntesis de circuitos digitales y sistemas complejos. Este lenguaje presenta diversas características que lo hacen uno de los HDL más utilizados en la actualidad.

Vhdl

Es un lenguaje orientado a la descripción o modelado de sistemas digitales; es decir, se trata de un lenguaje mediante el cual se puede describir, analizar y evaluar el comportamiento de un sistema electrónico digital.

VHDL es un lenguaje poderoso que permite la integración de sistemas digitales sencillos, elaborados o ambos en un dispositivo lógico programable, sea de baja capacidad de integración como un GAL, o de mayor capacidad como los CPLD y FPGA.

Ventajas del desarrollo de circuitos integrados con VHDL

A continuación se exponen algunas de las ventajas que representan los circuitos integrados con VHDL:

a) Notación formal: Los circuitos integrados VHDL cuentan con una notación que permite su uso en cualquier diseño electrónico.
b) Disponibilidad pública: VHDL es un estándar no sometido a patente o marca registrada alguna, por lo que cualquier empresa o institución puede utilizarla sin restricciones. Además, dado que el IEEE lo mantiene y documenta, existe la garantía de estabilidad y soporte
c) Independencia tecnológica de diseño: VHDL se diseñó para soportar diversas tecnologías de diseño (PLD, FPGA, ASIC, etc.) con distinta funcionalidad (circuitos combinacionales, secuenciales, síncronos y asíncronos), a fin de satisfacer las distintas necesidades de diseño.
d) Independencia de la tecnología y proceso de fabricación: VHDL se creó para que fuera independiente de la tecnología y del proceso de fabricación del circuito o del sistema electrónico. El lenguaje funciona de igual manera en circuitos diseñados con tecnología MOS, bipolares, BICMOS, etc., sin necesidad de incluir en el diseño información concreta de la tecnología utilizada o de sus características (retardos, consumos, temperatura, etc.), aunque esto puede hacerse de manera opcional.
e) Capacidad descriptiva en distintos niveles de abstracción: El proceso de diseño consta de varios niveles de detalle, desde la especificación hasta la implementación final (niveles de abstracción). VHDL ofrece la ventaja de poder diseñar en cualquiera de estos niveles y combinarlos, con lo cual se genera lo que se conoce como simulación multinivel.

f) Uso como formato de intercambio de información: VHDL permite el intercambio de información a lo largo de todas las etapas del proceso de diseño, con lo cual favorece el trabajo en equipo.
g) Independencia de los proveedores: Debido a que VHDL es un lenguaje estándar, permite que las descripciones o modelos generados en un sitio sean accesibles desde cualquier otro, sean cuales sean las herramientas de diseño utilizadas.
h) Reutilización del código: El uso de VHDL como lenguaje estándar permite reutilizar los códigos en diversos diseños, sin importar que hayan sido generados para una tecnología (CMOS, bipolar, etc.) e implementación (FPGA, ASIC, etc.) en particular.
i) Facilitación de la participación en proyectos internacionales: En la actualidad VHDL constituye el lenguaje estándar de referencia a nivel internacional. Impulsado en sus inicios por el Departamento de Defensa de Estados Unidos, cualquier programa lanzado por alguna de las dependencias oficiales de ese país vuelve obligatorio su uso para el modelado de los sistemas y la documentación del proceso de diseño. Este hecho ha motivado que diversas empresas y universidades adopten a VHDL como su lenguaje de diseño.

Desventajas del desarrollo de circuitos integrados con VHDL

Como se puede observar, VHDL presenta grandes ventajas; sin embargo, es necesario mencionar también algunas desventajas que muchos diseñadores consideran importantes:

a) En algunas ocasiones, el uso de una herramienta provista por alguna compañía en especial tiene características adicionales al lenguaje, con lo que se pierde un poco la libertad de diseño. Como método alternativo, se pretende que entre diseñadores que utilizan distintas herramientas exista una compatibilidad en

sus diseños, sin que esto requiera un esfuerzo importante en la traducción del código.

b) Debido a que VHDL es un lenguaje diseñado por un comité, presenta una alta complejidad, ya que se debe dar gusto a las diversas opiniones de los miembros de éste, por lo que resulta un lenguaje difícil de aprender para un novato.

Unidades básicas de diseño

La estructura general de un programa en VHDL está formada por módulos o unidades de diseño, cada uno de ellos compuesto por un conjunto de declaraciones e instrucciones que definen, describen, estructuran, analizan y evalúan el comportamiento de un sistema digital.

Existen cinco tipos de unidades de diseño en VHDL: declaración de entidad (entity declaration), arquitectura (architecture), configuración (configuration), declaración del paquete (package declaration) y cuerpo del paquete (package body). En el desarrollo de programas en VHDL pueden utilizarse o no tres de los cinco módulos, pero dos de ellos (entidad y arquitectura) son indispensables en la estructuración de un programa.

Las declaraciones de entidad, paquete y configuración se consideran unidades de diseño primarias, mientras que la arquitectura y el cuerpo del paquete son unidades de diseño secundarias porque dependen de una entidad primaria que se debe analizar antes que ellas.

Entidad

Una entidad (entity) es el bloque elemental de diseño en VHDL, Las entidades son todos los elementos electrónicos (sumadores, contadores, compuertas, flip-flops, memorias, multiplexores, etc.) que forman de manera individual o en conjunto un sistema digital. La entidad puede representarse de muy diversas maneras; por ejemplo, la arquitectura de un sumador completo a nivel de compuertas; ahora bien, esta entidad se puede

representar a nivel de sistema indicando tan sólo las entradas (Cin, A y B) y salidas (SUMA y Cout) del circuito. De igual forma, la integración de varios subsistemas (medio sumador) puede representarse mediante una entidad. Los subsistemas pueden conectarse internamente entre sí; pero la entidad sigue identificando con claridad sus entradas y salidas generales.

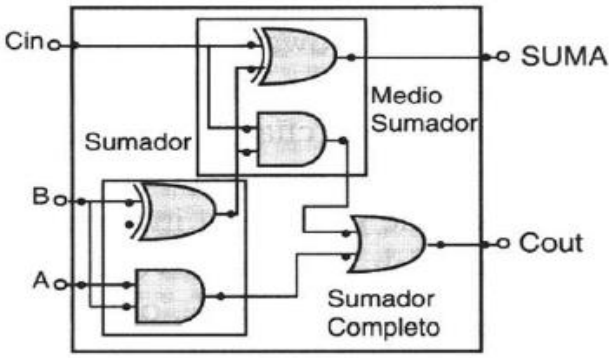
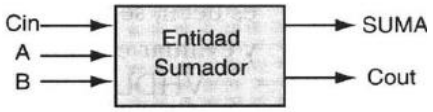


Figura 20. Descripción a nivel de compuertas de una entidad. Fuente: Maxines y Alcalá, 2002, p.48



. **Figura 21. Símbolo funcional de la entidad.** Fuente: Maxines y Alcalá, 2002, p.48

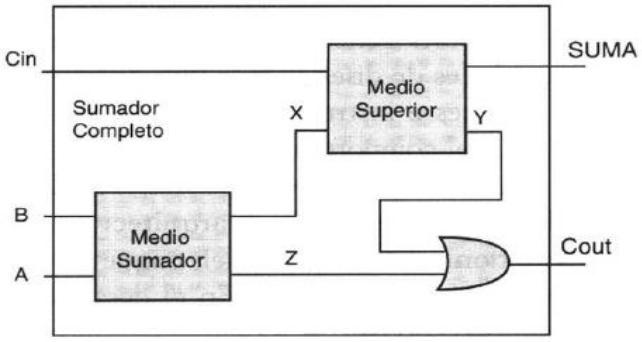


Figura 22. Diagrama a bloques representativo de la entidad. Fuente: Maxines y Alcalá, 2002, p.48.

Puertos de entrada y salida

Cada una de las señales de entrada y salida en una entidad son referidas como puerto, el cual es similar a una terminal (pin) de un símbolo esquemático. Todos los puertos que son declarados deben tener un nombre, un modo y un tipo de dato. El nombre se utiliza como una forma de llamar al puerto; el modo permite definir la dirección que tomará la información y el tipo define qué clase de información se transmitirá por el puerto. Por ejemplo, respecto a los puertos de la entidad que representan a un comparador de igualdad, las variables a y b denotan los puertos de entrada y la variable c se refiere al puerto de salida.

Figura 23. Comparador de igualdad. Fuente: Maxines y Alcalá, 2002, p.49.

Modos

Como ya se mencionó, un modo permite definir la dirección en la cual el dato es transferido a través de un puerto. Un modo puede tener uno de cuatro valores: in (entrada), out (salida), inout (entrada/salida) y buffer.

a) **Modo in:** Se refiere a las señales de entrada a la entidad. Este sólo es unidireccional y nada más permite el flujo de datos hacia dentro de la entidad.
b) **Modo out:** Indica las señales de salida de la entidad.
c) **Modo inout:** Permite declarar a un puerto de forma bidireccional, es decir, de entrada/salida; además permite la retroalimentación de señales dentro o fuera de la entidad.
d) **Modo buffer:** Permite hacer retroalimentaciones internas dentro de la entidad, pero a diferencia del modo inout, el puerto declarado se comporta como una terminal de salida.

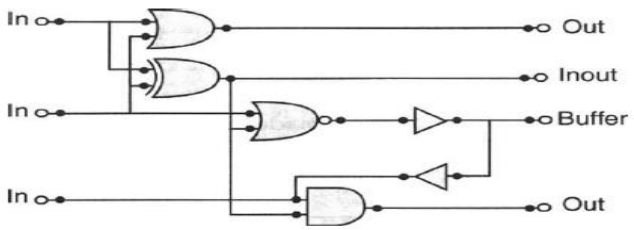


Figura 24. Modos y el curso de sus señales. Fuente: Maxines y Alcalá, 2002, p.49.

Tipos de datos

Los tipos son los valores (datos) que el diseñador establece para los puertos de entrada y salida dentro de una entidad; se asignan de acuerdo con las características de un diseño en particular.

Para decidir el tipo de datos que se debe utilizar hay que evaluar, fundamentalmente, dos factores:

1. El nivel de abstracción del modelo.
2. Las características del dispositivo que se modela.

Además, se debe tener en cuenta que VHDL es un lenguaje con reglas estrictas de asignación de valores: a un objeto sólo se le pueden asignar valores del tipo sobre el que está definido o el valor de un objeto de su mismo tipo.

Algunos de los tipos más utilizados en VHDL son:

Bit: el cual tiene valores de 0 y 1 lógico.

Boolean (booleano): que define valores de verdadero o falso en una expresión.

Bit_vector (vectores de bits): que representa un conjunto de bits para cada variable de entrada o salida.

Integer (entero): que representa un número entero.

Character: formado por los 128 caracteres ASCII.

Severety Level: formado por los valores **(NOTE, WARNING, ERROR,**

FAILURE).

Time: Definido para especificar unidades de tiempo.

String: Definido como un array de caracteres.

Tipos para operaciones de entrada/salida sobre ficheros:

Line: Puntero a **STRING**.

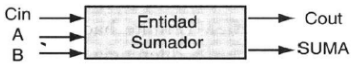
Text: Fichero de **STRINGS.**

Side: Enumerado sobre los valores **(RIGHT, LEFT).**

Declaración de entidades

Como se mencionó en la sección (Unidades básicas de diseño), los módulos elementales en el desarrollo de un programa dentro del lenguaje de descripción en hardware (VHDL) son la entidad y la arquitectura.

La declaración de una entidad consiste en la descripción de las entradas y salidas de un circuito de diseño identificado como entity (entidad); es decir, la declaración señala las terminales o pines de entrada y salida con que cuenta la entidad de diseño.




```
1       --Declaración de la entidad de un circuito sumador
2       entity sumador is
3       port    (A, B, Cin:    in bit;
4                SUMA, Cout: out bit);
5       end sumador;
```

Figura 25. Declaración de la entidad sumador de la figura 21. Fuente: Maxines y Alcalá,2002 p. 50.

Identificadores

Los identificadores son simplemente los nombres o etiquetas que se usan para referir variables, constantes, señales, procesos, etc. Pueden ser números, letras del alfabeto y guiones bajos (_) que separen caracteres y no tienen una restricción en cuanto a su longitud. Todos los identificadores

deben seguir ciertas especificaciones o reglas para que se puedan compilar sin errores.

Tabla 5.
Especificaciones para la escritura de identificadores

Regla	Incorrecto	Correcto
- El primer carácter es una palabra mayúscula o minúscula.	4suma	Suma4
- El segundo carácter no puede ser guion bajo.	S_4bits	S4_bits
- Dos guiones juntos no son permitidos.	Resta __4	Resta_4
- Un identificador no puede utilizar símbolos.	Clear#8	Clear_8

Fuente: Maxinez y Alcalá, p. 52

Diseño de entidades mediante vectores

La entidad sumador mencionada antes, usa bits individuales, los cuales sólo pueden representar dos valores lógicos (0 o 1). De manera general, en la práctica se utilizan conjuntos (palabras) de varios bits; en VHDL las palabras binarias se conocen como vectores de bits, los cuales se consideran un grupo y no como bits individuales. Como ejemplo considérense los vectores de 4 bits que se muestran a continuación:

vector_A = [A3, A2, A1, A0]
vector_B = [B3, B2, B1, B0]
vector_SUMA = [S3, S2, S1, S0]

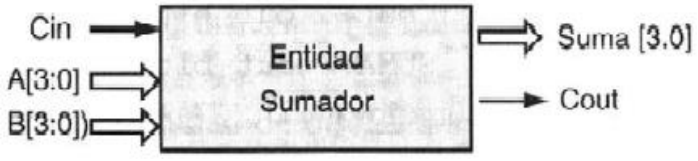


Figura 26. Entidad representada por vectores. Fuente: Maxinez y Alcalá, 2002, p.53

La manera de describir en VHDL una configuración que utilice vectores consiste en la utilización de la sentencia bit_vector, mediante la cual se especifican los componentes de cada uno de los vectores utilizados. La parte del código que se usa para declarar un vector dentro de los puertos es el siguiente:

port (vector_A, vector_B: in bit_vector (3 downto 0);

vector_SUMA: out bit_vector (3 downto 0));

Esta declaración define los vectores (A, B y SUMA) con cuatro componentes distribuidos en orden descendente por medio del comando:

3 downto 0 (3 hacia 0).

Una vez que se ha establecido el orden en que aparecen los bits enumerados en cada vector, no se puede modificar, a menos que se utilice el comando to que indica el orden de aparición en sentido ascendente.

Oto 3 (0 hasta 3).

Declaración de entidades mediante librerías y paquetes

Una parte importante en la programación con VHDL radica en el uso de librerías y paquetes que permiten declarar y almacenar estructuras lógicas, seccionadas o completas que facilitan el diseño.

Una librería o biblioteca es un lugar al que se tiene acceso para utilizar las unidades de diseño predeterminadas por el fabricante de la herramienta (paquete) y su función es agilizar el diseño. En VHDL se encuentran definidas dos librerías llamadas ieee y work. En la librería ieee se encuentra el paquete std_logic_l 164, mientras que en la librería work se hallan numeric_std, std_arith y gatespkg.

Figura 27. Contenido de las librerías ieee y work. Fuente: Maxinez y Alcalá, 2002, p.55.

En una librería también se permite almacenar el resultado de la compilación de un diseño, con el fin de utilizar en uno o varios programas. La librería work es el lugar establecido donde se almacenan los programas que el usuario va generando. Esta librería se encuentra siempre presente en la compilación de un diseño y los diseños se guardan en ella mientras no se especifique otra. La carpeta “otra” representa esta situación.

Cuando en el diseño se utiliza algún paquete es necesario llamar a la librería que lo contiene, para esto se utiliza la declaración (library ieee;), esto permite el uso de todos los componentes incluidos en la librería ieee. En el caso de la librería trabajo (work), su uso no requiere la declaración library, dado que la carpeta work siempre está presente al desarrollar un diseño.

Paquetes

Un paquete es una unidad de diseño que permite desarrollar un programa en VHDL de una manera ágil, debido a que contiene algoritmos preestablecidos (sumadores, restadores, contadores, etc.) que ya tienen optimizado su comportamiento. Por esta razón, el diseñador no necesita caracterizar paso a paso una nueva unidad de diseño si ya se encuentra almacenada en algún paquete en cuyo caso basta con llamarla y especificarla en el programa. Por lo tanto, un paquete no es más que una unidad de diseño formada por declaraciones, programas, componentes y subprogramas, que incluyen los diversos tipos de datos (bit, booleano,

std_logic_l), empleados en la programación en VHDL y que suelen formar parte de las herramientas en software.

El acceso a la información contenida en un paquete es por medio de la sentencia use, seguida del nombre de la librería y del paquete (use nombre_libreria.nombre_paquete.all;).

Ejemplo: use ieee.std_logic_1164.all;

Donde ieee es la librería, std_logic_1164 es el paquete y la palabra reservada all indica que se pueden usar todos los componentes almacenados en el paquete.

A continuación se nombraran los tipos de paquetes que utiliza VHDL:

a) El paquete std_logic_1164 (estándar lógico_1164) que se encuentra en la librería ieee, contiene todos los tipos de datos que suelen emplearse en VHDL (std_logic_vector, std_logic, entre otros).
b) El paquete numeric_std define funciones para realizar operaciones entre diferentes tipos de datos (sobrecargado); además, los tipos pueden representarse con signo o sin este.
c) El paquete std_arith define funciones y operadores aritméticos, como igual (=), mayor que (>), menor que (<), entre otros.

Arquitecturas

Una arquitectura (architecture) se define como la estructura que describe el funcionamiento de una entidad, de tal forma que permita el desarrollo de los procedimientos que se llevarán a cabo con el fin de que la entidad cumpla las condiciones de funcionamiento deseadas.

La gran ventaja que presenta VHDL para definir una arquitectura radica en la manera en que pueden describirse los diseños; es decir, mediante el algoritmo de programación empleado se puede describir desde el nivel de compuertas hasta sistemas complejos.

De manera general, los estilos de programación utilizados en el diseño de arquitecturas se clasifican como:

a) Estilo funcional
b) Estilo por flujo de datos
c) Estilo estructural

El nombre a estos estilos no es importante, ya que es tarea del diseñador escribir el comportamiento de un circuito utilizando uno u otro estilo que a su juicio le sea más acertado.

Descripción funcional

En una descripción funcional se expone la forma en que trabaja el sistema; es decir, las descripciones consideran la relación que hay entre las entradas y las salidas del circuito, sin importar como esté organizado en su interior.

si a = b entonces c = 1
si a ≠ b entonces c = 0

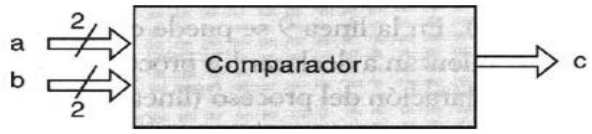


Figura 28. Descripción funcional de un comparador de igualdad de 2 bits. Fuente: Maxinez y Alcalá, 2002, p.57.

La descripción funcional se basa principalmente en el uso de procesos y de declaraciones secuenciales, las cuales permiten modelar la función con rapidez.

```
1       -- Ejemplo de una descripción funcional
2       library ieee;
3       use ieee.std_logic_1164.all;
4       entity comp is
5       port (a,b: in bit_vector( 1 downto 0);
6               c: out bit);
7       end comp;
8       architecture funcional of comp is
9       begin
10      compara: process (a,b)
11      begin
12            if a = b then
13                 c <='1';
14            else
15                 c<='0';
16      end if;
17      end process compara;
18      end funcional;
```

Figura 29. Arquitectura funcional de un comparador de igualdad de 2 bits. Fuente: Maxinez y Alcalá, 2002, p.58.

Descripción por flujo de datos

La descripción por flujo de datos indica la forma en que los datos se pueden transferir de una señal a otra sin necesidad de declaraciones secuenciales (if-then-else). Este tipo de descripciones permite definir el flujo que tomarán los datos entre módulos encargados de realizar operaciones. En este tipo de descripción se pueden utilizar dos formatos: mediante instrucciones when-else (cuando-si no) o por medio de ecuaciones booleanas.

En VHDL se manejan dos tipos de declaraciones: secuenciales y concurrentes. Una declaración secuencial de la forma if-then-else se halla dentro del proceso, donde su ejecución debe seguir un orden para evitar la pérdida de la lógica descrita. En cambio, en una declaración concurrente esto no es necesario, ya que no importa el orden en que se ejecutan.

```
1  --Ejemplo de declaración de la entidad de un comparador
2  library ieee;
3  use ieee.std_logic_1164.all ;
4  entity comp is
5  port  (a,b:   in bit_vector  (1 downto 0);
6           c:   out bit);
   end comp;
7
8  architecture f_datos of comp is
9  begin
10 c <= '1' when  (a = b)   else '0';  (asigna a C el valor
11 de 1 cuando a=b si no vale 0).
12 end f_datos;
```

Figura 30. Arquitectura por flujos de datos. Fuente: Maxinez y Alcalá, 2002, p.60.

Descripción Estructural

Esta descripción basa su comportamiento en modelos lógicos establecidos (compuertas, sumadores, contadores, etc.). El usuario puede diseñar estas estructuras y guárdalas para su uso posterior o tomarlas de los paquetes contenidos en las librerías de diseño del software que se esté utilizando.

Cada compuerta (modelo lógico) se encuentra dentro del paquete gatespkg, del cual se toman para estructurar el diseño. A su vez, este tipo de arquitecturas estándares se conoce como componentes, que al interconectarse por medio de las señales internas permiten proponer una solución. En VHDL esta conectividad se conoce como nelist o listado de componentes.

Para iniciar la programación de una entidad de manera estructural, es necesario la descomposición lógica del diseño en pequeños submodulos (jerarquizar), los cuales permiten analizar de manera practica el circuito, ya que la función de entrada/salida es conocida.

Es importante resaltar que una jerarquía en VHDL se refiere al procedimiento de dividir en bloques y no a que un bloque tenga mayor

jerarquía (peso) que otro. Esta forma de dividir el problema hace de la descripción estructural una forma sencilla de programar. En el contexto del diseño lógico esto es observable cuando se analiza por separado alguna sección de un sistema integral.

```
1  library ieee;
2  use  ieee.std_logic_1164.all;
3  entity comp is port (
4          a,b:  in bit_vector  (0  to  1);
5            c:  out bit);
6  end comp;
7  use work.compuerta.all ;
8  architecture estructural  of  comp  is
9  signal x:  bit_vector  (0  to  1);
10 begin
11    U0:  xnor2      port map  (a(0),  b(0),x(0)
12    Ul:  xnor2      port map  (a(l),  b(1),x(l)
13    U2:  and2       port map  (x(0),x(l),  c);

14 end estructural;
```

Figura 31. Descripción estructural de un comparador de igualdad de 2 bits. Fuente: Maxines y Alcalá, 2002, p.64.

Comparación entre los estilos de diseño

El estilo de diseño utilizado en la programación del circuito depende del diseño y de la complejidad del proyecto. Por ejemplo, un diseño puede describirse por ecuaciones booleanas, pero si es muy extenso quizá sea más apropiado emplear estructuras jerárquicas para dividirlo; ahora bien si se requiere diseñar un sistema cuyo funcionamiento dependa solo de sus entradas y salidas, es conveniente utilizar la descripción funcional, la cual presenta la ventaja de requerí menos instrucciones y el diseñador no necesita un conocimiento previo de cada componentes del circuito.

CAPÍTULO III

MARCO METODOLÓGICO

El marco metodológico es el procedimiento a seguir para alcanzar el objetivo de la investigación, está compuesto por el diseño, tipo y modalidad de investigación, fases de la investigación. Arias (2004) expone que “la metodología del proyecto incluye el tipo de investigación, las técnicas y los procedimientos que serán utilizados para llevar a cabo la indagación” es el ‘como’ se realizara el estudio para responder al problema” (p.45).

Tipo y Modalidad de Investigación

Este trabajo de investigación se sitúa en una perspectiva metodológica de investigación documental definida por Alfonso (1995) como proceso de construcción de conocimientos a partir de un trabajo sistemático de indagación, recolección, organización, análisis e interpretación de información en torno a un tema seleccionado y delimitado. A su vez este trabajo está apoyado de la investigación aplicada donde Lozada (2014) comenta que busca la generación de conocimiento con aplicación directa a los problemas de la sociedad o el sector productivo. Esta se basa fundamentalmente en los hallazgos tecnológicos de la investigación básica, ocupándose del proceso de enlace entre teoría y el producto.

La investigación documental es la parte esencial de un proceso de investigación científica que constituye una estrategia donde se observa y reflexiona sistemáticamente sobre realidades (teóricas o no) usando para ello diferentes tipos de documentos. Indaga, interpreta, presenta datos e informaciones sobre un tema determinado de cualquier ciencia, utilizando resultados que pudiesen ser base para el desarrollo de la creación científica.

Diseño de la Investigación

El diseño metodológico se ajusta a las fase del proceso de la investigación documental: preparatoria, descriptiva, interpretativa y construcción del documento final. A continuación se definen cada una de las etapas estipulando las actividades desarrolladas.

Fase preparatoria:

Esta fase inicial da respuesta a tres intenciones concretas: construir un marco teórico que permita contextualizar la investigación desarrollada, tomar las decisiones en torno al diseño de instrumentos adecuados a los objetivos y problema planteados y reflexionar en torno a la información obtenida.

- Lectura de bibliografía especializada relacionada con el objeto de estudio. Revisión de libros, artículos, investigaciones, etc. Con la finalidad de identificar temas clave de iniciar la construcción de un marco teórico para el informe final de la investigación.
- Conceptualización y elaboración de una base que permita la fundamentación de la investigación.
- Elaboración del primer borrador del marco teórico y del diseño de la investigación.
- Definición del diseño de la investigación, objetivos del estudio, técnicas de obtención de la información, cronología y metodología a utilizar.

En esta fase se abordará específicamente los conceptos de máquinas de estados finitas y algorítmicas desde el punto de vista algorítmica para luego plasmar en la metodología de desarrollo de hardware y permitir evaluar de forma gráfica los cambios en el tiempo que se ejecutan en la misma. Una vez solventada esa documentación se estudiaran diferentes metodologías de desarrollo de hardware existente para producir una metodología que aborde los problemas de manera general a lo particular.

Fase Descriptiva:

Comprende el trabajo de campo que se realiza con el fin de dar cuenta de los diferentes tipos de estudio que se han efectuado sobre el tema y subtemas, cuáles son sus referentes disciplinares y teóricos. Dentro de éste se establecieron los siguientes pasos: lectura exploratoria de cada uno de los artículos, libros y cualquier otro material obtenido en Internet.

Una vez establecida una metodología de descripción de hardware general se implementaran tres casos de estudio para evaluar las salidas obtenidas y la similitud en las mismas al ser ejecutadas por al menos tres programadores y de esa manera lograr depurar la misma y establecer los cambios necesarios para describir de forma expedita una propuesta idónea para el programador.

Fase Interpretativa:

Permite ampliar el horizonte de estudio por unidad de análisis donde se integraran las máquinas de estados finitos y algorítmicos a la metodología propuesta que cumplirá con el objetivo de dar uso de dichas máquinas en el proceso de descripción de hardware a través de VHDL.

Fase de construcción del documento final.

En esta fase se integrara la documentación, el análisis, las metodologías abordadas en la construcción de la propuesta y se dará difusión de la misma.

Grafico 1. Esquema del diseño de la metodología. Fuente: Materan. (2018)

CAPÍTULO IV

DESARROLLO DE LA PROPUESTA

Metodología para el desarrollo de código VDHL a través del uso de FSM y ASM

La siguiente metodología es una propuesta a través de la cual planteado un problema que requiere el diseño de circuitos secuencial que pueda generar el código VHDL a través del uso de Máquinas de Estados Finitos y Maquinas de Estados Algorítmica de manera directa para ello es necesario seguir una serie de etapas los cuales se indican a continuación:

Etapa Nº 1: Análisis del problema planteado

El primer paso es de gran importancia a la hora de diseñar cualquier tipo de circuito, ya que gracias a ese análisis entenderemos y especificaremos como debe ser el funcionamiento del circuito generando así ideas claras sobre lo que verdad se requiere ya que es primordial tener claro cómo debe funcionar nuestro circuito.

Etapa Nº 2: Identificar entradas y salidas:

Después de realizado el análisis se puede determinar cuáles son las señales externas que generan cambios en el funcionamiento del sistema es decir serán nuestras entradas al circuito. De igual manera identificaremos cuáles serán las salidas de nuestro circuito las cuales serían las señales que después de ser procesadas las entradas variaran según los requerimientos de funcionamiento del sistema.

Etapa Nº 3: Elaboración de FSM

El siguiente paso es desarrollar la máquina de estados finitos representándolo mediante un diagrama de estados para visualizar de manera más clara el funcionamiento del sistema y las transiciones entre

estados que ocurrirán de acuerdo a los valores que poseen las entradas en un determinado instante para ello es necesario primero seguir los siguientes pasos:

3. a Definir el número de estados posibles para dicho sistema

En este paso se definen los estados en los que se puede encontrar el sistema de acuerdo a cada una de las entradas posibles y representarlos con palabras claras y sencillas. Por ejemplo podemos identificar los estados de la siguiente manera S0, S1, S2, S3.

3. b Representar las transiciones de estado a través de un diagrama de estado:

Los diagramas de estado son una manera gráfica de representar las FSM lo cual es de gran ayuda a la hora de identificar las transiciones entre los estados del sistema

Los diagramas de estados se componen por círculos los cuales representan dichos estados del sistema y flechas que provienen de un estado a otro indicando la transición entre ellos según los valores de las entradas.

3. c Determinar el número de biestables:

Los biestables son dispositivos de memoria y nos permitirán representar los estados de la maquina en determinado instante el número de biestables que debemos implementar se determina según la cantidad de estados posibles que posea el sistema para lo cual se debe cumplir que

$2^n >= N$ estados

Donde n es el número de biestables y N es el número de estados posibles del sistema, es decir si necesitamos representar 4 estados la ecuación seria $2^n >= 4$ de tal manera n=2 ya que $2^2 = 4$ satisfaciendo la ecuación $2^2 >= 4$

3. d Codificación de los estados:

Para el diseño del circuito necesitamos representar los estados del sistema a través de señales lógicas 0 y 1 para ello se realiza la codificación de estados. Se le asigna a cada uno de los estados una combinación única de bits los cuales serán los valores posibles que podrán tomar los biestables implementados siguiendo el ejemplo anterior la codificación podría ser la siguiente:

Contamos con 4 estados S0, S1, S2, S3 e implementaremos 2 Flip-flop

S0 => 00
S1 => 01
S2 => 10
S3 => 11

3. e Realizar la tabla de estados, entradas y salidas

De donde surge esta tabla bueno proviene del diagrama de estados anteriormente elaborado en ella se representaran cuáles serán los estados siguientes y salidas de acuerdo a las entradas al sistema a partir de cualquiera de los estados posibles, es decir si estamos en el estado S0 cuál será el estado siguiente para cada una de las posibles entradas y así sucesivamente con cada uno de los estados. En dicha tabla se representaran los estados con las combinaciones de bits anteriormente fijadas para cada uno de ellos para así poder seguir con el siguiente paso.

3. f Tabla de excitación de los biestables

Sabemos que los valores de nuestros biestables son quienes representaran nuestros estados por ello es necesaria saber cuáles deben ser las entradas a los flip-flop para poder pasar de un estado determinado a otro según lo indique nuestra tabla de estados para ello se utilizan las tablas de excitación de los flip-flop.

- Primeros debemos elegir el tipo de biestables a utilizar J-K, D, S-R
- Luego de esto utilizamos las tablas de excitación ya conocidas según el biestable de nuestra elección. Con esta tabla determinaríamos los

valores de las entradas de los biestables necesarios para que ocurra un determinado cambio de estado para cada uno de los biestables que utilizaríamos.

3. g Obtención de funciones lógicas a través de mapas k

A partir de la tabla anterior generamos las funciones lógicas tanto para las salidas como para las entradas de los biestables a través de mapas de Karnaught obteniendo así lo necesario para el diseño del circuito requerido.

3. h Diseño del circuito

El último paso para la elaboración de la FSM es el diseño del circuito como tal el cual se genera gracias a las funciones lógicas anteriormente generadas.

Etapa Nº 4: Pasar de la FSM a una ASM:

Las ASM o máquina de estados algorítmico es otra manera de representar las FSM de manera más clara y precisa ya que define de mejor manera las condiciones necesarias para pasar de un estado a otro y de igual manera representa las acciones que se realizan en cada estado acercándonos así más a nuestro propósito el cual es generar el código en VHDl.

Como ya hemos elaborado nuestro diagrama de estados es más sencillo pasar a ASM solo debemos tomar en cuenta los siguientes aspectos:

- Los estados en FSM son representados con círculos y en ASM se representan con rectángulos los cuales son llamados cajas de estados
- Las transiciones entre estados son especificadas de manera más precisa estableciendo condiciones las cuales se representan con rombos
- Las salidas de tipo Mealy son representadas con rectángulos con bordes redondeados
- Los bloques ASM se ejecutan en un ciclo de reloj.

Aclarado esto podemos proceder a ejecutar los siguientes pasos los cuales nos ayudaran a definir nuestra ASM:

4. a Realizar la descripción del funcionamiento del circuito a través de un Pseudocódigo

En este paso se describe el funcionamiento del circuito a través de un pseudocódigo de manera secuencial ya que gracias a que el tipo de lenguaje de Pseudocódigo es más claro y comprensible hace que sea más fácil la representación de condiciones y pasos lo cual ayudara a la hora de la representación en el diagrama ASM

4. b Identificar condiciones necesarias para las diferentes transiciones entre estados:

A partir del diagrama de estados podemos identificar qué condiciones se deben cumplir para poder pasar de un estado a otro, esto se refleja en las transiciones entre los mismos ya que para pasar de un estado a otro los valores de las entradas deben poseer un valor especifico a esto se le llama condiciones de transición.

4. c Determinar acciones:

Tenemos claro que en cada uno de los estados las salida deben tener un valor determinado sabiendo esto debemos tener en cuenta que al pasar de un estado a otro estos valores varían, al cambio de estos valores se les denomina acciones las cuales se representan en las cajas de estados. En otras palabras las acciones son las asignaciones de valores a las variables que nos representan las salidas del sistema.

4. d Elaboración del diagrama ASM

Teniendo claro las transiciones entre estados, las condiciones y acciones para cada una de los estados podemos realizar el diagrama ASM a partir del diagrama de estados de la FSM.

En el diagrama ASM se representa la ruta a seguir para pasar de un estado a otro según se cumplan o no las condiciones anteriormente descritas.

Ejemplo para pasar del estado S0 a S1 la entrada debe ser X=1

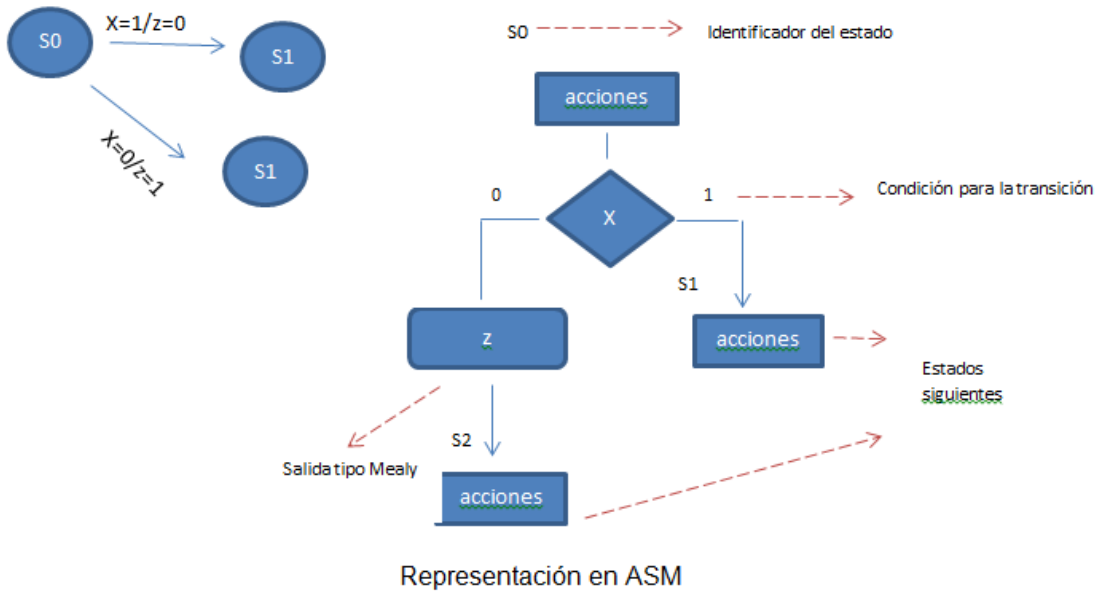


Figura 32. FSM llevada a ASM. Fuente: Materan. (2018)

Etapa Nº 5: Declaración de la entidad en VHDL

En el lenguaje de descripción de hardware VHDL se definen dos componentes primordiales uno de ellos es la entidad en la cual se establecen o declaran las entradas y salidas del sistema anteriormente definidas en el paso Nº 2

```
ENTITY nombre-entidad IS

  PORT (nombre_del_puerto-1: direccion_de_señal tipo_de_dato;
        nombre_del_puerto-2: direccion_de_señal tipo_de_dato;
        nombre_del_puerto-n: direccion_de_señal tipo_de_dato;)

END nombre-entidad;
```

El nombre entidad no tiene mucho que ver con la FSM que se trabaja, y no necesariamente tiene que estar relacionado con ella.

El nombre del puerto es el identificativo para las entradas y salidas.

Cuando todas las entradas y salidas son bits individuales se puede usar el tipo de dato **std_logic**.

La dirección de señal puede ser **IN** o **OUT**, dependiendo si se trata de una entrada o una salida.

Cuando se es dependiente de un ciclo de reloj, se define este puerto de la siguiente forma:

Clock: IN STD_LOGIC;

Para retornar a un estado inicial, se define el puerto de la siguiente forma:

reset: IN STD_LOGIC;

Etapa Nº 6: Declaración de la arquitectura

Este es el segundo componente en VHDL en el se declaran los estados de nuestra ASM además de variables y señales internas que podamos utilizar para definir el funcionamiento esperado lo cual es la segunda parte de la arquitectura del circuito como tal la cual está conformada por varios procesos o sentencias concurrentes según sean necesarios para el funcionamiento del mismo puede contener tantos procesos como sean necesarios en esta parte

ARCHITECTURE nombre de la arquitectura OF nombre-entidad IS

[declaraciones]

BEGING

[sentencias concurrentes]

END [nombre de la arquitectura];

Las declaraciones de estados: aquí se definen los estados y se crean las señales que tendrá un estado definido como su valor además se podrán declarar señales internas necesarias

Ejemplo:

SIGNAL señales internas a utilizar

TYPE Tipo_De_Estado IS (A, B, C, D);

SIGNAL Estado: Tipo_De_Estado;

Donde A, B, C y D representan los estados que contiene el diagrama.

Etapa Nº 7: Definición de los procesos o sentencias concurrentes

Los procesos o sentencias concurrentes son la parte más importante del código VHDL ya que en ellos es que se especifica detalladamente las operaciones, acciones y condiciones a evaluar para el funcionamiento de nuestro circuito nuestra metodología genera los procesos a partir de nuestra ASM ya que en ella se encuentran especificadas las acciones y condiciones necesarias para los cambios de estado que en si sería el funcionamiento del circuito

Se realiza siguiendo el siguiente formato:

```
PROCESS (lista_de_sensibilidad)
BEGIN
        [Configurar_estado_incial]
        [Transiciones_de_estados]
END PROCESS;
 [Instrucciones_de_salida]
```

En la arquitectura, Cuando un proceso VHDL se configura con señales de reloj y reinicio, este se plantea de la siguiente forma:

```
PROCESS (clock, reset)
 BEGIN
   IF (reset = ‘1’) THEN
         State <= A;
```

Configuración de estado inicial

```
   ELSIF else if (clk'event and clk='1')THEN
```

Configuración del reloj

La Transiciones de estados se realiza por medio del siguiente además en ellas se puede realizar la asignación de las salidas dependiendo de los estados en los que se encuentre en un determinado instante para ello se utiliza el siguiente formato:

```
CASE Estado IS
```

La sentencia siguiente dependerá del estado actual

```
WHEN Estado_n =>

        IF Entrada='1' THEN
        Asignacion de salidas
        Estado <= Estado_siguiente1;
        ELSE
        Asignacion de salidas
        Estado <= Estado_siguiente2;
        END IF;
```

Esta sentencia se coloca para los n números de estados que se tengan en el diagrama ASM, Tomando en cuenta su Transición a los siguientes estados con respecto a su entrad

```
WHEN others =>
                Asignacion de salidas
                Estado <= Estado inicial;

END CASE;
```

Esta sentencia se utiliza para restablecer el valor de A en caso de que el estado no obtenga ningún valor anterior

```
ENTITY nombre-entidad IS

  PORT (nombre_del_puerto-1: direccion_de_señal tipo_de_dato;
        nombre_del_puerto-2: direccion_de_señal tipo_de_dato;
        nombre_del_puerto-n: direccion_de_señal tipo_de_dato;)
END nombre-entidad;

ARCHITECTURE nombre de la arquitectura OF nombre-entidad IS

SIGNAL señales interneas a utilizar

TYPE Tipo_De_Estado IS (A, B, C, D);

SIGNAL Estado: Tipo_De_Estado;
BEGING
      PROCESS (lista_de_sensibilidad)

BEGIN
  IF (reset = '1') THEN
        State <= A;
        ELSIF elsif(clk'event and clk='1')THEN

CASE Estado IS  La sentencia siguiente dependerá del estado actual


WHEN Estado_n =>

            IF Entrada='1' THEN
                  Asignacion de salidas
                  Estado <= Estado_siguiente1;
            ELSE
                  Asignacion de salidas
                  Estado <= Estado_siguiente2;
            END IF;
WHEN others =>
                  Asignacion de salidas
                  Estado <= Estado inicial;


END CASE;
END PROCESS;
END [nombre de la arquitectura];

END CASE;
END PROCESS;
END [nombre de la arquitectura];
```

Figura 33.Código para definir componentes en VHDL: Fuente Materan. (2018)

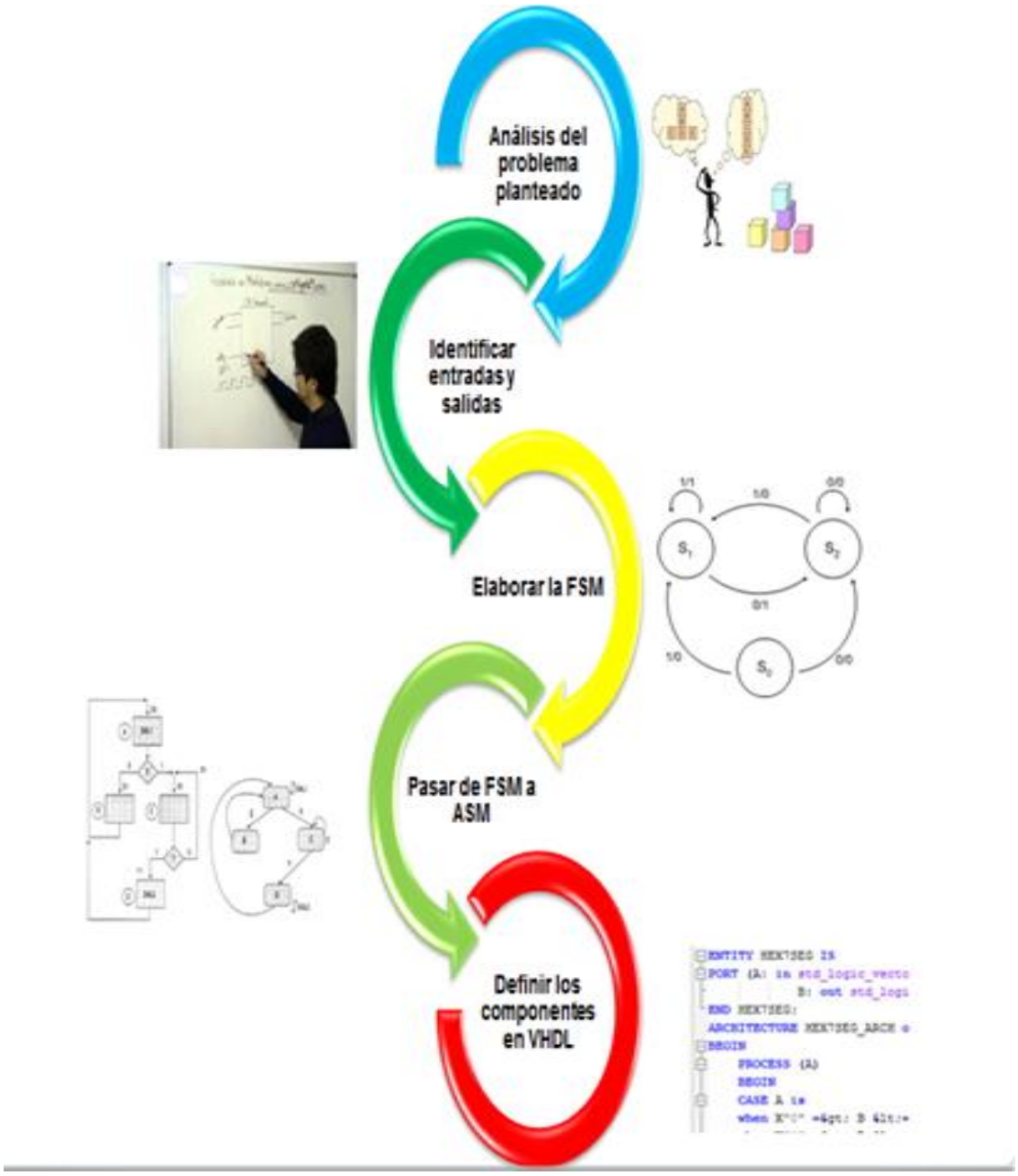


Grafico 2. Esquema general de la metodología para programar en lenguajes de descripción de hardware (VHDL) a través del uso de las FSM y ASM. Fuente: Materan, 2018.

Metodología para programar en lenguajes de descripción de hardware (VHDL) mediante del uso de las FSM y ASM.

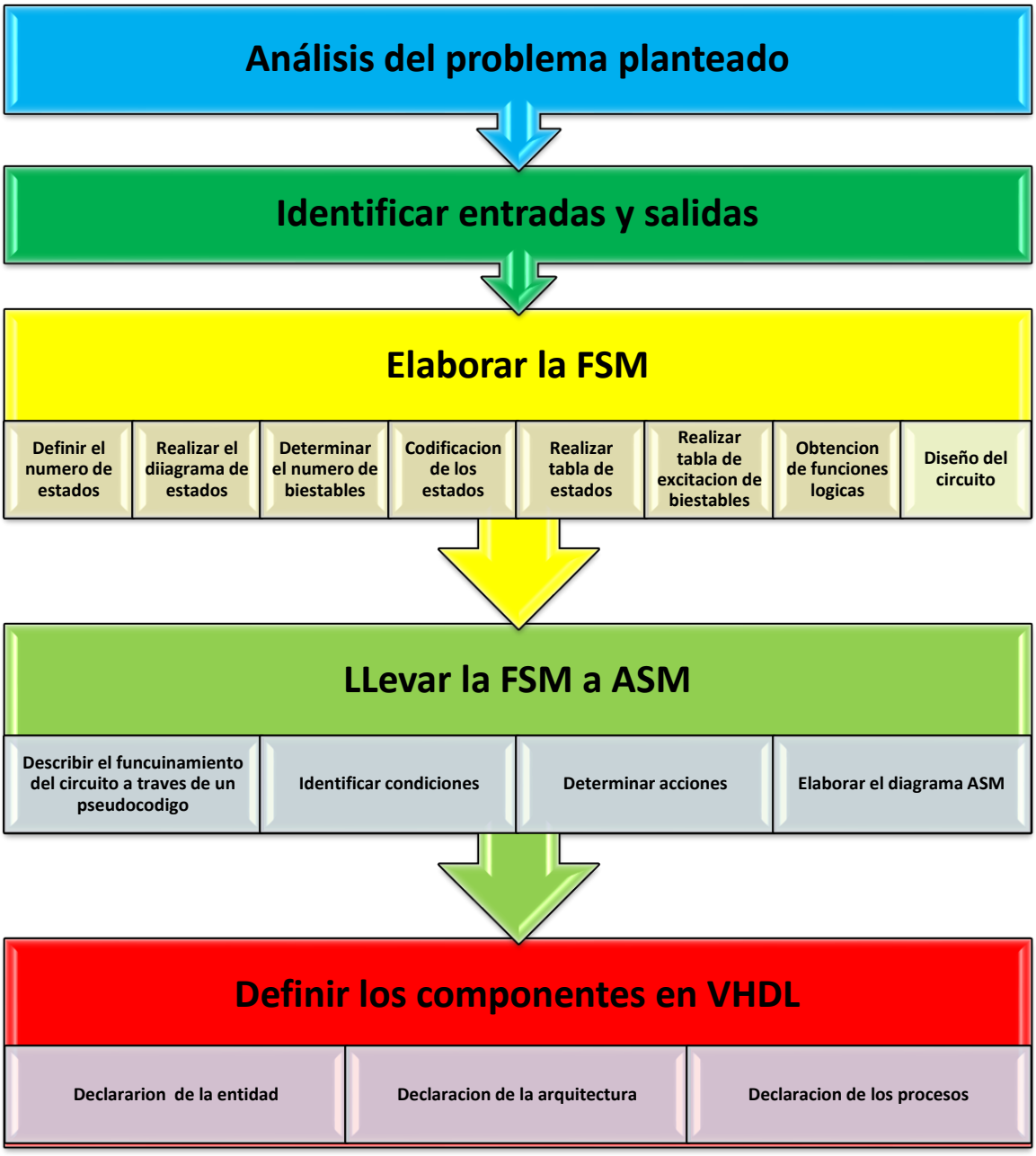


Grafico 3. Metodología para programar en lenguajes de descripción de hardware (VHDL) mediante el uso de las FSM y ASM. Fuente: Materan. 2018

CAPITULO V

CONCLUSIONES Y RECOEMENDACIONES

CONCLUSION

Las instituciones requieren de metodologías didácticas que le brinden herramientas al estudiante a la hora de resolver cualquier tipo de problema planteado para soluciones exitosas, preparándolos así para su desarrollo profesional en el ámbito laboral.

Para finalizar con el presente trabajo de investigación, se concluye principalmente dando respuesta al objetivo número uno (1), dos (2) y tres (3), enmarcado en la investigación; que es posible diseñar algoritmos que faciliten resolver cualquier tipo de problema mediante esta herramienta de carácter didáctico.

Todas las fases de estudio permitieron desarrollar esta metodología para los estudiantes de la asignatura arquitectura al computador en la carrera de ingeniería en sistemas de la Universidad de los Andes (NURR), facilitando el desarrollo de código en los lenguajes de descripción de hardware, es de importancia señalar que la metodología desarrollada se puede adaptar a cualquier tipo de problema planteado.

Culminado este proceso se han cumplido con todos los objetivos planteados en esta investigación cuyo resultado fue una metodología de diseño que representa una verdadera herramienta didáctica que le facilite al estudiante la tarea a la hora de enfrentarse con cualquier tipo de problema.

:

RECOMENDACIONES

Luego de haber concluido esta investigación, siempre se desea que haya una mejora continua del mismo; por lo tanto se hacen las siguientes recomendaciones:

1. Proponer la metodología de diseño desarrollada en la carrera de Ingeniería de la Universidad Valle del Momboy, con el fin de permitirle al estudiante aprender nuevos lenguajes de programación (lenguajes de descripción de hardware).
2. Incorporar a los grupos de investigación de la carrera de Ingeniería para investigar y dar aportes que logren una optimización de esta metodología.
3. De igual manera se les sugiere a las autoridades de la Universidad Valle del Momboy realizar cursos a los profesores para el uso de esta herramienta.

REFERENCIAS BIBLIOGRÁFICAS

Arias, F. (2004). El proyecto de Investigación. 4ta Edición. Carcas, Venezuela.

Cáceres, S., De Pablo, S., Cebrián, J.A., Sanz, F., Berrocal, M. (2015). Los diagramas ASM++ como herramienta aplicada en la enseñanza de la electrónica digital. Departamento de Tecnología Electrónica, Universidad de Valladolid, Valladolid, España.

Cassanovas, M. (2014). Máquinas de estados finitas. Centro Universitario de Desarrollo de Automatización y Robótica. Córdoba, Argentina.

Gutiérrez, J. (2008). Máquinas de estados finitos. Escuela Superior de Computo. México

Holguín, M. Orozco, A., Escobar, A., (2011). Metodología para al diseño de autómatas finitos con salidas en lenguaje ladder bajo el estándar IEC611131-3. Universidad Tecnológica de Pereira, Colombia.

Lara, E. (2002). Diseño de Sistemas Digitales con Lógica Programable. Universidad Autónoma de Nuevo León. México.

Lozada, J. (2014) Investigación Aplicada Definición, Propiedad Intelectual e Industria. Centro de investigación en Mecatronica y Sistemas interactivos, Universidad tecnológica Indoamericana. Quito, Ecuador.

Martin, B. (2006). Maquinas Algorítmicas como opción didáctica de sistemas digitales complejos. Universidad de Zaragoza.

Maxinez, D. y Alcalá J. (2002). El arte de programar sistemas digitales. Instituto tecnológico y de Estudios Superiores de Monterrey Campus, Estado de México.

Morales, L.; (2014). Propuesta de ejemplos integradores para la asignatura Digital VLSI, Universidad Central “Marta Abreu de las Villas, Santa Clara, Cuba.

Morris M, (1987). Diseño Digital, Primera Edición. Prentice Hall, México.

Muñoz, G (2002). Método simple para pasar de un algoritmo a un modelo en VHDL. Laboratorio de Automática Microelectrónica e inteligencia computacional, Universidad Distrital Francisco José de Caldas.

.

.