PONTIFICIA **UNIVERSIDAD CATÓLICA** DEL PERÚ

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

Diseño de una Arquitectura para un Sistema Neurodifuso ANFIS sobre un FPGA aplicado a la Generación de Funciones

Tesis para optar el Título de Ingeniero Electrónico, que presenta el bachiller:

Henry José Block Saldaña

ASESOR: Carlos Bernardino Silva Cárdenas

Lima, agosto de 2010

RESUMEN

En la presente tesis, se realizó el diseño de una arquitectura para un sistema neurodifuso ANFIS. Se tomó en consideración un sistema de orden cero de dos entradas y una salida, que cuenta con funciones de pertenencia triangulares en los antecedentes de las reglas difusas. Además, se tuvo en cuenta que el entrenamiento del sistema es realizado fuera de línea (*off-line*), en MATLAB.

La arquitectura diseñada se dividió en cuatro bloques: Fuzzificador, Permutador, Inferencia y Defuzzificador. Cada uno de estos bloques fue tratado como un subsistema y descrito por separado para facilitar su diseño. Posteriormente, se procedió a juntar los cuatro bloques, dando como resultado la arquitectura propuesta para el sistema neurodifuso ANFIS. Esta arquitectura fue descrita de manera modular y genérica mediante el lenguaje de descripción de hardware VHDL y fue implementada en los FPGA Spartan-3 XC3S200 de la empresa Xilinx y Cyclone II EP2C35 de la empresa Altera, utilizando las herramientas que se encuentran dentro de los entornos de desarrollo ISE 11 y Quartus II 9.1, respectivamente.

El sistema diseñado fue aplicado a la generación de funciones. Primero, se eligió una función no lineal y se llevó a cabo el entrenamiento del sistema en MATLAB para obtener los parámetros de los antecedentes y consecuentes de las reglas difusas. Después, estos parámetros fueron convertidos a una representación binaria en punto-fijo complemento a dos y almacenados en las memorias ROM del código en VHDL. Finalmente, se realizaron simulaciones sobre los dos FPGA, mencionados anteriormente, para verificar la operación del sistema y poder evaluar su desempeño. Entre los resultados obtenidos, destaca que el tiempo requerido por el sistema para calcular un valor de la función es menor a 10 μs (trabajando a una frecuencia de reloj de 50 MHz). Este valor es mucho menor al tiempo requerido por la aplicación en MATLAB, el cual fue de alrededor de 100 ms.

ÍNDICE

INTRODUCCIÓN

Los sistemas neurodifusos implican la combinación de las teorías de las redes neuronales y de los sistemas de inferencia difusa. Por un lado, las redes neuronales buscan reproducir la capacidad de razonamiento humano a través de su estructura y organización. Por otro lado, los sistemas de inferencia difusa permiten expresar el conocimiento de un humano "experto" mediante reglas *If-Then* simples, descritas en lenguaje natural.

Entre las diversas aplicaciones de estos sistemas, se puede mencionar su uso en el tratamiento inteligente de la información, la predicción de datos, el reconocimiento de patrones, el modelado de sistemas, la aproximación de funciones y el control automático. Aunque muchas de estas aplicaciones se realizan en software, una implementación en hardware presenta ciertas ventajas, debido a que permite el diseño de una arquitectura específica, lo cual lleva a un alto nivel de desempeño. Por consiguiente, las implementaciones en hardware son más convenientes cuando se requiere realizar un procesamiento en tiempo real.

En el presente trabajo, se propone el diseño en hardware de una arquitectura para un sistema neurodifuso ANFIS (Adaptive Network-based Fuzzy Inference System). El diseño de la arquitectura se realiza mediante el lenguaje de descripción de hardware VHDL (Very-high-speed integrated circuit Hardware Description Language). Posteriormente, se implementa en un FPGA (Field-programmable Gate Array) y se simula a través de su aplicación a la generación de funciones.

La tesis se divide en cuatro capítulos. En el primer capítulo, se presenta una breve introducción a los sistemas neurodifusos, así como algunas de sus aplicaciones y formas de implementación. En el segundo capítulo, se explica la estructura básica de un sistema neurodifuso ANFIS y el algoritmo de aprendizaje que permite su entrenamiento. Además, se da una descripción del lenguaje VHDL y de los FPGA. En el tercer capítulo, se presenta el diseño de la arquitectura propuesta y se detalla cada uno de los bloques que la componen. Finalmente, en el cuarto capítulo, se muestran las simulaciones realizadas y los resultados obtenidos.

CAPÍTULO 1. Los sistemas Neurodifusos

1.1. Una breve Introducción sobre los Sistemas Neurodifusos

Los sistemas neurodifusos surgieron de la necesidad de obtener y ajustar los parámetros de los sistemas difusos, ya sean sus conjuntos o sus reglas, mediante un método formal que no esté basado únicamente en el conocimiento humano o en la prueba y error. En especial, se buscaba ajustar de manera óptima los parámetros de los controladores difusos – una de las principales aplicaciones a nivel mundial de los sistemas difusos –. De modo que se vio la posibilidad de combinar las ya existentes teorías de las redes neuronales y de los sistemas de inferencia difusa. A continuación, se mencionan algunos de estos casos.

Primero, Nauck, Klawonn y Kruse plantearon algunas formas de integrar los principios de la teoría de control difuso con las redes neuronales y, además, presentaron un controlador neurodifuso que usa un algoritmo de entrenamiento para el ajuste de sus parámetros [1]. Segundo, Berenji y Khedkar propusieron la arquitectura de un sistema llamado GARIC (Generalized Approximate Reasoning-based Intelligent Control) para ajustar los parámetros de un controlador difuso a través de un algoritmo de entrenamiento por refuerzo [2]. Tercero, Jang propuso la arquitectura y el algoritmo de entrenamiento de un sistema de inferencia difusa basado en redes adaptativas, llamado ANFIS (Adaptive-Network-based Fuzzy Inference System), capaz de establecer una relación entre los datos de entrada y salida presentados [3]. Cuarto, Halgamuge y Glesner presentaron la arquitectura de una red neuronal perceptrón, conocida como FuNe (Fuzzy Neural), que sirve para construir sistemas difusos y ajustar sus parámetros, utilizando para esto datos de entrenamiento representativos [4]. Finalmente, Nauck y Kruse propusieron el sistema neurodifuso NEFCLASS (Neuro-Fuzzy system for the Classification of Data) para la clasificación de datos, basado en un perceptrón difuso [5].

En la actualidad, los sistemas neurodifusos están en continuo desarrollo y cada año se siguen proponiendo nuevas ideas.

1.2. Definición de un Sistema Neurodifuso

Como ya se mencionó anteriormente, un sistema neurodifuso implica la combinación de las teorías de las redes neuronales y de los sistemas de inferencia difusa. Por un lado, las redes neuronales buscan reproducir la capacidad de razonamiento humano a través de su estructura y organización; modelan la forma en que el cerebro humano realiza una tarea particular o función de interés. Pueden ser vistas como un procesador masivamente paralelo, compuesto por unidades de procesamiento más simples (las neuronas), capaz de adquirir conocimiento a través de un proceso de aprendizaje y almacenarlo en las fuerzas de sus interconexiones neuronales (pesos sinápticos) [6]. Por otro lado, los sistemas de inferencia difusa permiten expresar el conocimiento de un humano "experto" mediante reglas "If-Then" simples, descritas en lenguaje natural [7].

De un modo más estricto, Nauck define las principales características de todo sistema neurodifuso, las cuales se muestran a continuación [8]:

1. Un sistema neurodifuso es un sistema difuso que es entrenado por un algoritmo de aprendizaje derivado de la teoría de las redes neuronales.
2. Un sistema neurodifuso puede ser visto como una red neuronal de interacción hacia adelante (feedforward) de tres capas. La primera representa las variables de entrada; la del medio (oculta), las reglas difusas; y la tercera, las variables de salida. Algunos sistemas neurodifusos utilizan más de tres capas.
3. Un sistema neurodifuso puede ser siempre interpretado (antes, durante y después del aprendizaje) como un sistema de reglas difusas. Es posible tanto crear el sistema desde cero a partir de los datos de entrenamiento, como inicializarlo mediante un conocimiento previo en forma de reglas difusas.
4. El proceso de aprendizaje de un sistema neurodifuso toma en consideración las propiedades semánticas del sistema difuso subyacente. Esto conlleva a restricciones en las posibles modificaciones de los parámetros del sistema.
5. Un sistema neurodifuso aproxima una función desconocida n-dimensional que está parcialmente representada en los datos de entrenamiento. Las reglas difusas codificadas dentro del sistema representan muestras vagas.

Un sistema neurodifuso no debe ser visto como un sistema experto y no tiene relación con la lógica difusa en el sentido estricto.

1.3. Usos y Aplicaciones de los Sistemas Neurodifusos

Los sistemas neurodifusos se pueden usar y aplicar en aquellas áreas donde se han empleado con éxito tanto las redes neuronales como los sistemas de inferencia difusa. Se destaca el uso de estos sistemas en el tratamiento inteligente de la información, la predicción de datos, el reconocimiento de patrones, el modelado de sistemas, la aproximación de funciones y el control automático. Es más, estos usos se han dado en diversas áreas como la ingeniería, la medicina, la economía y el transporte. En las siguientes líneas, se mencionan algunas aplicaciones específicas de los sistemas neurodifusos.

Por una parte, Halgamuge y Glesner describen tres aplicaciones industriales del sistema FuNe en la clasificación de imágenes: la clasificación de imágenes de uniones soldadas, el reconocimiento de imágenes de sonar bajo el agua y el reconocimiento de dígitos manuscritos [4]. Por otra parte, Berenji y Khedkar utilizan el sistema GARIC y Jang, el sistema ANFIS para solucionar el problema de control de un péndulo invertido con un grado de movimiento sobre un carro [2] [9]. Asimismo, Jang aplica el ANFIS en el modelamiento de funciones altamente no lineales y en la predicción de una serie temporal caótica [3]. Finalmente, Kumar y Garg aplican un controlador neurodifuso para el control de dos brazos robóticos que cooperan entre sí [10].

1.4. Formas de Implementación de los Sistemas Neurodifusos

Existen diversos métodos y medios que permiten implementar los sistemas neurodifusos. Estos se pueden clasificar en dos grandes áreas: a nivel de software y a nivel de hardware. El tipo de implementación que se elija dependerá, principalmente, de los requerimientos de la aplicación en particular; sin embargo, existen ciertas ventajas y desventajas en cada tipo de implementación que es importante mencionar.

Por un lado, la mayoría de implementaciones en software se realizan mediante una computadora, aunque cabe indicar que también existen algunas que utilizan un (micro-) procesador que no forma parte de una computadora. Algunas ventajas que llevan a la elección de este tipo de implementación son la programación en alto nivel y el uso de herramientas de diseño, que facilitan el desarrollo del sistema en un menor tiempo. En cuanto a las desventajas, se puede mencionar que el procesamiento secuencial propio de una computadora no se adecua óptimamente a la arquitectura de los sistemas neurodifusos, que es paralela, lo que conlleva a un bajo desempeño y, en general, a que no se puedan realizar aplicaciones en tiempo real. Por el contrario, las implementaciones en hardware son más convenientes cuando se requiere realizar un procesamiento en tiempo real, debido al paralelismo propio de este tipo de sistemas. Además, se puede diseñar una arquitectura específica sobre la cual implementar el sistema, lo que permite obtener un alto desempeño. No obstante, también existen algunas desventajas, como son una mayor complejidad y tiempo de diseño, y un mayor costo.

CAPÍTULO 2. El Sistema Neurodifuso ANFIS

2.1. Presentación y Desarrollo del Tema

En la literatura se han planteado diversos sistemas neurodifusos, que han permitido dar solución a distintos problemas de alta complejidad. Uno de estos sistemas es el Sistema de Inferencia Difusa basado en Redes Adaptativas (ANFIS) [3], conocido también como Sistema de Inferencia Neuro-Difuso Adaptativo [9]. En las siguientes líneas, se desarrollará una breve descripción de este sistema, se mostrará la importancia de su implementación en hardware y se presentará el trabajo a realizar.

El ANFIS es un sistema de inferencia difusa que a través de un algoritmo de aprendizaje inspirado en la teoría de las redes adaptativas puede establecer una relación entre los datos de entrada y salida, modificando los valores de sus parámetros, tanto de los antecedentes como de los consecuentes. Además, esto no deja de lado que sea posible seguir utilizando el conocimiento humano en forma de reglas difusas *If-Then*. [3]

En cuanto a su implementación, el ANFIS puede ser implementado tanto en software como en hardware, siendo esta última esencial para permitir el desarrollo de nuevos dispositivos electrónicos portátiles y pequeños, que requieran de un procesamiento en tiempo real, de modo que es indispensable aprovechar el procesamiento paralelo y hacer uso óptimo de los recursos. Lo anterior solo es posible a través del diseño de una arquitectura en hardware. Para este fin, se puede utilizar el lenguaje de descripción de hardware VHDL (Very High-integrated Hardware Description Language) y un dispositivo lógico programable, entre los que destaca el FPGA (Field-programmable Gate Array).

Por lo tanto, en la presente tesis se diseñará la arquitectura del sistema neurodifuso ANFIS, la cual, a su vez, podrá servir como base para futuras investigaciones y desarrollos en el tema. Asimismo, se utilizará el lenguaje de descripción de hardware VHDL y un entorno de desarrollo que permitirá sintetizar el diseño sobre el FPGA. Finalmente, este sistema se simulará a través de su aplicación a la generación de funciones, por ser un método directo y eficaz para verificar su funcionamiento y mantener abierta la posibilidad a otras aplicaciones. En particular, las funciones no lineales son las de mayor interés por ser muy difíciles de generar por los sistemas convencionales.

2.2. El Sistema Neurodifuso ANFIS

2.2.1. Sistema de Inferencia Difusa

El ANFIS es en esencia un sistema de inferencia difusa. Por esta razón, se describirá en qué consisten estos sistemas y cuál es su funcionamiento. A continuación, en la figura 1, se presenta el diagrama general de un sistema de inferencia difusa.

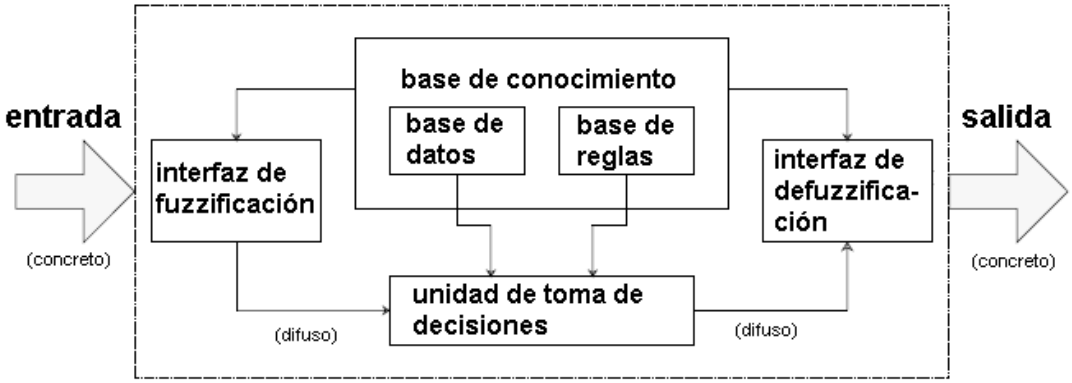


Figura 1. Sistema de Inferencia Difusa [3].

Este sistema está compuesto por cinco bloques [3]:

- Interfaz de fuzzificación: transforma los valores numéricos (concretos) en grados de pertenencia con valores lingüísticos.
- Base de datos: define las funciones de pertenencia de los conjuntos difusos que se utilizan en las reglas difusas.
- Base de reglas: contiene un número determinado de reglas difusas if-then, que tienen un antecedente y un consecuente.
- Unidad de toma de decisiones: realiza las operaciones de inferencia sobre las reglas.
- Interfaz de defuzzificación: transforma los resultados difusos en un valor numérico (concreto).

En relación con el razonamiento difuso, es decir, el proceso mediante el cual se obtiene una salida en función de las variables de entrada, este se realiza como se describe a continuación [3]:

1. Se comparan las variables de entrada con las funciones de pertenencia de los antecedentes para obtener los valores de pertenencia de cada etiqueta lingüística.
2. Se combinan los valores de pertenencia de los antecedentes mediante un operador de la T-norma, como el mínimo o el producto, para obtener el grado de activación o peso de cada regla.
3. Se genera el consecuente (concreto o difuso) de cada regla dependiente del grado de activación.
4. Se juntan los consecuentes para producir una salida concreta (valor numérico).

Existen diversos tipos de sistemas de inferencia difusa que dependen del razonamiento difuso y de las reglas difusas *If-then* que utilizan. Dentro de estos, el sistema difuso tipo *Takagi-Sugeno* [11] es de especial interés para la arquitectura del ANFIS que se describirá en el siguiente subcapítulo. En este sistema la salida de cada regla es una combinación lineal de las variables de entrada más un término constante, y la salida global es la media ponderada de la salida de cada regla. Por ejemplo, para dos entradas y dos reglas tipo *Takagi-Sugeno* de primer orden, estas tendrían la siguiente forma:

1. Regla 1: If x is A_1 and y is B_1, then $f_1 = p_1x + q_1y + r_1$,
2. Regla 2: If x is A_1 and y is B_1, then $f_2 = p_2x + q_2y + r_2$.

2.2.2. Arquitectura del ANFIS

El ANFIS puede representar tres tipos de sistemas de inferencia difusa, como se mencionan en [3]. Sin embargo, el más representativo y usado es el ANFIS cuyas reglas son del tipo *Takagi-Sugeno*, llamado simplemente ANFIS tipo-3. En la siguiente figura (figura 2) se muestra la arquitectura de un sistema ANFIS tipo-3 de dos entradas y una salida, que se utiliza a modo de ejemplo ilustrativo.

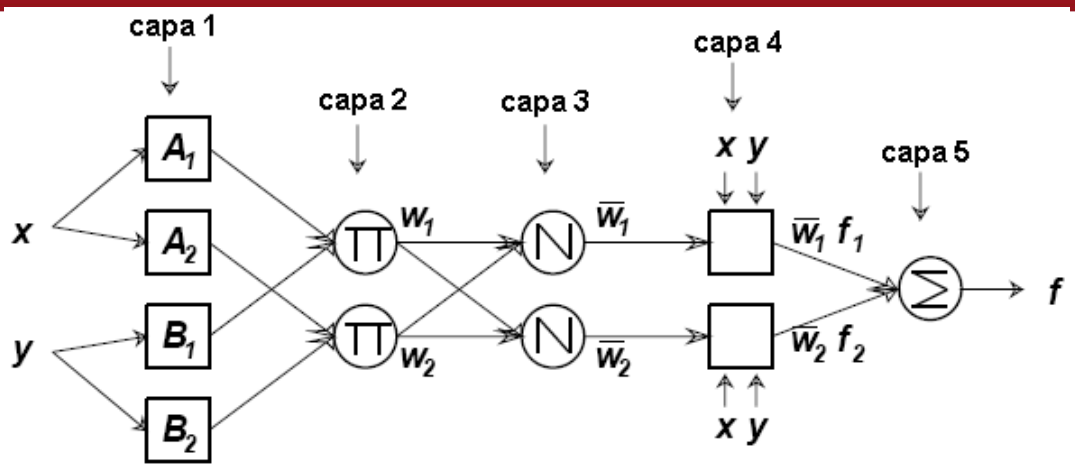


Figura 2. Arquitectura de un ANFIS (tipo-3) de dos entradas y una salida [3].

El ANFIS está compuesto de cinco capas. Los nodos en la primera y cuarta capa están dibujados como bloques cuadrados, lo cual significa que son adaptativos; es decir, que tienen parámetros que pueden variar. Los nodos en la segunda, tercera y quinta capa están dibujados como bloques circulares, que quiere decir que son fijos y no dependen de ningún parámetro variable. A continuación, se describe la función que realiza cada capa de acuerdo con [3]:

Capa 1. Cada nodo i en esta capa tiene la siguiente función de salida:

$$O_i^1 = \mu_{A_i}(x), \tag{1}$$

donde x es la entrada al nodo i y A_i es la etiqueta lingüística asociada a ese nodo.

Es decir, O_i^1 es la función de pertenencia de A_i, que es una función cuyos valores

de salida están entre 0 y 1. Se pueden usar diversas funciones de pertenencia como la función triangular, trapezoidal o la que tiene forma de campana, cada una

con sus propios parámetros. Los parámetros de estas funciones se conocen como parámetros de los antecedentes.

Capa 2. Cada nodo en esta capa multiplica las señales de entrada y envía el producto (grado de activación o peso) a la siguiente. Para el ejemplo:

$$O_i^2 = w_i = \mu_{A_i}(x) \times \mu_{B_i}(y), \qquad i = 1, 2. \tag{2}$$

Capa 3. En esta capa, el nodo-i calcula la razón entre el grado de activación o peso de la regla i y la suma de todos los grados de activación o pesos. Por ejemplo:

$$O_i^3 = \overline{w}_i = \frac{w_i}{w_1 + w_2}, \qquad i = 1, 2. \tag{3}$$

Capa 4. Cada nodo i en esta capa tiene la siguiente función de salida:

$$O_i^4 = \overline{w}_i f_i = \overline{w}_i (p_i x + q_i y + r_i), \tag{4}$$

donde $\{p_i, q_i, r_i\}$ es el conjunto de parámetros del nodo i, llamados parámetros de los consecuentes.

Capa 5. En esta capa solo hay un nodo, el cual calcula la salida global como la suma de todas sus entradas, es decir:

$$O_i^5 = \text{salida global} = \sum_i \overline{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i} \tag{5}$$

El proceso mediante el cual se obtiene la salida de todo el sistema se ilustra en la figura 3, que se muestra a continuación.

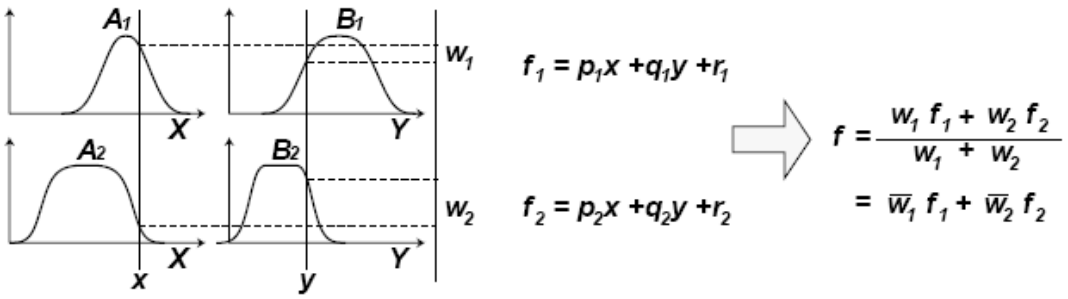


Figura 3. Razonamiento difuso del ANFIS de la figura 2 [3].

Finalmente, cabe mencionar que una arquitectura totalmente equivalente es la que se muestra en la siguiente figura 4.

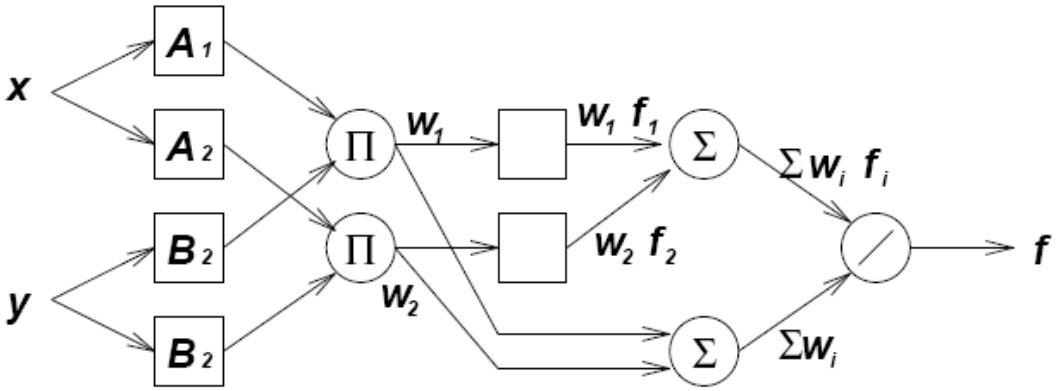


Figura 4. Arquitectura equivalente del ANFIS de la figura 2 [9].

2.2.3. Algoritmo de Aprendizaje Híbrido

El algoritmo de aprendizaje para el ANFIS es propuesto y explicado por Jang tanto en [3] como en [9], y consiste en un algoritmo híbrido de dos procesos: un proceso hacia delante y uno hacia atrás. Para esto, se utiliza un conjunto de datos de entrada y salida o conjunto de datos de entrenamiento. A continuación, se explica cada uno de estos.

En el proceso hacia delante, manteniendo fijos los parámetros de los antecedentes, las salidas de los nodos se propagan hasta la capa cuatro y los parámetros de los consecuentes se identifican por el método de mínimos cuadrados. Además, se obtiene una señal de error por cada nodo que se calcula comparando la salida obtenida con la deseada.

En el proceso hacia atrás, manteniendo fijos los parámetros de los consecuentes, las señales de error halladas anteriormente se propagan hacia atrás y los parámetros de los antecedentes se actualizan por el método del gradiente descendiente [9].

En la tabla 1, que se muestra a continuación, se resume este procedimiento.

Tabla 1. Algoritmo de aprendizaje híbrido para el ANFIS [9].

	Proceso hacia delante	**Proceso hacia atrás**
Parámetros de los antecedentes	Fijos	Gradiente descendiente
Parámetros de los consecuentes	Mínimos cuadrados	Fijos
Señales	Salidas de los nodos	Señales de error

2.2.4. El ANFIS como Aproximador Ilimitado

Jang demuestra, utilizando el teorema de Stone-Weierstrass, que un ANFIS con reglas simplificadas, es decir, que tienen la siguiente forma:

If x is A_i and y is B_i, then z is r_i.

donde r_i es un valor concreto, tiene poder de aproximación ilimitado para cualquier función no lineal en un conjunto compacto [3].

2.3. Dispositivo Lógico Programable FPGA

Los dispositivos lógicos programables (Programmable Logic Devices, PLDs) nacieron de la necesidad de contar con un dispositivo en el que se pudiese implementar circuitos lógicos combinacionales y que fuesen programables a nivel de hardware. Posteriormente, se desarrollaron PLDs con registros, lo cual permitió la implementación tanto de circuitos combinacionales como secuenciales [12].

El PLD es un chip de propósito general cuyo hardware puede ser reconfigurado. Su programación permite cambiar las interconexiones existentes entre los elementos lógicos dentro del dispositivo y, de esta manera, se puede implementar cualquier circuito lógico. Esto es posible gracias a los “fusibles” o “interruptores” semiconductores, que pueden actuar como un circuito abierto o cerrado. Existen diversos tipos de PLDs que se clasifican de acuerdo a la complejidad de su estructura y a la cantidad y variedad de recursos lógicos con los que cuentan: los SPLDs (Simple PLD), los CPLDs (Complex PLD) y los FPGAs (Field-programmable Gate Array) [12] [13].

Dentro de los PLDs, el FPGA (Arreglo de Compuertas Programable en Campo) es el más complejo y cuenta con una gran cantidad de recursos lógicos. Se compone fundamentalmente de celdas lógicas que incluyen, en la mayoría de los casos, un *flip-flop* tipo-D, una *look-up table* y un conjunto de multiplexores. Su estructura de interconexiones entre las celdas tiende a ser distribuida y más flexible que la de los otros dispositivos. Debido a todo esto, es el más apropiado para la implementación de sistemas digitales de gran tamaño [14].

2.4. Lenguaje de Descripción de Hardware VHDL

Los lenguajes de descripción de hardware (Hardware Description Languages, HDLs) permiten modelar y describir un circuito, ya sea desde el punto de vista estructural o de comportamiento, en un nivel de abstracción deseado. A diferencia de los lenguajes de programación convencionales, en los cuales las sentencias se realizan secuencialmente y una a la vez, los HDLs cuentan con sentencias que se ejecutan en paralelo o simultáneamente. Asimismo, toman en consideración diversos aspectos que permiten describir las características inherentes de los circuitos digitales.

Estas características están definidas por los conceptos de entidad, conectividad, concurrencia y tiempo. Primero, la entidad es el bloque básico de construcción, que se modela basándose en una parte de un circuito real. Segundo, la conectividad modela los cables de conexión entre estas partes, lo que permite que las distintas entidades interactúen entre sí. Tercero, la concurrencia describe el hecho de que varias entidades puedan estar activas a la vez y varias operaciones se realicen en paralelo. Finalmente, el tiempo que está relacionado con la concurrencia especifica el inicio y fin de cada operación.

Dentro de los HDLs, los más usados a nivel mundial son el Verilog y el VHDL, los cuales son estándares industriales. Ambos tienen las mismas capacidades, aunque presentan sintaxis diferentes, y se utilizan, por ejemplo, para la descripción de circuitos digitales en CPLDs y FPGAs [14].

2.5. Metodología de Diseño en VHDL

Existen diversas formas de realizar la descripción de un sistema digital en VHDL, así como distintas metodologías que facilitan el proceso de diseño del sistema. En el presente trabajo, se utilizan en especial dos metodologías: la metodología de diseño jerárquico y la metodología de diseño RT.

La primera es una metodología que consiste en dividir un sistema complejo y de grandes dimensiones en pequeñas partes o módulos, los cuales pueden ser construidos y descritos independientemente. Esto le permite al diseñador enfocarse en un módulo pequeño y manejable a la vez. Además, se puede reutilizar un módulo que ya haya sido diseñado, tanto en el mismo sistema como en otros.

La segunda es una metodología que permite describir las operaciones de un sistema mediante una secuencia de transferencia de datos y manipulaciones entre registros. En esta metodología se utilizan los diagramas ASMD (Algorithmic State Machine with Data path), los cuales representan a una máquina de estados, que funciona como unidad de control; y a un conjunto de unidades funcionales y registros, donde se realizan diversas operaciones. Entre sus principales ventajas, destaca la realización eficiente de algoritmos en hardware [14].

2.6. Definiciones Operativas

En el presente trabajo, se realizará el diseño de una arquitectura para un sistema neurodifuso ANFIS sobre un FPGA aplicado a la generación de funciones. Este diseño necesitará ser puesto a prueba y evaluado con el fin de verificar su funcionamiento y desempeño. Para esto, se utilizan ciertos criterios, los cuales se definen a continuación:

- **Frecuencia de trabajo:** corresponde a la velocidad en ciclos por segundo, también medida en hertzios (Hz), a la cual opera un sistema digital síncrono. Esto es, es la frecuencia de reloj (clock frequency) del sistema.

- **Tiempo de respuesta:** es el tiempo que demora el sistema en dar una respuesta válida ante una entrada determinada. En todo diseño se busca que sea el mínimo posible dentro de los requerimientos dados.

- **Recursos utilizados:** se refiere a la cantidad de recursos lógicos utilizados por el sistema. En un FPGA, corresponden a *flip-flops*, *look-up tables*, sumadores,

memorias, entre otros. Siempre se busca que se utilice la menor cantidad posible.

CAPÍTULO 3. Diseño de la Arquitectura para el Sistema Neurodifuso ANFIS

3.1. Objetivos

3.1.1. Objetivo General

Realizar el diseño de una arquitectura para un sistema neurodifuso ANFIS sobre un FPGA y verificar su funcionamiento a través de su aplicación a la generación de funciones.

3.1.2. Objetivos Específicos

- Diseñar la arquitectura para el sistema neurodifuso ANFIS.
- Describir la arquitectura diseñada mediante el lenguaje VHDL.
- En MATLAB, entrenar al sistema para una función que se quiera generar y obtener sus parámetros.

- Simular el diseño realizado sobre un FPGA utilizando los parámetros previamente hallados y verificar su funcionamiento.

- Realizar una arquitectura modular, genérica y portable, es decir, que se pueda implementar en más de un FPGA y no tenga dependencia con uno en particular.

3.2. Consideraciones para el Diseño

En primer lugar, se toma en cuenta que el entrenamiento del sistema ANFIS es fuera de línea (*off-line*), por medio de MATLAB. Por esta razón, el sistema implementado tendrá parámetros fijos guardados en memoria, pero que pueden ser modificados antes de la programación en el dispositivo FPGA.

En segundo lugar, el sistema neurodifuso ANFIS para el cual se diseñará la arquitectura presenta las siguientes características:

- Es un sistema de dos entradas: x, y. Tiene una única salida: f.
- Es de tipo *Takagi-Sugeno* de orden cero. De modo que las reglas difusas tienen la siguiente forma: *If x is* A_i *and y is* B_i*, then* f_i *is* r_i.
- Utiliza funciones de pertenencia triangulares en los antecedentes de las reglas, que dependen de tres parámetros: a, b y c.
- El número de funciones de pertenencia para cada entrada es el mismo.
- Se utilizan todas las combinaciones posibles entre los valores de pertenencia. Es decir, el número de reglas es igual al producto del número de funciones de pertenencia de las dos entradas.

Por ejemplo, si se tienen tres funciones de pertenencia por entrada, las reglas difusas serían las siguientes:

1. *If x is* A_1 *and y is* B_1*, then* f_1 *is* r_1 ;

2. *If x is A_1 and y is B_2, then f_2 is r_2* ;
3. *If x is A_1 and y is B_3, then f_3 is r_3* ;
4. *If x is A_2 and y is B_1, then f_4 is r_4* ;
5. *If x is A_2 and y is B_2, then f_5 is r_5* ;
6. *If x is A_2 and y is B_3, then f_6 is r_6* ;
7. *If x is A_3 and y is B_1, then f_7 is r_7* ;
8. *If x is A_3 and y is B_2, then f_8 is r_8* ;
9. *If x is A_3 and y is B_3, then f_9 is r_9* ,

donde *A* y *B* son funciones de pertenencia triangulares, y *r* es un valor concreto.

En tercer lugar, se utilizará una representación en punto-fijo complemento a dos para las señales que representan valores numéricos. La representación de un número binario de *n* bits, con *p* bits que representan la parte fraccional, es la que se muestra a continuación:

$$b_{n-p-1}b_{n-p-2}\cdots b_1 b_0 . b_{-1}b_{-2}\cdots b_{-p} \tag{6}$$

Valor: $$V = \frac{b}{2^p} \tag{7}$$

Rango: $$-2^{n-p-1} \leq b \leq 2^{n-1-p} - \frac{1}{2^p} \tag{8}$$

Finalmente, el sistema será parametrizable en lo siguiente:

- Número de funciones de pertenencia, por medio de la constante MF_NUM.
- Cantidad de bits de las entradas y de los parámetros, tanto para la parte entera como para la parte fraccional.

3.3. Diseño y Descripción de la Arquitectura del ANFIS

Para lograr un diseño óptimo, se tomaron en cuenta los siguientes aspectos:

1. Una arquitectura totalmente paralela, aunque puede operar rápidamente, no es eficiente, ya que requiere de muchos recursos. En particular, el número de multiplicadores es elevado. Por ejemplo, en el caso de trabajar con cinco

funciones de pertenencia por entrada, el número de multiplicadores es igual a 50 (veinticinco en la capa 2 y veinticinco en la capa 3).

2. Evaluar sólo las reglas activas, es decir, aquellas que tienen valores de pertenencia diferentes de cero, resulta en un sistema más eficiente y rápido.
3. Solo es necesario realizar una división, en la etapa final, de acuerdo a lo mostrado en la figura 4.

Por consiguiente, se propone una arquitectura que es en parte secuencial y en parte paralela, maximizando el uso de los recursos. Además, solo se evalúan las reglas activas y se realiza una división en la etapa final. El sistema propuesto se divide en cuatro subsistemas interconectados entre sí: Fuzzificador, Permutador, Inferencia y Defuzzificador, como se muestra en la figura 5 (no se muestran las conexiones internas).

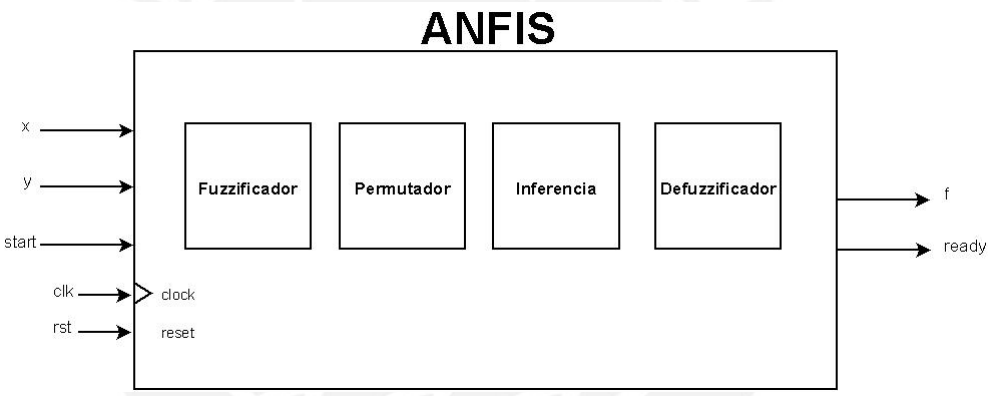


Figura 5. Diagrama de bloques del sistema ANFIS propuesto.

Las señales de entrada son las siguientes:

- ***x***, ***y***: valores numéricos de las entradas.
- ***start***: señal de control que da inicio a la operación del sistema.
- ***clk***: señal de reloj (*clock*).
- ***rst***: señal asíncrona de *reset* que inicializa el sistema.

Las señales de salida son las siguientes:

- ***f***: valor numérico que equivale al resultado calculado por el sistema.
- ***ready***: señal de control que indica que el sistema ha terminado de operar y puede recibir un dato nuevo.

A continuación, en la figura 6, se muestra un diagrama de tiempos del funcionamiento deseado del sistema.

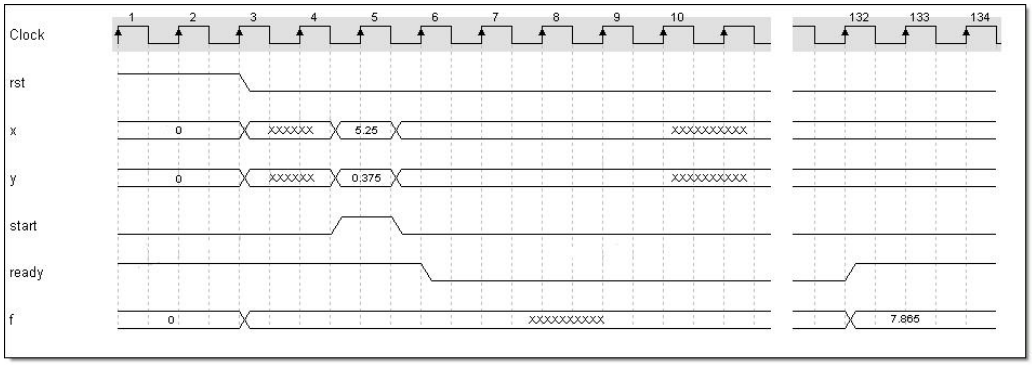


Figura 6. Diagrama de tiempos del sistema ANFIS propuesto.

Como se puede observar en la figura anterior, la señal *start* da inicio a la operación del sistema al cambiar a nivel alto y, al mismo tiempo, se colocan los datos de entrada. Solo es necesario que estos mantengan su valor durante un periodo de reloj. En el siguiente ciclo de reloj, la señal *ready* cambia a nivel bajo, lo cual significa que el sistema ha entrado en operación y se encuentra ocupado. Después de un tiempo determinado, la señal *ready* cambia a nivel alto y se puede leer el valor de la señal de salida, así como colocar nuevos datos de entrada.

En los siguientes subcapítulos, se describe y explica el funcionamiento de los subsistemas o bloques del sistema propuesto.

3.3.1. Bloque Fuzzificador

El diagrama de bloques del Fuzzificador se muestra en la figura 7. Los bloques que lo componen son los siguientes: dos registros, uno para cada entrada; dos memorias ROM, donde se almacenan los parámetros de los antecedentes; dos bloques que evalúan las funciones de pertenencia triangulares; y un controlador.

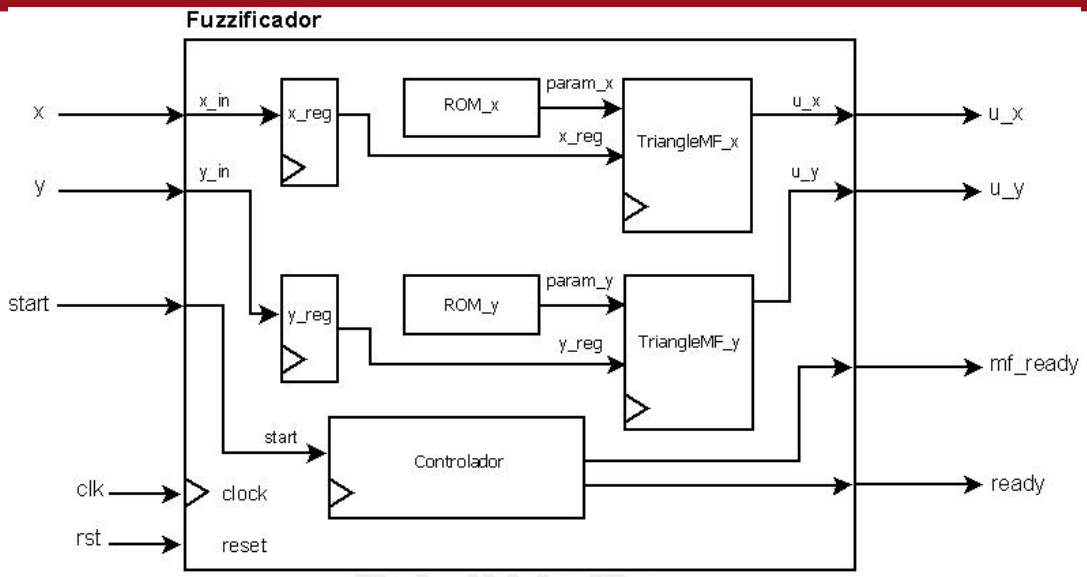


Figura 7. Diagrama de bloques del Fuzzificador. (No se muestran las señales internas de control).

Señales de entrada:

- ***x***, ***y***: valores numéricos de las entradas del ANFIS.
- ***start***: señal que da inicio a la operación del sistema.
- ***rst***, ***clk***: señales de *reset* y *clock*.

Señales de salida:

- ***u_x***, ***u_y***: valores de pertenencia calculados para las entradas *x*, *y*, respectivamente.
- ***mf_ready***: señal de control que indica que se han terminado de evaluar las funciones de pertenencia solicitadas.
- ***ready***: señal de control que indica que el sistema ha terminado de operar y puede recibir nuevos datos.

3.3.1.1. Función de pertenencia triangular

El bloque TriangleMF implementa la función de pertenencia triangular, cuya ecuación se presenta a continuación:

$$triangle(x; a, b, c) = \max\left(\min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right), \quad (9)$$

y cuya gráfica se muestra en la figura 8. Como se puede observar, los parámetros *a*, *b* y *c* representan los tres vértices del triángulo.

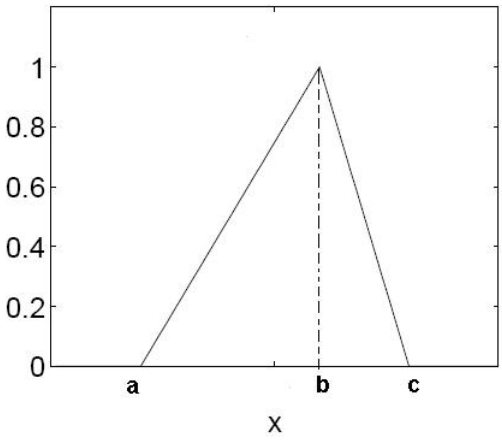


Figura 8. Función de pertenencia triangular.

La ecuación 10, que se presenta a continuación, es equivalente a la ecuación 9.

$$triangle(x;a,b,c) = \begin{cases} 0, & x \leq a \\ \frac{(x-a)}{(b-a)} = m_1 \times (x-a), & a < x < b \\ \frac{(c-x)}{(c-b)} = m_2 \times (c-x), & b < x < c \\ 0, & x \geq c \end{cases} \quad (10)$$

Como se puede observar en la ecuación 10, la función triángulo se puede separar en dos tramos de recta, cada uno con una determinada pendiente. Sobre la base de esta observación, se realiza el bloque TriangleMF. Además, se toma en consideración que las pendientes m_1 y m_2 son parámetros de entrada, como *a*, *b* y *c*, lo cual evita tener que realizar una división.

En la figura 9, se muestra el bloque TriangleMF con sus entradas y salidas (se describen las más significativas), y en la figura 10, se presenta el diagrama ASMD de este bloque que consiste de dos estados: *inicio_resta* y *mult*.

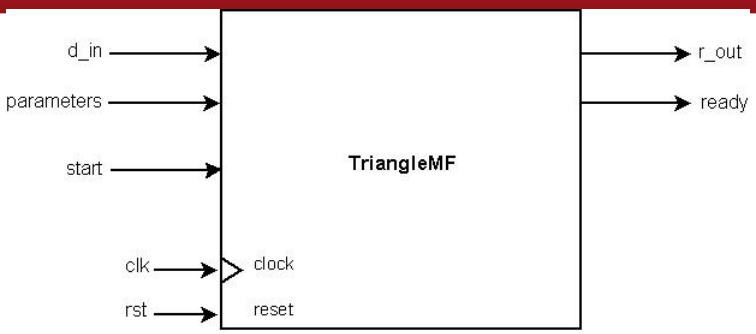


Figura 9. Bloque TriangleMF.

Señales de entrada:

- ***d_in***: valor numérico de entrada. (Corresponde a una de las entradas *x* o *y*).
- ***parameters***: vector de valores numéricos que contiene todos los parámetros de la función triangular. Estos son *a*, *b*, *c*, m_1 y m_2.

Señal de salida:

- **r_out**: valor numérico que corresponde al valor de pertenencia calculado.

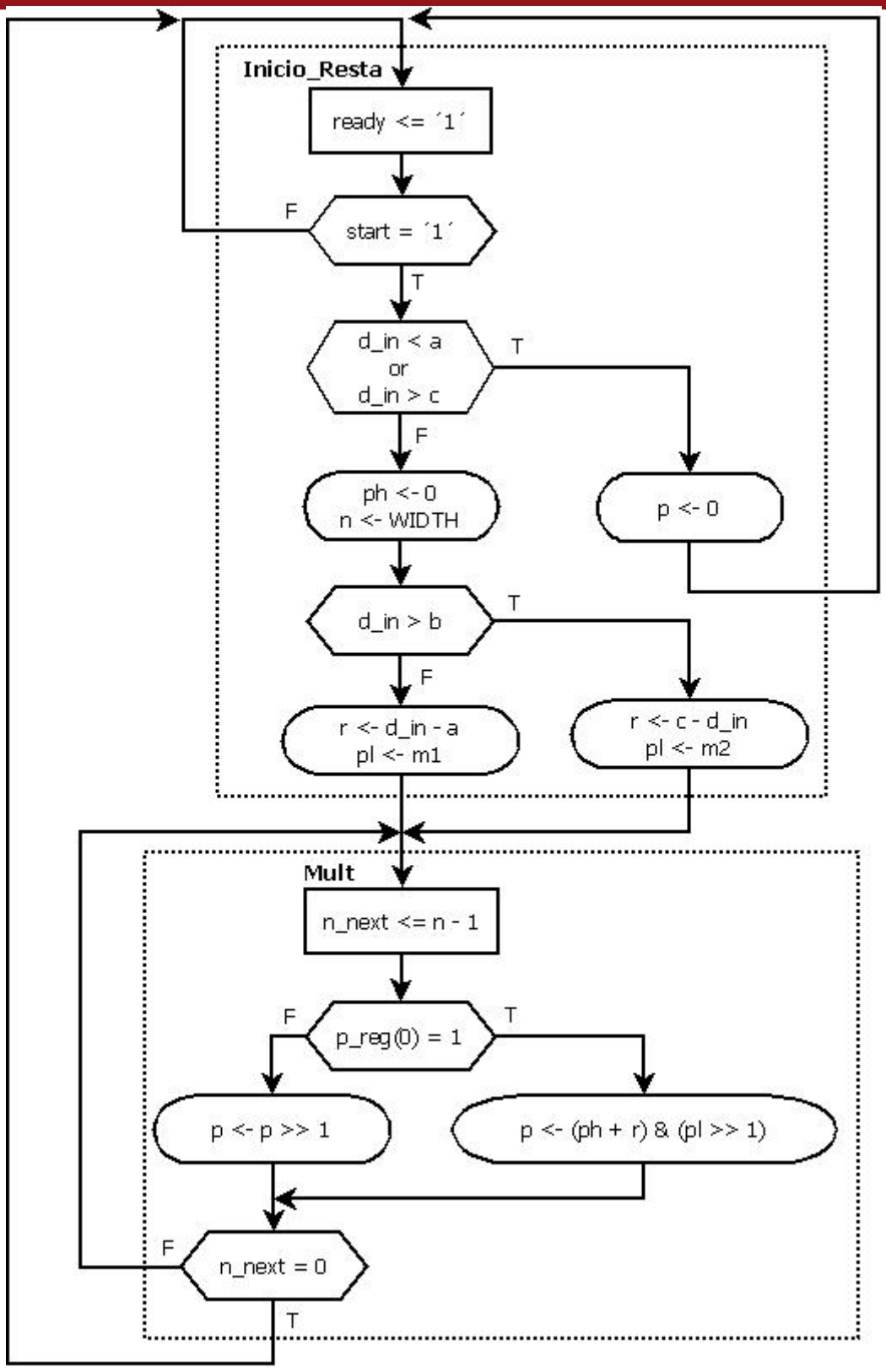


Figura 10. Diagrama ASMD del bloque TriangleMF.

3.3.1.2. Memorias de los Antecedentes

Cada memoria (ROM_x, ROM_y) almacena todos los parámetros (*a*, *b*, *c*, m_1 y m_2) de todas las funciones de pertenencia correspondientes a una entrada, en forma de vectores. A continuación, en la figura 11, se muestra un extracto de una porción de código en la que aparecen los parámetros guardados en la memoria ROM_x.

```
--    -- a_x --                -- b_x --                -- c_x --                -- m1_x --               -- m2_x --
0 => ("1110010001000010000010" & "1111101101011101010110" & "0000100010111001010l" & "0000101100010101010100" & "0001001100101001101O"),
1 => ("1010111011111100111010" & "0000000000000000011011" & "0101000011010010000O" & "0000001100101000111OO" & "0000001100101010111l"),
2 => ("1111011101000100001O" & "0000010010011110001O" & "0001101111011100100ll" & "0001001100101110010OO" & "0000101100010100001O1"),
3 => ("0000000000000000000O" & "0000000000000000000O" & "0000000000000000000O" & "0000000000000000000O" & "0000000000000000000O"),
```

Figura 11. Extracto de una porción de código de la memoria ROM_x.

Como se puede observar, en cada posición de memoria se almacenan todos los parámetros de una función de pertenencia. En la primera columna, se almacena *a*; en la segunda, *b*; en la tercera, *c*; en la cuarta, m_1; y en la quinta, m_2.

3.3.1.3. Controlador del Bloque Fuzzificador

Este bloque se encarga de coordinar las operaciones de los bloques internos del Fuzzificador. Su diagrama ASMD se muestra en la figura 12. Tiene tres estados: *inicio*, *op* y *espera*. A continuación, se describe cada estado.

En *inicio*, se espera hasta que se reciba la señal de *start* para iniciar la operación del Fuzzificador. Cuando esta señal cambia a nivel alto, se capturan los datos de entrada *x* y *y*, los cuales se almacenan en los registros *x_reg* y *y_reg*, respectivamente. Además, se inicializa el registro *addr* que indica la posición de memoria, que es utilizada por ROM_x y ROM_y.

En *op*, se inicia la operación de los bloques TriangleMF_x y TriangleMF_y mediante la señal *start_mfs*. Estos bloques operaran con los parámetros de las memorias ROM_x y ROM_y, respectivamente.

En *espera*, se espera hasta que ambas funciones de pertenencia hayan terminado de operar para activar la señal *mf_ready*. Además, se incrementa la dirección de memoria *addr*.

El proceso se repite hasta que se hayan evaluado todas las funciones de pertenencia para todas las direcciones de memoria, lo cual está determinado por la constante MF_NUM.

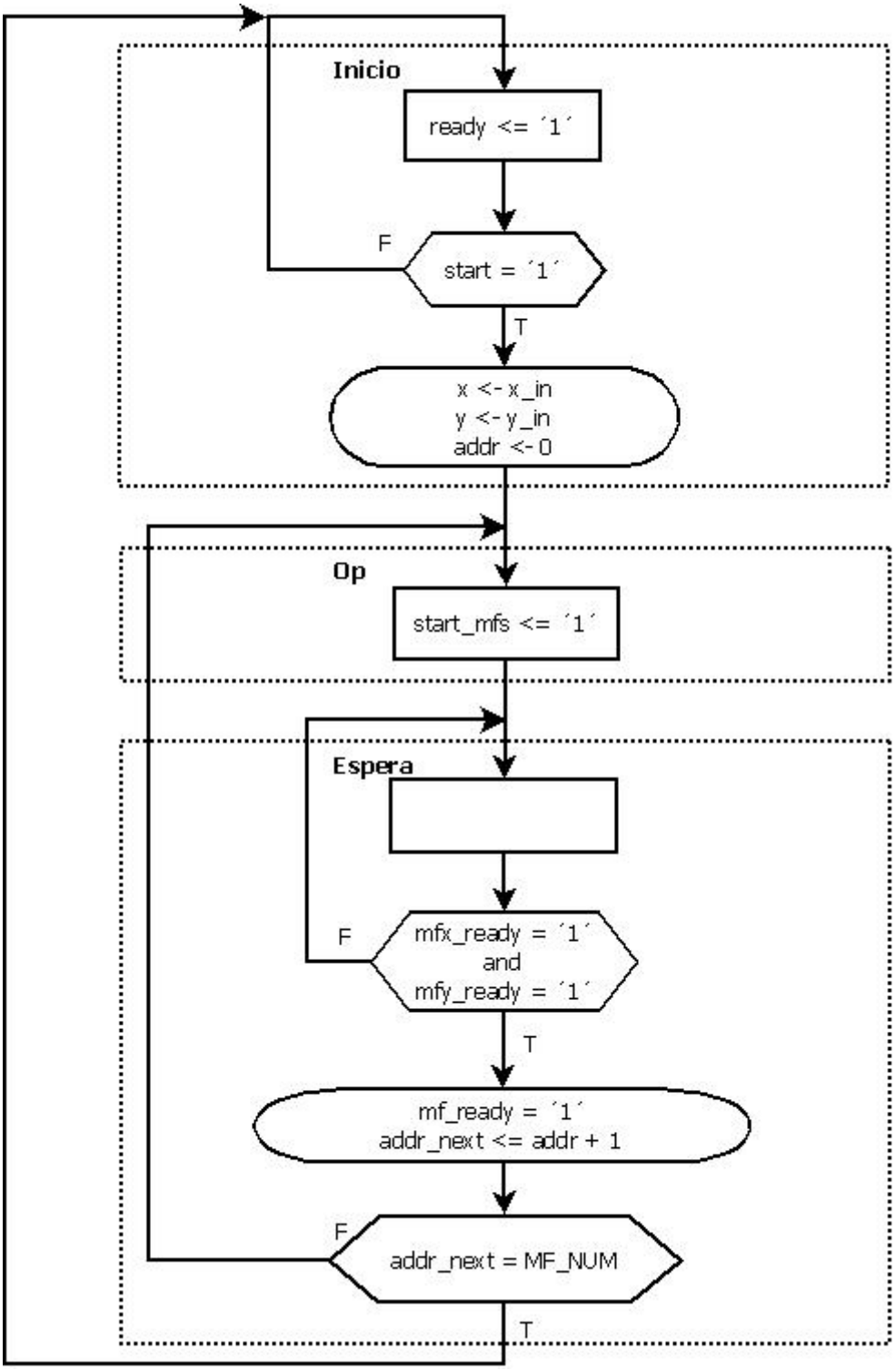


Figura 12. Diagrama ASMD del Controlador del Fuzzificador.

3.3.2. Bloque Permutador

El diagrama de bloques del Permutador se muestra en la figura 13. Está compuesto por dos registros de desplazamiento y un controlador.

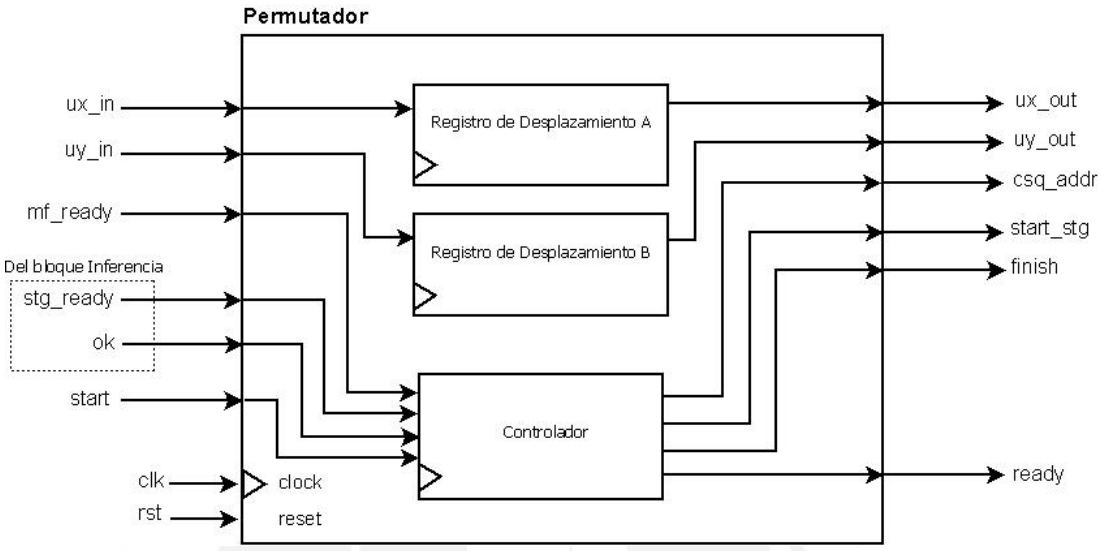


Figura 13. Diagrama de bloques del Permutador. (*No se muestran las señales internas de control).

Señales de entrada:

- ***ux_in***, ***uy_in***, ***mf_ready***: señales del bloque anterior.
- ***stg_ready***: señal de control del siguiente bloque, Inferencia, que se activa cuando este bloque está listo para recibir datos.
- ***ok***: señal de control del bloque Inferencia, que se activa cuando este bloque ha recibido la señal *finish*.

Señales de salida:

- ***ux_out***, ***uy_out***: valores de pertenencia de las entradas *x*, *y*, respectivamente. Cambian de acuerdo a todas las combinaciones posibles.
- ***csq_addr***: señal que contiene la dirección de memoria de la memoria de los consecuentes.
- ***start_stg***: señal de control que da inicio a la operación del bloque Inferencia.
- ***finish***: señal que indica que se han terminado todas las permutaciones posibles de los valores de pertenencia almacenados.

En la tabla 2, que se muestra a continuación, se ilustra el proceso de permutación para el caso MF_NUM = 3, luego de que los valores de pertenencia ya han sido guardados en los registros de desplazamiento respectivos.

Tabla 2. Proceso de permutación para MF_NUM = 3.

N° de secuencia	$R = [r(2) \quad r(1) \quad r(0)]$, $r(0) = salida$
1	$R_A = [\mu_{As}(x) \quad \mu_{A2}(x) \quad \mu_{A1}(x)]$, $R_B = [\mu_{Bs}(x) \quad \mu_{B2}(x) \quad \mu_{B1}(x)]$
2	$R_A = [\mu_{As}(x) \quad \mu_{A2}(x) \quad \mu_{A1}(x)]$, $R_B = [\mu_{B1}(x) \quad \mu_{Bs}(x) \quad \mu_{B2}(x)]$
3	$R_A = [\mu_{As}(x) \quad \mu_{A2}(x) \quad \mu_{A1}(x)]$, $R_B = [\mu_{B2}(x) \quad \mu_{B1}(x) \quad \mu_{Bs}(x)]$
4	$R_A = [\mu_{A1}(x) \quad \mu_{As}(x) \quad \mu_{A2}(x)]$, $R_B = [\mu_{Bs}(x) \quad \mu_{B2}(x) \quad \mu_{B1}(x)]$
5	$R_A = [\mu_{A1}(x) \quad \mu_{As}(x) \quad \mu_{A2}(x)]$, $R_B = [\mu_{B1}(x) \quad \mu_{B2}(x) \quad \mu_{Bs}(x)]$
6	$R_A = [\mu_{A1}(x) \quad \mu_{A2}(x) \quad \mu_{As}(x)]$, $R_B = [\mu_{Bs}(x) \quad \mu_{B1}(x) \quad \mu_{B2}(x)]$
7	$R_A = [\mu_{A2}(x) \quad \mu_{A1}(x) \quad \mu_{As}(x)]$, $R_B = [\mu_{Bs}(x) \quad \mu_{B2}(x) \quad \mu_{B1}(x)]$
8	$R_A = [\mu_{A2}(x) \quad \mu_{A1}(x) \quad \mu_{As}(x)]$, $R_B = [\mu_{B1}(x) \quad \mu_{Bs}(x) \quad \mu_{B2}(x)]$
9	$R_A = [\mu_{A2}(x) \quad \mu_{A1}(x) \quad \mu_{As}(x)]$, $R_B = [\mu_{Bs}(x) \quad \mu_{B1}(x) \quad \mu_{B2}(x)]$

3.3.2.1. Registro de Desplazamiento

El diagrama de bloques del Registro de Desplazamiento se muestra en la figura 14. Este registro, llamado R, está compuesto de varios registros conectados entre sí. El número de registros internos depende del número de funciones de pertenencia que se hayan definido (MF_NUM). Es decir, si se han definido tres funciones de pertenencia, entonces, se tendrán tres registros internos, uno por cada valor de pertenencia.

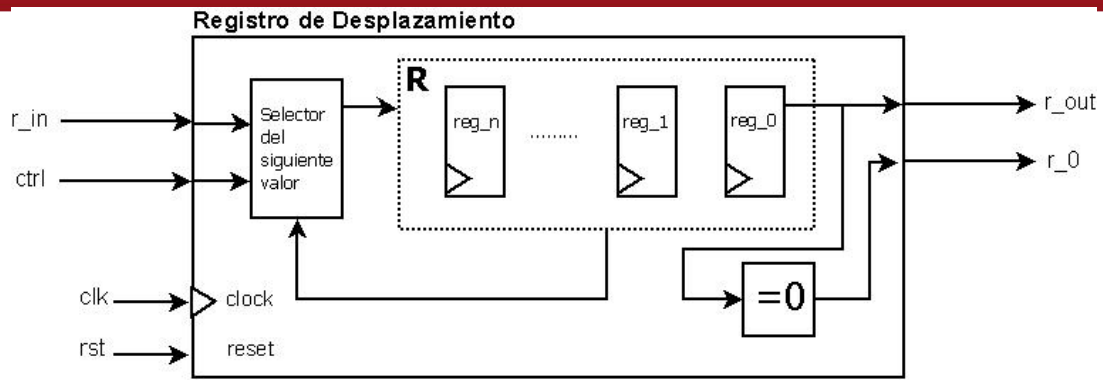


Figura 14. Diagrama de bloques del Registro de Desplazamiento. (*No se muestran todas las señales internas).

Señales de entrada:

- ***r_in***: valor numérico que corresponde a *ux_in* o a *uy_in*.
- ***ctrl***: señal de control que permite decidir el valor siguiente de todo el registro de desplazamiento.

Señales de salida:

- ***r_out***: valor numérico que corresponde a la salida del registro *0*.
- ***r_0***: señal de control que indica que el valor del registro *0,* que forma parte del registro de desplazamiento, es igual a cero.

La señal *ctrl* determina el valor siguiente del Registro de Desplazamiento R, de acuerdo a la tabla 3.

Tabla 3. Operaciones según la señal *ctrl* del Registro de Desplazamiento.

ctrl	**Operación**	**Significado**
"00"	$R \leftarrow R$	Se mantiene el valor.
"01"	$R[r(n) \dots r(1)\ r(0)] \leftarrow R[r(0)\ \dots\ r(1)]$	Se desplaza el registro hacia la derecha, circularmente.
"10"	$R[r(n) \dots r(1)\ r(0)] \leftarrow R[r_{in} \dots r(1)]$	Se desplaza el registro hacia la derecha, ingresando *r_in*.
"11"	$R \leftarrow R$	Se mantiene el valor.

3.3.2.2. Controlador del bloque Permutador

Este bloque se ocupa de controlar todas las operaciones que se deben realizar en el Permutador. En la figura 15, se presenta el diagrama ASMD de este controlador. Tiene ocho estados: *inicio*, *espera_dato*, *ini_n1*, *AoB_0*, *espera_mult*, *siguiente*, *ini_n2* y *fin*. A continuación, se describe cada estado.

En *inicio*, se espera hasta que se reciba la señal de *start* para iniciar la operación del Permutador. Cuando esta señal cambia a nivel alto, se inicializan los registros internos *n1* y *n2*, que funcionan como contadores, y *csq_addr*.

En *espera_dato*, se espera hasta que la señal *mf_ready* cambie a nivel alto para almacenar los valores de pertenencia *u_x* y *u_y* en los registros de desplazamiento respectivos. Este proceso se repite hasta que se hayan almacenado todos los valores de pertenencia.

En *ini_n1*, se reinicializa el valor del registro *n1*.

En *AoB_0*, se analiza si uno o los dos valores de pertenencia, en las salidas de los registros de desplazamiento, son iguales a cero. Si ese es el caso, entonces se saltea el estado *espera_mult*. Esto permite que solo se evalúen las reglas activas.

En *espera_mult*, se espera hasta que la señal *stg_ready* se active para enviar la señal *start_stg* que da inicio a la operación del bloque Inferencia.

En *siguiente*, se desplazan los registros A y B hacia la siguiente combinación determinada por los valores de los registros *n1* y *n2*. Este proceso se repite hasta que ambos registros sean iguales a cero, lo que significa que se han permutado todas las combinaciones posibles.

En *ini_n2*, se reinicializa el valor del registro *n2*.

En el último estado, *fin*, se le indica al bloque Inferencia mediante la señal *finish* que se ha concluido la operación y se espera hasta que este responda por medio de la señal *ok*.

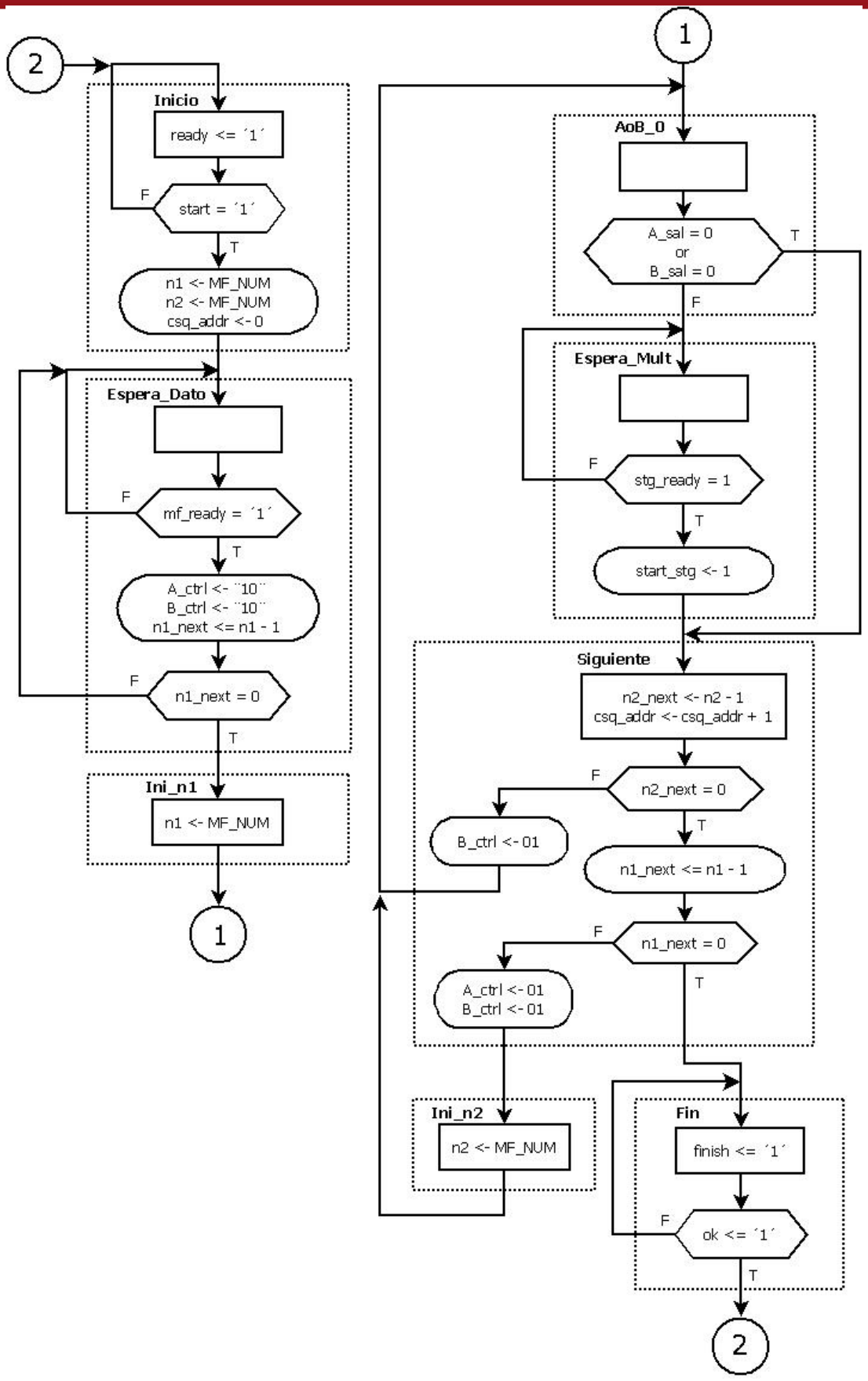


Figura 15. Diagrama ASMD del Controlador del Permutador.

3.3.3. Bloque Inferencia

El diagrama de bloques de Inferencia se muestra en la figura 16. Está compuesto por tres etapas y un controlador. La primera etapa está formada por un registro para guardar la dirección de memoria de los consecuentes y por un multiplicador, que multiplica los valores de pertenencia para obtener el grado de activación o peso de la regla correspondiente. La segunda etapa está formada por la memoria de los consecuentes, un multiplicador que multiplica el peso de la regla y el consecuente correspondientes, y un registro que almacena momentáneamente el peso de la regla. Finalmente, la tercera etapa está formada por dos acumuladores, uno para el numerador y otro para el denominador.

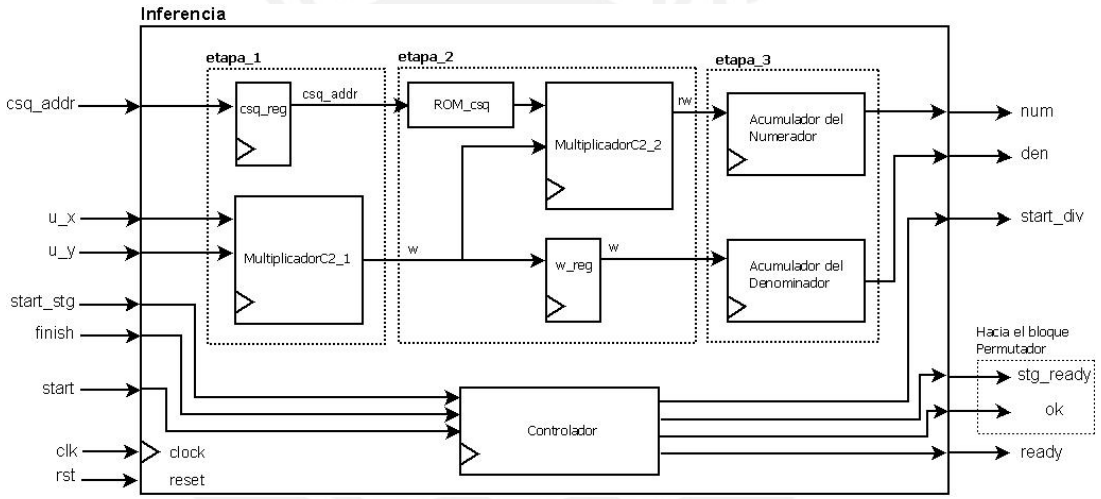


Figura 16. Diagrama de bloques de Inferencia. (*No se muestran las señales internas de control).

Señales de entrada:

- ***csq_addr***, ***u_x***, ***u_y***, ***start_stg***, ***finish***: señales del bloque anterior.

Señales de salida:

- ***num***, ***den***: valores numéricos que corresponden al numerador y el denominador.
- ***start_div***: señal de control que da inicio a la operación de división en el Defuzzificador.
- ***stg_ready***, ***ok***: señales descritas anteriormente.

En la tabla 4, se muestra un ejemplo de todo el proceso realizado por el bloque Inferencia. En este ejemplo, se consideran nueve reglas (MF_NUM = 3) y, además, que $\mu_{B3}(y) = 0$.

Tabla 4. Proceso por etapas del bloque Inferencia.

	Etapa 1	Etapa 2	Etapa 3
1	$w_1 = \mu_{A1}(x) \times \mu_{B1}(y)$	$rw = 0$, $w_{reg} = 0$	$num = 0$, $den = 0$
2	$w_2 = \mu_{A1}(x) \times \mu_{B2}(y)$	$rw_1 = r_1 \times w_1$, $w_{reg} = w_1$	$num = 0$, $den = 0$
3	$w_4 = \mu_{A2}(x) \times \mu_{B1}(y)$	$rw_2 = r_2 \times w_2$, $w_{reg} = w_2$	$num = 0 + rw_1$, $den = 0 + w_1$
4	$w_5 = \mu_{A2}(x) \times \mu_{B2}(y)$	$rw_4 = r_4 \times w_4$, $w_{reg} = w_4$	$num = num + rw_2$, $den = den + w_1$
5	$w_7 = \mu_{A3}(x) \times \mu_{B1}(y)$	$rw_5 = r_5 \times w_5$, $w_{reg} = w_5$	$num = num + rw_4$, $den = den + w_4$
6	$w_8 = \mu_{A3}(x) \times \mu_{B2}(y)$	$rw_7 = r_7 \times w_7$, $w_{reg} = w_7$	$num = num + rw_5$, $den = den + w_5$
7	---	$rw_8 = r_8 \times w_8$, $w_{reg} = w_8$	$num = num + rw_7$, $den = den + w_7$
8	---	---	$num = num + rw_8$, $den = den + w_8$

Al final, los valores del numerador y del denominador son los siguientes:

$$num = \sum rw = rw_1 + rw_2 + rw_4 + rw_5 + rw_7 + rw_8 \tag{11}$$

$$den = \sum w = w_1 + w_2 + w_4 + w_5 + w_7 + w_8 \tag{12}$$

3.3.3.1. Multiplicador de números con signo

Este bloque consiste en un multiplicador de números con signo en complemento a dos, que está basado en el algoritmo de multiplicación de Booth [15]. El diagrama ASMD que implementa este algoritmo se muestra en la figura 17.

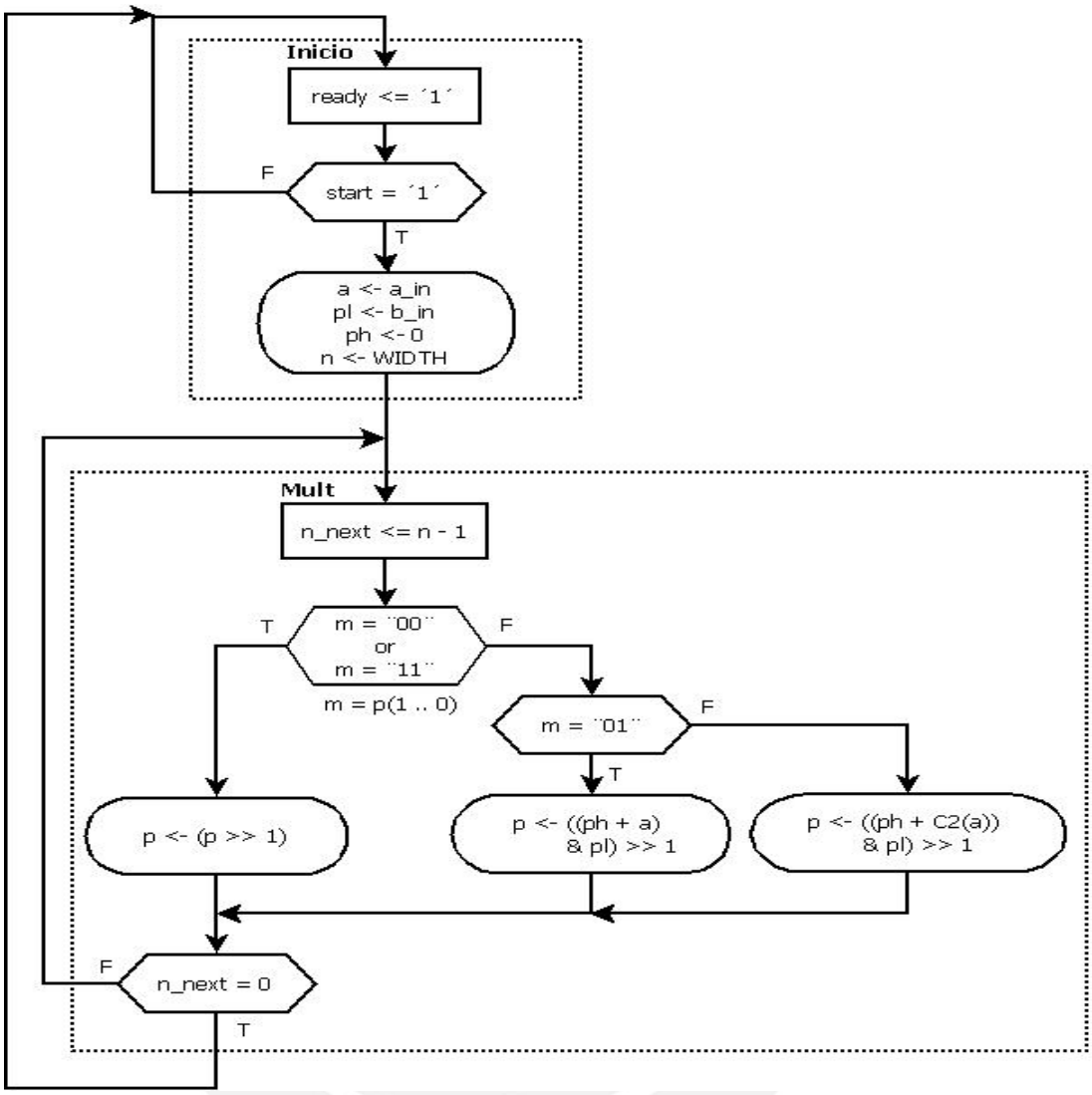


Figura 17. Diagrama ASMD del Multiplicador.

3.3.3.2. Memoria de los consecuentes

En esta memoria se almacenan los parámetros de los consecuentes. Para cada regla se tiene un parámetro, que es una constante. A continuación, en la figura 18, se muestra una porción de código de la memoria de los consecuentes, donde se puede observar que cada parámetro está almacenado en una posición de memoria. En este código, MF_NUM = 3, lo cual significa que hay nueve reglas.

```
--          -- Consecuentes --
   0 => "00010111010100011110", -- R1
   1 => "10111000011011111001", -- R2
   2 => "00010111010100011110", -- R3
   3 => "11111111111111100000", -- R4
   4 => "00000000000000100010", -- R5
   5 => "11111111111111100000", -- R6
   6 => "11101000101100101100", -- R7
   7 => "01000111100000101101", -- R8
   8 => "11101000101100101100", -- R9
   9 => "00000000000000000000",
```

Figura 18. Extracto de una porción de código de la memoria ROM_csq.

3.3.3.3. Acumulador

El diagrama de bloques del Acumulador se muestra en la figura 19. Este bloque está compuesto de un sumador, un bloque de multiplexaje y un registro. Implementa una función de sumatoria, es decir, se suma el valor almacenado con el valor de la entrada cada vez que se requiera. Además, se puede inicializar en cero en cualquier momento.

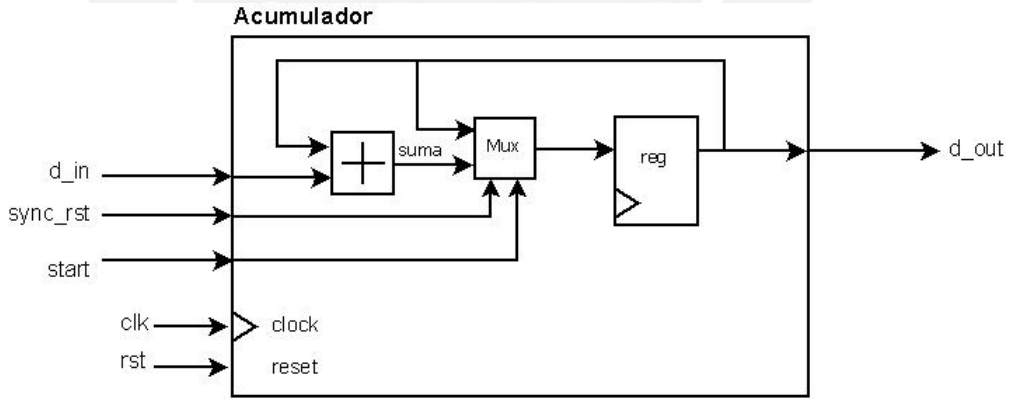


Figura 19. Diagrama de bloques del Acumulador.

Señales de entrada:

- ***d_in***: valor numérico de entrada que se quiere acumular.
- ***sync_rst***: señal de *reset* síncrono que inicializa el registro a 0.

Señales de salida:

- ***d_out***: valor numérico de salida que corresponde al valor acumulado en el registro.

3.3.3.4. Controlador del bloque Inferencia

Este bloque se encarga de controlar las operaciones dentro del bloque Inferencia. En la figura 20, se muestra el diagrama ASM del Controlador. Consiste de cinco estados: *inicio*, *etapa_ini*, *espera*, *etapa_fin1* y *etapa_fin2*. A continuación, se describe cada etapa.

En inicio, se espera hasta que la señal *start* indique el comienzo de la operación del sistema. Cuando esto ocurre, se inicializan todos los registros de las tres etapas que conforman el bloque Inferencia.

En *etapa_ini*, se indica mediante la señal *stg_ready* que se pueden recibir datos de entrada. El Permutador responderá con la señal *start_stg* cada vez que se requiera realizar una multiplicación de los valores de pertenencia diferentes de cero. Se activan todas las etapas del bloque Inferencia.

En *espera*, se espera hasta que todas las etapas hayan terminado de operar. Luego se regresa a *etapa_ini*. Este proceso se repite hasta que se active la señal *finish*, lo cual quiere decir que todos los valores de pertenencia presentados por el Permutador han sido multiplicados. En este caso, solo se activan las dos últimas etapas antes del cambio de estado.

En *etapa_fin1*, se espera hasta que la etapa dos haya concluido su operación.

En *etapa_fin2*, se espera a que la última etapa termine, es decir, el acumulador, lo cual dura un ciclo de reloj.

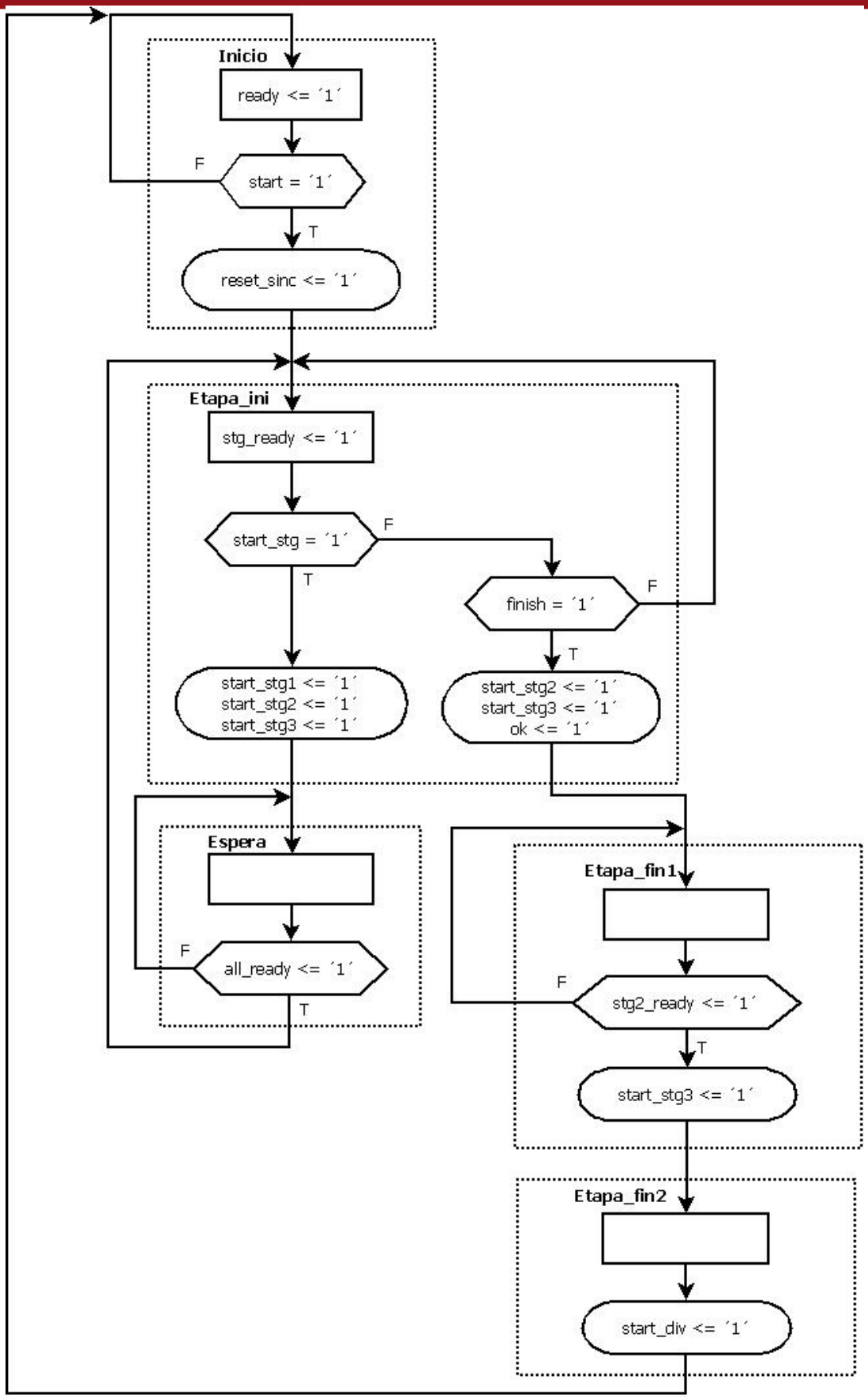


Figura 20. Diagrama ASMD del Controlador de Inferencia.

3.3.4. Bloque Defuzzificador

El diagrama de bloques del Defuzzificador se muestra en la figura 21. Está compuesto por un divisor y un controlador.

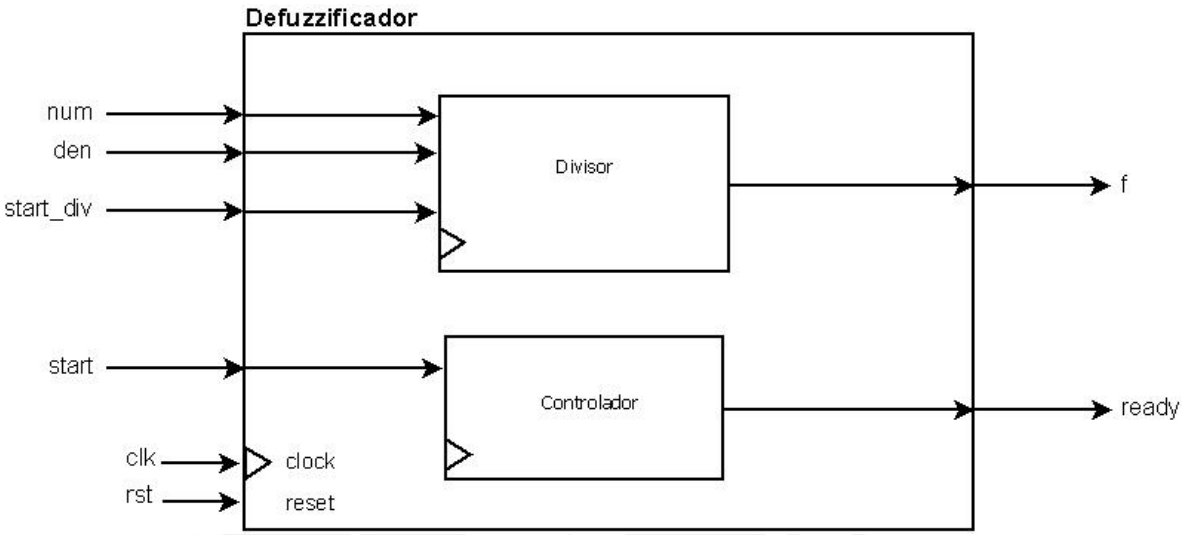


Figura 21. Diagrama de bloques del Defuzzificador. (*No se muestran las señales internas de control).

Señales de entrada:

- ***num***, ***den***: valores numéricos que corresponden al numerador (dividendo) y el denominador (divisor) de la división.
- ***start_div***: señal de control que da inicio a la operación de división.

Señal de salida:

- ***f***: valor numérico que corresponde al resultado de la división.

En la ecuación 13, que se presenta a continuación, se muestra el valor de la salida del Defuzzificador.

$$f = \frac{num}{den} = \frac{\Sigma_t \, rw_t}{\Sigma_t \, w_t} \qquad (13)$$

3.3.4.1. Divisor

Este bloque consiste en un divisor, que está basado en el algoritmo de división sin restauración [15]. El diagrama ASMD que implementa este algoritmo se muestra en la figura 22.

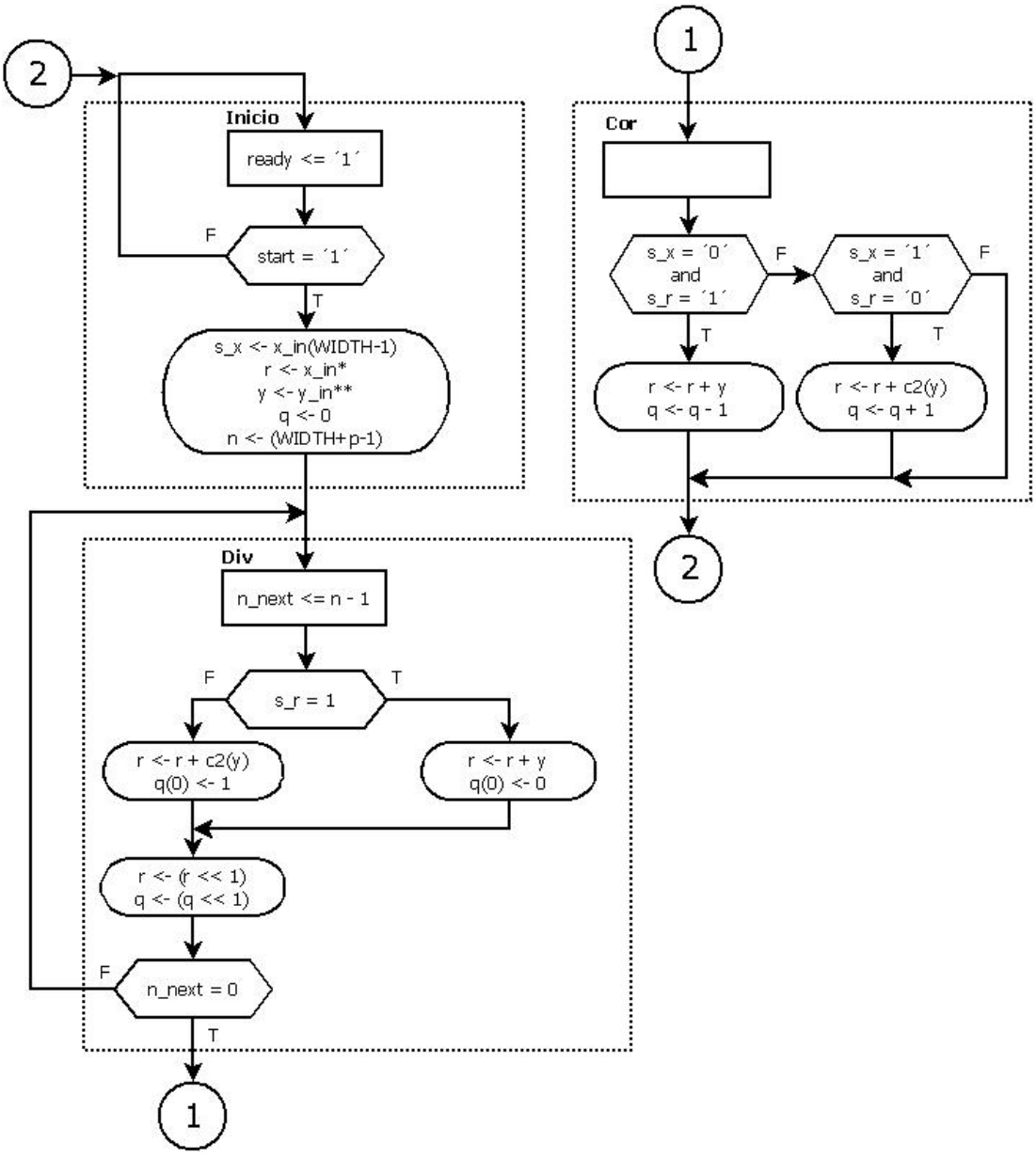


Figura 22. Diagrama ASMD del Divisor.

3.3.4.2. Controlador del bloque Defuzzificador

Este bloque se encarga de esperar hasta que el divisor termine de operar para activar la señal *ready* del Defuzzificador. En la figura 23, se muestra el diagrama ASM del Controlador. Consiste de tres estados: *inicio*, *espera*, *divide*.

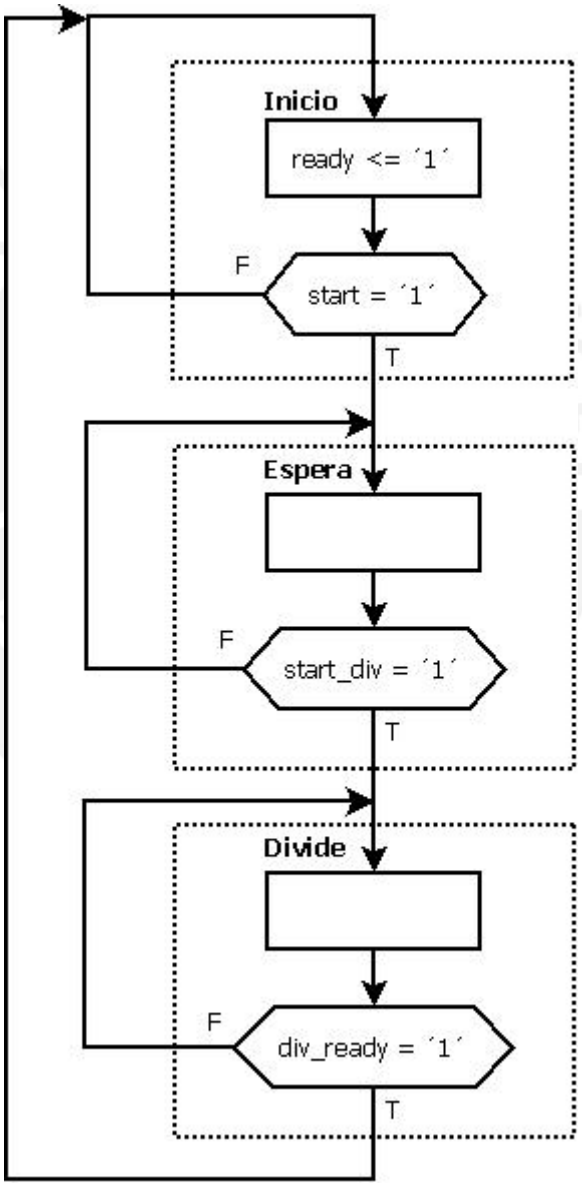


Figura 23. Diagrama ASM del Controlador del Defuzzificador.

3.4. Diseño en VHDL del ANFIS

En primer lugar, todos los bloques y diagramas presentados anteriormente, que forman parte del sistema ANFIS, fueron descritos mediante el lenguaje de descripción de hardware VHDL. Para algunos bloques, como TriangleMF_x y TriangleMF_y, Registro de Desplazamiento A y Registro de Desplazamiento B, y MultiplicadorC2_1 y MultiplicadorC2_2, solo fue necesario realizar la descripción de estos una vez, ya que luego fueron instanciados dos veces.

En segundo lugar, todos los bloques se describieron utilizando constantes genéricas, por lo que son parametrizables. Esto les permite adaptarse a los requerimientos del sistema ANFIS, ya que este también es parametrizable. A continuación, se muestran los parámetros globales de todo el sistema.

- **MF_NUM**: Número de funciones de pertenencia.
- **WIDTH_I**: Cantidad de bits de la parte entera de las entradas y de los parámetros de los antecedentes.
- **WIDTH_F**: Cantidad de bits de la parte fraccional de las entradas y de los parámetros de los antecedentes.
- **WIDTH_I_CSQ**: Cantidad de bits de la parte entera de los consecuentes.
- **WIDTH_F_CSQ**: Cantidad de bits de la parte fraccional de los consecuentes.
- **TRCT_1**: Precisión en la multiplicación del bloque Fuzzificador. Por defecto, TRCT_1 = 1. (2: Completa, 1: Truncado)
- **TRCT_2**: Precisión en la primera multiplicación del bloque Inferencia. Por defecto, TRCT_2 = 1. (2: Completa, 1: Truncado)
- **E**: Cantidad de bits necesarios de la parte entera de la salida.
- **P**: Cantidad de bits de la parte fraccional de la salida. Determina la precisión de la división que se realiza en el bloque Defuzzificador.

Finalmente, cabe mencionar que es recomendable que los parámetros TRCT_1 y TRCT_2 mantengan sus valores por defecto, ya que de otro modo se utilizará una mayor cantidad de recursos y el sistema demorará más en calcular un resultado, aunque este tenga una mayor precisión.

CAPÍTULO 4. Simulaciones y Resultados

En el presente capítulo, se presentan las simulaciones que permiten verificar el funcionamiento del sistema diseñado, así como también los resultados obtenidos para estas simulaciones. Primero, se elige una función de dos entradas y una salida que se quiere generar y se realiza el entrenamiento del ANFIS, utilizando el software MATLAB, versión 7.7.0 (R2008b), para poder obtener los parámetros de los antecedentes y de los consecuentes. Segundo, se realizan dos simulaciones sobre dos FPGA distintos con el fin de mostrar que la descripción en VHDL no tiene dependencia con un FPGA en particular. Por una parte, se simula el sistema ANFIS sobre el FPGA Spartan-3 XC3S200 [16], que cuenta con una baja cantidad de elementos lógicos, para lo cual se utiliza el simulador ISim del entorno de desarrollo ISE 11 de la empresa Xilinx, fabricante de este FPGA. Por otra parte, se simula el sistema sobre el FPGA Cyclone II EP2C35 [17], que cuenta con una cantidad mucho mayor de elementos lógicos, para lo cual se utiliza el simulador del entorno de desarrollo Quartus II 9.1 de la empresa Altera, fabricante de este FPGA. Finalmente, se presentan y analizan los resultados obtenidos en ambas simulaciones y se hace una comparación con el ANFIS de MATLAB.

4.1. Obtención de los Parámetros del ANFIS en MATLAB

En esta etapa, se utiliza el ANFIS de MATLAB [18]. Para esto, se ha realizado un script (ver Anexos) que permite, a partir de una función de dos entradas y una salida, entrenar al sistema ANFIS y obtener los parámetros de los antecedentes y consecuentes de las reglas. Estos parámetros son convertidos a valores binarios y son guardados en archivos de texto con el formato adecuado para que puedan ser copiados en las memorias ROM respectivas del sistema ANFIS en VHDL.

La función que se va a generar, la cual es una función no lineal, se expresa en la ecuación 14 y su gráfica se muestra en la figura 24.

$$f(x,y) = 10 \cdot x \cdot e^{(-x^2-y^2)}, \qquad x, y \in [-2, 2] \tag{14}$$

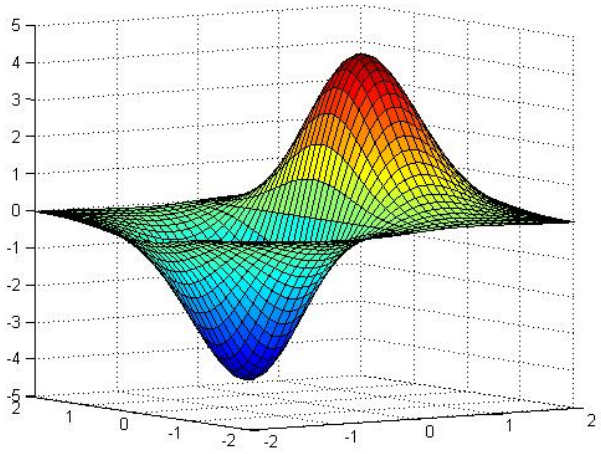


Figura 24. Gráfica de la función de la ecuación 14. (Gráfica con 1681 puntos).

Se fija el número de funciones de pertenencia por entrada a cinco, ya que con este número se logra una buena aproximación de la función. Tras ejecutar el script de MATLAB se obtiene que el sistema ANFIS ha aproximado la superficie que se muestra en la figura 25.

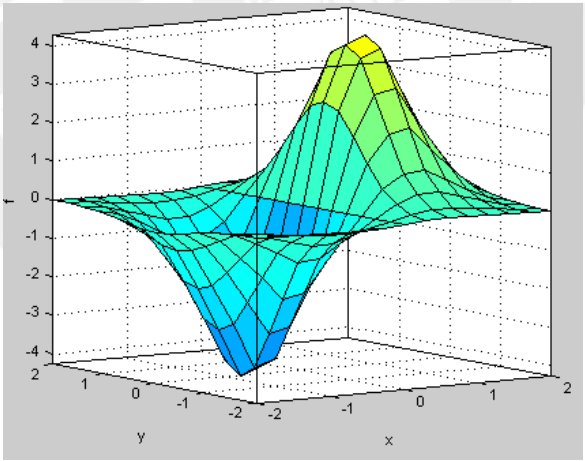


Figura 25. Superficie del ANFIS entrenado. (Gráfica con 101 puntos).

Asimismo, para validar el entrenamiento realizado, se eligen veinte valores al azar, que pertenecen al dominio de la función, y se grafica, para estos valores, la respuesta del ANFIS y la salida de la función. Esto se muestra en la figura 26. Como se observa, en la mayoría de los casos, el ANFIS aproxima con bastante exactitud a la función, aunque en algunos presenta una diferencia.

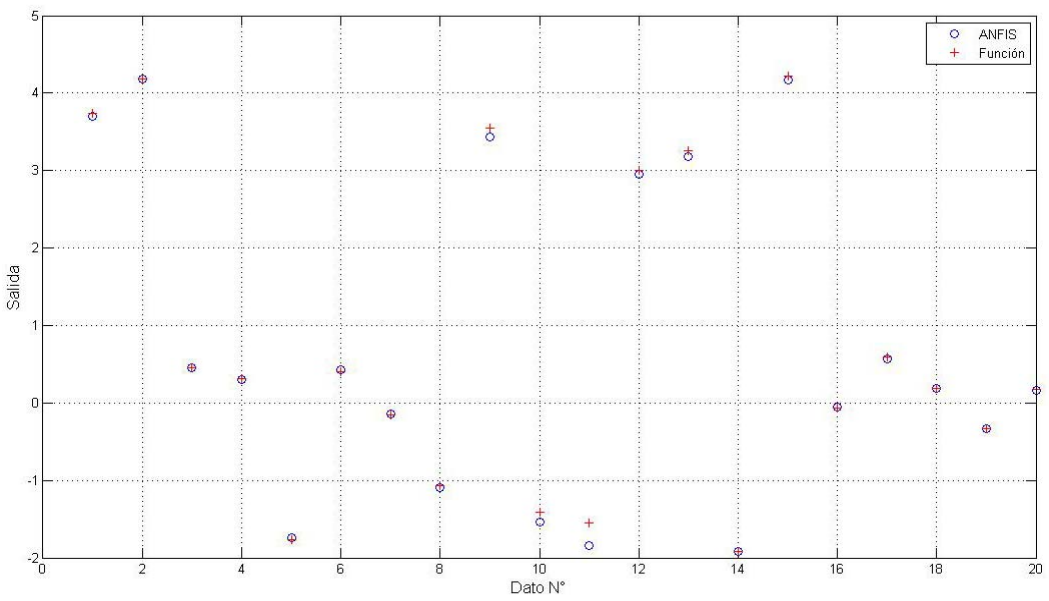


Figura 26. Respuesta del ANFIS y la función para los veinte valores al azar.

Finalmente, se obtienen los valores de los parámetros de los antecedentes y de los consecuentes. A continuación, se muestran las funciones de pertenencia para las entradas *x* e *y*, en las figuras 27 y 28, respectivamente. Además, se muestran los valores de los consecuentes en la tabla 5, los cuales han sido redondeados a cuatro decimales para una mejor visualización.

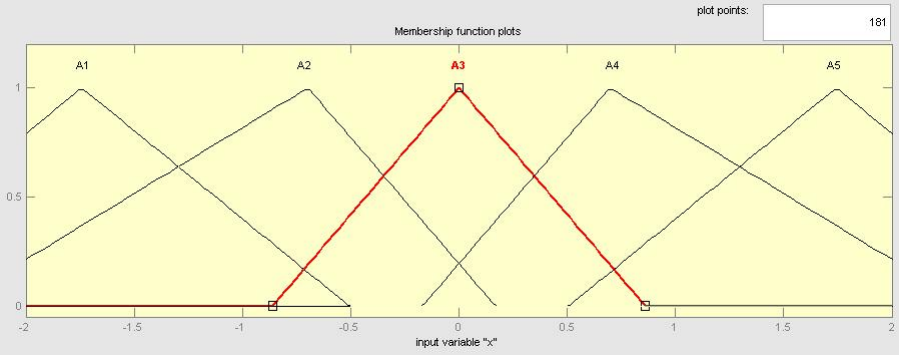


Figura 27. Funciones de pertenencia triangulares para la entrada *x*.

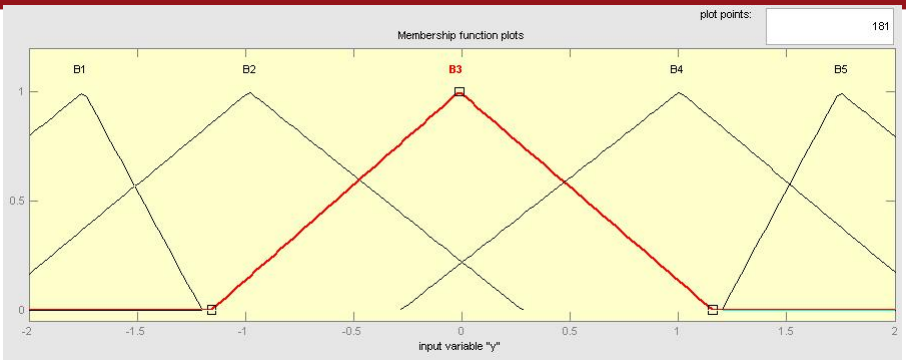


Figura 28. Funciones de pertenencia triangulares para la entrada *y*.

Tabla 5. Valores de los consecuentes.

$r_1 = -0.0331$	$r_2 = 0.2734$	$r_3 = 1.5620$	$r_4 = 0.2742$	$r_5 = -0.0360$
$r_6 = 0.1702$	$r_7 = -1.4077$	$r_8 = -8.0430$	$r_9 = -1.4121$	$r_{10} = 0.1852$
$r_{11} = 0$	$r_{12} = 0$	$r_{13} = 0$	$r_{14} = 0$	$r_{15} = 0$
$r_{16} = -0.1702$	$r_{17} = 1.4077$	$r_{18} = 8.0430$	$r_{19} = 1.4121$	$r_{20} = -0.1852$
$r_{21} = 0.0331$	$r_{22} = -0.2734$	$r_{23} = -1.5620$	$r_{24} = -0.2742$	$r_{25} = 0.0360$

4.2. Simulaciones

Para simular el sistema, se tomaron los veinte valores utilizados anteriormente para validar el ANFIS de MATLAB. Primero, se muestran las simulaciones realizadas en el ISIM del ISE 11 sobre el FPGA Spartan-3 XC3S200 y, luego, se muestran las simulaciones realizadas en el simulador del Quartus II 9.1 sobre el FPGA Cyclone II EP2C35. Ambas simulaciones son del tipo *Post-Place and Route Simulation* (*Timing*), es decir, que consideran los retardos en los componentes del FPGA. Además, se realizan a una frecuencia de reloj de 50 MHz.

4.2.1. Simulación en ISim del ISE 11

La figura 29 muestra la simulación realizada en el simulador ISim. En este caso, se ha utilizado una precisión de 8 bits en la parte fraccional de las entradas, de los parámetros y de la salida. En la figura, los valores de las entradas no se pueden visualizar, debido a que estos tienen una duración de un ciclo de reloj. En cambio, los valores de la salida tienen una duración mayor.

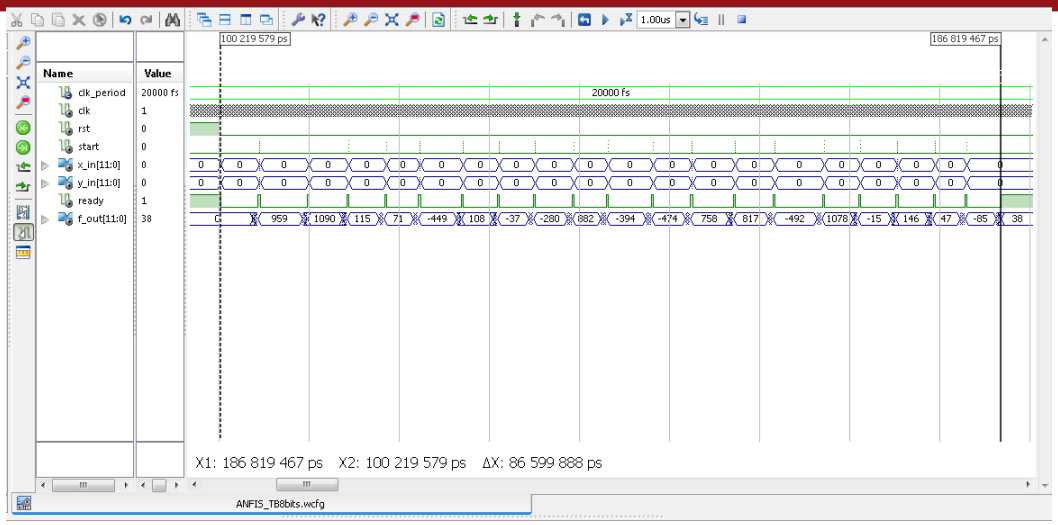


Figura 29. Simulación realizada en ISim para veinte pares de valores. (P = 8 bits).

En la tabla 6, se muestran los valores de las entradas y salidas de la figura anterior, así como el tiempo de respuesta del sistema para cada valor. Cabe mencionar, que los datos en el simulador se visualizan como números enteros, por lo cual es necesario convertirlos a punto-fijo, mediante la ecuación 15, que se muestra después de la tabla. Además, se han redondeado estos valores a cuatro decimales.

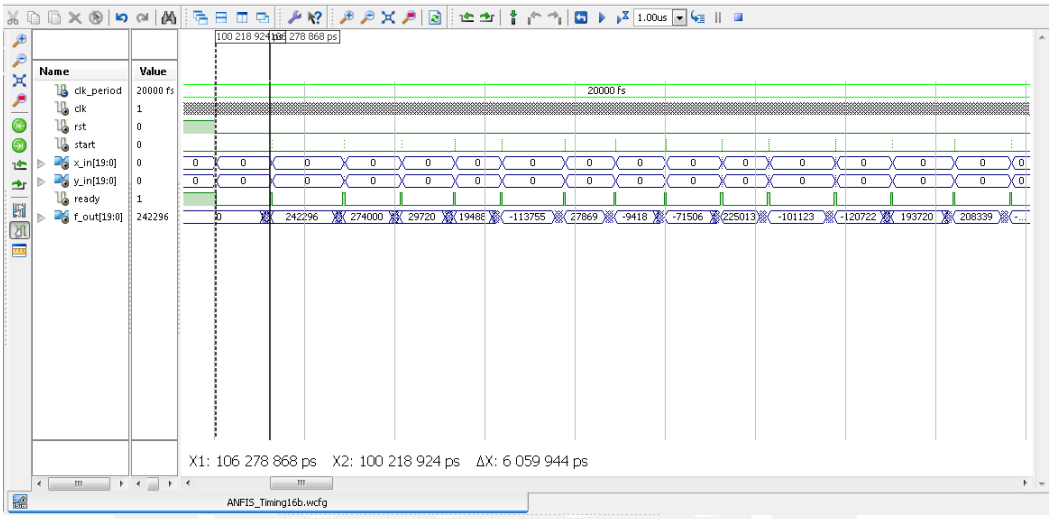
Tabla 6. Valores de las entradas y salidas de la figura 29.

	Entrada x	Entrada y	Salida f	Tiempo (μs)
1	0.5898	0.3281	3.7461	4.14
2	0.7148	0.1641	4.2578	5.24
3	0.5430	1.4805	0.4492	4.18
4	1.7813	-0.9414	0.2773	3.96
5	-1.1641	-0.7266	-1.7539	3.54
6	0.8359	-1.5234	0.4219	4.54
7	-1.0547	1.7578	-0.1445	3.78
8	-1.5234	0.5820	-1.09375	3.78
9	0.4297	-0.0820	3.4453	4.1
10	-0.1992	0.5586	1.5391	3.54
11	-0.1641	0.1797	-1.8516	4.78
12	0.6484	0.5898	2.9610	4.14
13	1.0820	0.1758	3.1914	4.56
14	-0.5977	0.8828	-1.9219	4.16
15	0.6484	0.0898	4.2110	5.24
16	-0.3359	1.9766	-0.0586	3.78
17	1.3672	-1.1250	0.5703	3.96
18	1.3320	-1.5781	0.1836	3.92
19	-0.9727	-1.5625	-0.3320	3.3
20	0.4531	-1.7461	0.1484	3.78

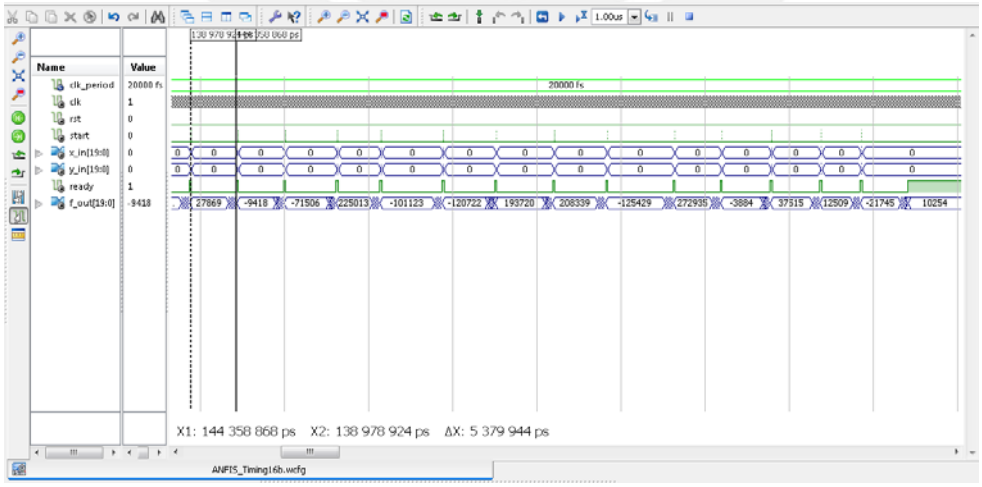
$$num_{fp} = num_{int} \times \frac{1}{2^P}, \quad (15)$$

donde *P* es la cantidad de bits de la parte fraccional. En este caso $P = 8$.

También se realizó una simulación utilizando una precisión de 16 bits en la parte fraccional de las entradas, de los parámetros y de la salida. La figura 30 muestra esta simulación.



(a)



(b)

Figura 30. Simulación realizada en ISim para veinte pares de valores. (P = 16).

En la tabla 7, se muestran los valores de las entradas y salidas de la figura anterior, así como el tiempo de respuesta del sistema para cada valor. Asimismo, se utiliza la ecuación 15 para convertir los valores a punto-fijo. En este caso $P = 16$. Nótese que el valor de las entradas cambia también, debido a la precisión.

Tabla 7. Valores de las entradas y salidas de la figura 30.

	Entrada x	Entrada y	Salida f	Tiempo (μs)
1	0.5905	0.3290	3.6971	6.06
2	0.7161	0.1630	4.1809	7.8
3	0.5432	1.4798	0.4535	6.1
4	1.7807	-0.9409	0.2974	5.72
5	-1.1643	-0.7277	-1.7358	4.98
6	0.8371	-1.5231	0.4252	6.78
7	-1.0551	1.7593	-0.1437	5.38
8	-1.5224	0.5822	-1.0911	5.38
9	0.4292	-0.0822	3.4334	6.02
10	-0.1994	0.5573	-1.5430	4.98
11	-0.1651	0.1789	-1.8421	7.18
12	0.6478	0.5892	2.9559	6.06
13	1.0811	0.1755	3.1790	6.64
14	-0.5991	0.8842	-1.9139	6.08
15	0.6480	0.0900	4.1647	7.8
16	-0.3354	1.9748	-0.0593	5.38
17	1.3677	-1.1253	0.5724	5.72
18	1.3317	-1.5768	0.1909	5.68
19	-0.9742	-1.5612	-0.3318	4.58
20	0.4538	-1.7456	0.1565	5.46

4.2.2. Simulación en Quartus II 9.1

En este caso, también se realizaron simulaciones para 8 bits y 16 bits en la parte fraccional de las entradas, de los parámetros y de la salida. Estas se muestran en las figuras 31 y 32, y los valores obtenidos son los mismos de la tabla 6 y 7, respectivamente.

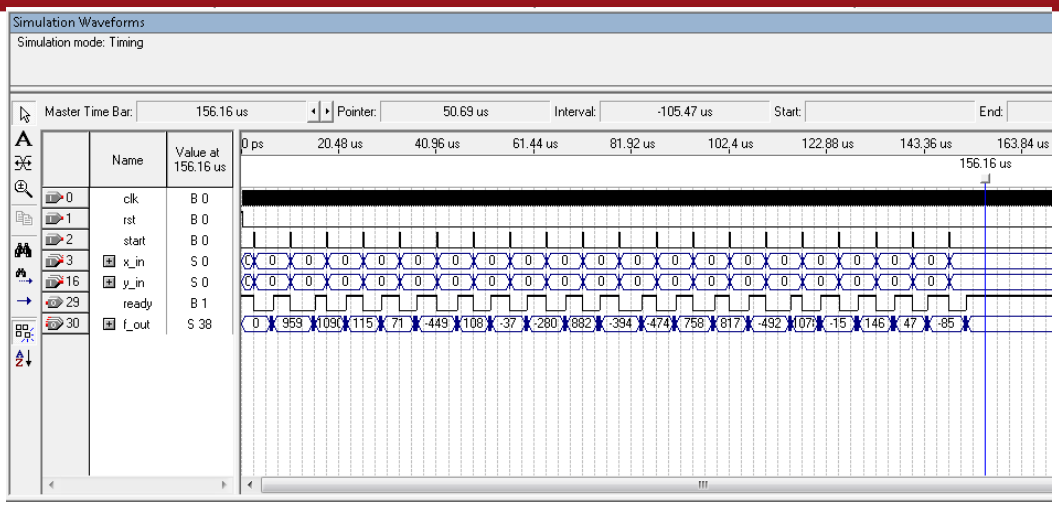
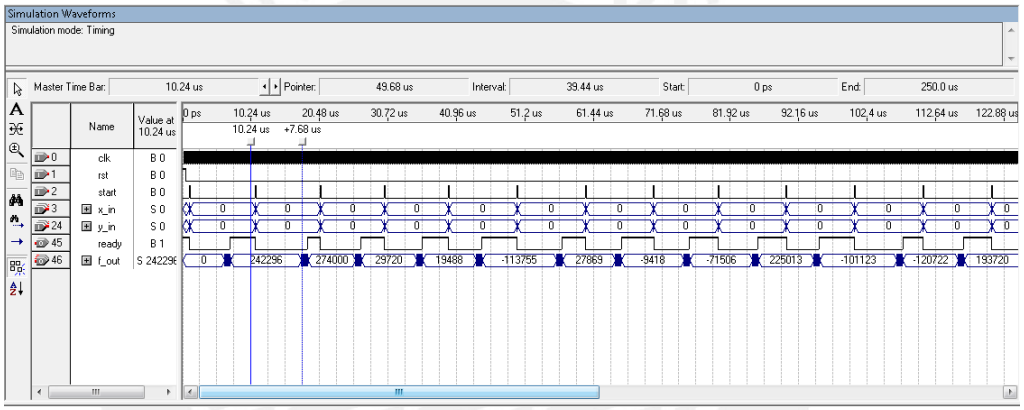
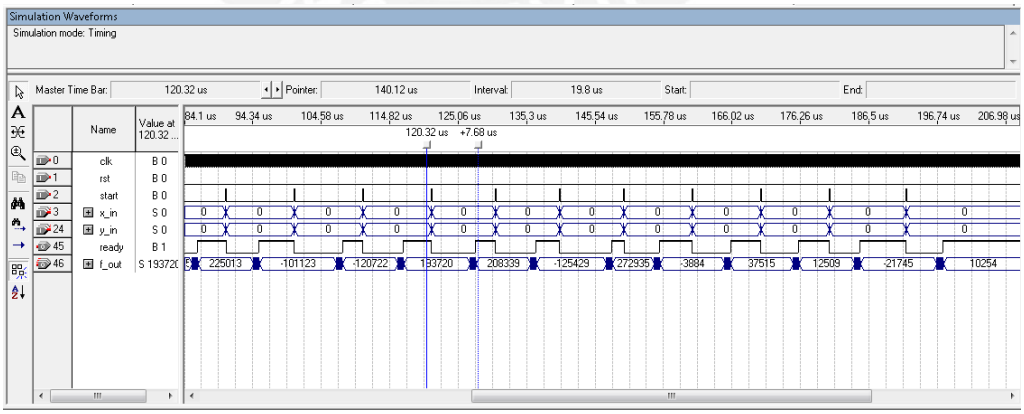


Figura 31. Simulación realizada en Quartus II 9.1. (P = 8).



(a)



(b)

Figura 32. Simulación realizada en Quartus II 9.1. (P = 16).

4.3. Resultados

En primer lugar, se presentan los resultados de los procesos de síntesis e implementación para ambos FPGA. Por un lado, en la figura 33 y 34, se muestra la cantidad de recursos lógicos que se utilizan del FPGA Spartan-3 XC3S200, cuando la precisión en la parte fraccional es de 8 bits y cuando es de 16 bits, respectivamente.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	422	3,840	10%	
Number of 4 input LUTs	928	3,840	24%	
Number of occupied Slices	512	1,920	26%	
Number of Slices containing only related logic	512	512	100%	
Number of Slices containing unrelated logic	0	512	0%	
Total Number of 4 input LUTs	944	3,840	24%	
Number used as logic	928			
Number used as a route-thru	16			
Number of bonded IOBs	40	173	23%	
Number of BUFGMUXs	1	8	12%	
Average Fanout of Non-Clock Nets	4.25			

Figura 33. Cantidad de recursos utilizados del FPGA Spartan-3 XC3S200. (P = 8).

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	674	3,840	17%	
Number of 4 input LUTs	1,435	3,840	37%	
Number of occupied Slices	790	1,920	41%	
Number of Slices containing only related logic	790	790	100%	
Number of Slices containing unrelated logic	0	790	0%	
Total Number of 4 input LUTs	1,460	3,840	38%	
Number used as logic	1,435			
Number used as a route-thru	25			
Number of bonded IOBs	64	173	36%	
Number of BUFGMUXs	1	8	12%	
Average Fanout of Non-Clock Nets	4.17			

Figura 34. Cantidad de recursos utilizados del FPGA Spartan-3 XC3S200. (P = 16).

Por otro lado, en la figura 35 y 36, se muestra la cantidad de recursos lógicos que se utilizan del FPGA Cyclone II EP2C35, cuando la precisión en la parte fraccional es de 8 bits y cuando es de 16 bits, respectivamente.

Quartus II Version	9.1 Build 222 10/21/2009 SJ Full Version
Revision Name	ANFIS
Top-level Entity Name	ANFIS
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	774 / 33,216 (2 %)
Total combinational functions	708 / 33,216 (2 %)
Dedicated logic registers	426 / 33,216 (1 %)
Total registers	426
Total pins	40 / 475 (8 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Figura 35. Cantidad de recursos utilizados del FPGA Cyclone II EP2C35. (P = 8).

Quartus II Version	9.1 Build 222 10/21/2009 SJ Full Version
Revision Name	ANFIS
Top-level Entity Name	ANFIS
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	1,172 / 33,216 (4 %)
Total combinational functions	1,045 / 33,216 (3 %)
Dedicated logic registers	678 / 33,216 (2 %)
Total registers	678
Total pins	64 / 475 (13 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Figura 36. Cantidad de recursos utilizados del FPGA Cyclone II EP2C35. (P = 16).

En el FPGA Spartan-3, se utiliza el 26% (P = 8 bits) y 41% (P = 16 bits) de los elementos lógicos disponibles, mientras que en el Cyclone II se utiliza el 2% (P = 8 bits) y 4% (P = 16 bits). La implementación en estos dos FPGA distintos permite comprobar que la descripción realizada en VHDL permite una fácil síntesis e implementación. Asimismo, el sistema se puede implementar no solo en un FPGA que tiene una gran cantidad de recursos lógicos, como el Cyclone II EP2C35, sino también en un FPGA con una cantidad de recursos lógicos mucho menor, como el Spartan-3 XC3S200.

En segundo lugar, se realiza una comparación entre los valores calculados por el ANFIS de MATLAB y los valores calculados por el ANFIS de VHDL para los veinte valores al azar, tanto para 8 bits como para 16 bits de precisión en la parte fraccional. Esto se muestra en la tabla 8. Nuevamente, se redondean los valores a cuatro decimales para una mejor visualización.

Tabla 8. Comparación con el ANFIS de MATLAB.

	ANFIS (MATLAB)	ANFIS (VHDL) , 8 bits	ANFIS (VHDL), 16 bits	Error (%), 8 bits	Error (%), 16 bits
1	3.6970	3.7461	3.6971	1.3281	0.0027
2	4.1807	4.2578	4.1809	1.8442	0.0048
3	0.4535	0.4492	0.4535	0.9482	0
4	0.2974	0.2773	0.2974	6.7586	0
5	-1.7358	-1.7539	-1.7358	1.0427	0
6	0.4253	0.4219	0.4252	0.7994	0.0235
7	-0.1437	-0.1445	-0.1437	0.5567	0
8	-1.0911	-1.09375	-1.0911	0.2429	0
9	3.4334	3.4453	3.4334	0.3466	0
10	-1.5430	-1.5391	-1.5430	0.2528	0
11	-1.8419	-1.8516	-1.8421	0.5266	0.0109
12	2.9560	2.9610	2.9559	0.1691	0.0034
13	3.1790	3.1914	3.1790	0.3901	0
14	-1.9139	-1.9219	-1.9139	0.4180	0
15	4.1645	4.2110	4.1647	1.1166	0.0048
16	-0.0593	-0.0586	-0.0593	1.1804	0
17	0.5725	0.5703	0.5724	0.3843	0.0175
18	0.1909	0.1836	0.1909	3.8240	0
19	-0.3318	-0.3320	-0.3318	0.0603	0
20	0.1565	0.1484	0.1565	5.1757	0

En el caso de 8 bits, el error promedio es de 1.3683 %, mientras que en el caso de 16 bits, es de 0.0034 %. La precisión del sistema no solo está limitada a estos casos, sino que se puede especificar libremente mediante las constantes genéricas del sistema. Dependiendo de los requerimientos de la aplicación, se podrá elegir la precisión que mejor se adecue.

Finalmente, se calcula el tiempo de respuesta del sistema promedio, en base a los datos mostrados en la tabla 6 y 7. De la tabla 6, el tiempo es de 4.1214 μs y de la tabla 7, el tiempo es de 5.9890 μs . Por otra parte, se calcula el tiempo que demora el ANFIS de MATLAB en dar una respuesta, para lo cual se utiliza la herramienta *Profiler* y un script. Aunque este tiempo presenta ciertas variaciones, es de alrededor de 100 ms . Comparado con este valor, el sistema propuesto es mucho más rápido y, por consiguiente, se presta para aplicaciones en tiempo real.

CONCLUSIONES

1. A partir de las simulaciones realizadas sobre los FPGA Spartan-3 XC3S200 y Cyclone II EP2C35, a través de las cuales se comprobó el correcto funcionamiento del sistema en la generación de funciones, se concluye que se ha realizado con éxito el diseño de la arquitectura propuesta para el sistema neurodifuso ANFIS, cumpliendo con el objetivo general de la presente tesis.

2. El diseño modular y genérico, es decir, en base a parámetros, permite reconfigurar al sistema fácil y rápidamente para que se pueda generar cualquier función requerida. Más aún, la aplicación del sistema a la generación de funciones no limita su uso y puede ser utilizado en otras áreas de aplicación en futuros trabajos.

3. La descripción en VHDL de la arquitectura diseñada es portable y no depende del FPGA elegido, lo cual se pudo comprobar al implementar el diseño tanto en el FPGA Spartan-3 XC3S200 como en el FPGA Cyclone II EP2C35. Asimismo, esto permite que en un futuro trabajo se pueda migrar la arquitectura diseñada a un chip integrado de aplicación específica, conocido como ASIC (Application-specific Integrated Circuit).

4. El diseño realizado hace un uso óptimo de los recursos para cumplir con la tarea designada. Se pudo comprobar que no es necesaria una arquitectura que sea totalmente paralela para lograr un buen desempeño y un tiempo de respuesta rápido.

5. El tiempo de respuesta del sistema es muy pequeño en comparación con el tiempo requerido por una aplicación en software como MATLAB. Esto implica que el diseño propuesto puede ser utilizado en una aplicación en tiempo real.

RECOMENDACIONES

En la presente tesis se realizó el diseño de una arquitectura para un sistema neurodifuso ANFIS. Para el diseño, se consideró un sistema de orden cero y de dos entradas y una salida. Así mismo, se utilizaron funciones de pertenencia triangulares en los antecedentes de las reglas difusas. Sin embargo, otras funciones de pertenencia, como las trapezoidales o las de forma de campana, pueden ser usadas. En ese caso, sería suficiente con reemplazar el bloque TriangleMF por el bloque correspondiente que implemente la función de pertenencia deseada, sin requerir mayores modificaciones. Por otra parte, se puede utilizar, en un futuro trabajo, la mayoría de los bloques del sistema actual para el diseño de la arquitectura de un sistema neurodifuso ANFIS de tres entradas y una salida.

En relación al entrenamiento del sistema para la generación de una función, cabe mencionar que mientras mayor sea la cantidad de funciones de pertenencia por entrada, mejor será la aproximación de la función por parte del sistema, aunque también será mayor el uso de recursos lógicos utilizados y el tiempo de respuesta. De igual manera, mientras mayor sea la cantidad de bits utilizados en la parte fraccional de todas las señales que representan valores numéricos, mayor será la precisión del sistema. Por lo tanto, dependerá de los requerimientos específicos de la aplicación fijar el grado de precisión necesario.

Finalmente, para una simulación sobre un FPGA de la empresa Altera, se recomienda utilizar el simulador ModelSim Altera, debido a que futuras versiones del entorno de desarrollo Quartus II no contarán con el simulador que se utilizó en la presente tesis. Del mismo modo, para una simulación sobre un FPGA de la empresa Xilinx, se puede utilizar el simulador ModelSim XE, el cual es más potente que el simulador ISim y permite realizar simulaciones en menor tiempo.

BIBLIOGRAFÍA

[1] D. Nauck, F. Klawonn, R. Kruse
1993 "Combining Neural Networks and Fuzzy Controllers", 8th Austrian Artificial Intelligence Conference, FLAI '93, Pages 35 – 46

[2] H. R. Berenji, P. Khedkar
1992 "Learning and tuning fuzzy logic controllers through reinforcements", IEEE Transactions on Neural Networks, Vol. 3, Pages 724 – 740

[3] J. S. R. Jang
1993 "ANFIS: Adaptive-Network-Based Fuzzy Inference Systems", IEEE Transactions on Systems, Man, and Cybernetics, Vol. 23, Pages 665 – 685

[4] S. K. Halgamuge, M. Glesner
1994 "Neural networks in designing fuzzy systems for real world applications", Fuzzy Sets and Systems, Vol. 65, Pages 1 – 12

[5] D. Nauck, R. Kruse
1995 "A Neuro-Fuzzy Approach for the Classification of Data", Paper of Symposium on Applied Computing, SAC'95

[6] Simon Haykin
1994 "Neural Networks: a comprehensive foundation", Macmillan

[7] Kazuo Tanaka
1997 "An Introduction to Fuzzy Logic for Practical Applications", Springer

[8] D. Nauck
1997 "Neuro-Fuzzy Systems: Review and Prospects", 5th European Congress on Intelligent Techniques and Soft Computing, EUFIT'97, Pages 1044 – 1053

[9] J. S. R. Jang, C. T. Sun
1995 "Neuro-Fuzzy Modeling and Control", Proceedings of the IEEE, Vol. 83, No. 3, Pages 378 – 406

[10] M. Kumar, D. Garg
2005 "Neuro-Fuzzy Controller Applied to Multiple Robot Cooperative Control", International Journal of Industrial Robot, Vol. 32, No. 3, Pages 234 – 239

[11] T. Takagi, M. Sugeno
1983 "Derivation of fuzzy control rules from human operator's control actions", Proceedings of the IFAC Symposium on Fuzzy Information, Knowledge Representation and Decision Analysis, Pages 55 – 60

[12] Volnei A. Pedroni
2004 "Circuit Design with VHDL", MIT Press

[13] Stephen Brown, Zvonko Vranesic
2005 "Fundamentals of Digital Logic with VHDL Design", McGraw-Hill

[14] P. P. Chu
2006 "RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability", Wiley-IEEE Press

[15] J. P. Deschamps, G. A. Bioul, G. D. Sutter
2006 "Synthesis of arithmetic Circuits: FPGA, ASIC and Embedded Systems", Wiley-Interscience ISBN: 0-471-68783-9

[16] Xilinx DS099 Spartan-3 FPGA Family data sheet
2009 <http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf> (Consultado por última vez en junio de 2010)

[17] Altera's Corporation
2005 "Cyclone II Device Handbook", Vol. 1

[18] The Mathworks™
2010 "anfis and the ANFIS Editor GUI", <http://www.mathworks.com/access//helpdesk/help/toolbox/fuzzy/fp715dup12.html>
(Consultado por última vez en junio de 2010)