UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Máquina digital de un clasificador fuzzy artmap

TESIS

Que para obtener el título de

Ingeniero Eléctrico Electrónico

P R E S E N T A

Carlos Eduardo Salazar Parra

DIRECTOR DE TESIS

Dr. Víctor Manuel Lomas Barrie

Ciudad Universitaria, Cd. Mx., 2020

Dedicado para todos aquellos

que buscan un propósito en el

complicado camino de la vida

Agradecimientos

A mis padres, pues gracias a su apoyo durante el transcurso de mi carrera terminé un ciclo en mi vida, sin su ayuda no lo habría logrado. Soy afortunado de recibir su bondad, cariño y paciencia.

A mis hermanos, por ayudarme en los momentos en los que necesitaba de un consejo y compañía. Estoy agradecido de que sean parte de mi vida.

A mi director de tesis, Dr. Víctor Lomas, gracias por creer que sería capaz de realizar este proyecto. Sus asesorías me ayudaron a adentrarme a nuevas y fascinantes áreas de conocimiento.

Gracias a mis sinodales por dedicar tiempo para enriquecer mi trabajo.

A la famila Ibarra González, por hacerme sentir parte de su familia y apoyarme en los momentos en los que más necesitaba ayuda. Estoy profundamente agradecido por su amistad.

A *ma petite*, por tu valioso cariño que me acompañó cuando me sentía perdido. Gracias por tu apoyo incondicional.

Gracias al Programa UNAM-PAPIIT por brindar los fondos para que este proyecto se realizara.

Índice

1 Introducción....1
1.1 Objetivo....1
1.2 Planteamiento del problema....1
1.3 Alcance....2
1.4 Metodología....3
1.5 Antecedentes históricos de las RNA....3
2 Red Neuronal Artificial Fuzzy ARTMAP....4
2.1 Redes Neuronales Artificiales....4
2.1.1 Definición....4
2.1.2 Modelo básico de una neurona artificial....4
2.1.3 Clasificación y tipos....5
2.1.4 Redes Neuronales Artificiales implementadas en hardware....7
2.2 Representación en punto fijo....8
2.2.1 Ventajas de las arquitecturas de punto fijo....10
2.2.2 Operaciones aritméticas en punto fijo....10
2.3 Teoría de la Resonancia Adaptativa....11
2.4 Fuzzy ARTMAP....13
3 Caso de Estudio: BOF & FAM....16
4 Hardware Reconfigurable....17
4.1 Estructura de una FPGA....18
4.2 System on a Chip....21
4.3 Zybo Zynq-7000....23
4.3.1 Procesador ARM....24
4.3.2 Processing System....25
4.3.3 Programmable Logic....26
5 Máquina digital de un clasificador Fuzzy ARTMAP....27
5.1 Arquitectura del clasificador....27
5.1.1 Descripción del hardware....28
5.1.2 Recursos de lógica programable....29
5.2 Comunicación con el sistema embebido....34
5.2.1 Protocolo AXI4 Lite....34
5.3 Operaciones del clasificador sobre el sistema embebido....38
6 Pruebas y resultados....43
7 Conclusiones....58
8 Referencias....59
8.1 Figuras y gráficas....61

1 Introducción

1.1 Objetivo

Diseño de una máquina digital de un clasificador Fuzzy ARTMAP (FAM) implementado en *hardware* reconfigurable. Dicha máquina es controlada y supervisada por un sistema embebido. El clasificador sirve para seleccionar objetos dentro de una celda de manufactura inteligente.

1.2 Planteamiento del problema

Las Redes Neuronales Artificiales (RNA) tienen la función de clasificar información. Para que la RNA pueda clasificar, se necesita un proceso de aprendizaje en el cual se le provee de datos con el fin de ser capaces de reconocer y clasificar diferentes objetos de estudio. El clasificador Fuzzy ARTMAP propuesto debe ser capaz de reconocer nuevos objetos de estudio cuando sea necesario y su diseño debe tener un énfasis en eficiencia para proporcionar una pronta respuesta para el sistema. Una alternativa a las RNA implementadas en máquinas secuenciales, es por medio de *hardware* reconfigurable como se menciona en el artículo *Implementación de Red Neuronal Artificial en FPGA* (Raeisi & Kabir 2006).

Para la aplicación de las RNA resulta novedoso una máquina digital en FPGA para que se puedan realizar tareas en paralelo, esto con el fin de obtener una respuesta más rápida en comparación a su implementación en máquinas secuenciales o en computadoras personales. Debido a esto se ha seleccionado el uso de *hardware* sobre *software*. El propósito del clasificador Fuzzy ARTMAP es para entrenarlo a reconocer piezas de manufactura como se ha visto en el artículo de investigación *Máquina de Visión para Ensamblado Robótico* (Cabrera, Juárez, Cabrera, & Castuera 2005).

El modelo de RNA que se seleccionó es la Fuzzy ARTMAP debido a su "capacidad de conocimiento incremental y estabilidad, pero principalmente por su rápido reconocimiento y respuestas de clasificación geométrica" (Cabrera, Juárez, Cabrera, & Castuera 2005). Las características anteriores nos ayudarán a crear una máquina de aprendizaje que clasifique la mayor cantidad de formas en el menor tiempo de respuesta posible.

1.3 Alcance

El alcance del proyecto es el siguiente:

a) Se propone la arquitectura de una máquina digital clasificadora por medio de una RNA implementada en *hardware* reconfigurable que sea entrenada fuera de línea.

b) La información que procesará la máquina digital es el descriptor de una pieza de manufactura que se obtiene a partir de una imagen, el método para extraer el descriptor es *Boundary Object Points* (BOF) (Cabrera et al., 2005).

c) El clasificador FAM tendrá comunicación con la arquitectura *Processing System* (PS) del ZYNQ-7 con el fin de enviar los datos que serán catalogados por la Fuzzy ARTMAP y recibir los resultados de cada categoría. También se podrá enviar nuevos datos en cualquier momento, por medio del PS, para que sean catalogados por la Fuzzy ARTMAP.

d) El clasificador tendrá el máximo número de categorías que puedan ser implementadas en la FPGA, de las cuales una será seleccionada como la ganadora. Para realizar las operaciones de los datos en la Fuzzy ARTMAP, se realizó una arquitectura con aritmética de números de punto fijo.

1.4 Metodología

Antes de comenzar el desarrollo de la FAM, se realizó una investigación sobre sus antecedentes de implementación en *hardware* reconfigurable, encontrando que al momento de la realización de esta tesis es un tema que no se ha abordado profundamente en la literatura. Los trabajos que sirvieron como punto de partida son el artículo de investigación donde se documentó por primera vez el concepto de FAM (Grossberg, Markuzon, Carpenter, Reynolds, & Rosen 1992), además de los elementos que la conforman. Para abstraer el concepto de la FAM e implementarlo en *hardware*, se basó en la tesis del Dr. Víctor Lomas (Lomas Barrie 2016), donde se muestra el esquema de un clasificador Tj. Para el siguiente paso se realizaron pruebas en la tarjeta de desarrollo asignada, Zybo Zynq-7000, para comprobar que cuenta con los recursos necesarios para realizar la FAM. Una vez comprobado la capacidad de recursos de la tarjeta, se seleccionó representar la información de la máquina digital en punto fijo. Como paso final se diseñó la máquina digital del clasificador FAM con base a las características de la tarjeta asignada.

1.5 Antecedentes históricos de las RNA

El primer artículo en hablar de las RNA se publicó en el año de 1943 (McCulloch & Pitts 1943), en este trabajo se describe por medio de modelos matemáticos una red neuronal artificial. Para 1958 se presentaría uno de los fundamentos de las RNA, el perceptron (Rosenblat 1958), la cual es una neurona artificial que tiene un número limitado de entradas binarias que son multiplicadas por pesos y como resultado se obtiene en la salida el valor de cero o uno. Ciertas discrepancias y limitaciones con la propuesta de Roseblatt crearon una época de desinterés sobre las RNA, por lo que tendrían que pasar aproximadamente 10 años para que se retomaran las investigaciones. Uno de los varios avances que se hizo en ese resurgimiento son las Redes Hopfield (Hopfield 1982), cuya aportación es el modelo de una red con nodos que puede recuperar información. En el año de 1992 el científico Grossberg S. junto con Carpenter G., Markuzon N., Reynolds J. y Rosen D. desarrollaron la teoría de la Fuzzy ARTMAP partiendo de la ART (del inglés *Adaptive Resonance Theory*). Dicha teoría se aproxima en cómo el cerebro humano puede procesar información para reconocer o asignar categorías a los eventos o al mundo que lo rodea. El resultado de la colaboración de Grosseberg y compañía dio como resultado una "RNA con aprendizaje supervisado para reconocer categorías y mapas multidimensionales en respuesta a una secuencia arbitraria de vectores de entrada binarios o análogos" (Grossberg et al. 1992). Poco después surgen las SVM (del inglés *Support Vector Machine*), las cuales alejarían el interés de la comunidad científica por las RNA ya que las SMV tienen un buen rendimiento en el análisis de información con una distribución desconocida (Auria & Moro 2008). Sería hasta poco después de inciado el siglo XXI que el interés de las RNA regresaría debido a factores como el incremento en la velocidad de procesamiento de los microprocesadores, el incremento de memoria RAM, cómputo paralelo, además de la baja en costos en tarjetas de *hardware* reconfigurable y un incremento significativo en el poder de computo de éstos (Kuon et al. 2007). Algunos trabajos recientes de RNA son las investigaciones realizadas para clasificación de imágenes en reconocimiento de piezas de manufactura en el Instituto de Investigaciones de Matemáticas Aplicadas y Sistemas (Cabrera, Juárez, Cabrera, & Castuera 2005).

2 Red Neuronal Artificial Fuzzy ARTMAP

2.1 Redes Neuronales Artificiales

2.1.1 Definición

Las RNA son modelos computacionales que se basan en el comportamiento de las redes de neuronas del ser humano, sin embargo, no son una representación exacta de éstas. Estos modelos son implementados en *hardware* o *software* con la intención de imitar la toma de decisiones como lo realiza el ser humano. Las RNA trabajan en paralelo distribuyendo su trabajo por un modelo basado en capas que interconecta sus elementos (Mitra & Chattopadhyay, 2016).
Las ventajas que ofrecen las RNA es la posibilidad de crear máquinas de aprendizaje que siendo entrenadas, pueden ser capaces de tomar decisiones o predecir comportamientos incluso si la información que les es brindada es poca. Las máquinas de aprendizaje son una de las distintas ramas de la Inteligencia Artificial (IA), la cual busca realizar tareas inteligentes para tomar decisiones con el mejor resultado y resolver problemas de manera eficiente.

A pesar de los beneficios que tienen las RNA es importante considerar sus desventajas como comportamientos inexplicables que la misma red no sabe cómo ni dónde se hayan generado (Mijwel, 2019), así como los problemas inherentes al seleccionar la arquitectura con que la RNA procesará los datos, por ejemplo, el uso de punto fijo y punto flotante puede causar respuestas no deseadas si éstos no son ajustados con base a los parámetros del problema que se desee resolver. Esta última desventaja se tratará con mayor detalle en el subcapítulo “Representación en punto fijo”.

2.1.2 Modelo básico de una neurona artificial

Para comprender el funcionamiento de las RNA es esencial comenzar por el elemento más básico que las conforma, la neurona. La Figura 2.1 muestra el módelo de una neurona artificial básica, la cual fue propuesta por McCulloch y Pits en el artículo *A logical calculus of the ideas immanent in nervous activity.*

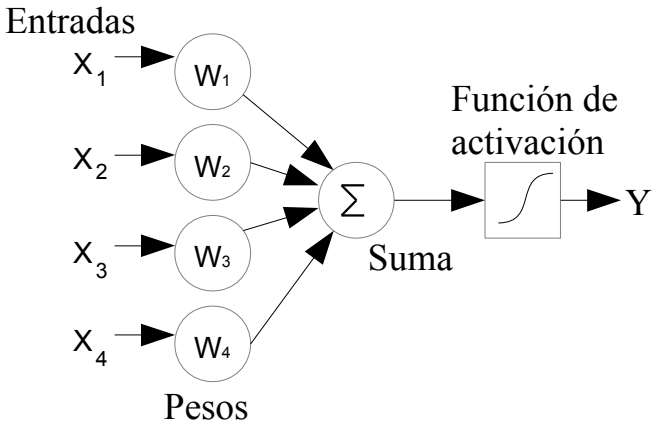


Figura 2.1: Modelo básico de una neurona artificial

A continuación se explica los elementos que conforman la neurona artificial:

- **Xi**: Valores de entrada de la neurona artificial.

- **Wi**: Pesos de la neurona artificial. Estos valores indican la importancia o intesidad de cada conexión en la neurona.

- **Suma** (S): Elemento encargado de sumar el producto de los pesos con sus respectivos valores de entrada (ver Ecuación 1).

$$S = \sum_{i=0}^{n} x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 + ... + x_n w_n$$

Ecuación 1: Suma

- **Función de activación**: esta función indica, dependiendo del valor obtenido en la unión sumadora, si la neurona se activa o no. Algunas de las funciones más utilizadas son las que se muestran a continuación.

1. **Función escalón**: se utiliza cuando la red neuronal trabaja con valores binarios. Su valor será 1 cuando el valor de salida de la unión sumadora sea mayor cero. Si la salida de la unión sumadora es menor o igual a cero, el valor de la función será 0.

2. **Función sigmoide**: esta función se utiliza cuando se necesita el uso de valores positivos entre 0 y 1. La función está descrita en la Ecuación 2.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Ecuación 2: Función sigmoide

2.1.3 Clasificación y tipos

Aunque las RNA parten del modelo básico de la neurona artificial, en la actualidad existe una gran variedad de ellas con características específicas, debido a esto tienen distintas aplicaciones en las áreas de ingeniería, medicina, economía, etc. Una manera de clasificar las diferentes RNA es por el método de aprendizaje tal como se muestra a continuación:

- **Aprendizaje supervisado**

Son consideradas aquellas máquinas de aprendizaje que por medio de un algoritmo procesan información que es brindada por el usuario. La máquina realiza las iteraciones necesarias hasta que el resultado sea el esperado. Este nombre es dado porque el usuario actúa como un profesor supervisando el aprendizaje. Las ventajas del aprendizaje supervisado son la libertad de escoger el número de

categorías, el extenso conocimiento que se tiene de los datos de entrada y por consecuencia es muy probable que se catalogue correctamente porque los datos que se reciben son conocidos.
Las desventajas de esta RNA es que no puede dar información mas allá de la que le haya sido brindada y es por ello que tampoco puede obtener información por sí misma.

Algunos ejemplos de algoritmos de aprendizaje supervisado son: regresión lineal, *Naïve* Bayes y búsqueda del vecino más cercano o *nearest neighbor search*.

- **Aprendizaje no supervisado**

Son las máquinas cuya función es describir un patrón o similitudes en información que previamente no ha sido clasificada. La característica de esta máquina es que no necesita intervención del usuario, el algoritmo por sí mismo se encarga de encontrar el comportamiento o característica de la información brindada.

A pesar de que el aprendizaje no supervisado es capaz de encontrar patrones en los datos por sí misma, se usan las máquinas de aprendizaje supervisado debido a que éstas tienen una mayor posibilidad de acertar en el resultado porque los datos que reciben ya se encuentran previamente catalogados a diferencia del aprendizaje no supervisado donde los datos son recibidos sin ninguna clase de categorización y por consecuencia la posibilidad de fallar es mayor.

Algunos algoritmos de aprendizaje no supervisado son: mapa auto-organizado, agrupamiento jerárquico y k-medias.

Una de sus diferentes aplicaciones es en el modelado estadístico.

- **Aprendizaje reforzado**

A diferencia de los dos clasificaciones anteriores, las máquinas de aprendizaje reforzado tienen un estado inicial donde son entrenadas para obtener una recompensa. Diferentes pruebas se hacen para cumplir el objetivo y seleccionar el proceso que tuvo como resultado la mayor recompensa. Así como un niño aprende a realizar ciertas acciones por medio de prueba y error dado un contexto o entorno, las máquinas de aprendizaje reforzado son diseñadas para imitar esta característica.

Algunos algoritmos de aprendizaje reforzado son: Estado de Acción Recompensa Estado de Acción o en inglés *State Action Reward State Action* (SARSA), aprendizaje Q y Gradiente de política de Profundidad Determinística o *Deep Deterministic Policy Gradient* (DDPG).

Algunas de las aplicaciones del aprendizaje reforzado es en los juegos de video o en la robótica.

En la Tabla 1 se muestran las características generales de algunas máquinas de aprendizaje, así como ejemplos de cada una de ellas.

	Aprendizaje supervisado	**Aprendizaje no supervisado**	**Aprendizaje reforzado**
Características	Los datos se encuentran previamente catalogados y a partir de ellos se selecciona una salida	Los datos de entrada no están catalogados y la máquina deberá aprender a seleccionar por sí misma	A la máquina se le asigna una tarea y la realizará en varias iteraciones para seleccionar la que dé una mayor recompensa
Algoritmos	*Naïve* Bayes, regresión lineal	Agrupamiento jerárquico, k-medias	Aprendizaje Q, SARSA

Tabla 1: Características y algoritmos de máquinas de aprendizaje

2.1.4 Redes Neuronales Artificiales implementadas en hardware

Las RNA tienen una considerable cantidad de tiempo en el interés de los investigadores, sin embargo, sus implementaciones han sido en mayor parte en *software*, por lo que el uso de *hardware* reconfigurable es un tema relativamente nuevo dado que el costo de las tarjetas de desarrollo era prohibitivo y su poder de computo no era suficiente. Avances en la tecnología de los FPGA en las últimas 3 décadas hicieron que la comunidad de investigadores reconsiderara el potencial que tendría la implementación de una RNA en *hardware*. Debido a su capacidad de realizar operaciones en paralelo, adaptación ad hoc del tipo de número y su representación binario, así como la reducción en costos, los FPGA serían una opción viable para desarrollar RNA.

Algunas de sus implementaciones abarcan diferentes campos, por ejemplo en el artículo *FPGA Implementation of a Neural Network for Real-Time Hand Tracking System,* se desarrolló una RNA multicapa en VHDL (del Inglés *Very High Speed Integrated Circuit Hardware Description Language*) para detectar una mano utilizando un sistema de reconocimiento. En el artículo mencionado anteriormente se concluye que "una solución en *hardware* usando una FPGA tiene la gran ventaja de compartir la idea fundamental de una RNA, procesar información en paralelo" (Krips et al., 2002).

El artículo *Implementation of an RBF Neural Network on Embedded Systems: Real-Time Face Tracking and Identity Verification*, se realizó una RNA de base radial en diferentes sistemas embebidos. El propósito de la RNA es el reconocimiento, seguimiento y verificación de rostros. El éxito de reconocimiento y verificación obtenido es 92% (FPGA), 85% (*Zero Instruction Set Computer* o ZISC) y 98.2% (*Digital Signal Processing* o DSP) (Paindavoine & Yang 2003).

Se han realizado diferentes propuestas de una RNA multicapa, por ejemplo en el artículo *Feedforward Neural Network Implementation in FPGA Using Layer Multiplexing for Effective Resource Utilization*, se propone el uso de LUTs (del inglés *Look Up Table*) para una sola neurona, además de diferentes configuraciones en las capas de multiplexaje. Dicha propuesta tuvo como resultado una reducción de recursos, así como un aumento en velocidad (Muthuramalingam et al. 2007).

Las RNA también se han utilizado para modelar sistemas caóticos, como se muestra en el artículo *Hardware design and implementation of a novel ANN-based chaotic generator in FPGA* . El trabajo modela el sistema caótico de Pehlivan-Uyaroglu en una FPGA usando VHDL. Los resultados obtenidos en este trabajo concluyen que "la precisión obtenida en la FPGA con el algoritmo RK5 es satisfactoria" (Koyuncu et al. 2016).

Las RNA también se han implementado para la predicción de producción de energía de un sistema fotovoltaico. El artículo *Hardware implementation of an artificial neural network model to predict the energy production of photovoltaic system* muestra una RNA Topología de Perceptron Multicapa, concluyendo que dicha RNA "es una buena alternativa para predecir el poder generado por sistemas fotovoltaicos", además anexa que "la implementación en *hardware* produce un bajo costo y resultados precisos" (Baptista, Abreu, & Travieso-Gónzalez 2016).

Otra aplicación es en el área de clasificación de objetos como se muestra en la tesis de Dr. Víctor Lomas (Lomas Barrie 2016) donde se implementa una RNA de clasificación de aprendizaje supervisado de nombre Fuzzy ARTMAP, la cual es entrenada por medio de imágenes capturadas por una cámara conectada a la tarjeta de desarrollo que extrae el descriptor de un objeto y se le clasifica con una Fuzzy ARTMAP. La implementación fue exitosa debido a que se obtuvo una esperada mejora en la rapidez de respuesta de la estructura paralela de la RNA.

Respecto a la RNA Fuzzy ARTMAP, en el artículo *Parallelization of Fuzzy ARTMAP Architecture on FPGA* se muestra una FAM implementada en VHDL con el propósito de clasificar imágenes del satélite ALSAT-2A. Aunque los resultados fueron exitosos, el artículo menciona que "tanto FPGAs y GPUs son capaces de cómputo de alto rendimiento, sin embargo su relativo rendimiento en varias aplicaciones todavía no es bien comprendido" (Yahiaoui et al. 2017).

A pesar de las ventajas señaladas en los casos anteriores, es necesario mencionar que existen retos que aún deben de superarse. Investigaciones sobre la eficiencia de las RNA en hardware expone que "la realización de RNA con una gran cantidad de neuronas en FPGA es una tarea difícil porque los algoritmos de las RNA requieren operaciones de multiplicación y es costosa su implementación en FPGA" (Mitra & Chattopadhyay 2016), es por eso que es necesario seleccionar el FPGA con la cantidad de módulos multiplicadores que satisfagan la densidad de neuronas y soporte el ancho del tipo numérico.

2.2 Representación en punto fijo

Al implementar una RNA en *hardware* es necesario seleccionar la representación numérica digital de las variables y constantes, de esta manera es posible cuantificar el comportamiento del caso de estudio. Para cuantificar dicho comportamiento se utiliza el sistema binario para que los datos sean almacenados y procesados en máquinas digitales. Operar sobre la parte entera no representa un problema, sin embargo, la parte fraccionaria debe representarse usando punto flotante o punto fijo.

El punto flotante es un método para representar números reales en máquinas digitales. Su diferencia respecto a punto fijo es que esta representación se conforma de una mantisa, un exponente y un bit de

signo según el formato IEEE754. La mantisa contiene el número al cual se moverá el punto decimal, el exponente indica cuantas posiciones se moverá y el bit más significativo contiene el signo. Una de las ventajas del uso de punto flotante es la flexibilidad que tiene para representar valores grandes o pequeños, sin embargo, una arquitectura de punto flotante requiere una mayor cantidad de recursos, por ejemplo, una Unidad de Punto Flotante (UPF) para que realice las operaciones aritméticas. En cambio, el uso del punto fijo consiste en colocar un punto que separa la parte entera y fraccionaria, donde dicho punto no cambiará de posición.

Por ejemplo

$$\mathbf{(4.5625)_{10}=(100.10010000)_2}$$

Representación con 3 bits parte entera y 8 bits parte decimal

Para la Tabla 2 el número decimal 4.5625 se representó en sistema binario con 3 bits para la parte entera y 8 bits para la parte fraccionaria. En este caso la representación binaria tiene el mismo valor si es convertido a su base original.

1	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**		**Total**
$1\bullet2^{2}$	$0\bullet2^{1}$	$0\bullet2^{0}$	$1\bullet2^{-1}$	$0\bullet2^{-2}$	$0\bullet2^{-3}$	$1\bullet2^{-4}$	$0\bullet2^{-5}$	$0\bullet2^{-6}$	$0\bullet2^{-7}$	$0\bullet2^{-8}$		$(4.5625)_{10}$

Tabla 2: Ejemplo de conversión de punto fijo a decimal

Sin embargo, no siempre es posible obtener el valor decimal esperado cuando se hace una conversión de punto fijo a decimal. En el siguiente ejemplo se muestra la conversión de decimal a punto fijo y ese mismo valor en punto fijo convertido de vuelta a decimal.

$$\mathbf{(3.141)_{10}=(11.00100100)_2}$$

Representación con 2 bits parte entera y 8 bits parte decimal

1	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**Total**
$1\bullet2^{1}$	$1\bullet2^{0}$	$0\bullet2^{-1}$	$0\bullet2^{-2}$	$1\bullet2^{-3}$	$0\bullet2^{-4}$	$0\bullet2^{-5}$	$1\bullet2^{-6}$	$0\bullet2^{-7}$	$0\bullet2^{-8}$	$(3.140625)_{10}$

Tabla 3: Ejemplo de conversión de punto fijo a decimal

Como se muestra en el ejejmplo anterior, la cantidad finita de bits limita el rango de valores que se pueden representar en la parte fraccionaria. Debido a estos casos es necesario llevar a cabo una estrategia de optimización de recursos, es decir, no se puede usar de manera ilimitada el ancho de un vector binario para representar un número, ni sacrificar precisión para ahorrarse espacio.

Es conveniente saber el rango de magnitudes con las que se trabajará para poder ajustar el número de

bits para la parte entera y decimal, reduciendo así el margen de error.

2.2.1 Ventajas de las arquitecturas de punto fijo

Un motivo fundamental para el desarrollo de la RNA en FPGA es replicar la mayor cantidad de arquitecturas Fuzzy ARTMAP, por lo cual se necesita el menor uso de recursos posible. Se seleccionó realizar una arquitectura de punto fijo, esto debido a que el uso punto flotante requiere de circuitos lógicos más complejos, "procesadores de punto fijo tienen circuitos lógicos simples comparados con procesadores de punto flotante" (Benefits of Using Fixed-Point Hardware 2017).

Otros beneficios de usar punto fijo sobre punto flotante son: reducción de recursos lógicos, menor consumo de energía y costo reducido (Finnerty & Ratigner 2017).

Implementar una arquitectura en punto fijo permite el uso de aritmética de enteros incluida en las bibliotecas *standar* de VHDL. También se reducen los recursos de la FPGA debido a que no es necesario implementar una UPF.

Además de los requisitos mencionados en los párrafos anteriores, también se requiere que la Fuzzy ARTMAP procese datos con rapidez. Es por ello que se ha seleccionado una arquitectura de punto fijo pues "la menor cantidad de instrucciones de aritmética de punto fijo inherentemente implica una ejecución más rápida" (Beheshti 2015).

2.2.2 Operaciones aritméticas en punto fijo

Las operaciones aritméticas en punto fijo requieren que ambos operandos tengan alineados su punto fijo. Si se realiza una suma, el resultado de la operación debe contener el número de bits del operando más uno (Ecuación 3).

N:Número de bits del operando

Nbpp=Número de bits por producto

$$Nbpp = N + 1$$

Ecuación 3: Cantidad de bits del producto de una suma en punto fijo

De igual manera que en la suma, la multiplicación requiere que ambos operandos se encuentren alineados. El producto de la operación genera una palabra del doble de bits del operando (Ecuación 4).

$$Nbpp = 2N$$

Ecuación 4: Cantidad de bits del producto de una multiplicación en punto fijo

2.3 Teoría de la Resonancia Adaptativa

El proceso con el cual los seres humanos aprenden nuevos conocimientos es un tema que ha sido de gran interés en el área de las ciencias cognitivas, su razón es que el cerebro humano tiene la facilidad de aprender de su entorno con relativa facilidad sin la necesidad de un tutor que lo guíe. Otra característica importante es la capacidad de almacenar conocimiento previamente adquirido para que éste sea usado para comparar o categorizar la información que se considere útil.

Se ha buscado replicar estas características del ser humano por medio de algoritmos que simulen dicho comportamiento. El problema que se tiene es que los algoritmos implementados pueden categorizar datos previamente adquiridos o nuevos datos pero no ambos. A esto se referie como el dilema de plasticidad y estabilidad. Con el objetivo de resolver el problema anterior, la Teoría de la Resonancia Adaptativa (ART por sus siglas en inglés) "ha sido desarrollada para evadir el dilema de estabilidad-plasticidad en redes competitivas de aprendizaje" (Weitzenfeld, Arbib, & Alexander 2002).

La Figura 2.2 muestra la estructura de una red ART, la cual tiene la característica de contar con un vector de entrada que contiene sólo valores binarios. Pasando a los elementos de la red ART, los círculos representan las neuronas de la red y las flechas son la comunicación de las capas y el subsistema orientador representado con la letra rho (***ρ***). Este subsistema contiene el factor de vigilancia, el cual "especifica el mínimo valor de la entrada que debe permanecer en el patrón correspondiente para que la resonancia ocurra" (Stephen Grossberg & Carpenter, 1998), en otras palabras, es el mínimo valor que acepta el sistema para que la categoría seleccionada se considere como ganadora.

La datos son recibidos por las neuronas de la capa ***F1*** y los pesos son enviados a la capa ***F2*** (que contiene las posibles categorías ganadoras) para que sean procesados. Para iniciar la clasificación, se parte de la hipótesis de que la primera neurona de la capa ***F2*** es la ganadora y ***F2*** envía de regreso la información a la capa ***F1.*** Si la información recibida en ***F1*** es igual o mayor al factor de vigilancia, la neurona propuesta se considera como la ganadora y se modifican los pesos de la red.

En caso de no cumplirse el factor de vigilancia, se analiza si existe alguna categoría que cumpla con las características de la hipótesis y de no existir dicha categoría, se crea una nueva. Si existe esa categoría, se desactiva para que no sea considerada en la siguiente hipótesis de la categoría ganadora, además los pesos de la red son modificados.

Los pasos anteriores se repiten hasta encontrar una categoría ganadora.

Por la retroalimentación de la red ART y capacidad de autorregularse, así como su capacidad de agregar nuevas categorías, esta red tiene plasticidad para contener información previa y elasticidad para añadir nuevo conocimiento.

Algunas de sus aplicaciones son control de robots a distancia, diagnósticos médicos y reconocimiento de objetos geométricos.

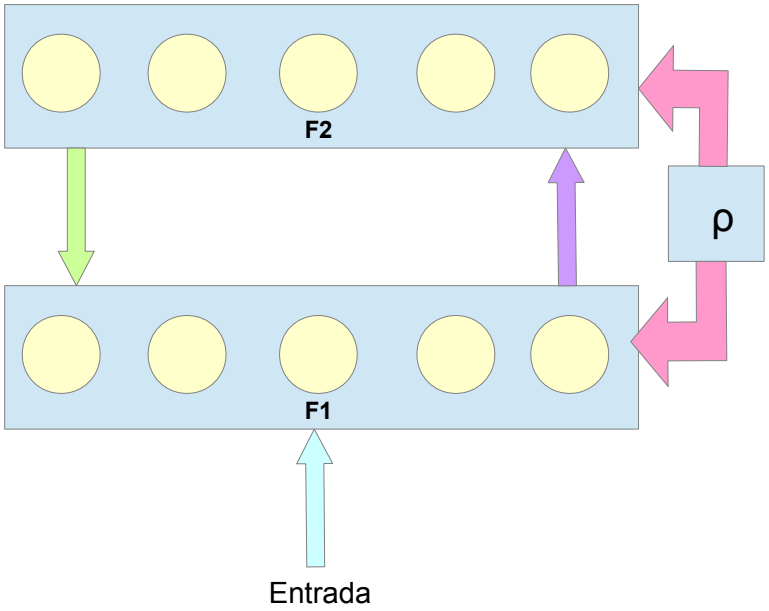


Figura 2.2: Estructura de una red ART

A partir de la red ART, se realizaron modificaciones a su diseño creando variantes de la misma. La Tabla 4 contiene información de las características de las diferentes redes ART.

RNA	Descripción	Supervisada
ART 1	Primera arquitectura ART creada. Agrupa valores binarios	No
ART 2	Rediseño de la arquitectrura ART 1.Su diferencia es que soporta valores continuos como entrada	No
Fuzzy ART	Añade lógica difusa al diseño de la ART	No
ARTMAP	Toma elementos del diseño de la ART 1 y ART 2 para unirlos	Sí
Fuzzy ARTMAP	Elementos de ART 1 y ART 2 son combinados con lógica difusa	Sí

Tabla 4: Tipos de redes ART

2.4 Fuzzy ARTMAP

La Fuzzy ARTMAP está compuesta por dos arquitecturas Fuzzy ART, ART_a es la encargada de recibir el vector de información de un patrón y la ART_b contiene los resultados deseados. Ambos módulos son conectados por una ART más que recibe el nombre de mapa de campo y cuya función es encontrar la relación de la información recibida en ART_a y los valores deseados guardados en ART_b. Dependiendo de la relación que se encuentre, el factor de vigilancia de ART_a podría modificarse.

La representación geométrica de la Fuzzy ARTMAP consiste en dos vectores bidimensionales u_j y v_j que forman una región R_j dentro de un cuadrado unitario como muestra en la Figura 2.3.

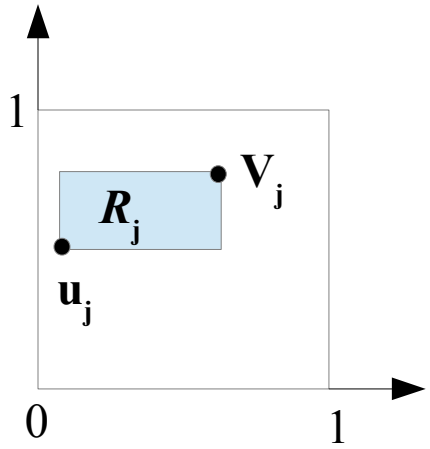


Figura 2.3: Región Rj

El vector de entrada ***a*** es concatenado con su complemento, formando así el vector I (ver Ecuación 5).

$$I = (a, a^2)$$

Ecuación 5: Vector I

Durante cada paso de aprendizaje, la región R_j se expande hasta $R_j \oplus a$ (Figura 2.4) es decir, "el mínimo rectángulo que contiene a R_j y a"(S Grossberg et al., 1992) . Los vértices de esta nueva región Rj están dados por $a \wedge u_j$ y $a \vee v_j$ donde el operador Fuzzy AND es $\wedge$ y el operador Fuzzy OR es $\vee$, tal como se muestra en el artículo *Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps.*

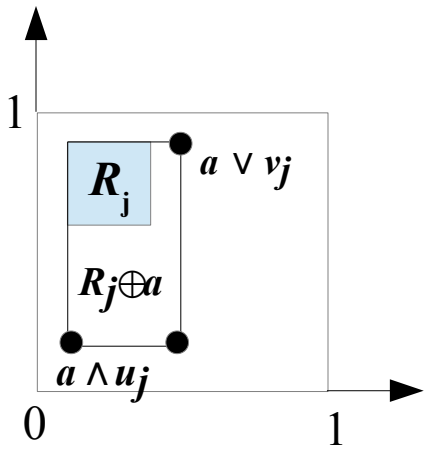


Figura 2.4: Región Rj⊕a

Las dimensiones de la región $R_j \oplus a$ está dada por la Ecuación 6:

$$|R_j \oplus a| \equiv |(a \vee v_j) - (a \wedge u_j)|$$

Ecuación 6: Región Rj⊕a

Para el correcto funcionamiento de la Fuzzy ARTMAP se debe entrenarla para después ser usada como clasificador.

- **Entrenamiento**

El proceso para entrenar la Fuzzy ARTMAP requiere de dos vectores:

I: Vector que contiene la información a clasificar.

$\boldsymbol{\omega}_j$: Vector que contiene los pesos.

Ambos vectores son introducidos en la Ecuación 7

$$T_j(I) = \frac{1}{\alpha + |\omega_j|}$$

Ecuación 7: Función de selección

Los elementos que conforman a T_j son:

- **Módulo de ω_j** : Se obtiene de la suma del valor absoluto de cada uno de los elementos del vector.

- **α**: Llamado parámetro de selección, es el encargado de escoger que el punto a mapear se encuentre en la menor región R_j.

- **Operador Fuzzy AND** ($\wedge$): Selecciona el valor más pequeño entre dos datos.

Para finalizar el proceso, la categoría ganadora es comparada con el factor de vigilancia para saber si se encuentra en resonancia y cumple con los parámetros esperados. La ecuación que describe esto es:

$$\frac{|I \bigwedge \omega_j|}{|I|} \geq \rho$$

Ecuación 8: Prueba de resonancia

- **Clasificación**

Para que la Fuzzy ARTMAP sea capaz de clasificar, es necesario que se haya entrenado previamente. El módulo ART_b no es utilizado y la clasificación se deja a la ART_a , "la cual entregará un vector que contiene sólo ceros excepto un uno.El índice de esta componente es el número de la categoría en la cual ha sido clasificado el vector a" (Busque & Parizeau, 1997).

3 Caso de Estudio: BOF & FAM

En el Laboratorio de Electrónica y Automatización para la Industria 4.0 del Instituto de Investigaciones en Matemáticas Avanzadas y Sistemas (IIMAS) se ha estado trabajando con una línea de investigación para reconocer objetos de ensamble dentro de una celda de manufactura inteligente. Una de estas líneas de investigación es utilizar el clasificador FAM para reconocer dichos objetos. Para ello es necesario extraer un descriptor único que contenga tales características que puedan ser utilizadas por una FAM. La BOF (del inglés *Boundary Object Points*) es un vector descriptivo que representa el contorno de un objeto. Sus características son la invariancia a la traslación, escalamiento y rotación. Dicho vector contiene 202 datos (donde cada uno de ellos es un valor entre 0 y 1) que serán procesados por la FAM. Si bien en este trabajo la BOF de un objeto es la señal de entrada de la FAM, los detalles de su obtención están fuera de su alcance de este trabajo y se muestra el procedimiento sólo con fines informativos.

El procedimiento para obtener la BOF es el siguiente:

1. Se extrae la región de interés del objeto (ROI) en escala de grises.

2. En la ROI se genera una imagen binaria, por medio de un umbral, donde el objeto se llena con un uno lógico, mientras que al fondo se le asigna un cero lógico.

3. Se convierte el resultado del paso anterior en una matriz con ceros y unos lógicos. A partir de ésta se genera una Matriz de Pesos por medio de la Ecuación 9.

$$I_{MP}(i,j) = \sum_{k=-1}^{1} \sum_{l=-1}^{1} I_b(i+l, j+k)$$

Ecuación 9: Matriz de pesos

Donde

$$0 \leq I_{MP}(i,j) \leq 9; \forall i,j \in \mathbb{N}$$

$$-1 \leq k \leq 1; -1 \leq l \leq 1; K[i,j] = 1 \forall i,j = 1,2,3.$$

4. Los puntos frontera del objeto se obtienen para poder calcular la distancia euclidiana entre cada punto frontera y el centroide.

$$D_n = \sqrt{(X_c - x)^2 + (Y_c - y)^2}$$

5. Se obtiene el resultado con mayor distancia entre el centroide y un punto frontera para normalizar cada distancia que haya sido obtenida. Esto da el vector descriptivo BOF.

4 Hardware Reconfigurable

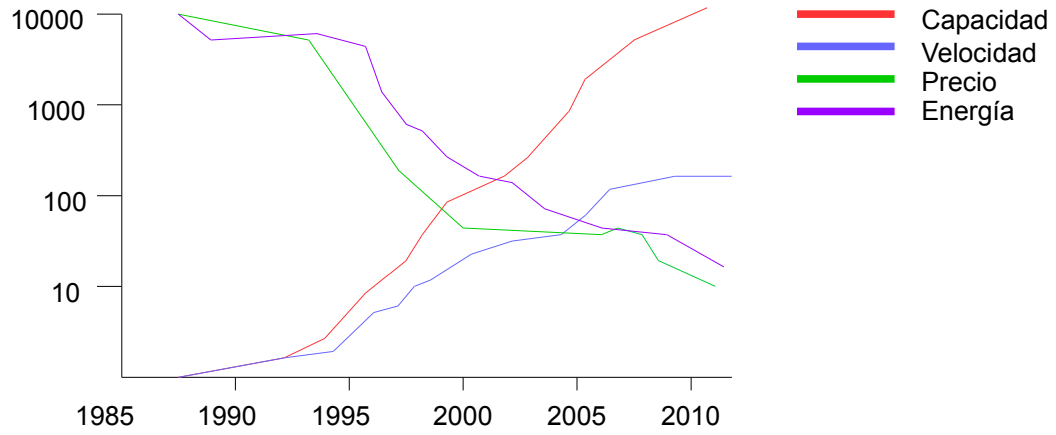
La creación de los circuitos integrados (CI) ASSP (del inglés *aplication specific standar product*) traería consigo las series 7400 y 4000, los cuales causarían una revolución en diferentes áreas de la electrónica digital. Su bajo costo y tamaño reducido los convertirían en una opción viable para diferentes sectores, siendo uno de ellos el desarrollo de sistemas de cómputo (Wang, Chang, & Cheng 2009). A pesar del éxito de éstos CI (Lojek 2010), los desarrolladores de *hardware* se dieron cuenta que necesitaban disposotivos que tuvieran mayor flexibilidad, es decir, que puedieran ser reconfigurados después de ser manufacturados con el propósito de facilitar el proceso de pruebas. Los dispositivos que buscarían satisfacer esta necesidad serían las *Programmable Array Logic* (PAL) y posteriormente las *Generic Array Logic* (GAL).

La necesidad de realizar dispositivos con circuitos más complejos, requeriría aumentar la cantidad de elementos lógicos. Debido a que las PAL y GAL no eran suficientes para satisfacer la demanda de dispositivos reconfigurables más robustos, se introdujo los *Complex Programmable Logic Device* (CPLD) los cuales pueden interconectar PALs entre sí y por consecuencia aumentar la cantidad de recursos lógicos disponibles.

Otra corriente de desarrollo de dispositivos reconfigurables tendría como resultado la creación del *Field Programmable Gate Array* (FPGA). Introducido al mercado en 1985 (Alfke 2007) por la compañia Xilinx, los FPGAs han beneficiado a diferentes sectores de la industria por su versatilidad y naturaleza reconfigurable. Algunas de sus aplicaciones son en el procesamiento de imagenes, comunicaciones, equipo médico, entre otros.

A pesar de las atractivas ventajas de las FPGA, al inicio de su producción no eran económicamente viables por su elevado precio (Trimberger 2015). Tuvieron que pasar varias decadas para que su producción redujera los precios de manufactura y así ser más accesibles al público general.

Como se muestra en la Gráfica 1, se tuvieron que hacer avances tecnológicos para que las características técnicas de los FPGAs tuvieran un incremento significativo para que fueran considerados como una alternativa. El aumento en poder de computo y reducción de costo de estos dispositivos, los ha vuelto una solución a problemas que requieran una rápida respuesta por la personalización específica en *hardware* que se puede generar en su descripción y su paralelización de tareas.



Gráfica 1: Comparativa de las características técnicas de los FPGA a través de las decadas.El precio y energía están escalados por 10000.

El aumento de las capacidades de las FPGAs, su considerable reducción en costos y tamaño, tendrían como consecuencia la diversificación de su uso y una mayor apertura para que sean consideradas como una alternativa.

4.1 Estructura de una FPGA

La estructura de una FPGA es diferente entre fabricantes e incluso entre familias del mismo desarrollador. Las FPGAs del fabricante Xilinx se componen de elementos CLB (del inglés *Configurable Logic Blocks*), Block RAM (del inglés *Block Random Access Memory*), *Input/Output Block* y DSP Blocks (del inglés *Digital Signal Processing Blocks*). Por otro parte, el equivalente de CLB para el fabricante Altera son los LAB (del inglés *Logic Array Blocks*) que se conforman de LE (del inglés *Logic Elements*) para la serie Cyclone IV o ALM (del inglés *Adaptative Logic Modules*) para la serie Stratix IV.

La tarea de cada uno de estos elementos básicos que conforman las FPGA son las siguientes:

- **CLB**: Elementos básicos en la arquitectura de una FPGA (desarrollado por Xilinx). Pueden ser configurados para realizar funciones lógicas. Estos bloques tienen dentro *slices* que a su vez se componen de *flip flops* y LUTs.

- **LUT**: La *Look Up Table* o LUT son bloques que contienen celdas para guardar información binaria. Son usadas principalmente para realizar operaciones aritméticas o funciones lógicas, podrían utilizar grandes cantidades de *hardware* y considerable espacio. Usar una LUT reduce el espacio usado por el *hardware* pues los resultados de las operaciones pueden ser guardados en sus celdas.

- **Flip Flop**: Los *Flip Flop* son dispositivos lógicos que permiten el almacenamiento de información binaria. Por cada bit de información deberá haber un *Flip Flop*.

- **Block RAM**: Memoria dentro de la FPGA, su capacidad depende del modelo. Su uso es para guardar información del programa a ejecutar y los datos que se procesarán en él.

- **Input/Output Block**: Bloque encargado de comunicar el FPGA con el mundo exterior tanto para recibir y enviar información.

- **DSP Blocks**: Diseñado para procesar digitalmente señales como lo son filtros, convoluciones, unidades MAC (del inglés *Multiply and accumulate*).

- **Slices**: Conjunto (utilizado por el fabricante Xilinx) que almacena *flip flops* y LUTs que comparten conexiones entre sí para reducir el tiempo de propagación de la información. Las *slices* agrupados conforman un **CLB**.

Los elementos mencionados anteriormente deben de comunicarse entre sí, de esa manera pueden realizar las tareas descritas por el usuario, por lo tanto la FPGA debe de contar con buses que puedan conectar cada bloque. La Figura 4.1 muestra la conexión que debe realizar los CLB con recursos de interconexión programable, que son configurados por por el usuario cuando realiza el diseño de su circuito lógico. Cada *slice* dentro del CLB no se encuentra comunicado, además de encontrarse organizados como columnas.

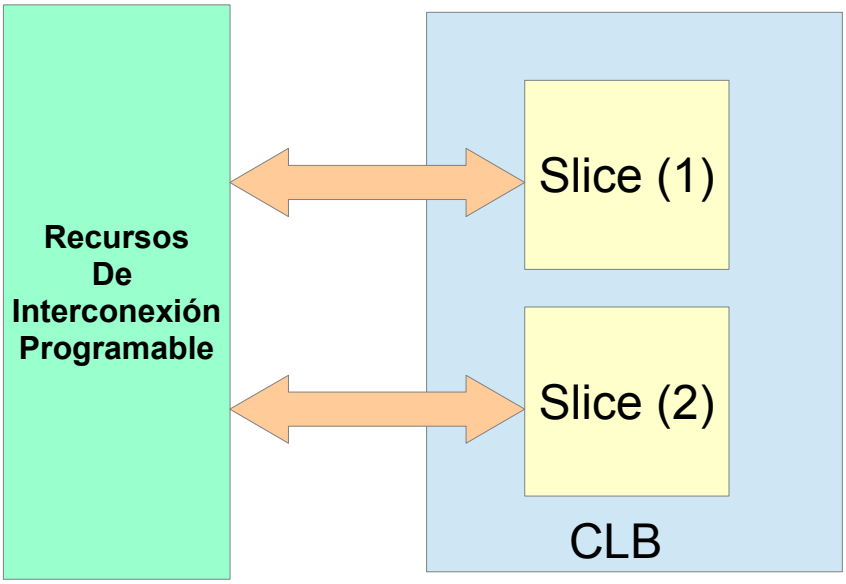


Figura 4.1: Conexión de CLB con Recursos de Interconexión Programable

Los recursos de interconexión programable se conforman de interruptores matriciales que permiten la conexión a una matriz de direccionamiento general. Se puede configurar para permitir o no la comunicación entre líneas que se encuentran a una distancia larga o corta, de tal manera que el retardo al envíar o recibir información entre bloques sea la menor posible.

Los bloques descritos anteriormente se pueden observar en conjunto en la Figura 4.2. En dicha figura se encuentran otros recursos, como lo son los bloques de ***entradas y salidas***, los cuales como lo indica su nombre, son *buffers* de salida o entrada que pueden ser configurados para comunicar elementos externos de la FPGA con elementos internos. Estos *buffers* pueden ser configurados en tres tipos de estados: activado, desactivado o alta impedancia. El *buffer* de salida puede ser configurado para la cantidad de corriente que se le dará a una carga o determinar la carga máxima que se puede añadir en una velocidad determinada. También es posible determinar qué tan rápido serán los tiempos de bajada o subida de la señal de la salida. Para el caso de las entradas, si la tarjeta donde se encuentra la FPGA cuenta con botones, se debe revisar con el fabricante si éstos cuentan con lógica negada o lógica positiva.

Cierta cantidad de retraso se puede añadir en la comunicación dentro de la FPGA, esto debido a la distancia entre los buses internos de los bloques en comunicación, es por ello que señales de gran importancia como son los pulsos de reloj, necesitan llegar en sincronía a todos los elementos que lo requieran. Para resolver los problemas de retraso se implementan *buffers* dedicados para pulsos reloj que se encuentran conectados a los distintos elementos que conforman la FPGA.

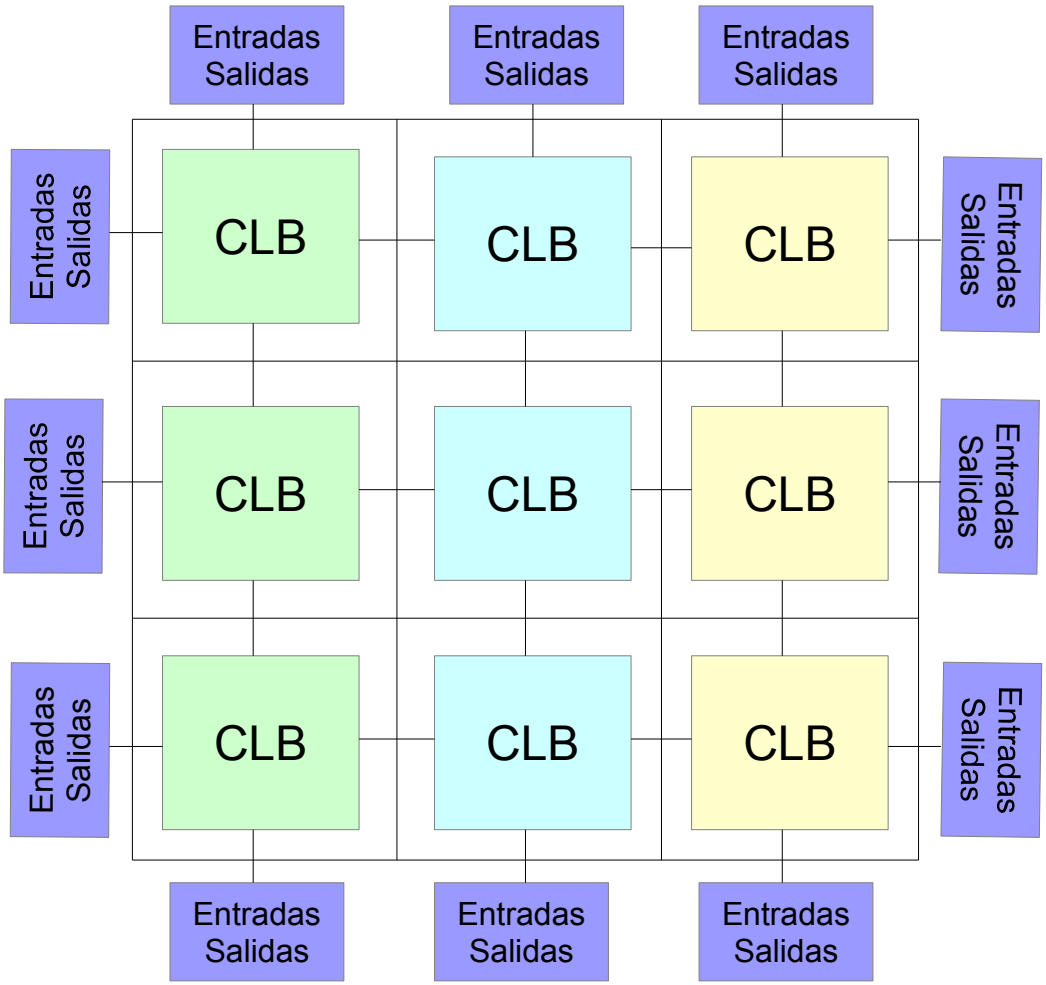


Figura 4.2: Estructura básica de una FPGA

La estructura por medio de bloques y la intercomunicación de los mismos permite el diseño de Propiedades Intelectuales o IP por sus siglas en inglés. El circuito digital diseñado por el usuario puede generarse como un bloque o IP, esto con el fin de ser reutilizados en diseños posteriores del usuario sin la necesidad de que sean desarrollados de nuevo.

4.2 System on a Chip

La necesidad de producir dispositivos electrónicos con menores dimensiones cada vez es mayor, por lo que es necesario ofrecer soluciones que reduzcan el tamaño de los componentes electrónicos.

Una solución a este problema son los SoC (del inglés *System on a Chip*), estos dispositivos ofrecen los componentes básicos de una computadora, junto con otros elementos, en un sólo chip. A diferencia de una computadora personal donde cada uno de sus elementos se encuentran separados, el propósito de un SoC es unir todos estos dispositivos, en un sólo chip con el fin de reducir costos de producción, reducir el gasto de energía, incrementar la velocidad y reducir el tamaño. Un ejemplo de SoC es el Zynq-7000 que contiene la tarjeta de desarrollo Zybo (Figura 4.3) creada por Xilinx. La descripción de los elementos que conforman la tarjeta Zybo se encuentran en la Tabla 5.

Algunos elementos que comúnmente se encuentran en un SoC son:

- **RAM**: La Memoria de Acceso Aleatorio es la encargada de guardar los programas que se ejecutarán y los datos que se procesarán.

- **CPU**: La Unidad de Proceso Central se encarga de llamar, decodificar y ejecutar los programas de cómputo.

- Los periféricos de un SoC dependerán entre cada modelo. Por ejemplo, para recibir o enviar video podría utilizar la interfaz HDMI mientras que para información de audio podrían tener entradas de 3.5 mm. Para tener comunicación con la red, algunas tarjetas tienen un conector modular 8P8C.

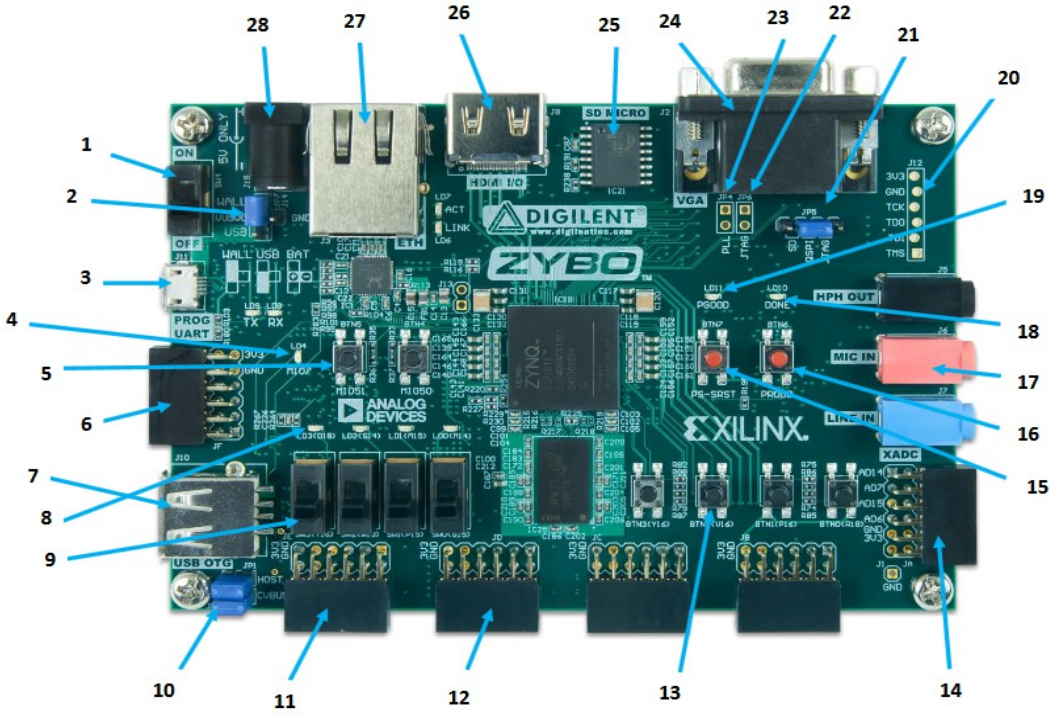


Figura 4.3: Tarjeta de desarrollo Zybo

Número	Descripción del componente	Número	Descripción del componente
1	Interruptor de encendido	15	*Pushbutton* de reinicio del procesador
2	*Jumper* de selección de alimentación y *header* de batería	16	*Pushbutton* de reinicio de lógica de configuración
3	Puerto UART/JTAG compartido	17	Conexiones de codec de audio
4	LED MIO	18	LED de finalización de configuración lógica
5	*Pushbuttons* MIO (2)	19	LED de confirmación de alimentación de la tarjeta
6	Pmod MIO	20	Puerto JTAG para cable opcional externo
7	Conectores OTG USB	21	*Jumper* de modo programación
8	LEDs de lógica (4)	22	*Jumper* habilitador de modo JTAG independiente
9	Interruptores deslizables lógicos (4)	23	*Jumper* PLL Bypass
10	*Jumpers* de selección USB OTG *Host/Device*	24	Conector VGA
11	*Standard* Pmod	25	Entrada microSD (al reverso)
12	Pmods de alta velocidad (3)	26	Conector HDMI S*ink/Source*
13	*Pushbuttons* lógicos (4)	27	Conector Ethernet RJ45
14	Pmod XADC	28	Entrada de alimentación de la tarjeta

Tabla 5: Componentes de la tarjeta Zybo

4.3 Zybo Zynq-7000

La familia Zynq 7000 de Xilinx tiene dispositivos que son catalogados con factores como su precio y subcatalogados por la cantidad de procesadores ARM Cortex-A9 que contienen. Esta sección describirá las características de la tarjeta Zybo Zynq-7000 puesto que se ha seleccionado para desarrollar este trabajo por ser la tarjeta de menor costo y contar con comunicación con dos procesadores ARM Cortex-A9.

La tarjeta de desarrollo Zybo contiene un SoC que pertenece a la familia Xilinx Zynq-7000. Cuenta con dos procesadores ARM Cortex-A9 y lógica FPGA de Artix-7 de la serie Xilinx 7. El número de pieza de la tarjeta Zybo es XC7Z010.

Los elementos de los que se compone la familia Zynq-7000 son un PS (del inglés *Processing System*) y PL (del inglés *Programmable Logic*). La Figura 4.4 muestra de manera general los elementos que conforman la arquitectura Zynq-7000.

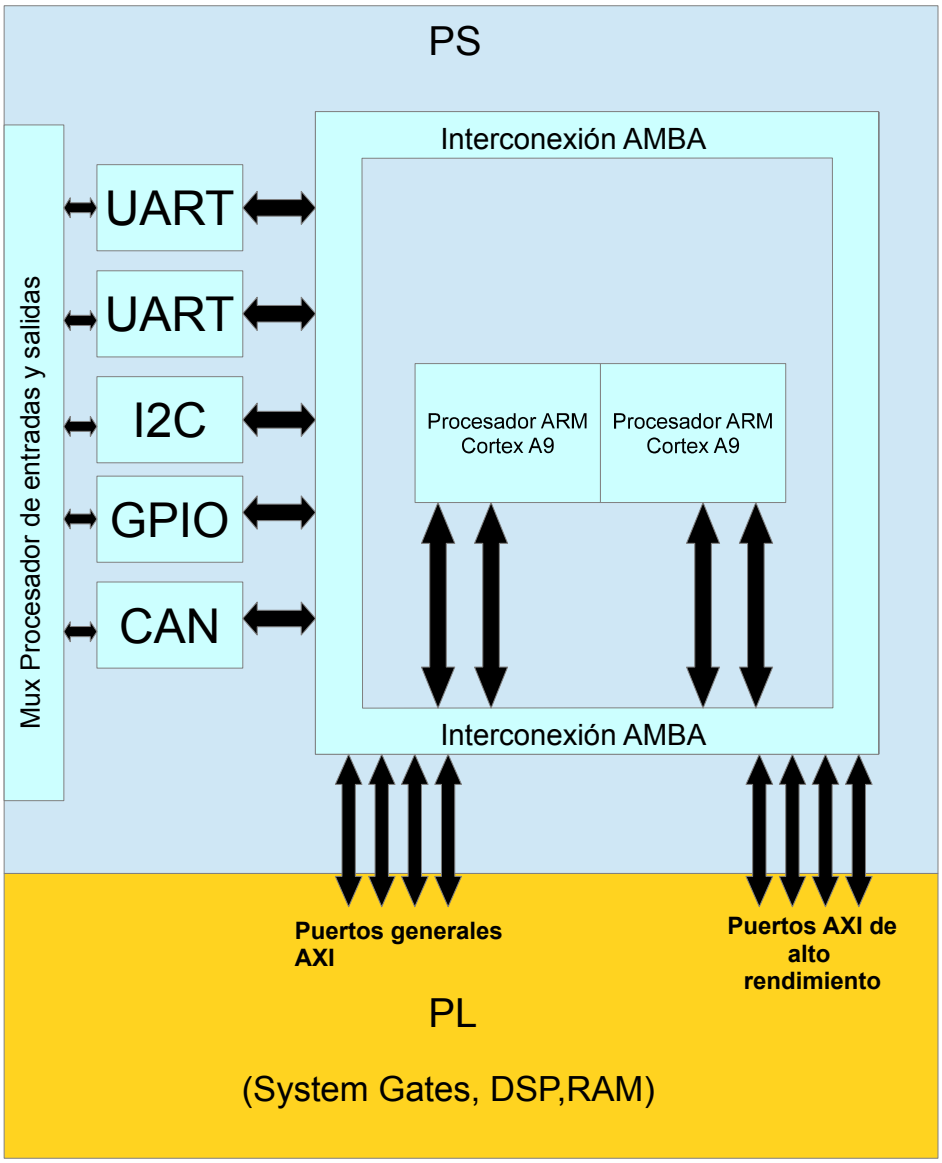


Figura 4.4: Arquitectura general de un Zynq APSoC

4.3.1 Procesador ARM

Los dispositivos de la familia Zynq-7000 de Xilinx cuentan con dos procesadores ARM Cortex-A9 en comunicación con lógica programable. Esta sección es una breve introducción a la historia del procesador ARM, así como las ventajas de su arquitectura.

Advanced RISC Machine o ARM por sus siglas en inglés, es una arquitectura que pertenece a la familia de procesadores RISC. *Reduced Instruction Set Computing o* RISC por sus siglas en inglés, son familias de procesadores con la característica de usar un conjunto de simples instrucciones para que sean ejecutadas en menor tiempo a altas velocidades de ciclos de reloj, mejorando así su eficiencia en consumo de energía. Gracias a su tamaño reducido, su implementación es popular en sistemas embebidos.

La arquitectura ARM fue introducida por la compañía Británica Acorn Computers en los años 80 y desde entonces se ha implementado en diversos de dispositivos debido a su velocidad, bajo consumo de energía y reducido espacio. Los procesadores ARM (ver Figura 4.5) son utilizados popularmente en consolas de videojuegos, reproductores de música, tabletas y celulares, entre otros, sin embargo, empresas como Xilinx crearon tarjetas de desarrollo que comunicaban la PL de una FPGA con un procesador ARM.

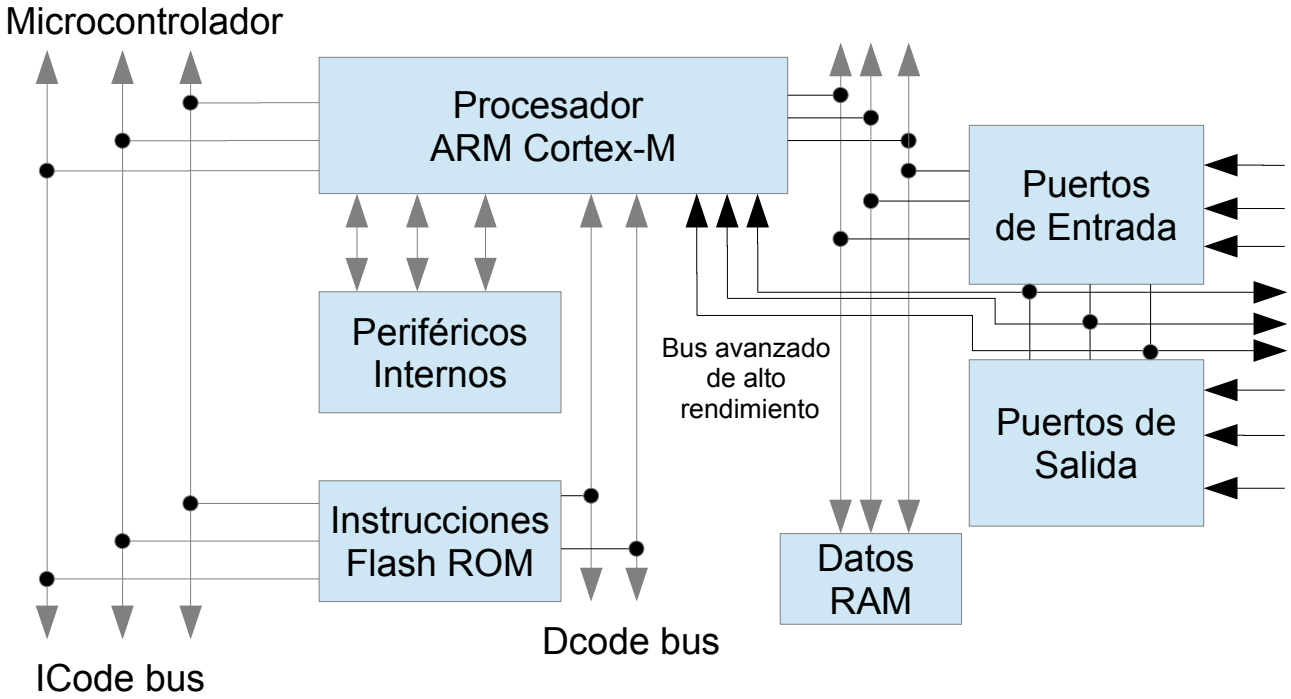


Figura 4.5: Arquitectura Harvard de un Microcontrolador basado en ARM Cortex M

4.3.2 Processing System

El PS es el elemento que ejecuta código máquina, el cual se puede programar por medio del SDK (del inglés *Software Development Kit*) de Xilinx utilizando C o C++, esto gracias a su procesador dual ARM Cortex-A9. Los elementos que conforman el PS son los siguientes:

- **Application Processing Unit**

También llamada APU por sus siglás en inglés, se conforma de una FPU (del inglés Floating Point Unit), MMU (del inglés *Memory Management Unit*), SCU (del inglés *Snoop Control Unit*), L1 cache, L2 cache y los procesadores ARM. La Figura 4.6 muestra la distribución de los elementos anteriores.

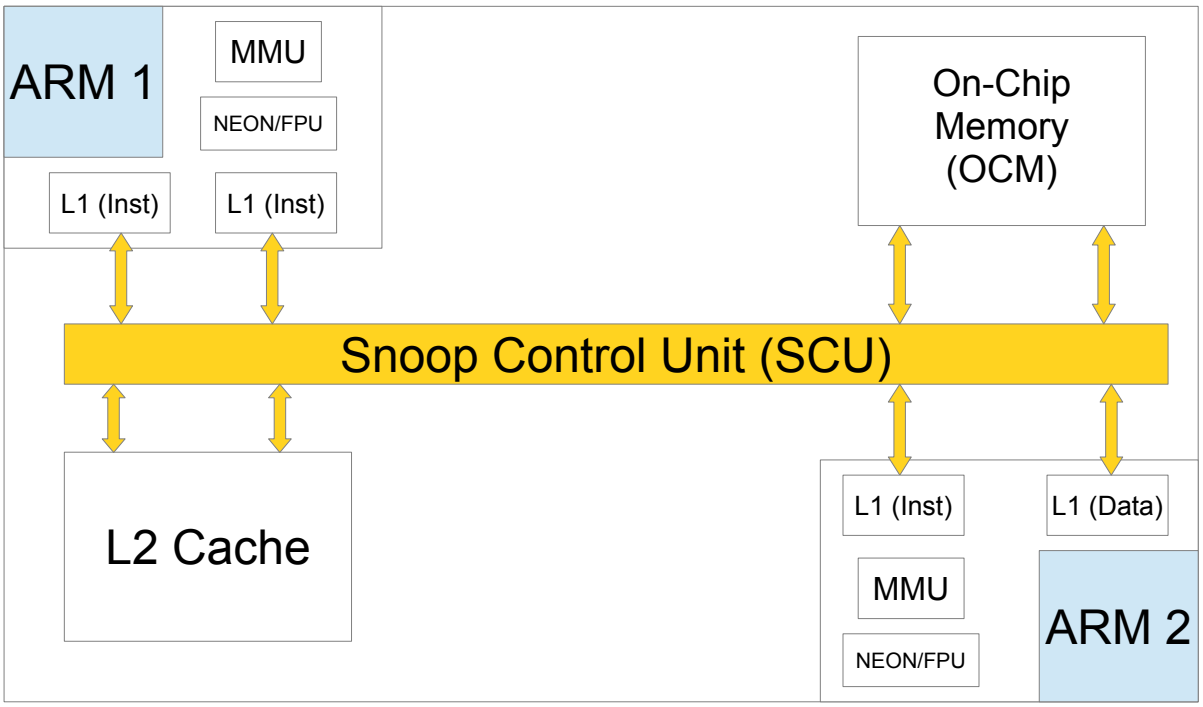


Figura 4.6: Estructura de la APU

A continuación se muestran las características de los elementos de la Figura 4.6 para el Zynq-7000 con número de pieza XC7Z010:

- L1 Cache: 32 KB para instrucciones y 32 KB de datos por procesador.
- L2 Cache: 512 KB.
- On-Chip Memory: 256 KB.
- Doble precisión de punto flotante para cada procesador.
- Doble procesador ARM Cortex-A9 con CoreSight.

4.3.3 Programmable Logic

La PL es el componente que contiene los recursos de *hardware* reconfigurable. La cantidad de recursos de PL dependerán del fabricante, así como del modelo de la tarjeta. El dispositivo Zynq-7000 con número de pieza XC7Z010 se basa en la serie Xilinx 7 con equivalente FPGA Artix-7 y contiene la siguiente cantidad de recursos:

- Convertidor analógico a digital (XADC) de 12 bits de doble canal operando como máximo a 1 MSPS (del inglés *Mega Samples Per Second*).
- 240 Kilobytes de BRAM .
- Puede operar con frecuencias de hasta 450 [Mhz].
- 80 *slices* de DSPs.
- 28 K de celdas lógicas programables.
- 4400 *slices* lógicos, cada uno con 8 *flip flops* y LUTs de 6 entradas.

La Figura 4.6 muestra los elementos que conforman un CLB dentro de la PL.

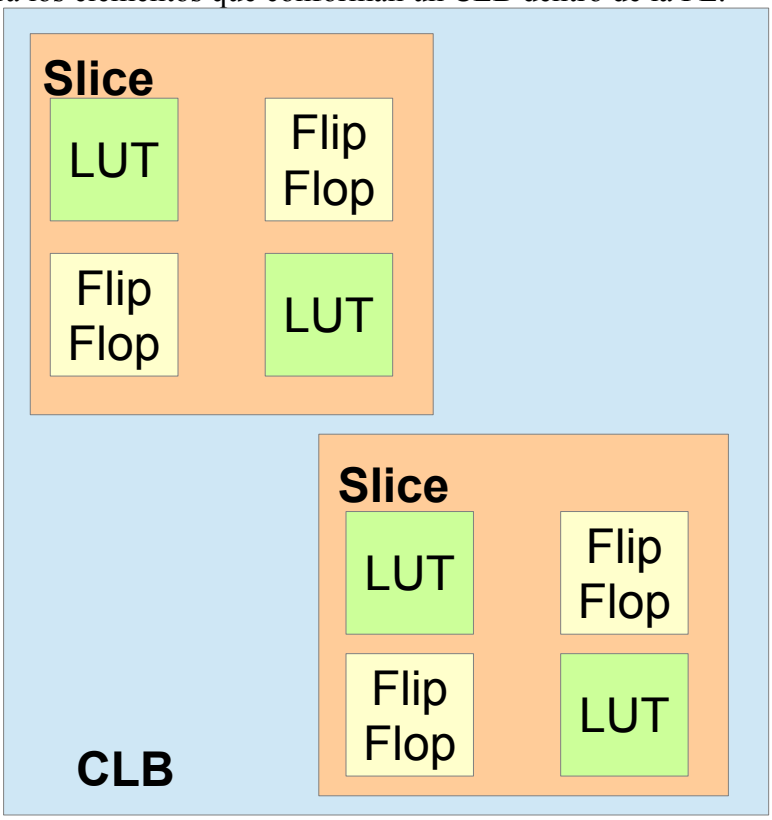


Figura 4.7: Conjunto de elementos de un CLB

5 Máquina digital de un clasificador Fuzzy ARTMAP

La máquina digital de un clasificador Fuzzy ARTMAP que aquí se presente, tiene como objetivo la clasificación de piezas de manufactura para seleccionar objetos dentro de una celda. La arquitectura del clasificador tiene una representación en punto fijo e interactúa con el PS de la tarjeta Zybo para recibir y enviar la información proveniente del generador BOF.

5.1 Arquitectura del clasificador

En esta sección se describirá la arquitectura del clasificador FAM implementado en la PL, así como la conexión de sus elementos y *hardware* utilizado.

Para iniciar el diseño en *hardware* de la Fuzzy ARTMAP se partió del clasificador Tj esquematizado en el artículo de Grossberg (S Grossberg et al., 1992). La Figura 5.1 muestra el diseño del esquema de un clasificador Tj, el cual debe ser replicado por cada objeto que se vaya a clasificar. A continuación se explicarán cada uno de los bloques que lo conforman.

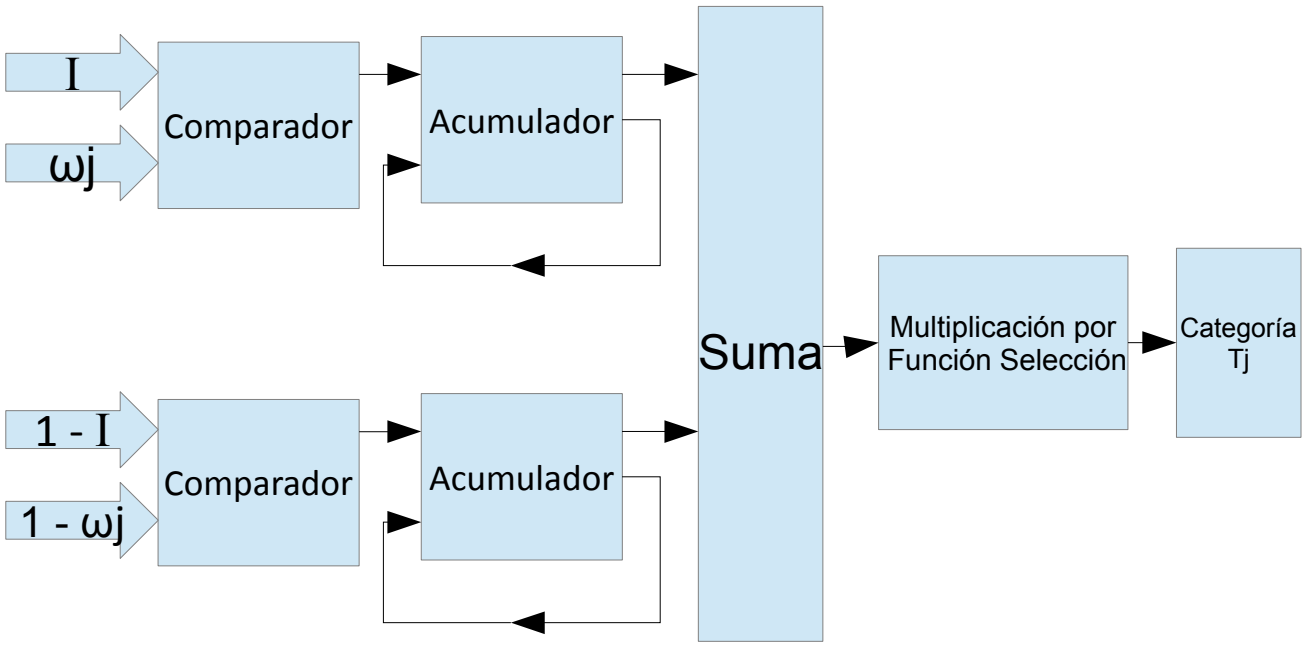


Figura 5.1: Esquema de un clasificador Fuzzy ARTMAP

- **Entradas**: Por cada clasificador se requiere de 4 entradas, éstas son para los pesos (**ωj**), BOF (**I**), y sus respectivos complementos **(1-I** y **1-ωj)**. Las entradas recibirán 202 datos, donde cada uno será de 32 bits para representar valores entre 0 y 1.

- **Comparador**: La función de este bloque es comparar los 202 datos del vector BOF (**ωj**), pesos (**I**) y sus complementos **(1-I** y **1-ωj)**. Se seleccionará el dato con menor magnitud.

- **Acumulador**: Este bloque acumulará los datos que el comparador haya seleccionado. El máximo valor que puede contener el acumulador es 202.

- **Suma**: Este bloque se encarga de sumar ambos acumuladores de 32 bits una vez que todos los datos hayan sido comparados.

- **Multiplicación**: Su tarea es multiplicar el valor obtenido en el bloque ***suma*** por la función selección (ver Ecuación 7). Dado que se multiplican dos datos de 32 bits, se deberá soportar un resultado de 64 bits.

Al finalizar los procesos de cada bloque, como resultado se obtiene una categoría Tj, la cual es representada en 64 bits.

La descripción del *hardware* de cada bloque mencionado se describirán a detalle en "5.1.2 Recursos de lógica programable".

5.1.1 Descripción del hardware

El siguiente paso en la creación de la Fuzzy ARTMAP es realizar la descripción de la arquitectura de su *hardware* con los requisitos de diseño mencionados en *"*Arquitectura del clasificador*"*. La tarjeta Zybo está diseñada para que *software* y *hardware* funcionen en conjunto, por lo que fue necesario distribuir las tareas de la FAM en el PL y PS, además de considerar como éstos se comunicarían. Las tareas que requieren de operaciones en paralelo se asignaron al PL, mientras que las tareas que dependen de las operaciones en paralelo se asignaron al PS.

Se planeó este proyecto para que la RNA estuviera comunicada por internet con el propósito de que los pesos de las categorías pudieran ser actualizados a distancia y en cualquier momento. El PS fue seleccionado para ser el elemento central que reciba la información y que la distribuya a sus respectivas IPs para su proceso. El ordenamiento de las categorías se asignó al PS con el motivo de reducir la cantidad de recursos utilizados en el PL, puesto que un algoritmo de ordenamiento en la PL demandaría recursos de éste y por consecuencia minimizando el número de módulos Fuzzy ARTMAP.

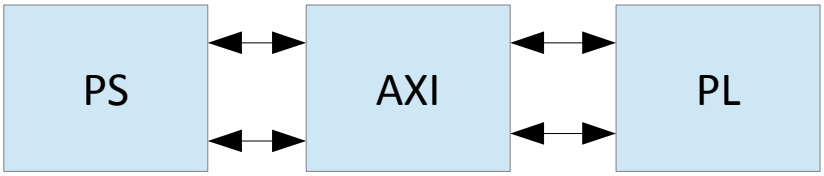


Figura 5.2: Elementos generales de la tarjeta Zybo

La Figura 5.2 muestra el sentido en que la información es enviada a través de los diferentes elementos que conforman la Zybo Zynq 7000. Es necesario el protocolo de comunicación *AXI* para que el PS y la PL puedan estar comunicados. Tanto el PS y AXI serán explicados a detalle en el siguiente capítulo.

Vivado Design Suite es una herramienta creada por Xilinx para el desarrollo de circuitos digitales a través de lenguajes de descripción de *hardware* principalmente. Este software permite el uso de distintas herramientas para la síntesis y análisis del *hardware*. Por medio de este programa se integran los componentes que conectan al AXI con la PL.

Se seleccionó Xilinx por *Vivado Design Suite,* el cual cuenta con herramientas de diseño versátiles para unir procesadores ARM y PL, así como un énfasis de su trabajo en conjunto. Dentro de las tarjetas de desarrollo FPGA Zynq 7000 de Xilinx, se seleccionó la Zybo puesto que cuenta con dos procesadores ARM que pueden ser comunicados con una PL, de esta manera las tareas pueden ser distribuidas entre uno y otro para optimizar recursos.

5.1.2 Recursos de lógica programable

El lenguaje de descripción de *hardware* utilizado es VHDL, esto para continuar con el mismo lenguaje con el que se trabajan los proyetos en el Laboratorio de Electrónica y Automatización para Industria 4.0 y así mantener un ambiente de desarrollo homogéneo.

Para desarrollar el *hardware* se ha considerado su comunicación con el PS. El Código 1 muestra la descripción de los puertos de entrada (reciben información del PS) y de salida (envían información al PS) de la entidad FAM. Para esta explicación se ha considerado la clasificación de 4 categorías.

```
entity FuzzyARTMAP3 is
 Port ( a:in std_logic_vector(31 downto 0);
        --a: Vector que activa y desactiva los módulos FAM
        clk:in std_logic;
        --clk: pulso de reloj que sincroniza los circuitos digitales
        entrada,entrada2,entrada3,entrada4,entrada5,entrada6:in std_logic_vector(31 downto 0);
        --entrada: son las entradas que se comunican con el PS para enviar los pesos y sus complementos
        entrada7,entrada8:in std_logic_vector(31 downto 0);
        BOF,BOFC,indice:in std_logic_vector(31 downto 0);
        --BOF: Entrada que se comunica con el PS para recibir el vector BOF
        --BOFC: Entrada que se comunica con el PS para recibir el vector BOF complementado
        --indice: Entrada que se comunica con el PS para recibir la posición de la ROM
        --donde la información se almacenará
        seleccion:in std_logic_vector(31 downto 0);
        --seleccion: entrada que recibe el valor de la función selección
        salidaf1,salidaf2,salidaf3,salidaf4,sumaBOF:out std_logic_vector(31 downto 0);
        --salidaf: Estas salidas envían al PS el valor de cada acumulador total
        b,c,d,e:out std_logic_vector(63 downto 0));
        --b,c,d,e: Estas salidas envían al PS el valor de cada categoría Tj
end FuzzyARTMAP3;
```

Código 1: Entidad de la Fuzzy ARTMAP para cuatro categorías

La arquitectura en la PL se diseñó con los elementos anteriores para que primero todo los datos a procesar sean recibidos y almancenados en memorias ROM. Una vez recibidos todos los datos, se procede a activar los módulos FAM y a su vez el contador que recorrerá cada una de las direcciones de memoria de la ROM para que cada dato sea comparado y acumulado. Cuando el contador alcanza el valor 202, quiere decir que toda la información se procesó y que los resultados de cada categoría Tj y cada acumulador total están listos para ser enviados al PS. Con este diseño como punto de partida, se procede a describir el *hardware* de cada elemento.

El Código 2 y Código 3 muestran los elementos que en conjunto son un módulo FAM.

Los 16 bits menos significativos de la entrada ***a*** tiene la función de activar el número de módulos Fuzzy ARTMAP. Los 16 bits restantes de la entrada ***a*** son para activar o desactivar los cálculos de la FAM. Por ejemplo, si se desea activar 10 módulos FAM se deben colocar los valores de la siguiente manera:

(0000000A)

Activación de 10 módulos FAM

Para desactivar y reiniciar los cálculos de la Fuzzy ARTMAP se deben colocar los valores como se muestra a continuación:

(00010000)

Reinicio de los cálculos de la FAM

Los puertos ***entrada*** son los encargados de recibir la información de los pesos para ser almacenados en memorias ROM (ver Figura 5.3). Habrá una ROM por cada vector de pesos y sus complementos. Se necesitan 202 localidades de memoria por cada ROM debido a que el vector BOF contiene 202 datos. La información recibida en la ROM será acumulada (ver Código 2 y Código 3) y puesto que el máximo valor de cada dato es 1, en el peor de los casos se tendrá un valor de 202 en el acumulador. Para evitar el desbordamiento en el acumulador se requiere mínimo de 8 bits para poder representar 202 en sistema binario. Como el tamaño del dato en cada localidad de memoria es de 32 bits y 8 de ellos son para representar la parte entera, los 24 bits restantes representarán la parte decimal. Como se sabe el rango de valores que tendrán los acumuladores dentro de la Fuzzy ARTMAP, realizando pruebas (ver página 43) se llegó a la conclusión de que la cantidad de bits mencionados son los óptimos para que se cubra el rango de valores posibles, además de reducir el error inherente al utilizar representació en punto fijo.

La entrada ***indice*** indica en cual dirección de la ROM se almacenan los pesos, BOF y sus complementos. El valor de índice será actualizado desde el PS.

Los puertos BOF y BOFC reciben el vector descriptivo BOF y su complemento que será almacenado en las memorias ROM con las características mencionadas anteriormente.

Para este ejemplo (ver Código 2 y Código 3), los datos del vector pesos se almacenan en ***ROM*** y sus complementos en ***ROM2***. La BOF y su complemento se guarda ***ROM9*** y ***ROM10*** respectivamente. Los 202 datos del vector pesos y BOF serán comparados, seleccionando el menor dato el cual será acumulado. Al finalizar la comparación de todos los datos, ***acumulador*** y ***acumulador2*** serán sumados dando como resultado un acumulador total (***salidaf1***).

Un contador será el encargado de recorrer cada una de las direcciones de memoria de las ROM (ver Figura 5.3). Cuando se recorran todos las direcciones, se para la comparación de los datos de la BOF y el vector pesos.

La entrada ***selección*** es la encargada de llevar la Función de Selección (Ecuación 7). Una vez que cada uno de los pesos y BOF haya sido comparado y acumulado, se multiplicará la Función de Selección por el valor contenido en el acumulador total. Este resultado representa una categoría Tj.

```
--Primer módulo FAM
process(clk,habilitador)
begin
--habilita o deshabilita este módulo FAM
if habilitador(0)='1' then

if rising_edge(clk) then
--Si el contador no ha llegado a 201 se siguen comparando
--los pesos y BOF
if  aux=201 then

else
--ROM: Contiene los pesos de la primera categoría
--ROM9: Contiene la BOF
if unsigned(ROM(aux))<=unsigned(ROM9(aux)) then
acumulador<=acumulador+unsigned(ROM(aux));
else
acumulador<=acumulador+unsigned(ROM9(aux));
end if;
end if;
end if;
else
--Si se deshabilita este módulo, el acumulador se reinicia
acumulador<="00000000000000000000000000000000";
end if;
```

Código 2: FAM con datos no complementados

```
process(clk,habilitador)
begin
--Habilita o deshabilita este módulo FAM
if habilitador(0)='1' then

if rising_edge(clk) then
--Si el contador no ha llegado a 201 se siguen comparando
--los pesos y BOF
if  aux=201 then

else
--ROM2: Contiene los pesos complementados de la primera categoría
--ROM9: Contiene la BOF complementada
if unsigned(ROM2(aux))<=unsigned(ROM10(aux)) then
acumulador2<=acumulador2+unsigned(ROM2(aux));
else
acumulador2<=acumulador2+unsigned(ROM10(aux));
end if;
end if;
end if;
else
--Si se deshabilita este módulo, el acumulador se reinicia
acumulador2<="00000000000000000000000000000000";
end if;

end process;
```

Código 3: FAM con datos complementados

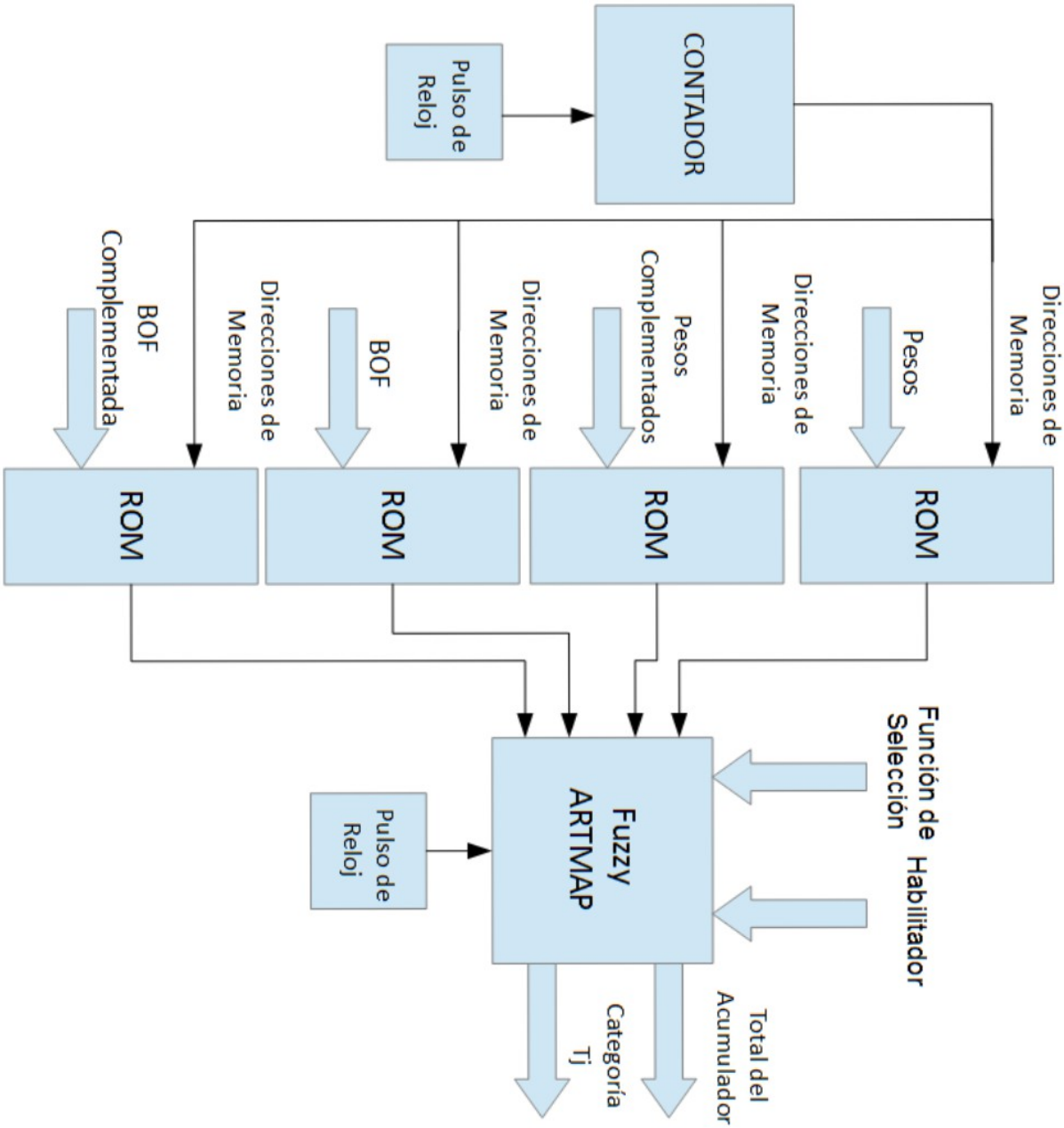


Figura 5.3: Elementos generales en la PL

5.2 Comunicación con el sistema embebido

El protocolo de comunicación *AXI* permite que la información procesada en la PL pueda ser enviada al doble procesador ARM Cortex o viceversa. Una vez finalizada la arquitectura de la FuzzyARTMAP se procedió a realizar una IP de ella. En el entorno de desarrollo de Vivado hay herramientas que permiten la conexión de la IP personalizada con el procesador Zynq. Una forma de realizar la conexión entre éstos es por medio de un *AXI4 Lite*.

5.2.1 Protocolo AXI4 Lite

El protocolo AXI o *Advanced eXtensible Interface* forma parte de buses de microcontroladores que pertenencen a ARM AMBA (AXI Reference Guide 2011). Además de su uso en microcontroladores , después se utilizaría para realizar la conexión entre diferentes bloques que pertenecen a un SoC. En la familia de dispositivos Zynq de Xilinx, permite la conexión entre el PS y el PL. La Figura 5.4 muestra los canales que tiene el protocolo AXI.

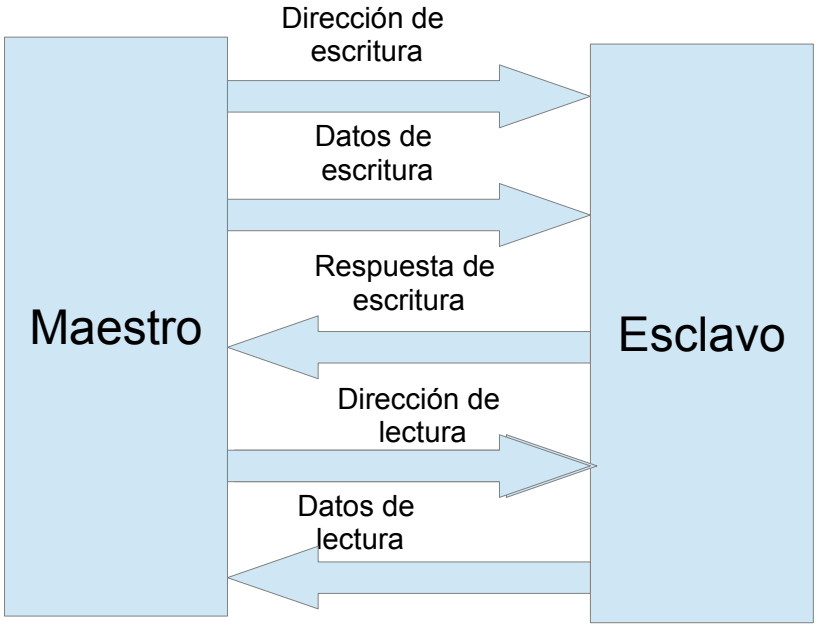


Figura 5.4: Canales de protocolo AXI

Los tipos de AXI que existen son:

- **AXI4**: cuenta con mayor rendimiento, proporciona una dirección y transfiere una gran cantidad de datos de hasta 256 palabras.

- **AXI4 *Lite***: Soporta la transferencia de un dato por conexión . Es el AXI más simplificado pues envía una dirección y sólo un dato.

- **AXI4 Stream**: Permite la transferencia de datos a altas velocidades.No utiliza direcciones porque la información es enviada sólo entre receptor y emisor.

La FAM necesita ser monitoreada y recibir o enviar pocos datos desde el PS. Una conexión simple, con herramientas de monitoreo y control básicas desde el PS y una poca cantidad de datos enviados es suficiente. *AXI4 Lite* fue seleccionado por cumplir con los requerimientos mencionados.

AXI4 Lite cuenta con los siguientes canales:

- Canal de lectura de direccionamiento: El procesador envía una señal a la interfaz esclava de que desea iniciar una transacción de lectura.
- Canal de lectura de información: envía información transferida durante el proceso.
- Canal de escritura de direccionamiento: El procesador envía una señal a la interfaz esclava de que desea iniciar una transacción de escritura.
- Canal de escritura de información: El procesador envía información a la interfaz esclava.
- Canal de respuesta de escritura: Este canal permite que la interfaz esclava indicar que la información ha sido recibida o que algún error sucedió.

Este protocolo cuenta con señales *handshaking* que pertenencen a los cinco canales mencionados.Estas señales son *Ready* y *Valid,* la primera indica que el receptor ya puede recibir direcciones o información mientras que la segunda indica si la información o direcciones son válidas.

Las señales de reconocimiento del Canal de Escritura de Direccionamiento son:

- **S_AXI_ARADDR**: Señal dirigida por la interfaz Maestra que contiene el bus de direcciones del *AXI* a la interfaz esclava.
- **S_AXI_ARVALID**: Señal de validación para indicar que S_AXI_AWADDR puede ser muestreado por el periférico esclavo.
- **S_AXI_ARREADY**: Esta señal envía un pulso para mostrar que el periférico esclavo está listo.

Las señales de reconocimiento del Canal de Lectura de Información son:

- **S_AXI_RDATA**: Bus de datos del periférico esclavo al *AXI*.
- **S_AXI_RVALID**: Señal que indica que el Maestro puede leer información del S_AXI_RDATA.

- **S_AXI_RREADY**: Indica que ya se pueden aceptar valores en el Maestro.

- **S_AXI_RRESP**: Indica si las transacciones se realizaron exitosamente o si hubo un error.Los errores y sus valores se muestran a continuación.

  a)OKAY: Indica que la información se recibió exitosamente sin errores. Su valor es **00**.

  b)EXOKAY: Este estado sólo es usado en AXI4. Su valor es **01**.

  c)SLVERR:El esclavo ha recibido la dirección de la transacción con éxito pero hay un error Su valor es **10**.

  d)DECERR: Indica que la dirección provista no existe en el espacio de direcciones del AXI Su valor es **11**.

Para el Canal de Lectura de Información, la información es enviada del periférico esclavo al Maestro.

Las señales en el Canal de Escritura de información son:

- **S_AXI_WDATA**: Bus de datos del Maestro o *AXI* al periférico esclavo.

- **S_AXI_WVALID**: Señal de validación, indica que S_AXI_RDATA puede ser muestreado por el Maestro.

- **S_AXI_WREADY**: Señal para validar que el Maestro está listo para aceptar información.

- **S_AXI_WSTRB**: Señal que indica la cantidad de bytes que serán utilizados en el bus de información y leídos por el esclavo.

La información de los diferentes canales que componen *AXI4 Lite* se extrajo del documento *Designing a Custom AXI-lite Slave Peripheral* (Griffin 2014).

Una vez que se conocen las señales del protocolo AXI, se inicia el proceso de creación de la IP personalizada (IP FAM) y se configura el AXI. Escogemos el nombre que le deseamos dar, así como la cantidad de bits de cada dato ,el número de registros y el tipo de AXI. En la Figura 5.5 podemos ver la declaración del *AXI4 Lite* y a continuación se describen a detalle los elementos que se encuentran en dicha figura.

- **Interface Type**: Pestaña de selección del tipo de AXI a seleccionar (4, Lite y stream).

- **Interface Mode**: Se selecciona si el AXI será esclavo o maestro.

- **Data Width** (Bits): Selección de cantidad de bits con las que trabajará el AXI.

- **Number of Registers**: Selección del número de registros que tendrá el AXI. El mínimo valor es 4 y el máximo es 512.

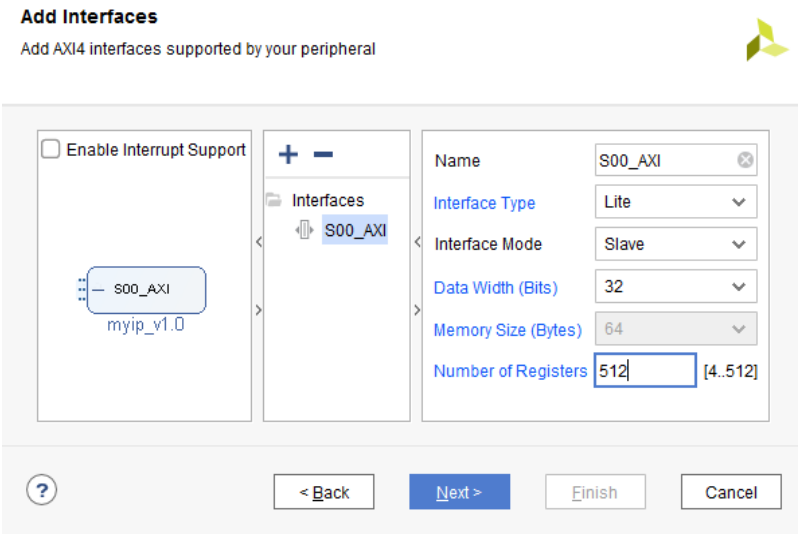


Figura 5.5: Características del AXI4 Lite

Para configurar *AXI 4 Lite* es necesario declarar los puertos y señales que se deben comunicar con la IP FAM y el *AXI*. Se agregó al proyecto de la IP FAM el archivo *VHD* de la Fuzzy ARTMAP como muestra la *Figura 5.6.*

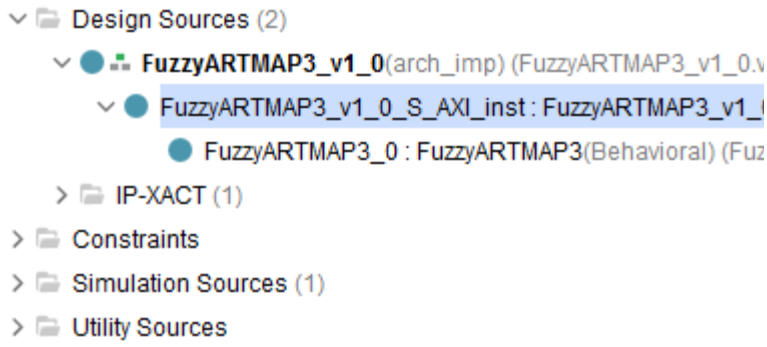


Figura 5.6: Fuentes de la IP FAM

Todos los puertos de la Fuzzy ARTMAP deben de ser declarados en el AXI de la IP FAM para que no se generen conflictos al realizar la síntesis. Estos puertos se asignan dentro del AXI para que puedan ser accedidos por el PS.

También se declaró las señales ***sale*** para que los 64 bits de salida de la IP se asignen en dos registros de 32 bits. Esta medida fue necesarioa debido que el máximo número de bits de un registro es 32.

Una última modificación a este archivo es asignar las señales a cada registro de salida a la señal ***reg_data_out***.
Para finalizar se registran todos los cambios realizados para dejar la IP FAM lista para su conexión con el PS.

La Figura 5.7 tiene el resultado final del diseño de bloques que representan la conexión de la Fuzzy ARTMAP, *AXI4 Lite* y el PS.

Con el diagrama de bloques se genera el *bitstream*, archivo con la configuración que se diseñó para la tarjeta Zybo. Una vez generado el *bitstream,* se exporta al SDK de Xilinx donde se creará el código de programación, el cual es ejecutado en el procesador ARM del PS. La función del programa es leer los registros que contiene los datos de las categorías Tj.

5.3 Operaciones del clasificador sobre el sistema embebido

El Código 4 muestra la dirección de memoria en base hexadecimal para acceder a los registros.

```
/* Definiciones para el periférico FUZZYARTMAP3_0 */
#define XPAR_FUZZYARTMAP3_0_DEVICE_ID 0
#define XPAR_FUZZYARTMAP3_0_S_AXI_BASEADDR 0x43C00000
#define XPAR_FUZZYARTMAP3_0_S_AXI_HIGHADDR 0x43C0FFFF
```

Código 4: Dirección base de los registros

Se genera un archivo *FuzzyARTMAP.h*, éste contiene el valor de *offset* que debe ser sumado a la dirección base para tener acceso al registro que se desee.

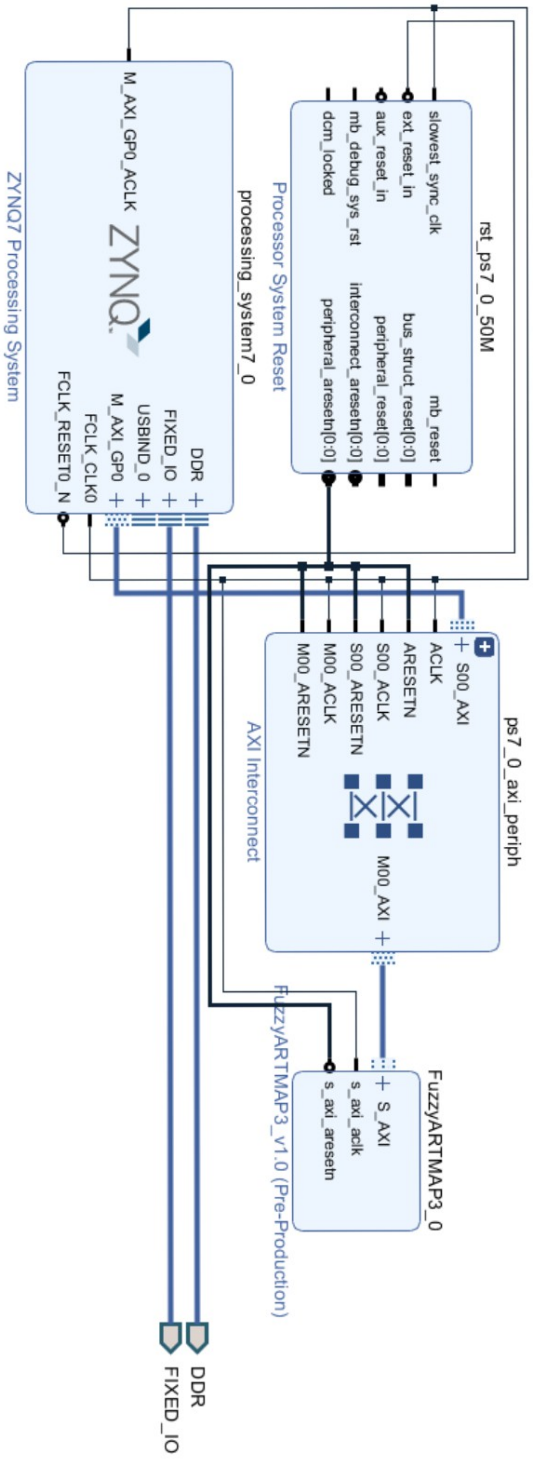


Figura 5.7: Comunicación de la Fuzzy ARTMAP con el sistema embebido

En la Figura 5.8 se muestra el diagrama de flujo del programa principal que se ejecuta en la parte del PS del Zynq 7000. Los bloques en verde son las tareas que se realizan en la PL. El lenguaje de programación es C.

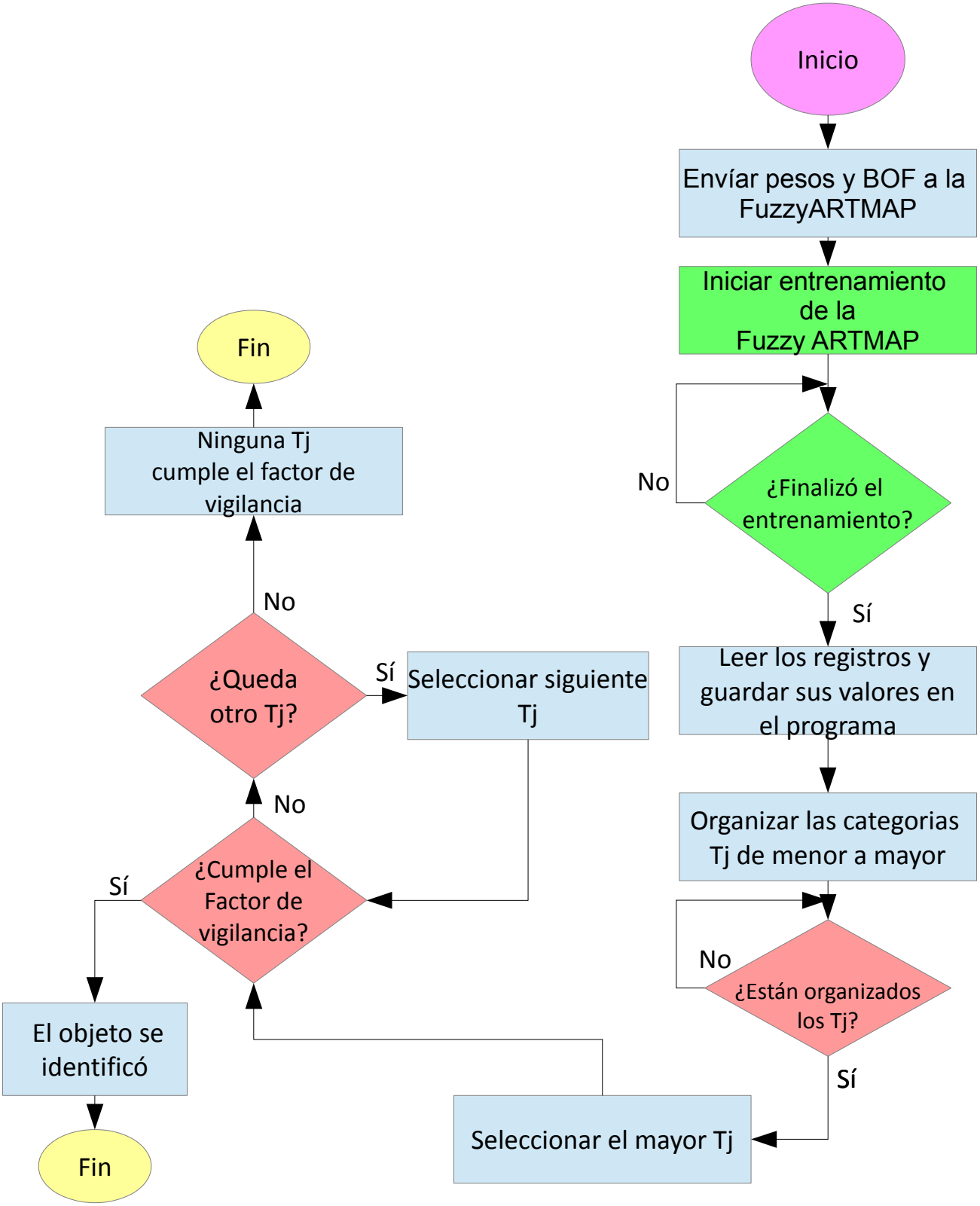


Figura 5.8: Digrama de flujo en el PS y su interacción con el PL

Los pesos y BOF son guardados en una matriz donde cada renglón son los datos de los pesos, BOF y sus complementos. Cada renglón contiene 202 datos. Se muestra en el Código 5 que por medio de dos ciclo *for*, los pesos, BOF y sus complementos son enviados a la PL. El ciclo *for* anidado es el encargado de enviar la información de cada renglón de la matriz, mientras que el otro ciclo *for* direcciona el renglón de la matriz. La instrucción *Xil_Out32* envía información a la PL y debe tener como argumento la dirección de memoria donde se desea transferir los datos y la información en sí.

```
offsets=0x00000000;
    contador=0x00000000;
    for(i=0;i<4;i++){
        for(j=0;j<202;j++){
        Xil_Out32(XPAR_FUZZYARTMAP3_0_S_AXI_BASEADDR+4,contador);
        Xil_Out32(XPAR_FUZZYARTMAP3_0_S_AXI_BASEADDR+offsets,pesos[i][j]);

        }
        contador=contador+0x00000001;
        offsets=offsets+0x00000004;
    }
```

Código 5: Envío de información del PS al PL

El registro con el *offset* de valor 4 tiene la función de indicar la dirección de memoria donde se van a guardar los pesos, BOF y sus complementos. Durante cada ciclo *for* diferentes posiciones del arreglo pesos y BOF serán accedidos y enviados a la PL.

Para iniciar los cálculos de la Fuzzy ARTMAP se debe enviar en el primer registro una palabra de 32 bits, ver Código 6, donde el valor en los 16 bits más significativos activa o desactiva el contador que recorrerá todas las localidades de memoria de las ROM. Los 16 bits menos significativos activan la cantidad de módulos Fuzzy ARTMAP que se deseen utilizar. El registro con *offset* con valor de 96 tiene la tarea de llevar el valor de la Función de Selección a la PL.

```
Xil_Out32(XPAR_FUZZYARTMAP3_0_S_AXI_BASEADDR,0x00010000);
Xil_Out32(XPAR_FUZZYARTMAP3_0_S_AXI_BASEADDR,0x00000004);


Xil_Out32(XPAR_FUZZYARTMAP3_0_S_AXI_BASEADDR+96,0x0001446f);
```

Código 6: Activación de los módulos FAM y envío de la Función de Selección al PL

Cuando se termina de hacer los cálculos de la FAM, se envía la información del PL al PS por medio de la instrucción *Xil_In32*, la cual sólo requiere como argumento la dirección de memoria donde se guarda el resultado. Una vez multiplicados los acumuladores de cada módulo por la Función de Selección, se tiene el valor de *Tj* de cada categoría en dos palabras de 32 bits. En el Código 7 el primer ciclo for obtiene los resultados de la FAM y los 32 bits más significativos son guardados en el vector ***resultados*** mientras que los bits restantes son guardados en el vector ***resultados2.*** De igual manera los resultados de los acumuladores cada módulo FAM son enviados al PS.

```
  offsets=0x00000064;

        for(i=0;i<30;i++){


resultados[i]=Xil_In32(XPAR_FUZZYARTMAP3_0_S_AXI_BASEADDR+offsets);
                    offsets=offsets+0x00000004;

resultados2[i]=Xil_In32(XPAR_FUZZYARTMAP3_0_S_AXI_BASEADDR+offsets);

        }
        offsets=0x000000B8;
        for(i=0;i<15;i++){

             acumulador[i]=Xil_In32(XPAR_FUZZYARTMAP3_0_S_AXI_BASEADDR+offsets);
             offsets=offsets+0x00000004;
        }
```

Código 7: Envío de los resultados del PL al PS

El PS trabaja con 32 bits, es por ello que los resultados en 64 bits deben ser truncados. El Código 8 muestra como utilizando los operadores de corrimiento a la izquierda y derecha se tiene como resultado los bits más significativos de cada palabra. A estos dos resultados les es aplicado el operador OR binario y se obtiene una sola palabra de 32 bits donde se tiene 4 bits para la parte entera y los restantes para la parte decimal. Cada resultado es guardado en el arreglo ***categorias***.

```
u32 categorias[15];
        for(i=0;i<15;i++){

             salidaB=resultados[i];
             salidaB2=resultados2[i];
             categorias[i]=(u32)salidaB <<12 | (u32)salidaB2 >>20;
        }
```

Código 8: Corrimiento y aplicación del operador OR a los valores Tj

También se leen los registros que almacenan los valores de cada acumulador de los módulos para que éstos sean guardados en el PS.

Los *Tj* se organizan utilizando el algoritmo *QuickSort* puesto que su complejidad en promedio es *O(nlog(n))* y en el peor caso es $O(n^2)$. El beneficio de este algoritmo es que si se tiene una gran cantidad de categorías, el ordenamiento de éstas será lo más rápido posible. Dentro de este algoritmo se guarda en otros vectores la posición original de cada Tj, así como el valor del acumulador de su respectivo módulo Fuzzy ARTMAP.

Cuando los valores Tj estén organizados, se empezará con el que tenga el mayor valor para ser comparado con el factor de vigilancia, si lo cumple este valor será catalogado como el ganador, en caso contrario se seleccionará el siguiente valor Tj hasta encontrar alguno que satisfaga el factor de vigilancia.

6 Pruebas y resultados

En la presente sección se muestran las pruebas que se realizaron a la máquina digital de un clasificador Fuzzy ARTMAP, así como los resultados obtenidos. Las pruebas que se realizaron consisten en tomar una fotografía a 15 objetos diferentes para clasificar cada uno de ellos. Esta fotografía debe contener sólo el objeto de interés con un fondo blanco (ver Figura 6.1).

Figura 6.1: Estado de Sonora como figura de prueba

Para realizar las pruebas del clasificador sobre la FPGA, se conformo un conjunto de datos del descriptor BOF de 15 piezas distintas. Estos datos se obtuvieron fuera de línea utilizando MATLAB para generar los pesos y la BOF. En la Figura 6.2 se puede observar como en el programa en Matlab la región de interés se encuentra en negro, mientras que el fondo está en blanco.

Figura 6.2: Procesamiento de la imagen en Matlab para obtener su BOF

Pruebas y resultados

Una vez generados los pesos y la BOF de las 15 figuras, se procede a almacenar esta información en el PS para ser procesada en la PL. El experimento consiste en realizar 15 pruebas, donde en cada una se selecciona un objeto diferente para ser reconocido.

En conjunto, las tablas 7, 8 y 9 muestran los resultados de las 15 pruebas realizadas. La columna ***Prueba*** indica el número de la prueba y número de la categoría Tj que se desea reconocer. La columna ***Figura*** contiene las figuras que se van a reconocer, siendo la figura ***Durango*** la categoría Tj1, Bulldog la categoría Tj2, etc.

Las columnas ***Tj*** contienen dos celdas, una de ellas indica la ***posición*** que obtuvo esa categoría en cada prueba, mientras que la otra celda indica el ***valor*** o resultado que se obtuvo de Tj en cada prueba. El valor de cada Tj va de 0 hasta 1, donde la categoría Tj ganadora es la que tenga el valor más alto. La posición con el valor Tj más pequeño es la 15, mientras que la posición 1 es para la que tiene un valor de Tj más grande. Como ejemplo se tomará la **Prueba 6.** En esa prueba se desea reconocer la figura ***estrella***, la cual tiene el Tj número 6. Los valores Tj en la Tabla 8 indican qué posición y valor obtuvo cada Tj en la prueba número 6. Observando los diferentes valores Tj de la prueba 6, el valor más alto corresponde al ***Tj6*** con la posición número 1 con un ***valor*** de 0.9950432.

La Tabla 6 muestra si la figura seleccionada en cada prueba fue reconocida por la FAM.

Prueba	Reconoció
1	Sí
2	Sí
3	Sí
4	Sí
5	Sí
6	Sí
7	Sí
8	Sí
9	Sí
10	Sí
11	Sí
12	Sí
13	Sí
14	Sí
15	Sí

Tabla 6: Resultados del reconocimientos de las 15 figuras cada prueba

		Tj1		Tj2		Tj3		Tj4		Tj5	
Prueba	**Figura**	**Posición**	**Valor**	**Posición**	**Valor**	**Posición**	**Valor**	**Posición**	**Valor**	**Posición**	**Valor**
1	Durango	1	0.9950432	4	0.9194493	9	0.8979938	13	0.855589	14	0.8533332
2	Bulldog	7	0.9194493	1	0.9950432	2	0.9373012	10	0.9071935	9	0.9093427
3	Hidalgo	10	0.8979938	2	0.9373012	1	0.9950432	4	0.9194126	7	0.9156752
4	Engrane	11	0.855589	6	0.9071935	4	0.9194126	1	0.9950432	3	0.9244942
5	Oso	12	0.8533332	6	0.9093427	5	0.9156752	4	0.9244942	1	0.9950432
6	Estrella	8	0.9247975	6	0.9291378	9	0.9241856	14	0.874214	12	0.8916262
7	Tortuga	10	0.8932459	3	0.9319113	8	0.9068414	4	0.9285941	2	0.9344673
8	León	7	0.9174513	6	0.9352604	10	0.8997939	14	0.8711699	12	0.8833752
9	Sonic	7	0.9082267	8	0.9001973	11	0.8759211	14	0.835917	13	0.8566677
10	Rana	8	0.9267741	5	0.9352649	9	0.9159844	14	0.8630162	13	0.8863315
11	Pato	8	0.8998143	7	0.9016073	10	0.8805066	14	0.837163	13	0.8532551
12	Sinaloa	5	0.8910434	9	0.8623201	10	0.8601414	14	0.8017501	13	0.8034898
13	Sonora	7	0.9192752	9	0.8935508	10	0.8798937	14	0.8184014	13	0.840994
14	Guerrero	12	0.8658014	5	0.9154147	4	0.918518	8	0.8994422	10	0.9401689
15	Oaxaca	11	0.8009718	7	0.8496098	6	0.8652014	2	0.9163144	4	0.8876513

Tabla 7: Resultados de las pruebas desde Tj1 hasta Tj5

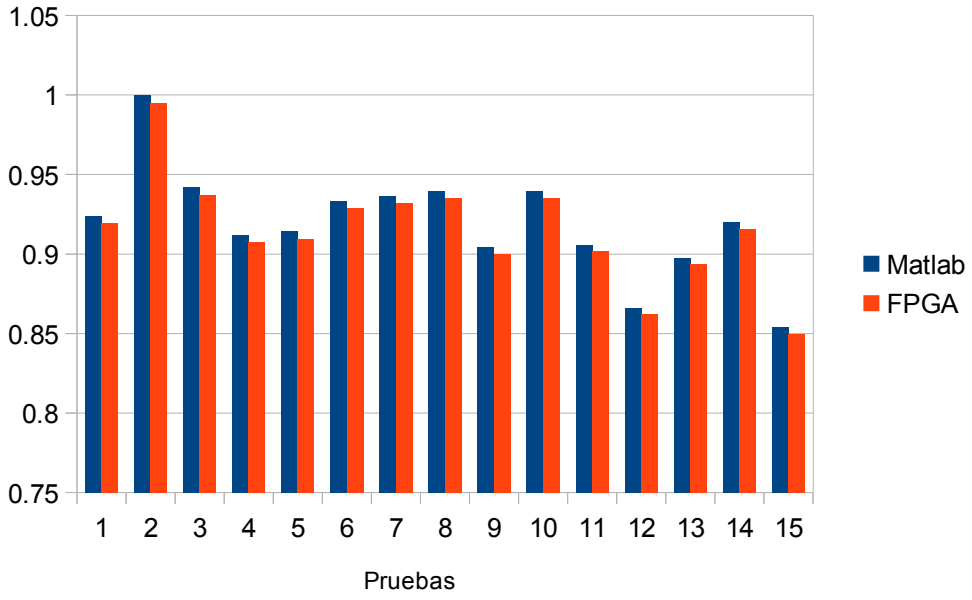
		Tj6		Tj7		Tj8		Tj9		Tj10	
Prueba	**Figura**	**Posición**	**Valor**	**Posición**	**Valor**	**Posición**	**Valor**	**Posición**	**Valor**	**Posición**	**Valor**
1	Durango	3	0.9247975	10	0.8932459	6	0.9174513	7	0.9082267	2	0.9267741
2	Bulldog	6	0.9291378	5	0.9319113	4	0.9352604	12	0.9001973	3	0.9352649
3	Hidalgo	3	0.9241856	8	0.9068414	9	0.8997939	13	0.8759211	6	0.9159844
4	Engrane	8	0.874214	2	0.9285941	9	0.8711699	13	0.835917	10	0.8630162
5	Oso	7	0.8916262	3	0.9344673	10	0.8833752	11	0.8566677	9	0.8863315
6	Estrella	1	0.9950432	10	0.9226811	3	0.9477665	4	0.9337963	2	0.9560744
7	Tortuga	6	0.9226811	1	0.9950432	7	0.9106265	11	0.883266	9	0.906431
8	León	3	0.9477665	9	0.9106265	1	0.9950432	5	0.9359446	4	0.9450298
9	Sonic	6	0.9337963	10	0.883266	5	0.9359446	1	0.9950432	3	0.9382463
10	Rana	2	0.9560744	10	0.906431	3	0.9450298	4	0.9382463	1	0.9950432
11	Pato	5	0.9330782	11	0.8783762	3	0.9491904	2	0.9547282	6	0.9277224
12	Sinaloa	7	0.8817188	11	0.8206834	8	0.8713961	3	0.8966734	4	0.8911961
13	Sonora	5	0.9276774	11	0.8623613	8	0.9158022	2	0.9381196	6	0.9273864
14	Guerrero	6	0.9048998	3	0.9245923	9	0.8927989	13	0.8624696	7	0.9044818
15	Oaxaca	8	0.8215269	5	0.8817657	9	0.8105278	12	0.7762989	10	0.8085139

Tabla 8: Resultados de las pruebas desde Tj6 hasta Tj10

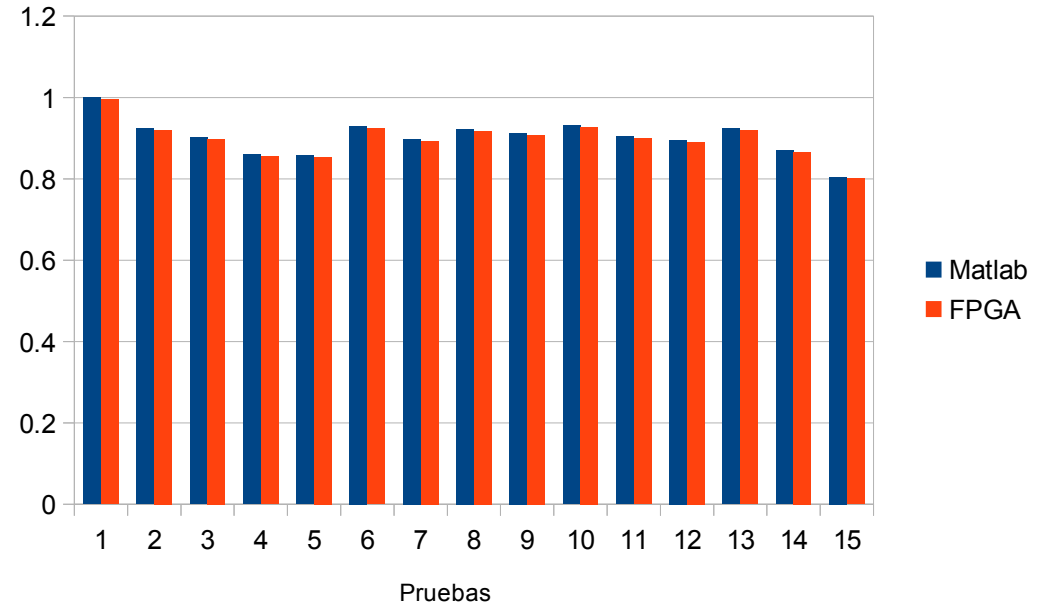
		Tj11		Tj12		Tj13		Tj14		Tj15	
Prueba	**Figura**	**Posición**	**Valor**	**Posición**	**Valor**	**Posición**	**Valor**	**Posición**	**Valor**	**Posición**	**Valor**
1	Durango	8	0.8998143	11	0.8910434	5	0.9192752	12	0.8658014	15	0.8009718
2	Bulldog	11	0.9016073	14	0.8623201	13	0.8935508	8	0.9154147	15	0.8496098
3	Hidalgo	11	0.8805066	15	0.8601414	12	0.8798937	5	0.918518	14	0.8652014
4	Engrane	12	0.837163	15	0.8017501	14	0.8184014	7	0.8994422	5	0.9163144
5	Oso	13	0.8532551	15	0.8034898	14	0.840994	2	0.9401689	8	0.8876513
6	Estrella	5	0.9330782	13	0.8817188	7	0.9276774	11	0.9048998	15	0.8215269
7	Tortuga	13	0.8783762	15	0.8206834	14	0.8623613	5	0.9245923	12	0.8817657
8	León	2	0.9491904	13	0.8713961	8	0.9158022	11	0.8927989	15	0.8105278
9	Sonic	2	0.9547282	9	0.8966734	4	0.9381196	12	0.8624696	15	0.7762989
10	Rana	6	0.9277224	12	0.8911961	7	0.9273864	11	0.9044818	15	0.8085139
11	Pato	1	0.9950432	9	0.8890397	4	0.9360341	12	0.8653297	15	0.774968
12	Sinaloa	6	0.8890397	1	0.9950432	2	0.9366479	12	0.8170604	15	0.7472478
13	Sonora	4	0.9360341	3	0.9366479	1	0.9950432	12	0.8560807	15	0.7638991
14	Guerrero	12	0.8653297	15	0.8170604	14	0.8560807	1	0.9950432	10	0.89187426
15	Oaxaca	13	0.774968	15	0.7472478	14	0.7638991	3	0.8918742	1	0.9950432

Tabla 9: Resultados de las pruebas desde Tj11 hasta Tj15

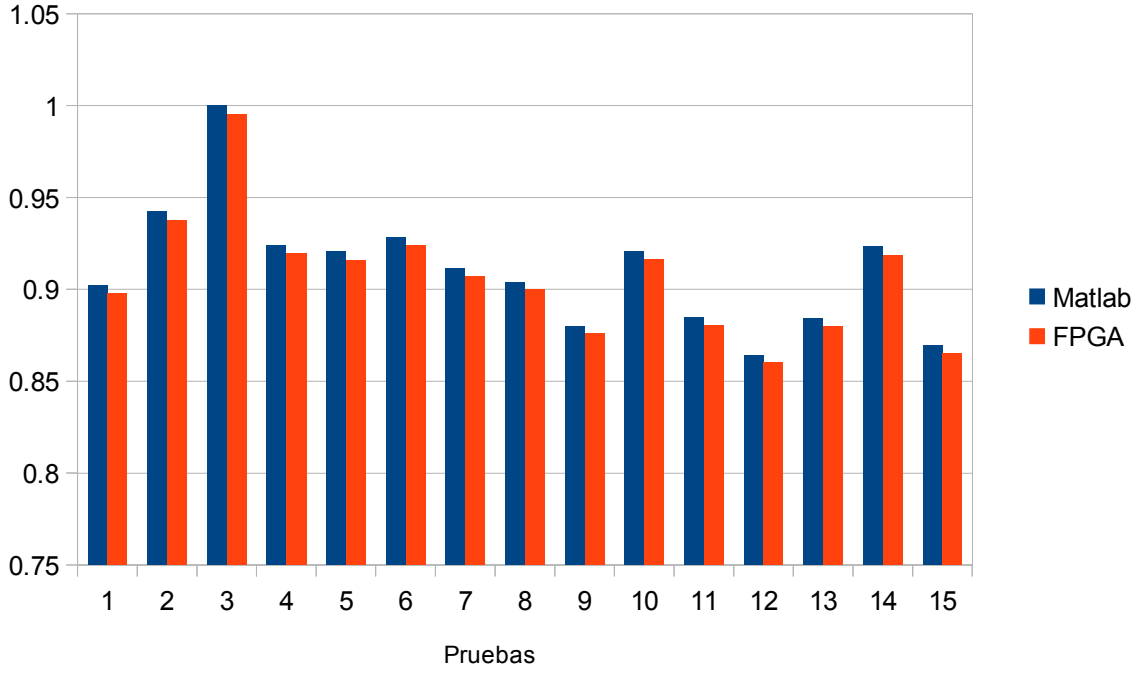
Las siguientes tablas comparan los resultados obtenidos en la FAM contra los obtenidos en Matlab. En el eje de las abscisas se encuentra el número de la ***prueba***, mientras que en el eje de las ordenadas se encuentra el valor de ***Tj*** obtenido en dicha prueba.



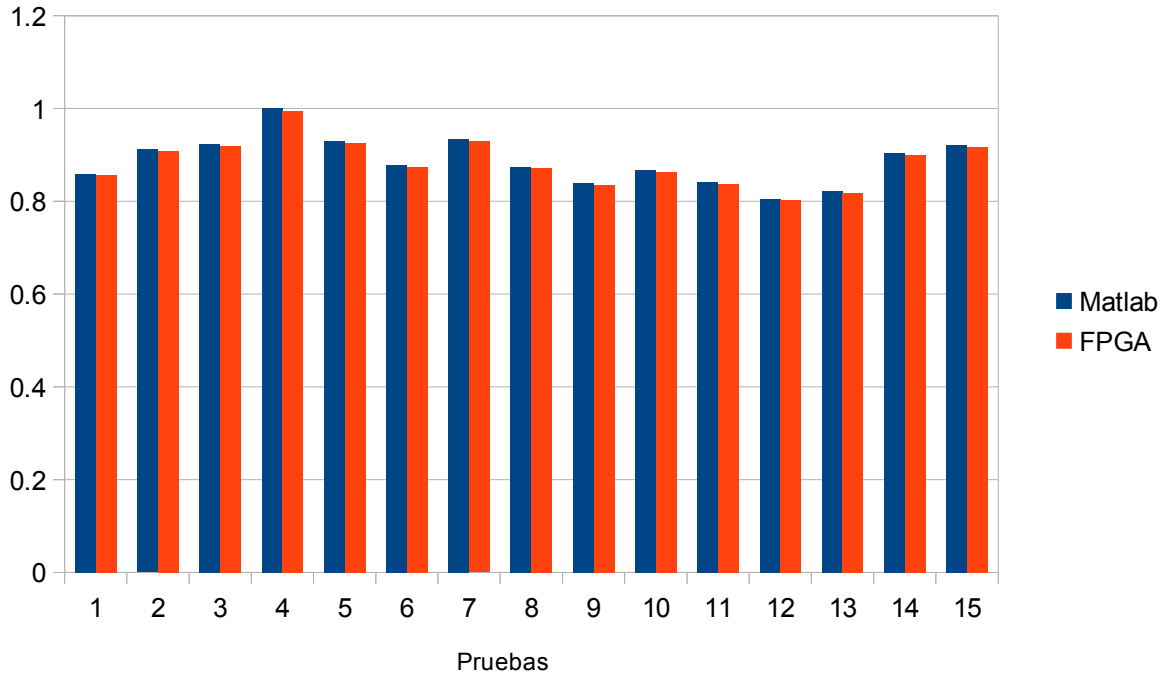
Gráfica 2: Comparación de los valores de Tj2 en la FPGA y Matlab



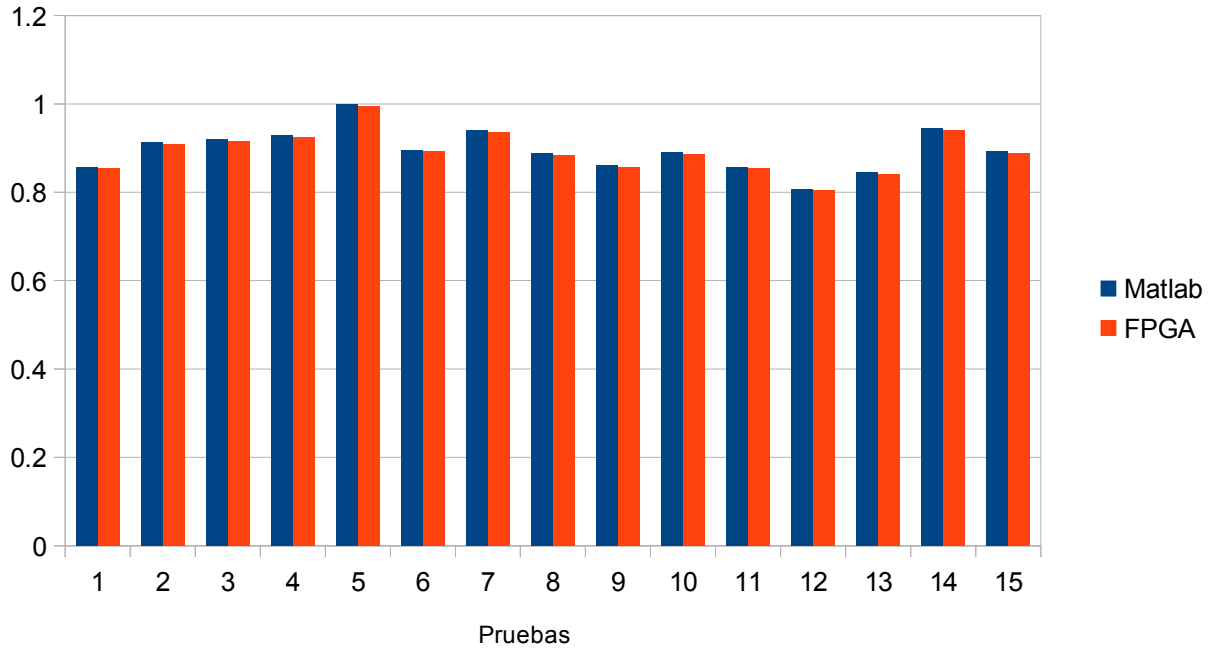
Gráfica 3: Comparación de los valores de Tj1 en la FPGA y Matlab



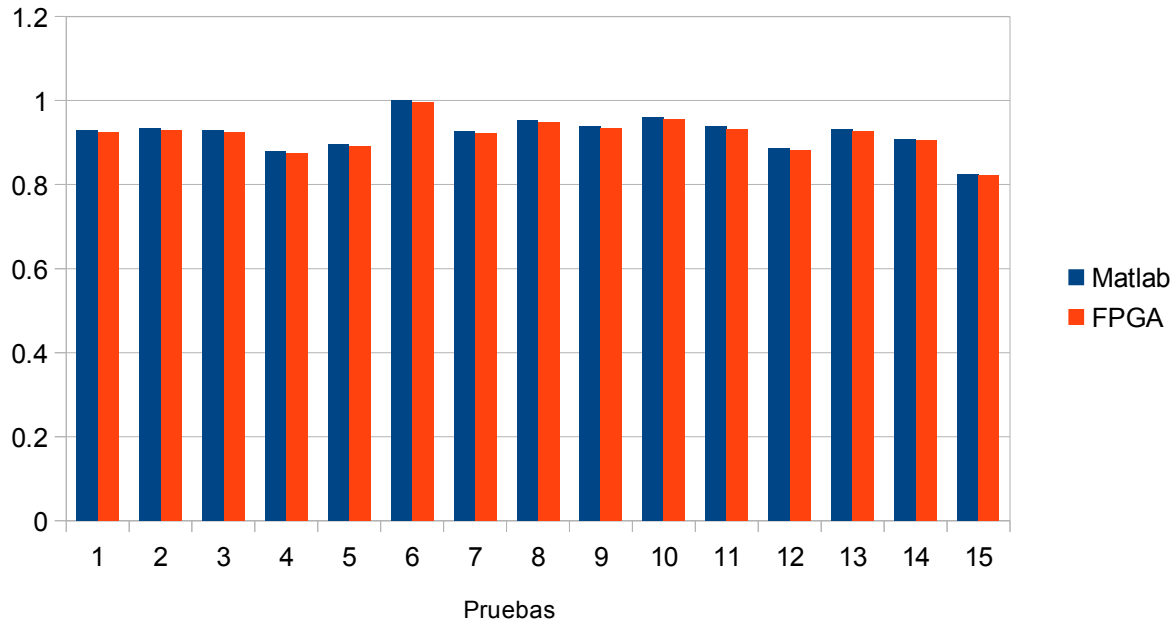
Gráfica 5: Comparación de los valores de Tj3 en la FPGA y Matlab



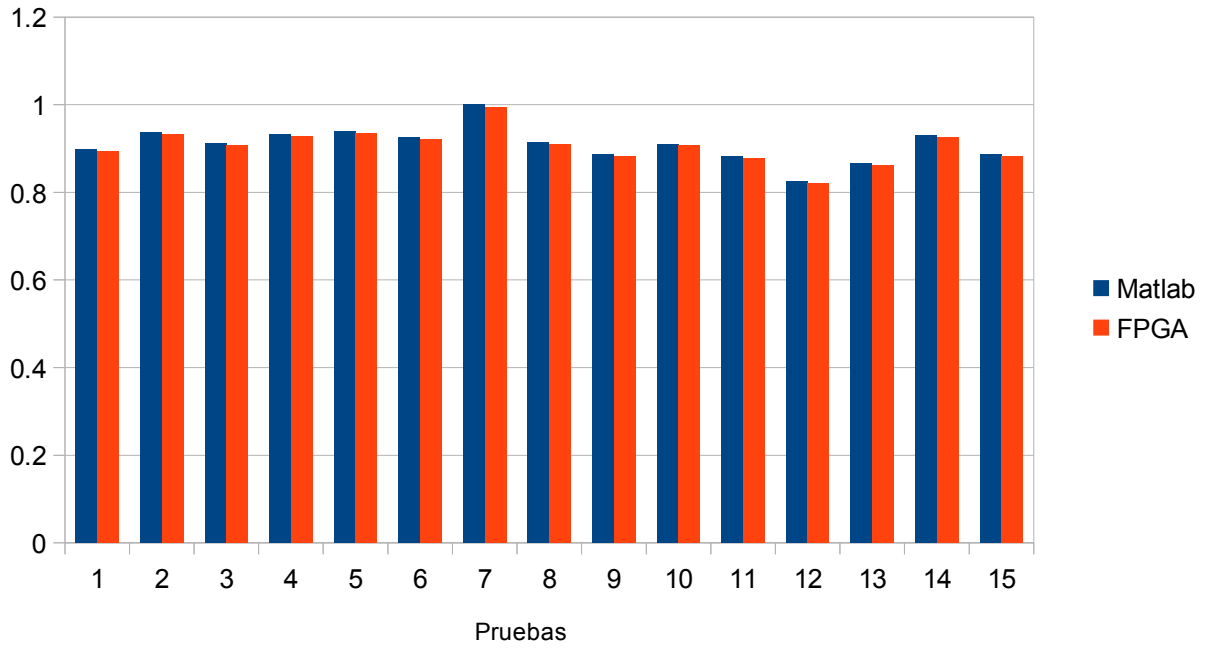
Gráfica 4: Comparación de los valores de Tj4 en la FPGA y Matlab



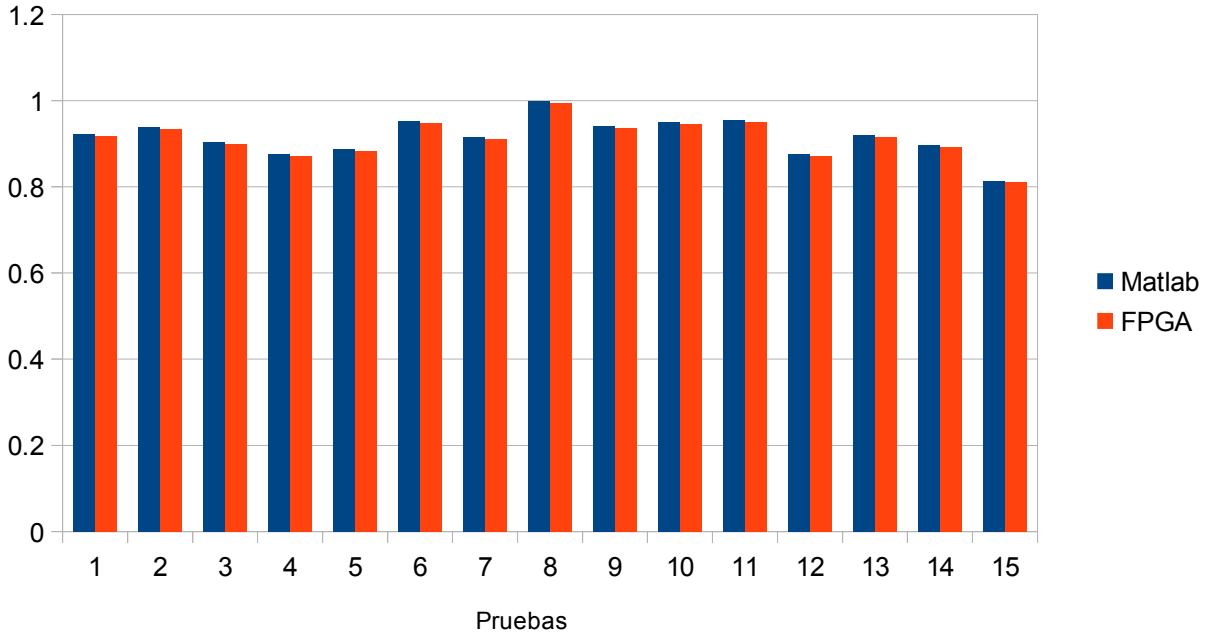
Gráfica 6: Comparación de los valores de Tj5 en la FPGA y Matlab



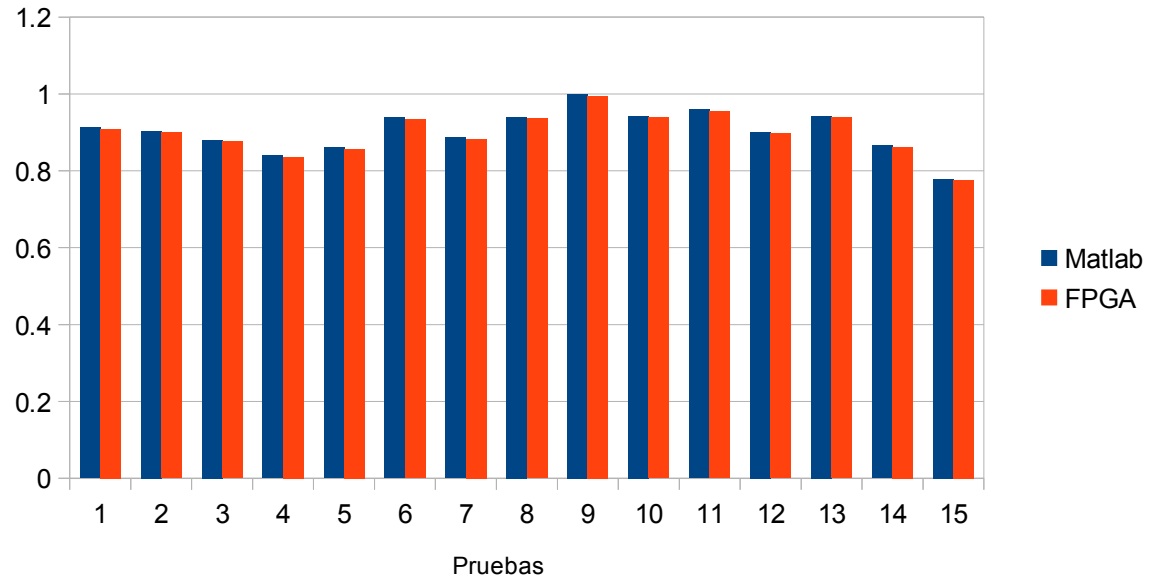
Gráfica 7: Comparación de los valores de Tj6 en la FPGA y Matlab



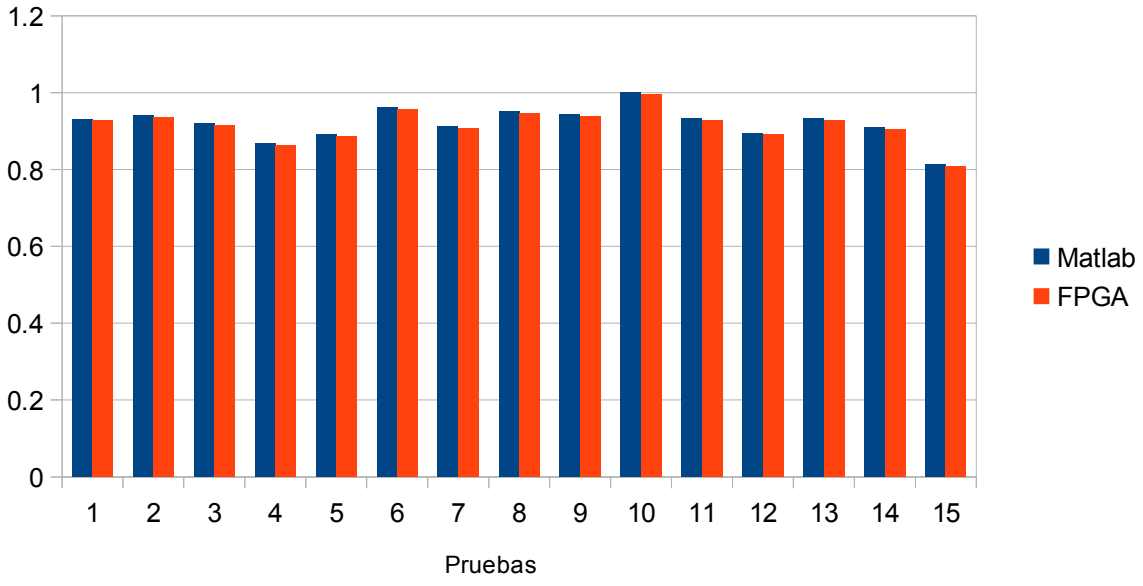
Gráfica 8: Comparación de los valores de Tj7 en FPGA y en Matlab



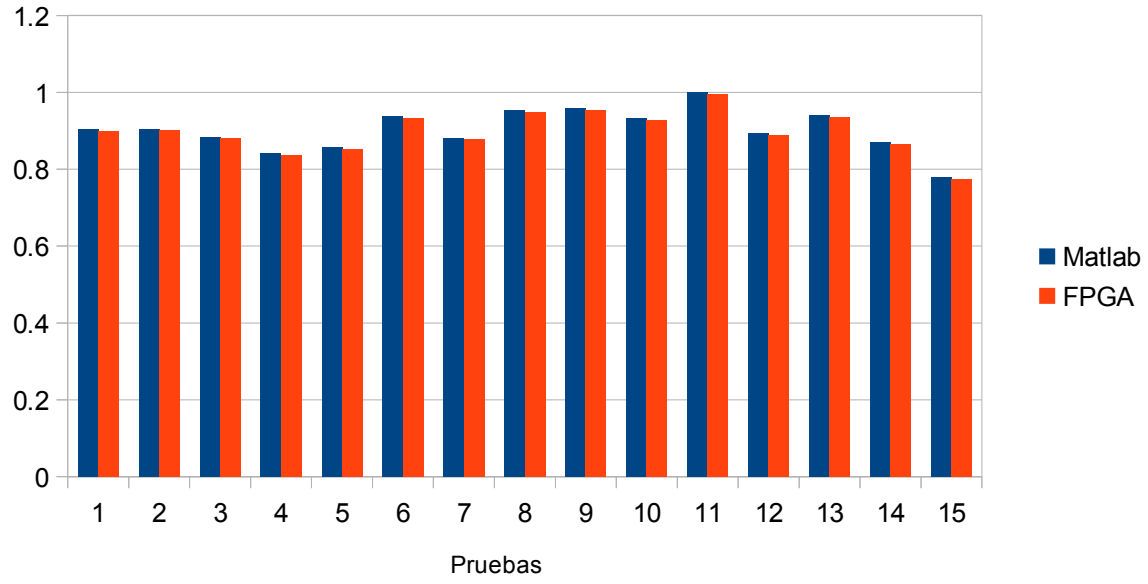
Gráfica 9: Comparación de los valores de Tj8 en FPGA y en Matlab



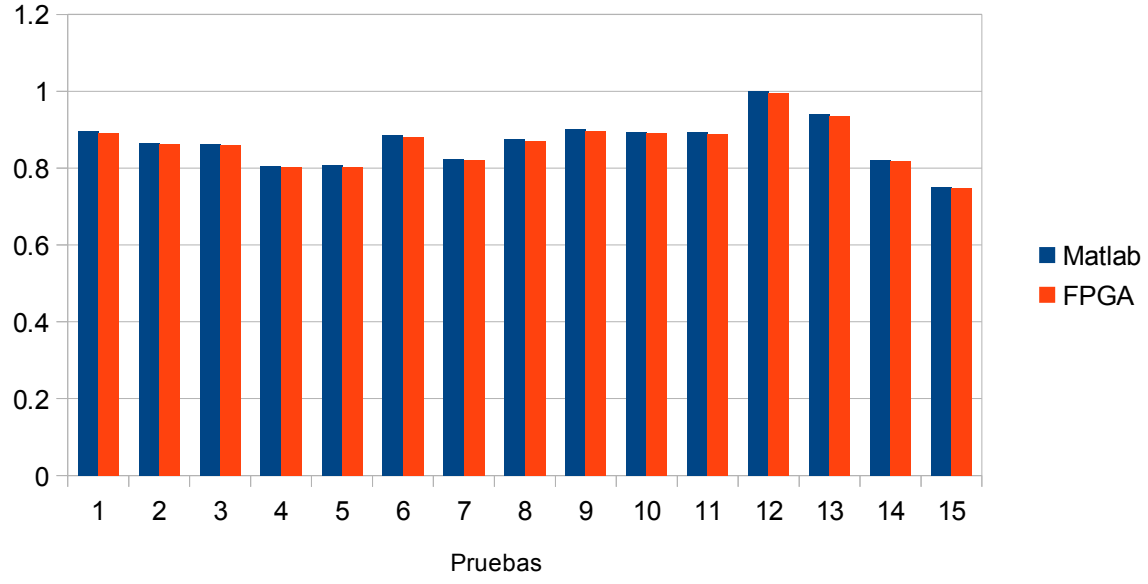
Gráfica 10: Comparación de los valores de Tj9 en FPGA y Matlab



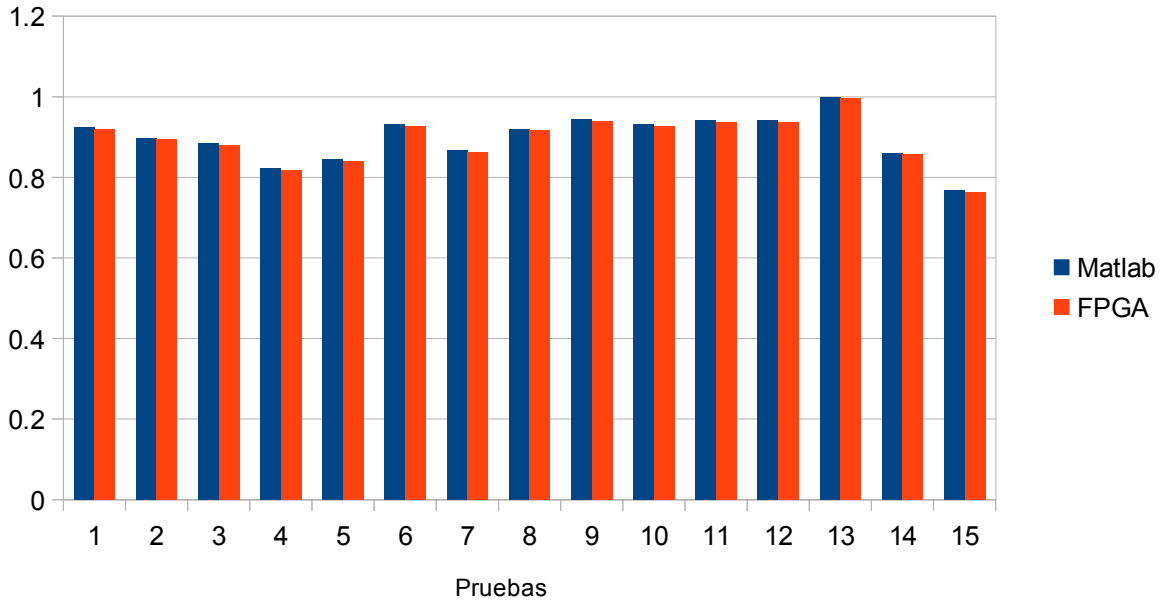
Gráfica 11: Comparación de los valores Tj10 en FPGA y Matlab



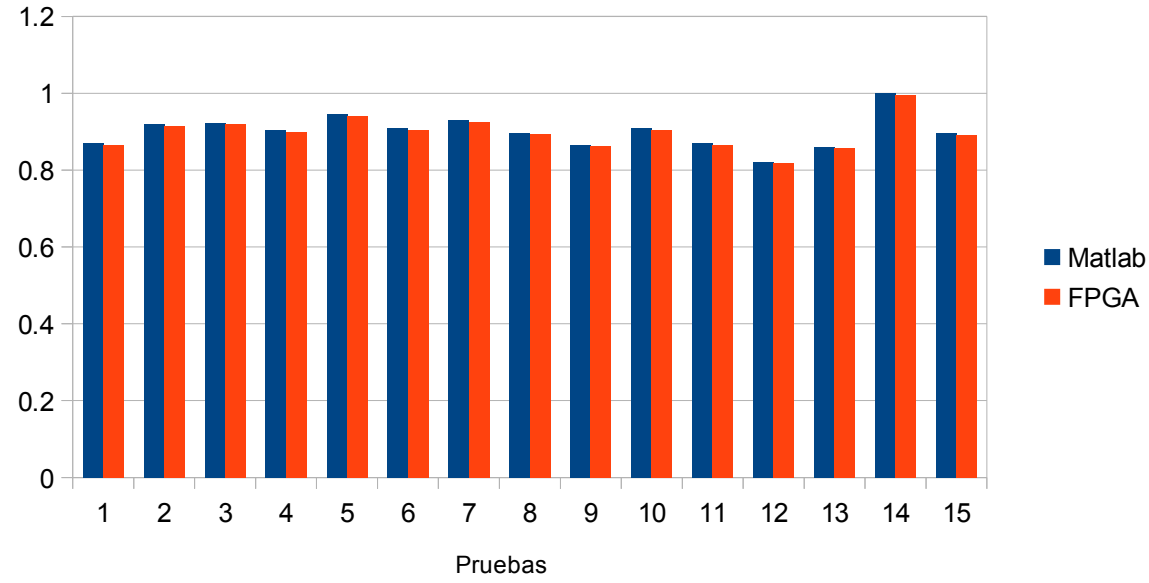
Gráfica 12: Comparación de los valores Tj11 en FPGA y en Matlab



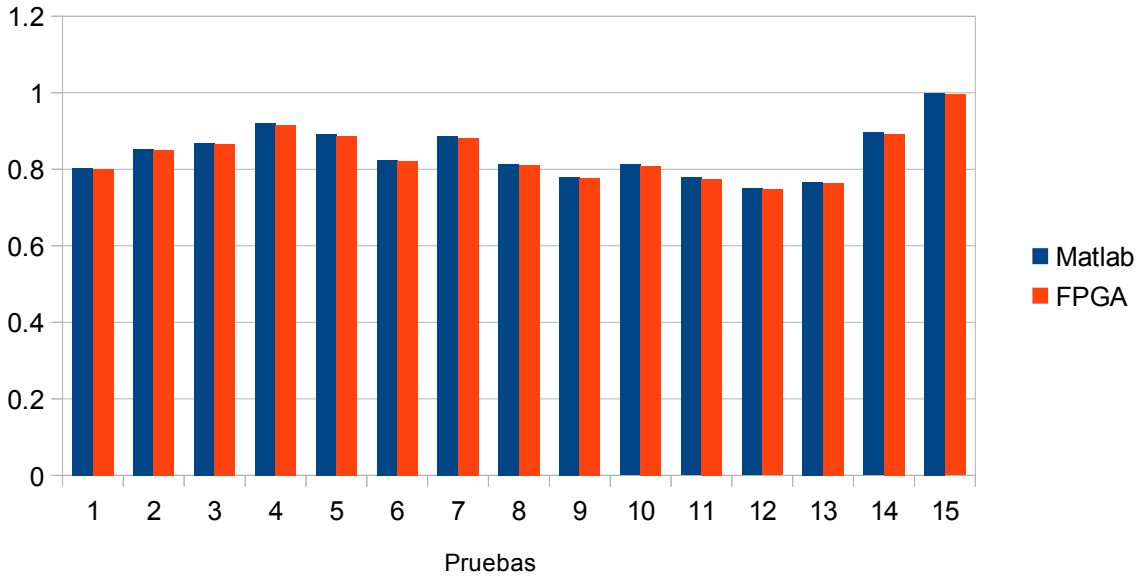
Gráfica 13: Comparación de los valores Tj12 en FPGA y en Matlab



Gráfica 14: Comparación de los valores Tj13 en FPGA y en Matlab



Gráfica 15: Comparación de los valores Tj14 en FPGA y en Matlab



Gráfica 16: Comparación de los valores Tj15 en FPGA y en Matlab

Para cuantificar el error de la FAM en FPGA comparado con la FAM en Matlab, se calculó el Error Cuadrático Medio (ECM), ver Ecuación 10. El ECM se calculó comparando los 15 valores de cada Tj con los 15 valores Tj obtenidos en Matlab.

$$ECM = \sqrt{\frac{\sum_{i=1}^{n} P_i - O_i{}^2}{n}}$$

Ecuación 10: Error Cuadrático Medio

Tj	Error Cuadrático Medio
1	0.00440047
2	0.00439958
3	0.0044031992
4	0.0042265
5	0.00437252
6	0.00441996
7	0.0042857
8	0.00440141
9	0.00428047
10	0.00442193
11	0.00431933
12	0.00396634
13	0.00426912
14	0.00429917
15	0.00398267

Tabla 10: Error cuadrático medio de los valores Tj

Los resultados de la Tabla 10 comprueban que los valores obtenidos por la FAM en FPGA son bastante aceptables, además de contar con un error mínimo comparado con los resultados de Matlab. La cantidad de bits seleccionados en la FPGA es considerable óptimo.

A continuación se muestra el tiempo que tarda *Quicksort* en organizar las 15 categorías Tj considerando el peor de los casos. De igual manera se ha registrado el tiempo en que tarda en realizarse la prueba de resonancia, es decir, encontrar la categoría Tj ganadora (ver Tabla 11).

Quicksort	**Prueba de resonancia**
10.43 [µs]	3996.48 [µs]

Tabla 11: Registro de tiempos de Quicksort y prueba de resonancia

En la Tabla 12 se registraron los porcentajes de recursos utilizados en la FPGA para diferentes cantidades de módulos FAM.

	Cantidad de módulos FAM			
Recursos de la FPGA	5	10	15	20
LUTs	6%	14%	31%	35%
LUTRAMs	9%	21%	33%	41%
Flip Flops	1%	3%	16%	15%
DPS	9%	18%	27%	36%

Tabla 12: Porcentaje de recursos utilizados en la FPGA

7 Conclusiones

La implementación de una RNA en *hardware* reconfigurable es novedoso. Se ha comprobado con el desarrollo de este trabajo que es posible y deseable tener una RNA en un FPGA, sin embargo, diferentes retos se tienen que superar para hacer esto posible.

Trabajar con datos en el mundo digital requiere tomar la decisión de cómo dichos datos serán representados en la máquina digital. Para este trabajo la representación en punto fijo fue la mejor opción debido al menor uso de recursos comparado con punto flotante, además de conocer el rango de valores con los que se trabajará. El uso de punto fijo requiere de una interpretación, un acuerdo entre dispositivos para que puedan procesar la información correctamente. Llegar a este acuerdo entre el PS y PL resultó una dificultad pues mientras el primero trabaja con 32 bits, en la PL se trabajó con 64 bits. Fue necesario truncar los resultados con corrimientos en el PS, pero a pesar de realizar esto el error que se obtuvo es aceptable y no afectó los resultados de la FAM. La RNA Fuzzy ARTMAP demostró ser una sólida opción para la clasificación de diferentes objetos. Aunque la clasificación de los objetos de prueba fue exitosa, es importante reconocer que es posible que la Fuzzy ARTMAP no clasifique correctamente como se expone en su primer artículo (S Grossberg et al., 1992), por lo que resultaría útil realizar más pruebas de su comportamiento. Si se desea aumentar las posibilidades de que la Fuzzy ARTMAP catalogue correctamente, es importante crear un algoritmo que generé un vector descriptivo con características lo suficientemente diferentes entre cada objeto a clasificar.

Dividir las tareas que harían la PL y PS se realizó en la etapa de planeación con el motivo de poder tener la mayor cantidad de módulos FAM en la PL. Es por ello que las tareas de ordenamiento y prueba de resonancia fueron realizadas en el PS. Sería posible implementar un algoritmo de ordenamiento en la PL al costo de menores recursos para módulos FAM.

Mantener en sincronía el PS y PL fue un reto puesto que era necesario coordinar todos los módulos FAM para que procesaran la información al mismo tiempo. Si los datos no se encontraban en sincronía, se corría el riesgo de que estuvieran desorganizados, provocando un error en los cálculos de la FAM. La solución a este problema fue tener un registro en el PS que indicara a la PL la dirección de memoria de una ROM, donde cada dato será guardado. Esta solución a pesar de ser efectiva, tuvo como consecuencia el uso de más recursos en la PL, por lo que una mejora se podría realizar haciendo un *streaming* de los datos directamente a los módulos FAM y así eliminar el uso de las memorias ROM.

A pesar de la reducción de módulos FAM por uso de ROMs y el truncamiento de número de bits de los datos, los resultados fueron exitosos. La implementación tiene el potencial de clasificar diferentes objetos de estudio en una pequeña tarjeta de *hardware* reconfigurable, por lo que su aplicación en espacios reducidos, como en celdas de manufactura, es atractivo y a un precio asequible. Realizando mejoras en el diseño sería posible incrementar en gran cantidad el número de objetos que pueden ser clasificados, por lo que continuar su desarrollo contribuiría al área de clasificadores en las máquinas de aprendizaje.

8 Referencias

Alfke, P. (2007). *20 Years of FPGA Evolution*. Xilinx.

Auria, L., & Moro. (2008). *Support Vector Machines (SMV) as a Technique for Solvency Analysis*.

AXI Reference Guide. (2011). Xilinx.

Baptista, D., Abreu, S., & Travieso-Gónzalez, C. (2016). *Hardware implementation of an artificial neural network model to predict the energy production of a photovoltaic system*. Microprocessors and Microsystems.

Beheshti, D. (2015). *Fixed Point Performance of Interpolation/Extrapolation Algorithms for Resource Constrained Wireless Sensors*. New York Institute of Technology.

Benefits of Using Fixed-Point Hardware. (2017). Mathworks.com. https://www.mathworks.com/help/fixedpoint/gs/benefits-of-fixed-point-hardware.html

Busque, M., & Parizeau. (1997). *A Comparison of Fuzzy ARTMAP and Multilayer Perceptron for Handwritten Digit Recognition*. Laboratoire de Vision et Système Numériques.

Cabrera, M., Juárez, I., Cabrera, R., & Castuera, J. (2005). *Machine Vision Approach for Robotic Assembly*.

Finnerty, A., & Ratigner, H. (2017). *Reduce Power and Cost by Converting from Floating Point to Fixed Point*. Xilinx.

Griffin. (2014). *Designing a Custom AXI-lite Slave Peripheral*. SILICA.

Grossberg, S, Markuzon, N., Carpenter, G., Reynolds, J., & Rosen, D. (1992). *Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps*. IEEE TRANSACTIONS ON NEURAL NETWORKS.

Grossberg, Stephen, & Carpenter, G. (1998). *Adaptative Resonance Theory*.

Hopfield, J. (1982). *Neural networks and physical systems with emergent collective computational abilities*. Proceedings of the national academy of sciences.

Koyuncu, ˙Ismail, Pehlivan, I., & Alçın, M. (2016). Hardware design and implementation of a novel

ANN-based chaotic generator in FPGA. *Department of Electric-Energy, Anadolu University, Department of Electrical-Electronics Engineering, Department of Control and Automation,*.

Krips, M., Lammert, T., & Kummert, A. (2002). FPGA Implementation of a Neural Network for a Real-Time Hand Tracking System. *Department of Electrical and Information Engineering*.

Kuon, I., Tessier, R., & Rose, J. (2007). *FPGA Architecture: Survey and Challenges*. Now Publishers.

Lojek, B. (2010). *History of semiconductor engineering*. Springer.

Lomas Barrie, V. M. (2016). *Aprendizaje y reconocimiento invariante de objetos en ensamble con robots empleando redes neuronales artificiales implementadas en FPGA*.

McCulloch, W., & Pitts, W. (1943). *A logical calculus of the ideas immanent in nervous activity.Bulletin of Mathematical Biology.*

Mijwel, M. (2019). *Artificla Neural Networks Advantages and Disadvantages*.

Mitra, S., & Chattopadhyay, P. (2016). *Challenges in Implementation of ANN in Embedded System*. International Conference on Electrical, Electronics, and Optimization Techniques.

Muthuramalingam, A., Anitha, D., & Himavathi, S. (2007). Feedforward Neural Network Implementation in FPGA Using Layer Multiplexing for Effective Resource Utilization. *IEEE TRANSACTIONS ON NEURAL NETWORKS*.

Paindavoine, M., & Yang, F. (2003). Implementation of an RBF neural network on embedded systems: Real-time face tracking and identity verification. *IEEE TRANSACTIONS ON NEURAL NETWORKS*.

Raeisi, R., & Kabir, A. (2006). *Implementation of Artificial Neural Network on FPGA*.

Rosenblatt, F. (1958). *A logical calculus of the ideas immanent in nervous activity*. Psychological Review.

Trimberger, S. (2015). *Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology*.

Wang, L.-T., Chang, Y.-W., & Cheng, K.-T. (2009). *Electronic Design Automation*. Morgan Kaufmann.

Weitzenfeld, A., Arbib, M., & Alexander, A. (2002). *The Neural Simulation Language: A system for*

Brain Modeling.

Yahiaoui, R., Alilat, F., & Loumi, S. (2017). *Parallelization of Fuzzy ARTMAP Architecture on FPGA: Multispectral Classification of ALSAT-2A Images*. IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS.

8.1 Figuras y gráficas

Grossberg, S., Markuzon, N., Carpenter, G., Reynolds, J., & Rosen, D. (1992).Representación de los pesos Fuzzy ART.[Figura 2.3,Figura 2.4]. *Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional.*

Trimberger, S. (2015). Comparación de precios, velocidad,capacidad y energía en los FPGAs de Xilinx a través de las decadas [Gráfica 1].*A Retrospective on the First Thirty Years of FPGA Technology.*.

Xilinx. Tarjeta de desarrollo Zybo Zynq-7000 ARM/FPGA SoC [Figura 4.3,Tabla 5]. Recuperado de: https://store.digilentinc.com/zybo-zynq-7000-arm-fpga-soc-trainer-board/

Valvano, J. (2015). Arquitectura harvard de un microcontrolador ARM Cortex-M [Figura 4.5]. *Embedded Systems: Introducción a los microcontroladores ARM Cortex-M.*

Digilent. (2017). Arquitectura Zynq AP SoC [Figura 4.4].Recuperado de: *Zybo FPGA Board Reference Manual*

Fallahlalehzari,F. (Desconocido). Application Processing Unit Structure.[Figura 4.6]. Recuperado de: https://www.aldec.com/en/company/blog/144—introduction-to-zynq-architecture