UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
INGENIERÍA ELÉCTRICA-SISTEMAS ELECTRÓNICOS

APRENDIZAJE Y RECONOCIMIENTO INVARIANTE DE OBJETOS EN ENSAMBLE CON ROBOTS EMPLEANDO REDES NEURONALES ARTIFICIALES IMPLEMENTADAS CON FPGA

TESIS
QUE PARA OPTAR POR EL GRADO DE:
DOCTOR EN INGENIERÍA

PRESENTA:
VÍCTOR MANUEL LOMAS BARRIÉ

TUTOR PRINCIPAL
DR. JUAN MARIO PEÑA CABRERA, IIMAS
COMITÉ TUTOR
DR. HECTOR BENÍTEZ PÉREZ, IIMAS
DR. ISMAEL LÓPEZ JUÁREZ, CINVESTAV-SALTILLO

CIUDAD UNIVERSITARIA, CD. MX. OCTUBRE 2016

JURADO ASIGNADO:

Presidente: DR. FERNANDO ARÁMBULA COSÍO.

Secretario: DR. HÉCTOR BENÍTEZ PÉREZ.

Vocal: DR. JUAN MARIO PEÑA CABRERA.

1er. Suplente: DR. BORIS ESCALANTE RAMÍREZ.

2do. Suplente: DR. FABIÁN DEMETRIO GARCÍA NOCETTI.

Lugar o lugares donde se realizó la tesis: Ciudad de México, México

TUTOR DE TESIS:

DR. JUAN MARIO PEÑA CABRERA

_____

FIRMA

(Segunda hoja)

Dedico éste trabajo a mi amada esposa.

Agradezco al pueblo de México que a través de CONACYT y la CEP me proporcionaron recursos económicos para mi formación doctoral.

Agradezco a mis padres y hermanos.

Agradezco todo el apoyo brindado por el Dr. Juan Mario Peña Cabrera.

Índice general

Índice de figuras

Listado de códigos

Índice de tablas

Resumen

Los sistemas embebidos tienen como finalidad conjuntar tecnologías de electrónica, cómputo y telecomunicaciones en componentes de tamaño reducido y bajo consumo de potencia eléctrica, para resolver problemas en tiempo real. Esto quiere decir que la respuesta de un sistema embebido busca responder lo más rápido posible al cambio de estado de sus variables de entrada.

Por otro lado, los sistemas de visión para el reconocimiento de objetos implementan una gran variedad de técnicas, algoritmos y métodos. En algunos casos es necesario un poder de cómputo considerable para llevarlos a cabo en el menor tiempo posible. Otros algoritmos son eficaces para realizarse en pequeñas computadoras, sin embargo, no son tan robustos como se desea. Existe una relación proporcional entre el tamaño de la computadora (dimensiones, peso, consumo de potencia, etc) y el poder de cómputo necesario para resolver un algoritmo.

El sistema embebido (SE) presentado en este trabajo utiliza un algoritmo para el reconocimiento invariable de objetos en una línea de producción, haciendo uso de tecnologías de hardware reconfigurable para la obtención de video así como la abstracción de formas que culminan con el reconocimiento de un objeto.

El reconocimiento está basado en un método conocido como BOF (*Boundary Object Function*, Función de Frontera de un Objeto), que consiste en obtener de una imagen binaria un vector de las distancias entre el centroide y el borde del objeto. Dicha representación discreta de la forma de ese objeto es aprendida por una red neuronal artificial. El reconocimiento consiste en utilizar la red neuronal artificial como un clasificador de objetos. El modelo de red neuronal artificial está basado en la Teoría de Resonancia Adaptativa, en particular en una red FUZZY ART MAP. Éste método de reconocimiento de objetos es invariante a la rotación, traslación y escalamiento.

El SE contiene una Máquina Digital de Reconocimiento de Objetos (MDRO), la cual realiza las tareas mencionadas anteriormente; el diseño y arquitectura de dicha máquina es una aportación original del autor. La cual representa una mejora al algoritmo y ejecución para el reconocimiento de objetos en una computadora digital.

Abstract

The embedded systems joins to the electronic, computing and telecommunications technologies in such a small components and low electric power consumption, in order to solve real times problems. It means that the embedded system output must be as fast as the input variables change.

On the other hand, the computer vision systems for object recognition result in endless techniques, algorithms and methods. In some cases considerable computing power is needed to execute them in less time. Another algorithms are good enough to fit in small computers nevertheless the robustness is not desirable as should be. There is a inverse relationship between the computer size (height, width, deep, weight, electric power consumption, etc.) and the computing power needed to solve the algorithm.

The embedded system presented in this thesis, works with an algorithm for invariant objects recognition in a production line, using reconfigurable hardware techniques for video capture, abstraction form and the recognition of the object.

The recognition is based on the BOF (Boundary Object Function) method, it consists to obtain the border-centroid vectors from a binary image. This discrete representation of the object shape is learned by a artificial neural network (ANN). Thus, the recognition process is a objects sorter. The ANN model is based on Adaptive Resonance Theory, the Fuzzy ARTMAP ANN. This object recognition method is invariant to rotation, translation and scaling.

The embedded system contains a Digital Object Recognition Machine (DORM), which performs the tasks mentioned above; the design and architecture of the machine is an original contribution of the author. Which represents an improvement to the algorithm object recognition running on a digital computer.

Capítulo 1

Introducción

En el Departamento de Electrónica y Automatización del Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS), UNAM, se trabaja en una línea de investigación referente a la manufactura inteligente. Un aspecto fundamental en ésta área, es el o lo sistema(s) cognitivos para el reconocimiento de piezas u objetos que se van a manipular, maquinar, seleccionar o bien ensamblar dentro de una celda de manufactura.

Un método para la identificación de partes a través de un sensor de video y una computadora, es el *Boundary Object Function* (BOF, Función de frontera de un objeto) - Fuzzy ARTMAP [Peña-Cabrera, 2005]. En el cual se captura un cuadro, se procesa, se extrae un descripto único y posteriormente una Red Neuronal Artificial (RNA), previamente entrenada, discrimina un ganador de un conjunto de descriptores.

Aún cuando dicho método es robusto y eficiente, su implementación en computadoras de propósito general hacen de él un algoritmo lento. Sobre todo cuando la RNA debe elegir a un ganador dentro del descriptor único de varios objetos. El trabajo que a continuación se presenta, es una mejora utilizando arquitecturas de cómputo reconfigurable.

1.1. Objetivo

El objetivo es el diseño y desarrollo de una arquitectura de una Máquina Digital de Reconocimiento de Objetos que contribuya a mejorar la eficiencia del algoritmo BOF-Fuzzy ARTMAP. Dicho modelo sirve, entre otras aplicaciones, para el reconocimiento visual de piezas en una celda de manufactura.

El procedimiento consiste en la extracción de un descriptor único de una pieza de manufactura, su BOF, a partir de una imagen digital. Posteriormente se realiza su clasificación usando una RNA, en particular la RNA Fuzzy ARTMAP [Grossberg, 1992]. La pieza es utilizada en una celda de manufactura para el ensamblado con robots. La cual tiene características muy particulares en las cuales el modelo es muy apto para poderla reconocer.

La Máquina Digital de Reconocimiento de Objetos (MDRO) forma parte de un sistema embebido que digitaliza una escena, extrae características de los objetos presentes y los reconoce a partir de conocimiento previo. Por lo tanto, éstas funcionalidades están integradas en un solo chip (SystemOnChip) el cual forma parte de la componente cognitiva de una celda de manufactura en una línea de producción.

La eficiencia del sistema se comparará en términos de la velocidad de procesamiento y los recursos requeridos entre el sistema embebido (SE) y una computadora de propósito general (CPG).

1.2. Hipótesis

Capturar, preprocesar, filtrar y procesar datos de video digital se pueden realizar de una manera más eficiente en un máquina digital *ad hoc* que en una computadora de propósito general (CPG). En particular, la obtención del BOF de un objeto digitalizado y posteriormente su clasificación a través de una RNA FUZZY ARTMAP puede ser más eficiente en una arquitectura basada en cómputo reconfigurable, que en una arquitectura de computo del tipo Von Neumann o Harvard.

1.3. Contribución

La principal contribución es la arquitectura de una máquina digital para el reconocimiento de objetos, inmersa en un sistema embebido. Su diseño, desarrollo, implementación y pruebas son una innovación y constituye un aporte al conocimiento. Esto es en términos de operaciones con datos digitales para el procesamiento de video en tiempo real en una celda de manufactura. Tal y como se describe en detalle más adelante.

La principal contribución consiste en el diseño, desarrollo, implementación y pruebas de la arquitectura de una máquina digital, inmersa en un sistema embebido, para el reconoci-

miento de objetos. Dicha arquitectura es una innovación y constituye un aporte al conocimiento en términos de operaciones con datos digitales para el procesamiento de video en tiempo real en una celda de manufactura. Como se describe en detalle más adelante.

El sistema es capaz de procesar video, a una velocidad de 24 FPS (*Frames per second*), en el cual se obtiene un descriptor único de un objeto digitalizado y su clasificación dentro de varias categorías, de manera continua y sin demoras o retrasos de tiempo que hagan perder uno o varios cuadros, contrario a lo que ocurre en otro tipo de implementaciones.

En términos específicos se mencionan a continuación aquellas técnicas y métodos que se innovaron en el contexto de este trabajo, para la solución en diferentes etapas del procesamiento de los datos.

La primera mejora sobre el algoritmo original [Peña-Cabrera, 2005], es el uso de espacio de memoria para almacenar cada imagen que compone el video. Al ser un procesamiento en línea, no es necesario almacenar cada pixel en memoria del tipo SRAM. Pixel a pixel es procesado y posteriormente desechado. Solo es necesario almacenar la información binaria, de las dos primeras líneas del cuadro para rellenar el *buffer* de procesamiento. Además ese *buffer* se representa en un solo bit por cada pixel.

El extractor de borde permite el procesamiento de 9 pixeles simultáneamente, es decir en un ciclo de reloj. Además, en término de recursos utilizados, la representación de cada pixel se hace en un arreglo de registros del tipo std_logic_vector, que para términos prácticos, se puede considerar como la unidad mínima de información en una FPGA. En una CPG el almacenamiento de un pixel se hace en un registro de memoria de 64 bits (para sistemas de 64 bits), desperdiciando así casi el 99 % de recursos.

Otro punto a destacar es la generación del vector BOF, ya que a partir de dos módulos CORDIC, que calculan el módulo y el ángulo de cada vector frontera, simultáneamente se almacena y se ordena en una memoria BRAM. El ángulo es la dirección y el módulo es el dato. De manera que en un proceso muy simple se seleccionan determinados vectores hacia los puntos frontera sin importar el tamaño relativo del objeto digitalizado con el tamaño físico del cuadro.

En definitiva la velocidad de procesamiento del SE en comparación con la CPG es contundente, como se muestra en el capítulo de resultados. También hay características que solo puede proporcionar un SE sobre una CPG. Por mencionar algunas, se tiene que el consumo de potencia eléctrica evidentemente es menor en el SE. El espacio físico que ocupa un SE es notoriamente más reducido que el de una computadora personal, hecho que favorece la uti-

lización de este SE en una celda de manufactura. También el tema económico es un aspecto a considerar. El cálculo del costo entre una tecnología y otra podría resultar poco claro en términos de uso/precio. Una CPG puede usarse para reconocer objetos mientras realiza otro tipo de cálculos. En cambio el SE está limitado en sus funciones y capacidades. Sin embargo, el costo de una FPGA, como el utilizado para éste trabajo, es más competitivo que el de una CPG.

Una de las razones que motivaron para la realización de este trabajo, fue la capacidad de replicar el diseño tantas veces como pueda caber un componente en una FPGA. Por lo que la MDRO se diseñó para replicarla en la FPGA. Esto con el propósito de procesar segmentos del cuadro simultáneamente. Así, es posible tener un cierto número de MDRO procesando en paralelo en un mismo SE.

1.4. Justificación

En aplicaciones en tiempo real para el reconocimiento de objetos por medio de visión artificial, es necesario tener herramientas que proporcionen robustez en el algoritmo de detección, rapidez, menor consumo de potencia eléctrica, que ocupen el menor espacio posible, que sean ligeros y económicos. Por lo tanto, un sistema embebido conectado a un sensor de video que procese en tiempo real cada escena y reconozca objetos o piezas en una línea de producción, así como su posición, aún cuando varíe el escalamiento, la rotación y desplazamiento, es de gran utilidad para mejorar la producción en cualquier tipo de industria automatizada.

El cómputo de la BOF y RNA Fuzzy ARTMAP se ha implementado en computadoras de escritorio o portátiles haciendo uso de lenguajes de programación de alto nivel para un sistema operativo determinado y utilizando un procesador central de propósito general y con memoria de programa del tipo RAM. El sistema operativo es el encargado de administrar todos los recursos de una computadora. Y se quiera o no, el procesador es utilizado por este la mayor parte del tiempo, lo cual implica que si otro programa requiere hacer uso de dicho procesador tendrá que compartirlo con el sistema operativo, ya sea en intervalos de tiempo o en hilos. Esto sin lugar a dudas retrasa el tiempo de ejecución de un programa cualquiera.

Si se requiere que un sistema de reconocimiento de objetos de una celda de manufactura robotizada trabaje en tiempo real, es necesario poner atención en los detalles del uso de recursos de nuestro programa. Pues entre menos veces y menos tiempo sea interrumpido el

ciclo del proceso, más rápido se realizará el reconocimiento en línea.

Con la implementación de arquitecturas de hardware hechas a la medida, se puede llegar a soluciones verdaderamente eficientes. Es por eso que en este trabajo se plantea el uso de circuitos reconfigurables, FPGA (**Field Programmable Gate Array**), para su solución. Es decir, que con la implementación de la BOF y la RNA Fuzzy ARTMAP computadas directamente en una arquitectura de hardware *ad hoc* se omite el uso de un sistema operativo que consume tiempo y la estructura central de procesamiento, se convierte en múltiples instancias lógicas de cálculo tanto para el descriptor como el clasificador.

1.5. Alcances

En este trabajo describe el diseño, desarrollo y pruebas de una arquitectura en cómputo reconfigurables de un sistema embebido para extracción de características de video; por lo que a partir de la captura cruda de video de una escena dónde es necesario reconocer un objeto se tiene el diseño de un módulo digital que binariza la imagen, extrae el centroide del objeto, obtiene los puntos frontera y construye una función de datos discretos que posteriormente son reconocidos a través del clasificador de una RNA.

Así como en el planteamiento original del reconocedor se establecen limitaciones para el reconocimiento, aquí se aplican los mismos criterios. Se considera pues, que el centroide del objeto a reconocer está inmerso en su propia área. Existe una diferencia sustancial de contraste entre el fondo de la imagen y el objeto. La iluminación es homogénea sobre toda la superficie del objeto procurando evitar brillo o reflejos sobre el lente de la cámara.

Para éste diseño se parte del hecho de que el objeto a reconocer es único en la escena que captura la cámara. Es decir, no existen más objetos y por ende no se consideran traslapes entre objetos ni las fronteras se unen en ningún píxel. Sin embargo es conveniente aclarar, que ésta Máquina Digital de Reconocimiento de Objetos (MDRO) puede replicarse tantas veces como sea necesario, y sobre todo pueda contenerlas el dispositivo reconfigurable. Dada su estructura es posible realizar la identificación de varios objetos en la escena por medio de software o por hardware. Así será posible reconocerlos todos.

1.6. Metodología

La metodología parte del análisis del flujo de datos del método BOF-FuzzyARTMAP así como de los accesos a memoria cuando el algoritmo se ejecuta en una CPG. Una vez obtenidos las tendencias, se diseño la estrategia que posteriormente definió tanto la arquitectura particular de cada componente como la arquitectura de todo el sistema.

Al tratarse de un sistema de procesamiento en tiempo real, se puso énfasis en los puntos donde era prioritario tener disponibilidad de procesamiento, a fin de que no se constituyera un cuello de botella. Vino entonces la determinación de que partes del proceso debiesen ser instrumentados como máquinas digitales o que pudiesen resolverse de mejor manera en el software del SE. Aunque siempre fue prioritario, el dejar el procesado intensivo de datos, para la parte reconfigurable y en el SE ejecutar solo acciones de control, comunicación y configuración. Ya que precisamente es ahí, donde se buscaría que las facilidades de concurrencia o paralelismo, que otorga el hardware reconfigurable, atacara las partes exhaustivas del algoritmo.

A partir de bosquejos, diagramas de tiempo, diagramas de funcionalidad y pruebas de manejo de datos en lenguajes de alto nivel, se construyeron los primeros diseños de los componentes. Y a la par se fueron probando y depurando.

Cuando se tuvieron las características, requerimientos y arquitecturas de cada módulo se procedió a implementarlos de forma separada en lenguaje de descripción de hardware, particularmente en VHDL. Y haciendo uso de los simuladores de la herramienta del fabricante (Xilinx), se crearon bancos de prueba con un conjunto de datos virtuales, obtenidos a partir de imágenes reales procesadas en python.

Después de la depuración de cada módulo fue necesario la selección del hardware donde se implementaría el sistema de reconocimiento. La decisión estuvo basada en seleccionar plataformas de hardware que no fuera de alto costo, de reducido tamaño físico y que tuviera componentes de conectividad de uso común. El consumo de potencia eléctrica también fue un requisito para la selección del circuito integrado.

A partir de la selección del chip y su arquitectura, se analizaron las características y requerimientos del sistema embebido. Vino un estudio para determinar la mejor forma de integrar la MDRO con el procesador central y la unidad de memoria.

Posteriormente se realizó el encapsulamiento de toda la aplicación en el circuito integrado y se procedió a realizar las pruebas unitarias y globales del sistema para mejorar la eficiencia.

Además hubo un proceso para reducir el espacio de los componentes y la latencia entre módulos.

Al tratarse de una herramienta que su principal campo de acción es en tiempo real se probaron características relacionadas con el desempeño y recurso de hardware utilizados. Estas pruebas se hicieron teniendo como contraparte, algoritmos implementados en CPG utilizando lenguajes de programación de alto nivel. El algoritmo de la BOF con la RNA FuzzyARTMAP se programó y se obtuvieron tiempos de ejecución para los lenguajes de programación C++ y Matlab. Aún cuando Matlab no siempre administra los datos para obtener el mejor desempeño, sí tiene algunas funciones primitivas que han sido optimizadas incluso mejor que en C. Por lo tanto se considera una herramienta apropiada para al menos saber el comportamiento del algoritmo en otro lenguaje de programación.

Las pruebas se realizaron con segmentos del algoritmo donde existe independencia de datos, es decir que no es necesario el resultado de un proceso para continuar con el siguiente paso. Y es ahí donde se hizo la comparación.

1.7. Estructura de la tesis

El capítulo 2 aborda las investigaciones realizadas referentes al tema de reconocimiento de objetos utilizando inteligencia artificial; así como también los trabajos realizados con dispositivos reprogramables en la solución de problemas en tiempo real para la visión artificial.

De esta manera el Estado del Arte sirvió como guía en la estrategia y conveniencia de realizar esta investigación.

En el capítulo 3 se habla acerca del método utilizado en este trabajo para extraer un descriptor único de un objeto presente en una escena. Este descriptor único es conocido como Función de Frontera de un objeto, BOF. Como parte del desarrollo de ésta investigación ser realizaron varias implementaciones en un lenguaje de programación de alto nivel para comprender y analizar el algoritmo tanto para la obtención de la BOF como el entrenamiento y clasificación de una RNA Fuzzy ARTMAP. En particular se describe un algoritmo que presenta una optimización para CPG de la obtención de la frontera de un objeto. Así mismo, en este capítulo se habla de las características principales de las RNA y en particular de la utilizada en este trabajo, la Fuzzy ART MAP. Además de que se mencionan las razones por las cuales se utilizó en particular esa red. Por último, se describe el algoritmo de esa

red acompañada de una implementación generalizada en un dispositivo digital. Dado que la parte fundamental de la investigación e implementación es sobre un dispositivo de lógica reprogramable, también en este capítulo se describen características fundamentales de estos dispositivos y su evolución hasta lo que actualmente se conoce como FPGA. Su estructura y lógica interna son descritas con la finalidad de conducir al lector a un entendimiento de las limitaciones y ventajas que se tuvieron para convertir el algoritmo de la BOF y al RNA Fuzzy ARTMAP en una MDRO. Al final de éste capítulo se habla del diseño y la arquitectura de un sistema embebido.

En el capítulo 4 se hace una descripción detallada de la arquitectura de la MDRO y su implementación en VHDL. Comprende desde la obtención de la imagen por parte del sistema embebido hasta el diseño del clasificador Fuzzy ARTMAP, sin olvidar la arquitectura del cálculo de la BOF. También se incluyen en este capítulo los detalles del sistema embebido que enmarca la operación y control de la MDRO. Conceptos como el videocontrolador o el acceso directo a memoria son tratados de forma general.

En el capítulo 5 están las pruebas y resultados producto de la experimentación de este trabajo de investigación. Con tablas y gráficas se expresa descriptivamente indicadores de desempeño que comparan al sistema embebido desarrollado para esta tesis y dos algoritmos de dos diferentes lenguajes de programación ejecutándose en una CPG.

Por último, en el capítulo 6 se expresan las conclusiones primero de forma general, seguidas por una sección para cada etapa.

Capítulo 2

Estado del Arte

En los procesos de manufactura semiautomática o automática se requieren etapas de clasificación, supervisión, de control de calidad, de ensamblaje, de transporte o bien de embalaje. Si bien competir con la agudeza, precisión y buen criterio del ojo humano y su sistema neurológico es osado debido a la gran complejidad del tratamiento de la información que allí se procesa, cuando se tratan de altas velocidades de producción o incluso la carga de trabajo, conviene utilizar sistemas que por su propia cuenta tomen decisiones y así mejorar la producción en masa. [Herakovic, 2010]

No se trata de desplazar al ser humano por máquinas como parte del medio de producción, se trata de que la tecnología coadyuve a mejorar los procesos para bajar los costos y en consecuencia mayor gente pueda acceder a los satisfactores. De la misma forma ocurre con evitar riesgos innecesarios al humano en procesos productivos sumamente peligrosos.

Es por ello que los sistemas de visión artificial han y siguen siendo sumamente requeridos en la industrialización de procesos en donde anteriormente se realizaban de manera artesanal.

A continuación se presenta el Estado del Arte que llevaron a definir las guías de éste trabajo. La primera sección aborda conceptos para ubicar al lector en el campo de estudio. Posteriormente se presenta un estudio cuantitativo de la producción científica, a nivel mundial, de los temas circundantes al objeto del trabajo. Lo anterior es con el objetivo de justificar el interés de contribuir en específico, a ésta rama de la ingeniería.

Las últimas tres secciones de éste capítulo abordan conceptos, fundamentos y las tecnologías aplicadas que recientemente se han generado alrededor de éste tema.

2.1. Reconocimiento artificial de objetos

La visión es el proceso de descubrir que hay en el mundo y dónde se encuentra. Es el proceso de identificar imágenes en el mundo externo y obtener una descripción útil y con información relevante para el observador. [Marr, 1982]

El reconocimiento artificial de objetos es una amplia y compleja área de estudio. Por lo cual, es importante ubicar el campo de acción y las líneas de investigación vinculantes con el presente trabajo. Es por ello que se realizó un esquema donde se conectan los temas de estudio, las implementaciones y los conceptos englobados en el término *Machine Vision*. La figura 2.1 ilustra, en su cuadro superior, los principales temas de estudio que se requiere conjuntar para reconocer objetos. En el cuadro intermedio se presentan los diferentes dispositivos donde se han implementado soluciones para reconocer objetos. Por último, la parte inferior muestra algunas aplicaciones en la industria.

A partir de diferentes disciplinas de la ciencia (p. ej. física, neurobiología, matemáticas) se generan áreas de estudio referentes a la visión por computadora y el procesamiento de imágenes, solo por mencionar algunas. Para poder llevar a cabo, pruebas de concepto, implementaciones, aplicaciones, pruebas de desempeño, se utilizan dispositivos electrónicos (p. ej. computadoras personales, DSP, clusters, FPGA, etc.). Y que en la mayoría, son aplicados para solucionar problemas en la medicina, educación, entretenimiento o en la industria (p. ej. manufactura avanzada, ensamble con robots, inspección visual, etc.)

Es importante señalar que la presente investigación se realiza enfocada a la visión computarizada a través de una FPGA para aplicaciones en líneas de ensamble. Para mostrar la evolución que ha tenido el tema, a continuación se presentan datos bibliométricos de los temas que se consideraron más importantes para el presente estudio. Los estudios bibliométricos permiten visualizar el interés de publicar en revistas académicas sobre un tema y, a partir de los artículos publicados es posible ubicar el crecimiento o decrecimiento del tema y las principales áreas de conocimiento, instituciones, autores y países interesados en el tema. Para este estudio bibliométrico se decidió utilizar búsquedas combinadas que consideraran todas las formas de denominar los términos teóricos que conciernen a esta tesis, por lo que se realizaron las búsquedas de:

- *Robot Vision & Embedded System*
- *Pattern Recognition & Embedded System*

- *Neural Network & Embedded System*

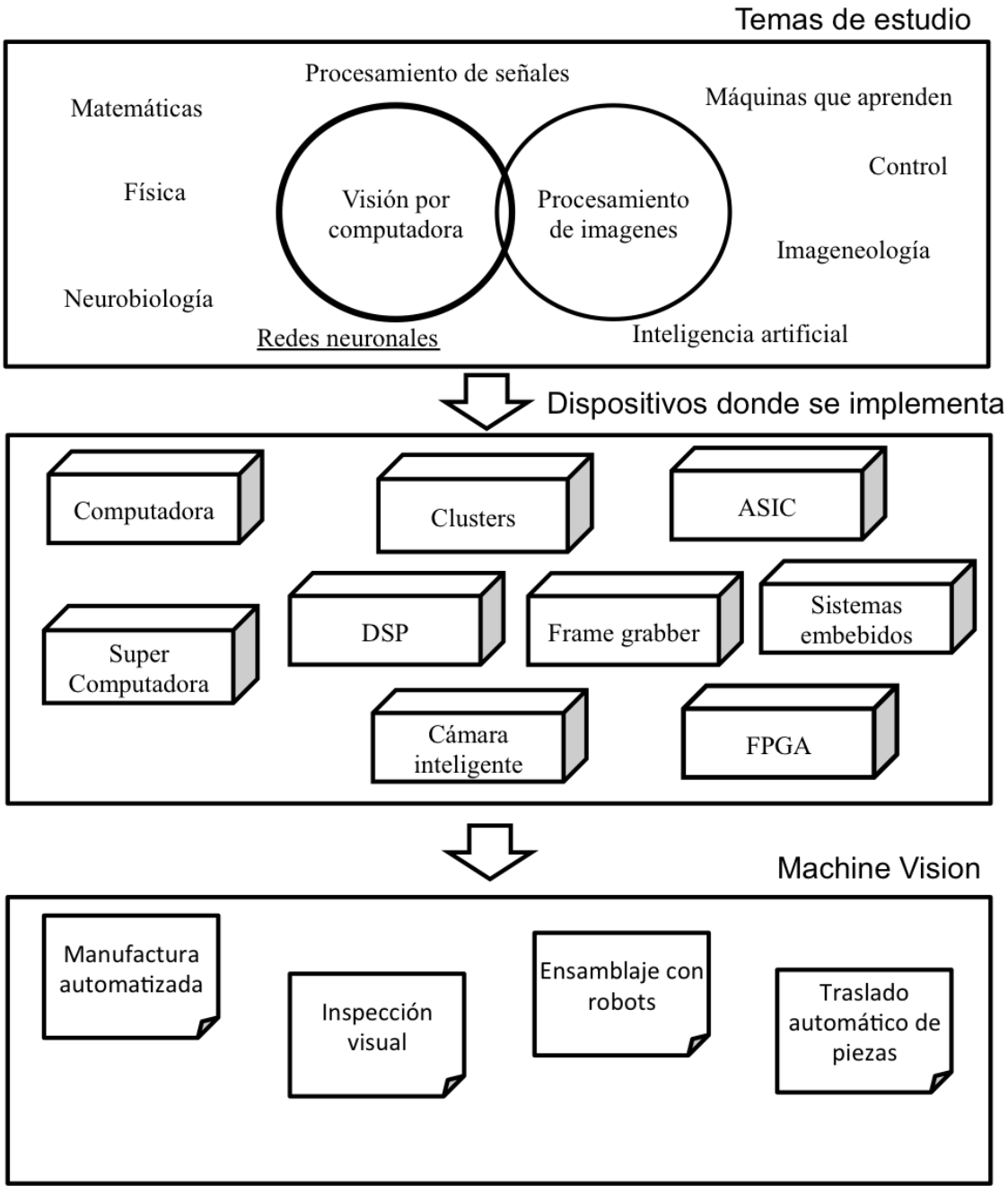

Figura 2.1: Cuadro de temas relacionados con el reconocimiento automático de objetos

Para obtener los resultados que aquí se presentan se utilizó la base de datos de información científica Web of Science, suministrado por el *Institute for Scientific Information* (ISI).

La búsqueda se realizó en Diciembre del 2015. En general, se encontraron los siguientes resultados, ver figura 2.1.

Tema	Robot Vision & Embedded System		Pattern Recognition & Embedded System		Neural Network & Embedded System	
Tipo de Artículos	Revista	Conferencia	Revista	Conferencia	Revista	Conferencia
Total artículos publicados	69	76	373	259	1,197	874

Tabla 2.1: Resumen de artículos encontrados.

En la tabla se muestra que el tema *Robot Vision & Embedded System* tiene una cantidad mayor de artículos en conferencia que en revista, aunque las cifras son muy parecidas. En contraste, en las otras dos búsquedas se detectan más artículos de revistas que de conferencias.

Al analizar la publicación anual de artículos en revistas a partir de 1994, se observa que el tema con más artículos a través del tiempo es *Neural Network & Embedded System*, publicando un promedio de 52 artículos por año. En segundo lugar con 16 artículos anuales en promedio, se muestra la publicación de artículos en el tema *Pattern Recognition & Embedded System*. Por ultimo, los temas menos referidos son *Robot Vision & Embedded System*, con un promedio de 3 artículos al año. Los tres temas muestran una tendencia creciente, lo cual refleja el interés de los investigadores por publicar sobre los temas (figura 2.2).

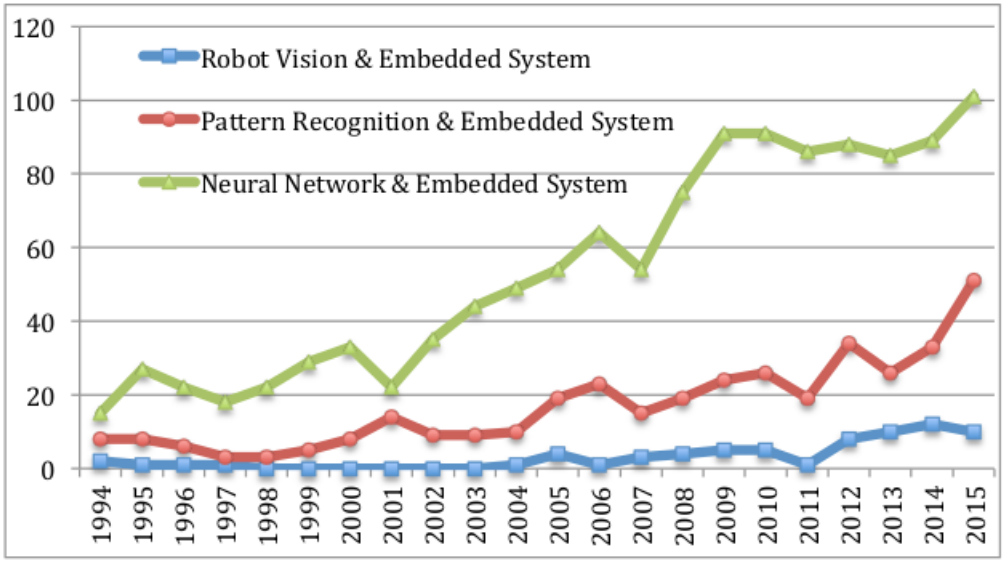

Figura 2.2: Gráfica de artículos encontrados.

Asimismo se identificaron las revistas y congresos en que se publican más artículos de cada tema En el caso de *Robot Vision & Embedded System*, se encontraron 120 revistas y congresos. En la figura 2.2 se presentan los 12 con más artículos publicados.

No	Revista o Congreso	Art
1	IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)	9
2	IEEE International Conference on Robotics and Automation (ICRA)	5
3	Computer Vision and Image Understanding	3
4	Robotics and Autonomous Systems	3
5	ACM Transactions on Embedded Computing Systems	2
6	Annals of Daaam for & Proceedings of the International Daaam Symposium Engineers	2
7	Autonomous Robots	2
8	Biological Cybernetics	2
9	IEEE Transactions on Automatic Control	2
10	Intelligent Robots and Computer Vision	2
11	International Journal of Advanced Manufacturing Technology	2
12	Sensors	2

Tabla 2.2: Artículos encontrados para *Robot Vision & Embedded System*.

En el caso de *Pattern Recognition & Embedded System*, se encontró un total de 491 revistas y congresos. En la tabla se presentan los que tuvieron más de 4 artículos en el periodo de 1994 a 2015. Destaca la revista *Pattern Recognition* con 23 artículos. Así mismo, sobresale

la presencia de revistas y congresos del Instituto de Ingeniería Eléctrica y Electrónica (IEEE) (ver figura 2.3).

No	Revista o Congreso	Art
1	Pattern Recognition	23
2	Neurocomputing	7
3	Pattern Recognition Letters	7
4	IEEE Engineering In Medicine and Biology Society	6
5	Sensors	6
6	IEEE Transactions on Pattern Analysis and Machine Intelligence	5
7	IEICE Transactions on Information and Systems	5
8	International Joint Conference on Neural Networks	5
9	Expert Systems With Applications	4
10	IEEE International Conference on Acoustics, Speech and Signal Processing	4
11	IEEE Transactions on Intelligent Transportation Systems	4
12	Journal of Intelligent & Fuzzy Systems	4
13	Plos One	4

Tabla 2.3: Artículos encontrados para *Pattern Recognition & Embedded System*.

En tanto que para *Neural Network & Embedded System*, se encontraron más de 1,100 revistas y congresos. En la figura 2.4 se muestran los nombres de revistas y congresos en que se publicaron más de 10 artículos, destacando la revista: Expert Systems With Applications con 33 artículos.

No	Revistas y Congresos	Art
1	Expert Systems With Applications	33
2	Neural Networks	28
3	IEEE Transactions on Neural Networks	27
4	International Joint Conference on Neural Networks	25
5	Neurocomputing	24
6	IEEE International Joint Conference on Neural Network Proceedings	16
7	IEEE Transactions on Industrial Electronics	15
8	IEEE International Conference on Systems, Man and Cybernetics	14
9	Artificial Neural Networks and Machine Learning - ICANN	12
10	IEEE International Conference on Fuzzy Systems	12
11	IEEE Transactions on Systems Man and Cybernetics Part B-Cybernetics	12
12	International Joint Conference on Neural Networks, Proceedings	12
13	Sensors	11

Tabla 2.4: Artículos encontrados para Neural Network & Embedded System.

También es interesante conocer los países a los que pertenecen los autores e instituciones

con mayor número de publicaciones (ver figura 2.5). En la tabla se muestra los primeros 15 países con más artículos publicados en revistas y congresos. En el caso de *Robot Vision & Embedded System*, Estados Unidos ha publicado poco más de la cuarta parte de los artículo, seguido de Francia con casi el 7 % del total de artículos. Del mismo modo, en el caso de *Pattern Recognition & Embedded System*, Estados Unidos encabeza la lista con poco menos del 19 %, seguido por China con el 12.5 % de los artículos. En tanto que en el tema de *Pattern Recognition & Embedded System*, las instituciones Chinas ocupan el primer lugar con el 16.5 % de los artículos, mientras que Estados Unidos ha publicado casi el 15 %.

No	País	Robot Vision & Embedded System	Pattern Recognition & Embedded System	Pattern Recognition & Embedded System
1	USA	25.5%	18.8%	14.8%
2	France	6.9%	2.4%	2.6%
3	Germany	6.2%	3.3%	3.0%
4	China	5.5%	12.5%	16.5%
5	Italy	5.5%	6.0%	4.2%
6	Taiwan	5.5%	5.7%	7.9%
7	Spain	4.8%	3.8%	3.8%
8	Switzerland	3.4%	1.7%	0.9%
9	South Korea	3.4%	1.3%	2.7%
10	Canada	2.8%	5.7%	2.5%
11	Brazil	2.8%	1.7%	1.9%
12	Mexico	2.8%	1.4%	2.1%
13	Austria	2.8%	0.9%	0.5%
14	Japan	2.1%	4.4%	5.5%
15	England	2.1%	3.2%	4.3%

Tabla 2.5: Artículos por país.

Al buscar las instituciones que más publican se observa que en el caso de *Robot Vision & Embedded System*, la Ecole Polytechnic Federal Lausanne y National Taiwan Univ Science & Technology lideran la lista de artículos publicados. En la búsqueda *Pattern Recognition & Embedded System* los autores de Chinese Academy of Sciences y University of Milan han publicado 7 artículos respectivamente. Por último, *Neural Network & Embedded System*, la National Cheng Kung University publicó 24 artículos entre 1994 y 2015.

Como se ha mostrado, existe mucho trabajo de investigación en reconocimiento de objetos, formas, procesamiento digital de imágenes y sistemas comerciales para capturar y desplegar información visual de escenas, así como sistemas integrales para aplicaciones de visión en industrias de compañías tan grandes como National Instruments (National Instruments,

2016 y 2015) , Data Translation, Inc (Data Translations, 2015), Matrox Electronic Systems (Matrox, 2016), hardware de adquisición de imágenes muy variados desde arreglos lineales de fotodiodos, *scanners*, *frame grabbers*, hasta sofisticadas cámaras que forman imágenes en tres dimensiones como las de Daitron Inc (Daitron, 2015), Genex Technologies Inc (Genex, 2015), Leutron (Leutron, 2015), Newton Research Labs (Newton, 2014) y tarjetas digitalizadoras muy sofisticadas como las de BitFlow (BitFlow2014), Engineering Design Team, Inc (EDT, 2015) y ADLINK Technology Inc (ADLINK, 2015), que ofrecen conceptos implementados de sistemas en tiempo real, esto es adquisición o reacción a eventos sin pérdida de información tan rápido como lo hace un problema real. Conceptos de estabilidad, manejo de información en ambientes hostiles, arquitecturas abiertas y sistemas interactivos conocidos como *mouse-driving image processing systems*, como el DT/Image-Pro de Data Translation que puede hacer convoluciones espaciales, filtrados, *mask edge detection*, análisis estadísticos de puntos, líneas, histogramas, mediciones de área, *frame averaging*, operaciones lógicas y aritméticas, rotaciones, mejoramiento de imágenes, etc. Métodos de procesamiento digital de imágenes para lo que se conoce como: *Edge Detection*, *Silohuette Recognition*, *Image Enhancemente*, *Feature Descriptor*, *Filtering*, *Binarization*, Segmentation, 3-D Imaging Systems, etc. y desarrollo de algoritmos para reconocimiento de patrones, de formas, análisis de escenas, etc., son temas muy ricos en contribuciones ya logradas por excelentes autores como Rosenfeld [22], Bribiesca [Bribiesca, 1980], Guzmán [Guzman, 1969], González [Gonzalez, 1977], y Capson [Capson, 2002].

2.2. Redes Neuronales

Las RNA son una forma de emular las características propias de los seres humanos en cuanto a memorizar y asociar hechos, situación, que resulta del todo aplicable cuando se tienen problemas que no pueden expresarse a través de un algoritmo y en donde se tiene una característica común: la experiencia. Tal situación lleva al ser humano a resolver un sin número de problemas con la experiencia acumulada, así que parece ser viable llegar a la solución de problemas con las características planteadas diseñando y construyendo sistemas capaces de reproducir esta característica humana, pudiendo entonces, hablar de modelos artificiales simplificados para adquirir conocimiento a través de la experiencia.

Dentro de estas consideraciones, nos presentamos ante el reto de diseñar sistemas que auxilien al hombre en la fabricación y ensamble de otros sistemas utilizando diversas partes

especializadas. En este caso, nos encontramos con dos problemas a resolver: el reconocimiento invariante de objetos en una línea de ensamble y el aprendizaje incremental de ellos de manera similar a como lo realiza el ser humano basándose en la experiencia y asociación de hechos.

Existen excelentes y novedosos trabajos al respecto, pero mayoritariamente como esfuerzos puntuales y muy objetivos para la solución de un problema específico, tal es el caso de las redes neuronales para el tratamiento de la información cuya unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso humano "la neurona". El cerebro humano contiene aproximadamente 12 billones de neuronas, cada una de ellas con múltiples entradas y una salida, pudiéndose así interconectar y procesar la información para resolver prácticamente cualquier problema. Uno de los primeros modelos matemáticos de una neurona fue el propuesto por McCulloch y Pitts en 1943 . [Mcculloch, 1943, 115] en donde se sientan las bases de las redes neuronales actuales y se reafirma que una red neuronal a diferencia de una máquina algorítmica clásica no se programa, si no que, "se educa"mediante una regla de aprendizaje que se puede producir de tres maneras: aprendizaje supervisado, aprendizaje no supervisado y el aprendizaje autosupervisado. En 1958 Rosenblatt presentó el Perceptron [Rosenblatt, 1958], red neuronal con aprendizaje supervisado basado en una modificación de la regla de aprendizaje propuesta por Hebb [Hebb, 1949]. En los años 60's se propusieron otros modelos denominados Adaline y Madeline, que utilizan una regla de aprendizaje denominada Delta, en donde se considera una función de error [Widrow, 1960]. La era moderna de las redes neuronales artificiales (ANN) surge con la técnica de aprendizaje de propagación hacia atrás Back Propagation [Bryson, 1969]. Otros modelos que permiten un aprendizaje no supervisado como el mapa-auto-organizable (SOM) de Kohonen [Kohonen, 1990] o los basados en la teoría de resonancia adaptiva (ART) de Grossberg y Carpenter [Carpenter, 2003] y los modelos de control de motor de Bullock, Gaudiano y Grossberg [Gaudiano, 1992] y [Bullock, 1991] y los clasificadores de Hopefield y Hamming [Hopfield, 1982], tienen gran aceptación actualmente para muchas aplicaciones.

La investigación doctoral apunta hacia la utilización de redes neuronales artificiales ART para desarrollar la capacidad de aprendizaje incremental para el sistema de visión para ensamble con robots. Dado que éste modelo de redes neuronales, ha de mostrado tener características muy favorecedoras para el tipo de aplicación en el que se van a emplear. Ya que no tienen pérdida catastrófica de información al seguir aprendiendo nuevas categorías y convergen en un menor número de épocas, entre otras importantes características.

ART son las siglas para *Adaptive Resonance Theory* introducida por Stephen Grossberg

en 1976 , siendo actualmente en la Universidad de Boston en The Center for Adaptive Systems & The Department of Cognitive and Neural System el lugar en donde este tipo de redes se ha investigado mayoritariamente [11]. Existen diferentes tipos de redes ART como las clásicas:ART-1, ART-2, ART-2A, ARTMAP, Fuzzy ART, Distributed ART y ARTMAP (dARTy DARTMAP), Gaussian ARTMAP, arbo ART y otras, autores como: Carpenter [Carpenter, 1991, 4], y otros se dedican a estos temas . También existen simuladores como: ART-1 simulator, ART Gallery, SNNS, fuzzy ARTMAP para MATLAB y otros [Hung, 1995], [Nageswaran, 2009] y [Anagnostopoulos, 2002].

2.3. Redes Neuronales implementadas en FPGA

Las RNA habitualmente son implementadas en CPG a través de lenguajes de programación de alto nivel, como puede ser MATLAB, C o Java. Por ejemplo, en cuanto a trabajos desarrollados que involucren el reconocimiento de objetos para manufactura utilizando la extracción de la BOF como descriptor único y utilizando la RNA Fuzzy ARTMAP como clasificador se tiene el trabajo de [Peña-Cabrera, 2005, p. 189] que fue el primero en aparecer en la literatura y que se obtuvo un reconocimiento de 100 % de los objetos utilizados. Con un tiempo de reconocimiento promedio de 1:50.6 minutos, incluyendo la extracción del descriptor único y la clasificación de la RNA FUZZY ARTMAP. Para ese caso se utilizó una CPG Pentium III a 800 [Mhz] con 512 [MB] en RAM. Se empleó una cámara PULNIX 6710 blanco y negro a una resolución de 640x480 pixeles. Otro caso descrito en [Reyes-Acosta, 2009] trata de la concatenación de la BOF y la Forma de la Sombra (SFS, *Shape from Shading*) para generar un vector característico de un objeto que es aprendido y clasificado por una RNA Fuzzy ARTMAP. En éste caso se utilizó una cámara Af602c con un sensor de 1/2 pulgada a una resolución de 656x490 en una CPG Intel Xeon a 1.86 [GHz] con 3 [GB] de RAM. El algoritmo fue implementado en Visual C++.NET. Por último se tiene el trabajo desarrollado en [Francisco, 2011] donde se utilizó una cámara Bumblebee a 640x480 pixeles para el reconocimiento de un objeto usando vectores de información de 2D (contorno), 2.5D (indicador de forma local) y 3D (profundidad a partir de estéreo). Todo a partir de la extracción de la BOF, SFS e información proveniente del histograma. En éstos últimos no se pone énfasis en el tiempo transcurrido de reconocimiento puesto que lo importante en ese caso es ampliar el rango tipo de objetos. Igualmente fue desarrollado utilizando Visual C++.NET.

La utilización de lenguajes de alto nivel para la implementación de RNA ejecutándose

en CPG proporciona una manera simple para hacer pruebas de concepto o mejoras en el desempeño de reconocimiento de objetos, como se vió en el párrafo anterior. Sin embargo, al compilar dicho algoritmo en una CPG se pierde la posibilidad de procesar en paralelo la RNA lo cual repercute en su desempeño. Por lo que el reto consiste en implementar una RNA en una estructura que permita el paralelismo. Bien podrían usarse opciones tecnológicas como el multiprocesamiento en computadoras tipo cluster, aprovechar la tecnología multi hilo de los procesadores modernos, la utilización de máquinas vectoriales o masivamente paralelas [Martín, 2007].

Gracias al surgimiento de la tecnología VLSI (*Very Large Scale Integration*) en los ochentas, se retomo el uso de las RNA debido al bajo costo en el desarrollo de dispositivos de cómputo y a que posibilitó su realización directa en el hardware. Este último consiste en construir físicamente un sistema cuya arquitectura refleje en mayor o menor medida la estructura de la RNA. Sin embargo esta estructura debe de ser capaz de albergar un gran número de conexiones y un flujo de datos elevados por lo que limita el tamaño de la RNA que pueda ser implementada.

Hay dos formas de realizar esta implementación a través de dispositivos ASIC (*Application Specific Integrated Circuit*) o FPGA. Estos últimos han sido ampliamente usados debido al rápido desarrollo de prototipos. Además de que presentan rendimientos similares a los de las supercomputadoras a un precio significativamente inferior., ya que su estructura interna del sistema de procesamiento se puede ajustar a cada instante al problema que está resolviendo.

Su estructura interna permite realizar multiplicaciones a gran velocidad y con una longitud de palabra únicamente limitada por el mismo programador o por el tamaño físico de su arquitectura. Esta característica hace que los FPGA sean tomados en cuenta para realizar inteligencia artificial a través de RNA.

La investigación doctoral que se propone, puede contribuir al desarrollo y experiencia en la utilización de éstos métodos para beneficios significativos en la industria, de hecho, el propio programa que respalda el otorgamiento del grado esperado plantea la necesaria interacción con un problema real con interés real de alguna industria, por lo que que la realización de un sistema como el propuesto justifica y contribuye a lograr estas metas.

2.4. Procesamiento entre hardware dedicado y una CPG

Es por demás interesante y objeto de varias investigaciones el comparar la eficiencia de un mismo algoritmo implementado en hardware o en software. Si bien la velocidad de los procesadores para CPG se incrementa año con año y la capacidad de almacenamiento RAM aumenta constantemente las soluciones en hardware a veces resultan lo suficientemente competitivas y en varios casos son evidentemente mas veloces que su versión implementada en software.

Que tanto es mas rápida la solución planteada en hardware reconfigurable o en software es una pregunta difícil de contestar. Existen varios aspectos a tomar en cuenta. Por ejemplo un indicador de desempeño para una solución basada en software seria la velocidad del procesador. Dato que puede ser fácilmente extraído de las hojas de datos publicadas por el fabricante. Sin embargo podría no ser suficiente información. Dado que el recurso principal no es utilizado al 100 % por el algoritmo. Se debe compartir con el sistema operativo. Por lo que es necesario experimentar ejecutar varias veces el algoritmo para sacar una estadística del tiempo promedio que es necesario para calcular cierto proceso. Empíricamente se puede determinar el número de iteraciones necesarias para conocer el tiempo promedio. En los experimentos descritos en [Vanhoucke, 2011] se utilizan 5 iteraciones para cada prueba

Para el caso de un FPGA no son necesarias las consideraciones anteriores, en [Cullinan, 2012, p. 26] se aclara que el tiempo consumido en un proceso será el mismo si así se repitieran muchas veces. Dado que una vez iniciado el proceso no sufre interrupciones repentinas. Basta con leer la máxima frecuencia del FPGA publicada por el fabricante para saber a que velocidad correra el proceso.

En [Grozea, 2010] se realiza una comparación para resolver problemas de ordenamiento de datos y extracción de N-gramas entre un FPGA (Tarjeta ML507, Virtex-5 a 0.55 [GHz]), un GPU (*Graphics Processing Unit*) (2 tarjetas Nvidia Quadro FX 5600, con 1.5 [GB] de RAM a una velocidad de 1.35 [GHz]) y una CPG con multi núcleo (Dell Precision T7400 con 2 procesadores Xeon 5472 quad-core, a una velocidad de 3 [GHz] y 16 [GB] de RAM) Los resultados obtenidos fueron: para el FPGA se tiene una velocidad de procesamiento de 1.875 [Gbit/s] y comunicación [1 Gbit/s]; el CPG con multi núcleo se tienen 2 [Gbit/s] para ambas cosas, procesamiento y comunicación; y para el GPU se alcanzó una velocidad de 8 [Mbit/s]. La competencia está entre el FPGA y el CPG multi núcleo, que al final obtiene el mejor rendimiento. Sin embargo no es por esa razón que concluyen que es mejor la implementación con CPG sobre la hecha con el FPGA. Aducen que la complejidad para implementar

el algoritmo en el FPGA es mayor que en CPG. Tardaron 2 días en sintetizar el proyecto para el FPGA, en cambio para el CPG se codificó en un lenguaje de programación de alto nivel usando la biblioteca OpenMP que es una implementación para distribuir el cómputo en los núcleos de una CPG. Al final un modelo reciente de FPGA tendrá mayor velocidad que el usado y superará a los resultado obtenidos con el CPG, aun cuando la velocidad del CPG aumente. Debido a la forma en que lo realiza.

El resultado del problema planteado en [Wray, 2010, p. 2] resulto ser 377 veces mas rápida la implementación en FPGA que en software. Usando un FPGA de la familia Virtex 5 de Xilinx. Una ventaja sustancial de las soluciones planteadas con FPGA es la propiedad de reconfigurarse parcialmente, capacidad que es aprovechada en esa investigación. Ya que las posibilidades de auto adaptarse, de acuerdo a la evolución del proceso, resultan muy benéficas para el procesado de datos.

La comparación de desempeño no solo ha sido entre soluciones de hardware y software, también se han producido trabajos entre FPGA y soluciones en ASIC [Kuon, 2007]. No solo se compara la velocidad de procesamiento, también hay resultados con respecto a la potencia eléctrica consumida, espacio dentro del chip y recursos utilizados.

No siempre el mejor desempeño lo tiene el dispositivo que realice más rápido el procesamiento del algoritmo, también importa el tiempo y recursos que son necesarios para hacer llegar los datos al núcleo del proceso. Por ejemplo en [Cullinan, 2012, p. 54] se calcula una FFT (*Fast Fourier transform*) usando una FPGA, un CPU de una CPG y en una GPU. El código implementado en la GPU es 311 veces rápido que usando FPGA, y 3,351 veces mas rápido que en una CPG. Sin embargo el tiempo para transferir los datos a la GPU es de 69 [ms] por lo que la solución en la FPGA queda en primer lugar al ser 26.67 veces más rápido que la GPU y 10.76 veces más rápida que en la GPU.

Claro, una GPU depende de una CPG para inicializarla, configurarla, conectarla puesto que es vista como un periférico por parte del sistema operativo. En la FPGA pueden implementarse soluciones como un periférico más de una CPG o funcionar independientemente para el caso de un sistema embebido.

Capítulo 3

Descriptor, Redes Neuronales y FPGA

En éste capítulo se presenta el Marco Teórico de los temas fundamentales en los cuales se basó el trabajo de investigación. Ellos son el Descriptor Único, la RNA FUZZY ARTMAP y los circuitos de lógica reprogramable FPGA.

En la sección Descriptor Único se detalla el procedimiento para obtener la BOF de un objeto en una escena, a partir de una imagen obtenida por un sensor de video a color. Temas como la obtención del umbral, la segmentación, cálculo del centroide y de puntos frontera son abordados.

Posteriormente, se describen las RNA, sus características y se hace énfasis en la Teoría de Resonancia Adaptativa, En particular el modelo de la RNA FUZZY ARTMAP, tanto para el entrenamiento como la clasificación de categorías. Después se abordan las implicaciones de modelar dicha RNA en una FPGA.

Al final de capítulo se mencionan las características de las FPGA y la forma de programarlas.

3.1. Descriptor único

En el reconocimiento artificial de patrones u objetos existe una gran variedad de algoritmos basados en diferencias de color, la frecuencia de este, vecindad de pixeles, la distribución de color, extracción de características, agrupamiento de pixeles o propiedades geométricas solo por mencionar algunos. El método de extracción de la BOF está basado en este último criterio, el de propiedades geométricas.

Al obtener algunas propiedades geométricas del objeto a reconocer se puede conformar un vector que describa su comportamiento geométrico. Al utilizar un método de inteligencia artificial para clasificar la información, es necesario que ese vector, al cual llamaremos vector descriptivo sea único. Es decir que exista una relación biunívoca entre el vector descriptivo de un objeto y el objeto mismo.

También cabe mencionar que las propiedades de invariancia a la rotación, traslación y escalamiento son fundamentales para un sistema robusto de reconocimiento de objetos. Esta invariancia se refiere a que no importa si el objeto inicialmente aprendido está rotado θ grados con respecto al que se quiere reconocer. O bien que se encuentre en otra parte de la imagen en la que al principio estaba. Y sobre todo que no importe si el acercamiento de la cámara al objeto se ve alterado desde la posición inicial.

En esta sección se describe la forma de obtención de la Función de Frontera de un Objeto.

3.1.1. Función de Frontera de un Objeto (BOF)

Se trata de un método para calcular una función discreta de todas las distancias, y sus correspondientes ángulos, del centroide de un objeto hacia todos y cada uno de sus puntos frontera.

Cabe aclarar, que cuando se refiere a un objeto en realidad se habla de la proyección de ese objeto en un plano de 2D. Por lo que, para abreviar, se considera al objeto como dicha proyección en 2D en el resto de éste trabajo.

Es así, un método que convierte las características de forma de un objeto en un vector, la BOF, que lo represente [Peña-Cabrera, 2005, p. 119]. En la Figura 3.1 se tiene un cuadrado, y al interior se muestran diversos vectores del centroide al borde.

Para cada vector del centroide al borde, se calcula su distancia. Dicha distancia es almacenada en un vector, a este vector se le llama BOF. El número de elementos de éste vector está en función del número de distancias calculadas desde el centroide hacia la frontera del objeto. Entre más vectores del centroide al borde existan, más características tendrá la función.

De tal manera que el cálculo de la BOF para otro tipo de objetos, e incluso una combinación de ellos quedaría como se muestra en la figura 3.2. Se observa, por ejemplo, que para el caso de un círculo, su BOF es una constante de valor igual al radio de éste.

Para obtener la BOF de un objeto se siguen los siguientes pasos:

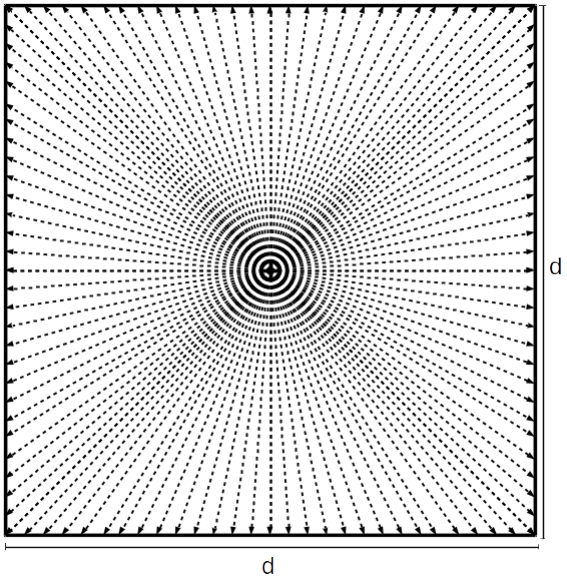

Figura 3.1: Vectores de distancia del centroide al borde del objeto

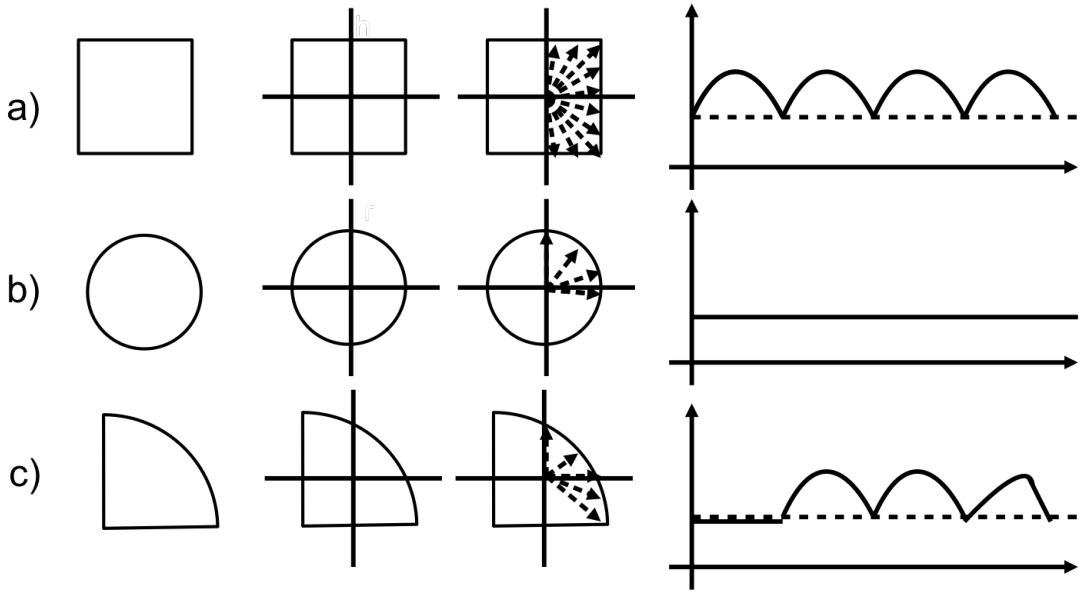

Figura 3.2: BOF de varias figuras geométricas

1. Adquirir la imagen del objeto y su espacio de trabajo.
2. Convertir la imagen a escala de grises [0-255].
3. Calcular el histograma de la imagen.
4. Obtener el mejor umbral que separe el objeto del fondo de la escena.

5. Convertir la imagen original en una imagen binaria.
6. Calcular el centroide del objeto.
7. Obtener los puntos frontera del objeto.
8. Calcular la distancia euclidiana de todos los puntos frontera con el centroide.
9. Normalizar todas las distancias. Y conformar el vector BOF.

3.1.2. Obtención de la imagen

La forma de adquirir una imagen digital de una escena y los objetos que la componen se utilizan dispositivos análogo-digitales que convierte un haz de luz en un valor numérico. Dichos sensores, dependiendo de su construcción, son lineales o de matriz, a color o a escala de grises. Los usados para este trabajo son sensores de matriz y a color.

Si bien el sensor utilizado entrega 3 componentes de color RGB (rojo, verde y azul) se hace una conversión a escala de grises de 8 bits. Ya que el alcance de ésta investigación se basó en la obtención de la imagen en escala de grises.

Existen diversas técnicas de conversión de color a escala de grises. Unas enfocadas a una mejor ecualización, otras a resaltar aspectos de una banda determinada, o bien la oclusión de una o dos bandas, etc. En trabajos anteriores a este, se ha observado que una ponderación de la representación de la luz para cada banda es conveniente. Por lo que se realiza un promedio de las tres bandas y se convierte a escala de grises.

3.1.3. Histograma, umbral e imagen binaria

Para poder determinar la BOF de un objeto es necesario que la imagen del objeto sea binaria, es decir que el objeto esté representado por un ’1’ lógico y el fondo por un ’0’ lógico.

Un método para convertir una imagen a color o en escala de grises a una imagen binaria, es determinar un umbral que delimite el valor de cada pixel de acuerdo a su intensidad de tono. El valor de cada pixel mayor a ese umbral será sustituido por un ’0’ lógico y si es menor entonces será sustituido por un ’1’ lógico.

Para determinar el valor del umbral es necesario conocer el histograma de la imagen.

Histograma

El histograma es el conteo de todos los pixeles de la imagen que corresponden a un mismo nivel. Si la imagen está en escala de grises, y cada pixel está representado por **n** diferentes tonos, entonces existirán **n** niveles. Por otro lado si la imagen es a color y ésta está representada por 3 bandas, existirá un histograma por cada banda de **n** niveles.

Umbral

Determinar el mejor umbral representa el poder segmentar lo mejor posible uno o varios objetos de su fondo. Es una tarea por demás estudiada y que su propia determinación representa un análisis más profundo, dado que para determinadas condiciones hay algoritmos con mejores resultados que otros en menoscabo de su tiempo de ejecución.

Dado que se trata de un procesamiento en tiempo real, es necesario determinar el umbral en el menor tiempo posible. Ya que hay encadenados otros procesos a la espera del valor del umbral de ese cuadro.

Un método que resulta favorable para la determinación del umbral en un sistema de visión en tiempo real, es el de encontrar el menor valor de pixel después de haber encontrado el máximo. En la figura 3.3 se muestra un histograma de una imagen con 255 niveles de grises. El lóbulo más grande representan el conteo de pixeles correspondientes al fondo de la imagen. Por el contrario, el lóbulo menor son pixeles que representan al objeto.

De acuerdo con la definición anterior el primer valor de pixel mínimo, después de haber encontrado el máximo, se considera un umbral apropiado para separar el objeto del fondo de la imagen. Aunque en la figura anterior se muestra un área que pudiese abarcar pixeles que formen parte del objeto y del fondo, la correcta iluminación permite disminuir dicha area y obtener el mejor umbral.

Imagen binaria

Una vez obtenido el umbral se aplica la siguiente regla de correspondencia 3.1 a la imagen en escala de grises I_{eg} para obtener una imagen binaria I_b.

$$I_b(i,j) = \begin{cases} 1 & si\ I_{eg}(i,j) > umbral \\ 0 & de\ otra\ forma \end{cases} \tag{3.1}$$

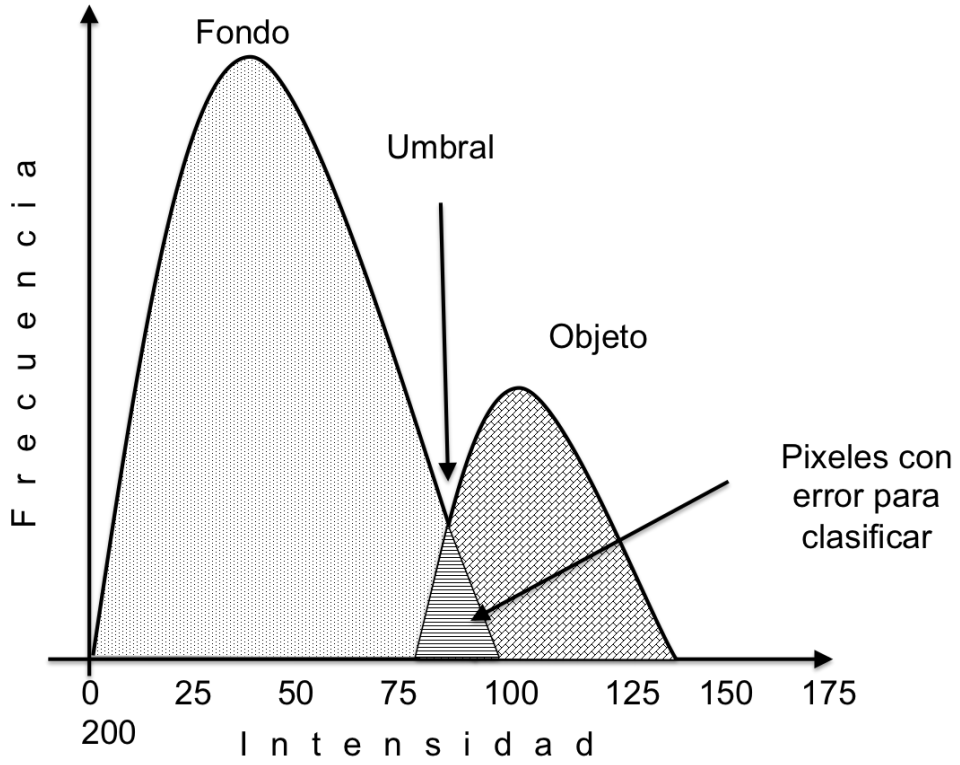

Figura 3.3: Determinación del mejor umbral en un histograma

Esto es, si los pixeles con coordenadas i, j son menores al umbral, el valor de los pixeles será 1, de lo contrario serán 0.

3.1.4. Cálculo del centroide

El centroide (X_c, Y_c) de una forma es el punto donde se encuentra concentrada toda la masa [Horn, 1986]. Por lo tanto todas las fuerzas que actúan sobre el objeto confluyen en dicho punto. Geométricamente, y en particular para el método BOF, el centroide representa el punto de partida para calcular los vectores de distancia desde este punto a la frontera del objeto. Una forma de obtener el centroide de un objeto de dos dimensiones en el dominio discreto es la siguiente:

$$X_c = \frac{1}{A} \sum_i \sum_j i \tag{3.2}$$

$$Y_c = \frac{1}{A} \sum_i \sum_j j \tag{3.3}$$

dónde A es el área del objeto proyectado en un plano 2D e i y j son las coordenadas discretas dónde tiene presencia el objeto. En términos de una imagen binaria las ecuaciones 3.2 y 3.3 se pueden expresar de ésta forma:

$$X_c = \frac{\sum A_x}{\sum A} \tag{3.4}$$

$$Y_c = \frac{\sum A_y}{\sum A} \tag{3.5}$$

dónde A_x y A_y son los puntos pertenecientes al objeto a lo largo de los ejes x y y respectivamente.

3.1.5. Obtención del borde de un objeto

Una forma de obtener los pixeles que conforman los puntos frontera de un objeto es a partir de la transformación de una imagen binaria a una matriz de factores de pesos [Peña-Cabrera, 2005, p. 102]. La cual consiste en desplazar de izquierda a derecha y de arriba hacia abajo, sobre todos y cada uno de los pixeles que integran una imagen, un operador matricial, que está conformado por la ecuación 3.6

$$I_{MP}(i,j) = \sum_{k=-1}^{1} \sum_{l=-1}^{1} I_b(i+l, j+k) \tag{3.6}$$

Dónde: $0 \leq I_{MP}(i,j) \leq 9$;

$\forall i, j, k, l \in \mathbb{N}$.

Con ésta transformación se obtiene que los pixeles que no forman parte del objeto seguirán siendo 0 y los que sí forman parte o al menos están cerca del objeto son mayores a 0. Entre más grande sea el valor se garantiza que el pixel es parte del objeto. Sólo habrá que definir un umbral para discriminar entre auténticos puntos frontera y puntos cercanos a la frontera pero que no pertenecen al objeto.

3.1.6. Conformación de la BOF

Una vez obtenido el centroide y los puntos frontera, se calculan las distancias correspondientes. La ecuación 3.7 representa la distancia euclidiana entre dos puntos de un plano cartesiano [Francisco, 2011, p. 34].

$$D_n = \sqrt{(C_x - x)^2 + (C_y - y)^2} \tag{3.7}$$

El número de distancias es igual al número de puntos frontera. Sin embargo, debido al número excesivo de vectores distancia, se puede reducir a cierta proporción. Por ejemplo, calcular las distancias del centroide a aquellos puntos frontera que tengan un ángulo de 2 grados .

3.1.7. Método alterno para obtener la BOF

Con la intención de evitar el proceso de binarizar una imagen, se desarrolló un método alterno para calcular la BOF de un objeto. El algoritmo fue publicado en una revista científica arbitrada durante el doctorado [Peña-Cabrera et al., 2013], que trajo consigo la publicación de algunas memorias en congresos internacionales [Peña-Cabrera et al., 2012] y [Lomas-Barrie et al., 2015].

El método consiste en trazar varias rectas que usen al centroide como pivote. Para cada una de esas rectas, se explora el valor de los pixeles que atraviesa dicha recta. Al tener el valor de cada pixel este es comparado con un umbral y se determina si es punto frontera o no.

En este procedimiento sólo se requiere conocer el umbral adecuado para distinguir entre el fondo y el objeto en la escena, lo que implica que no es necesario binarizar la imagen, pues se realiza la comparación del umbral con los pixeles que son seleccionados por una función. Al no binarizar la imagen, el proceso consumo menor tiempo.

A continuación se describe el algoritmo:

1. Se obtiene el centroide de la imagen.

2. A partir del centroide se sigue la trayectoria que define una recta $y = \tan(\varphi n)\, x + b_0$; donde φn es el ángulo que varía de θ a n, y b_0 es la ordenada al origen que está en función del ángulo φn, y y x son los índices de la matriz de la imagen y representan la

posición de un píxel, ver figura 3.4

3. De acuerdo a dicha trayectoria, cada píxel es evaluado y si su valor es de 255 se regresa a la posición $x - 1$ y se vuelve a calcular y, dicha coordenada será el pixel frontera.
4. Se calcula la distancia de ese punto frontera con el centroide. Con la ecuación 3.7
5. La distancia D_n se almacena en el vector $BOF[n]$.
6. Se incrementa φn con una precisión ρ.
7. Se repiten los pasos del 2 al 6 hasta que φn alcanza el valor de n. Ver figura 3.5 y figura 3.6

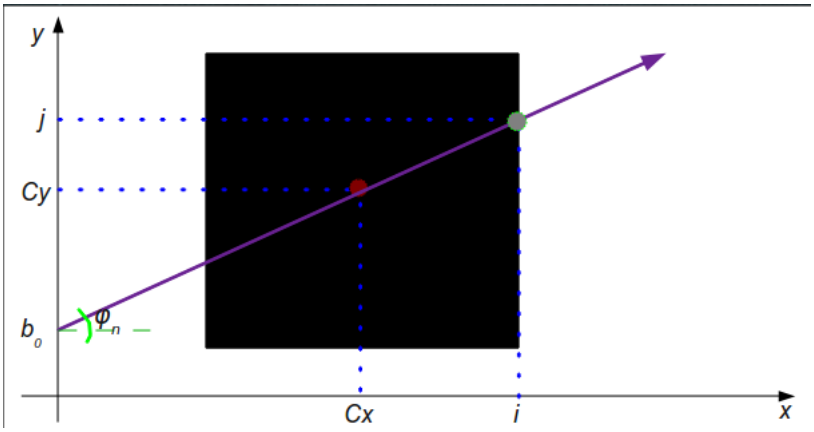

Figura 3.4: Primer rastreo de un punto frontera

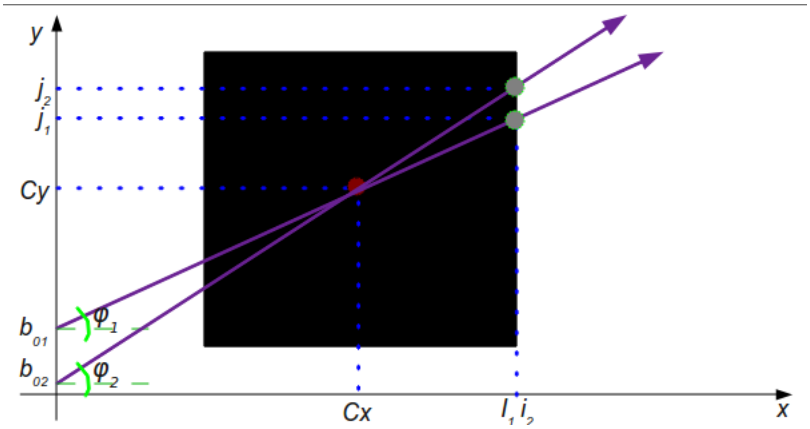

Figura 3.5: Segundo rastreo de un punto frontera

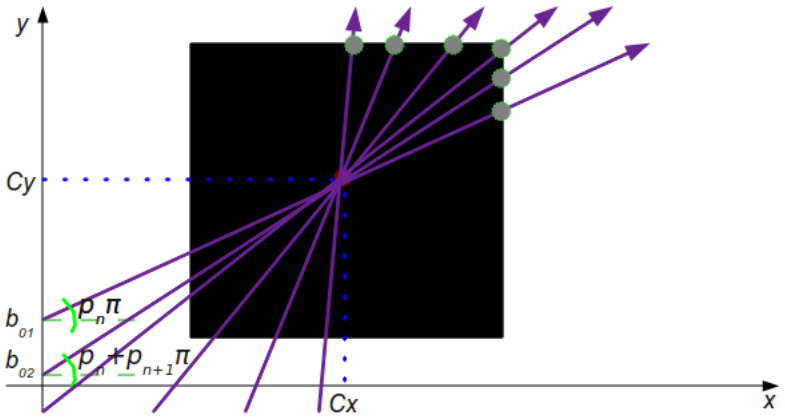

Figura 3.6: N-esimo rastreo de un punto frontera

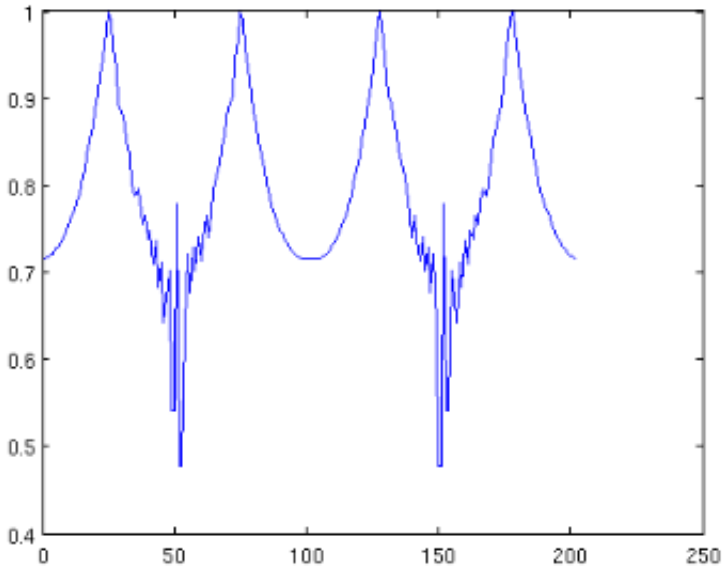

Figura 3.7: Gráfica de distancias del centroide a puntos frontera normalizada

Al final las distancias del objeto al centroide se pueden observar en la Figura 3.7.

Una desventaja de éste método sobre el descrito en la sección anterior, radica en el hecho de que es necesario conocer el centroide, además de que en términos de manejo de memoria, se necesita almacenar la imagen en su totalidad para poder mapear el rastreo de cada punto frontera; lo cual resulta contraproducente para la optimización del algoritmo en FPGA. Sin embargo, sería más rápido si se pudiesen solventar los recursos de memoria. Esto es debido a que sólo se consulta un menor número de celdas de memoria en comparación al primero, en el que toda la imagen debe ser procesada.

3.2. Redes Neuronales Artificiales

En ésta sección se detalla la parte cognitiva del SE y la MDRO, que está basada en el modelo de una RNA. Al principio se da un panorama de las RNA, después acerca de las redes ART y al final de su implementación en FPGA.

El cerebro humano, y el sistema neurobiológico de muchos seres vivos, son sistemas biológicos que organizan, aprenden, aplican, imaginan e intuyen el conocimiento.

No sólo se trata de tomar decisiones para discriminar un objeto u otro cuando se le presentan; o tomar una decisión de un conjunto de opciones, sino también es capaz de experimentar, actualizar la información vieja u obsoleta y aplicar criterios a través de los cuales se busca el camino más óptimo.

Lograr simular un sistema tan complejo como ese, ha sido de gran interés durante ya varias décadas; investigadores de varios países han tratado de simularlos. La sóla idea de tener una máquina que pueda tener las mismas deducciones tal cual las tiene un perro, un gato o un ser humano sería algo fascinante para el mundo de la automatización.

No sólo expertos en ingeniería han tratado de copiar su funcionamiento, disciplinas como la filosofía, psicología o neurología están interesados en obtener modelos similares, factibles y sobre todo confiables de un sistema neurológico.

Las RNA son un conjunto de tecnologías asociadas a modelos de redes neuronales biológicas pero implementadas como ecuaciones, algoritmos y esquemas. Éstas pueden recrearse en sistemas de cómputo o bien electrónicos. Las soluciones van desde sistemas analógicos o digitales, hasta una combinación de ambas, las híbridas.

La forma de conseguir que las RNA funcionen se basan en principios básicos de sus símiles, las biológicas. La estructura fundamental es la misma. En la figura 3.8 se presenta un modelo de RNA muy simple.

Las entradas (x_n) y la salida (y) pueden ser continuas o discretas. Los pesos son componentes que establecen su valor de acuerdo al resultado esperado, es decir, se calculan de acuerdo a un entrenamiento en función de un determinado resultado. Una función de propagación, que resulta ser la suma de todas las entradas (x_n) multiplicadas por los pesos (w_n) es activada o no por la función de activación. Por lo tanto, se tiene la ecuación 3.8

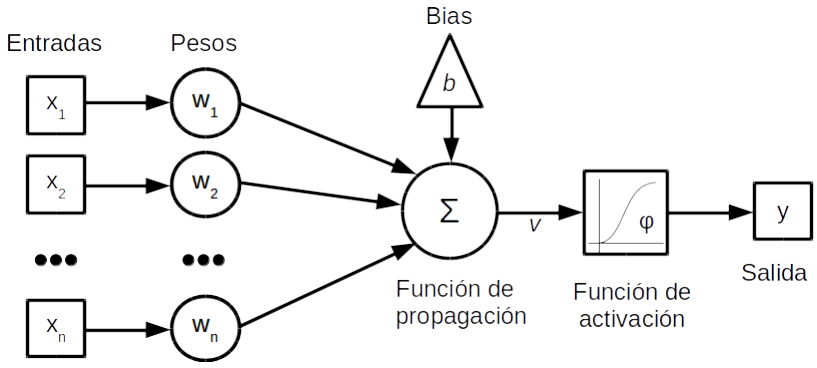

Figura 3.8: Red neuronal artificial básica.

$$y = \phi(v) = b + \sum_{i=1}^{n} x_i w_i \tag{3.8}$$

El sesgo b tiene como objetivo determinar un valor de salida para cuando $x_1 = x_2 = ... = x_n = 0$. Cuando una neurona artificial es insuficiente para procesar más entradas que a su vez están influenciadas por mayor numero de pesos, entonces se conforma una red de neuronas. Dicha red esta conformada por capas. Existen capas de entrada de donde proviene cada señal a procesar, capas ocultas y finalmente una capa de salida. La figura 3.9 muestra una representación de una capa de entrada, una oculta y una capa de salida.

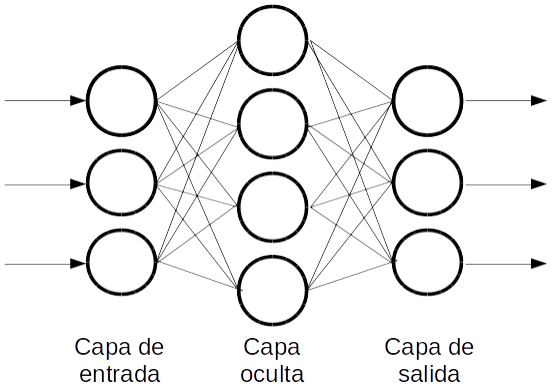

Figura 3.9: Red Neuronal Artificial.

Existe una gran variedad de modelos de RNA, desde la más básica (el perceptrón), pasando por los Mapas Auto organizados (SOM) hasta la familia de redes basadas en la Teoría de Resonancia Adaptativa, sólo por mencionar algunas.

El fundamento de este trabajo está enfocado en el modelado en hardware reconfigurable

de un tipo en especial ART conocido como Fuzzy ART MAP.

3.2.1. Redes Neuronales del tipo ART

El término **resonancia**, se refiere al llamado **estado de resonancia** de la red, en el que una categoría de vector prototipo se aproxima lo suficiente con el vector de entrada en cuestión para que el sistema no genere una señal de *reset* a la siguiente capa neuronal.

La familia de redes ART es basta, y han sido generadas a apartir de la evolución de los algortimos y las necesidades. En la tabla comparativa 3.2.1 se muestran miembros y características de la familia ART. Principalmente ART es concebida como una RNA de aprendizaje no supervisado y se aprecia que hasta la familia Fuzzy ART seguía siendo de la misma manera. Sin embargo, cuando el concepto de ARTMAP fue desarrollado, los posteriores miembros debiesen implementar el aprendizaje supervisado. Además, cabe desatacar que las aplicaciones de las diferente RNA ART inlfuyen en el tipo de dato de entrada, es decir algunas son del tipo binario y en otras continuo. Para el caso de la Fuzzy ARTMAP se establece que tiene una etrada híbrida.

RNA	Descripción	Tipo de entradas	Supervisada
ART-1	Red ART	Binarias	No
ART-2	Mayor capacidad	Continua	No
ART-2A	Mayor capacidad.	Binarios/Continua	No
ART-3	Basada en transmisores químicos	Continua	No
Fuzzy ART	Implementación de Lógica Difusa sobre ART	Continua	No
ARTMAP	Combinación de ART-1 y ART-2	Binarias	Sí
Fuzzy ARTMAP	Lógica difusa a ARTMAP	Binarias/continuas	Sí

Tabla 3.1: Comparación familia ART

3.2.2. Fuzzy ARTMAP

La red Fuzzy ARTMAP tiene características muy peculiares por las cuales se escogió utilizarla para el reconocimiento e objetos. En general, al tratarse de una combinación de

las redes ART-1 y ART2 con la componente de la teoría de la Lógica Difusa, la hace sumamente competitiva con respecto a otros modelos de RNA. Entre sus características podemos mencionar que la Fuzzy ARTMAP es:

- Robustas y tolerantes a fallas,
- es flexible,
- su información es difusa,
- la entrada puede ser incompleta y
- sobre todo, es áltamente paralela.

Los conceptos de plasticidad y estabilidad son ampliamente descriptivos de esta red, para ejemplificarlo se presenta la siguiente analogía: Una persona puede aprender nuevos conceptos (objetos, personas, ideas, entidades vivientes, etc.) pero no olvida las anteriores. Afortunadamente Fuzzy ARTMAP combina aceptablemente ambos parámetros. Que a decir de otros modelos RNA, la Fuzzy ARTMAP tiene en particular que su entrenamiento es en línea, converge en un menor número de épocas de entrenamiento y que es una versión de la arquitectura ART1 pero supervisada.

En la figura 3.10 se muestra el esquema de una RNA Fuzzy ARTMAP. La red ART_a opera sobre las entradas, y ART_b sobre las salidas. y en el papa de campo F^{ab} se vincula las entradas con las salidas. El parámetro de vigilancia rho determina si se tolera la diferencia entre los datos de entrada y los patrones almacenados, esto con el propósito de no generar más categorías innecesarias.

Entrenamiento

Acontinuación se describe el procedimiento para el entrenamiento de una Fuzzy ARTMAP.

1. Inicializar todos los pesos
$$w = 0 \tag{3.9}$$
2. Obtener el complemento de todos los elementos de la entradas
$$I = (a, a_c) = (a_1, a_2, \cdots, a_M, a_{c1}, a_{c2}, \cdots, a_{cM}) \tag{3.10}$$

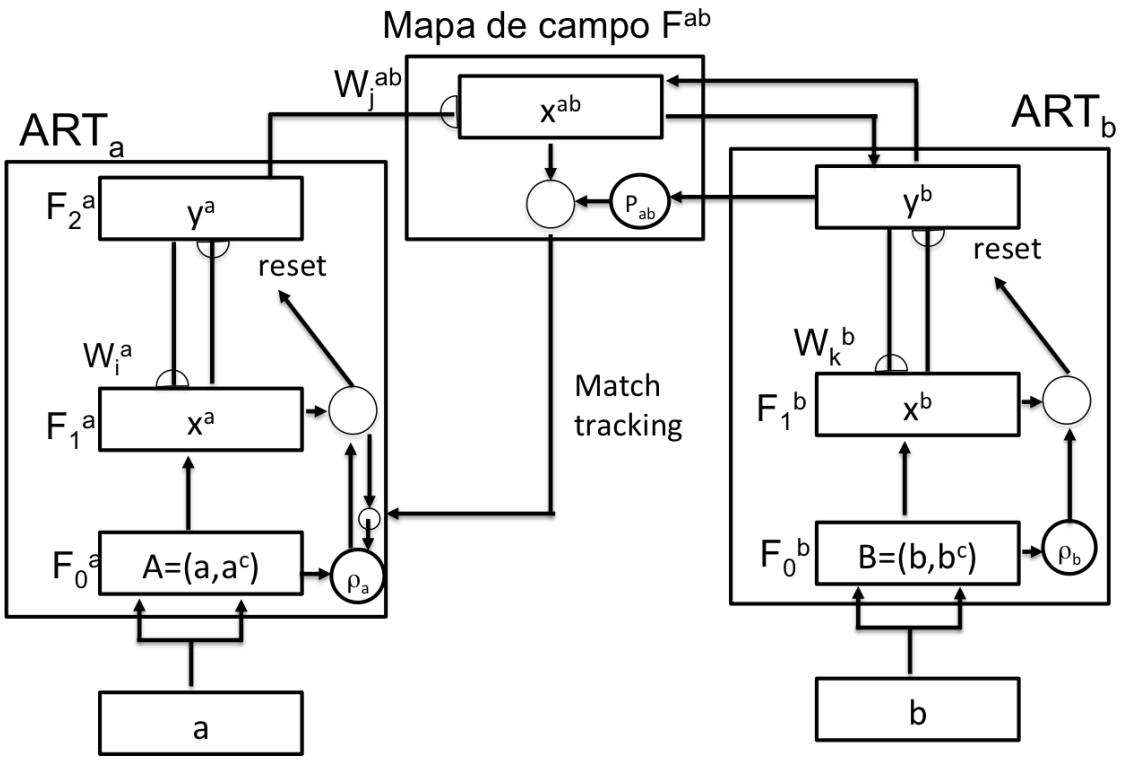

Figura 3.10: Esquema FuzzyARTMAP. [Grossberg, 1992]

3. Calcular la elección de categoría

$$T_j(I) = \frac{|I \wedge w_j|}{\alpha + |w_j|} \tag{3.11}$$

Donde

$$(p \wedge q)_i = min(p_i, q_i) \tag{3.12}$$

y

$$|P| = \sum_{i=1}^{M} |p_i| \tag{3.13}$$

α es el parámetro de elección y $\alpha > 0$; ρ es el parámetro de vigilancia con rango [0,1]

4. Encontrar el J-ésimo nodo tal que

$$T_J = max\{T_J\}; j : 1, \cdots, N \tag{3.14}$$

Las salida es y en F_2 cuando y_J y $y_j = 0; J \neq j$

5. Si hay resonancia, es decir si

$$\frac{|I \wedge w_j|}{|I|} \geq \rho \tag{3.15}$$

entonces se modifican los pesos con

$$w_J^{nuevo} = \beta \cdot \left(I \wedge w_J^{viejo}\right) + (1 - \beta) \cdot w_J^{viejo} \tag{3.16}$$

donde β es la taza de aprendizaje [0-1]

De lo contrario $w_J = 0$

6. Repetir los pasos 4 y 5 hasta que se cumpla lo siguiente

$$\frac{\left|y^b \wedge w_j^{ab}\right|}{|y^b|} \geq \rho_{ab} \tag{3.17}$$

7. Para una nueva categoría, repetir los pasos del 2 al 6

Clasificación

Para el proceso de clasificación se ejecutan lo pasos del 1 al 5 pero ahora los valores de entrada, son las señales provenientes del exterior, en éste caso, el vector BOF.

Para obtener el T_J es necesario ordenar las categorías T_j de forma ascendente.

Capacidades de reconocimiento de la BOF con el método alternativo

En el método alternativo, presentado en la sección anterior, se realizaron pruebas para determinar el número mínimo de elementos de la BOF para que la RNA Fuzzy ARTMAP pudiera reconocerlos. Cuando la densidad de puntos de la función a reconocer es alta, hay más probabilidades de que la RNA pueda reconocer al objeto, sin embargo, requiere más iteraciones sobre su base de conocimiento, lo cual implica mayor proceso de cómputo. Por tanto reducir al mínimo la densidad de puntos hace más eficiente el reconocimiento.

Para estas pruebas, se calculó la BOF con diferentes cantidades de puntos. El parámetro **pre** indica un factor de precisión en la función que genera el número de iteraciones. En este caso solo se trata de un cuadrado (figura 3.11). Para ésta prueba, el factor **pre** es de 0.1, 0.05

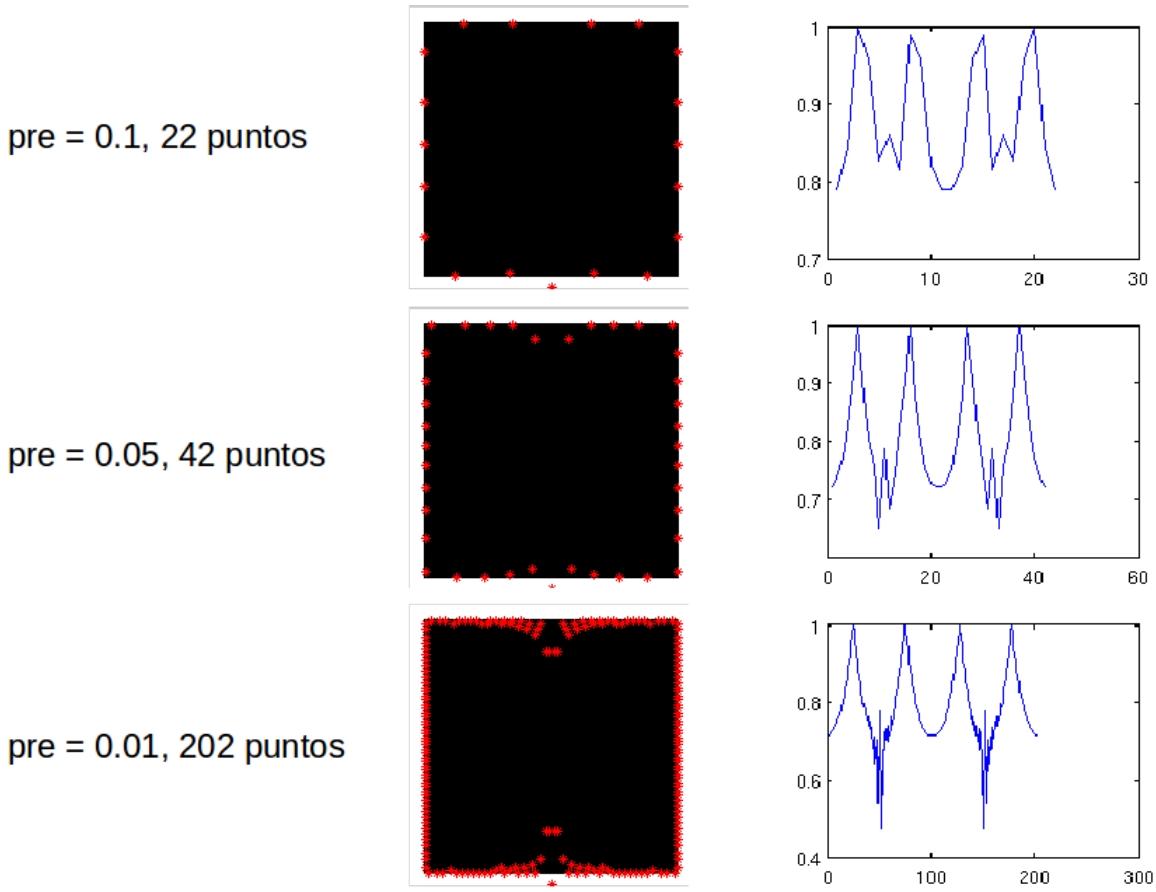

Figura 3.11: BOF con varias densidades de puntos frontera

y 0.01 lo cual genera 22, 42 y 202 puntos frontera. El mayor número de correspondencias se tuvo con 42 puntos. Y para probar la eficiencia se aplicó el algoritmo a otras tres formas, figura 3.12.

En la tabla 3.2 se muestra el resultado de la BOF (método alternativo)-Fuzzy ARTMAP para los objetos de la figura anterior.

3.2.3. Fuzzy ARTMAP en lógica reprogramable

Una de las razones para utilizar FPGA en el diseño e implementación de una RNA es la facilidad para reconfigurarla, como consecuencia el proceso de diseño es menos caro que un hardware ASIC, y también brinda al usuario la oportunidad de cambiar los pesos de la RNA basado en sus necesidades [Raeisi, 2006].

La intención de realizar una búsqueda sobre las características de una FPGA tiene como

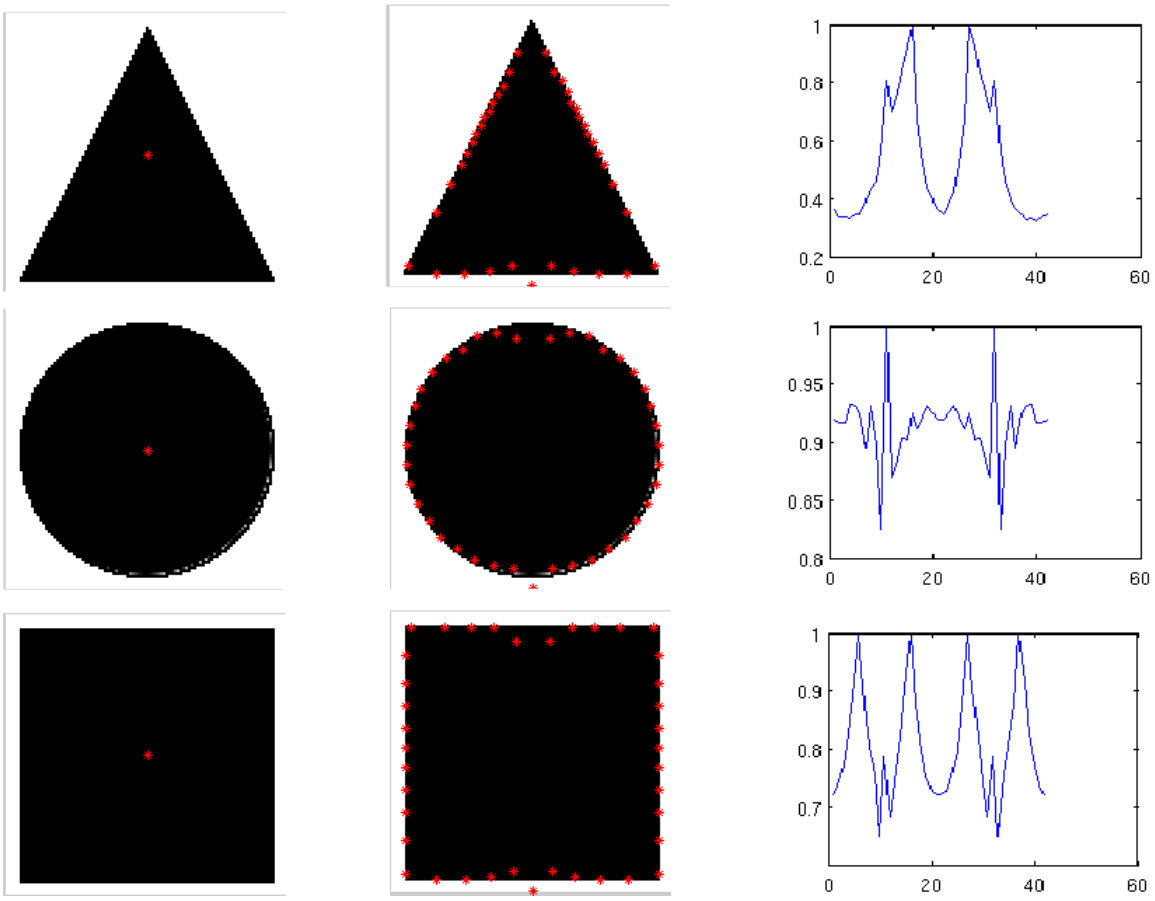

Figura 3.12: BOF de varias figuras con 42 puntos

objetivo determinar cual o cuales son los dispositivos que son más aptos o que presentan ventajas para la realización de este proyecto.

De ésta manera, el seleccionar en particular un dispositivo de lógica reconfigurable adecuado permitirá auxiliar en mayor medida las pruebas que se lleven a cabo a fin de conseguir el objetivo final.

En la literatura consultada, no se percibe una inclinación a arquitectura o tipo particular de una FPGA para la implementación de una RNA. De hecho, desde la década de los 90, cuando se concibió la idea de implementar una RNA sobre una FPGA [Zhu, 2003], la capacidad de integración aunada a la velocidad de dichos dispositivos en aquella época no resultaba desventajoso; puesto que ya se incluía el diseño paralelo, que finalmente prevalece al elegir las FPGAs sobre el resto de las implementaciones de RNA.

Figura(S) Entrena	Resolución	Fig Clasifica	Resolución	¿Acertó?
T,C,c	100x100	T	100x100	Sí
T,C,c	100x100	C	100x100	Sí
T,C,c	100x100	c	100x100	Sí
T,C,c	100x100	T	16x16	Sí
T,C,c	100x100	C	16x16	Sí
T,C,c	100x100	c	16x16	No
T,C,c	100x100	C rotado 45	100x100	Sí
T,C,c	100x100	T rotado 90	100x100	No
T,C,c,M	100x100	M	100x100	Sí
T triangulo, C Cuadrado, c círculo, M mancha 1				

Tabla 3.2: Pruebas de reconocimiento

Implicaciones técnicas de una RNA sobre una FPGA

De acuerdo a [Bonnici, 2006] existen limitaciones técnicas para la construcción de una RNA en cómputo reconfigurable. Entre otras, se puede mencionar el tamaño de la red neuronal y la latencia que existe entre la activación de una capa de neuronas y otra. Para el primer caso, lo importante es saber el número de pesos y su tamaño. Dado que la relación que existe entre estos dos parámetros da una idea de la capacidad que debe tener la FPGA para poder integrar una RNA. Y para la latencia, se busca eficientar el trazado de rutas de conexión dentro de la FPGA. Ésto se debe a que cada bloque lógico del FPGA se conecta por una línea de datos hacia otro bloque lógico, lo cual genera un retraso de la señal. El trazado de las rutas conexión se realiza de acuerdo a las restricciones del fabricante, y la herramienta ISE de Xilinx genera una ruta óptima [Wang, 2005]; Que bien puede ser revisada para acelerar la comunicación entre capas de neuronas. El diseño propuesto en este trabajo, presenta una estrategia complementaria para reducir los efectos de este fenómeno.

Los pesos de la RNA deberán estar disponible en la sección de lógica reconfigurable del sistema, ya que se requere acceder a ellos para realizar el cómputo de las salidas. Existen dos opciones para almacenarlos es dispositivos FPGA en memoria tipo DRAM, la cual es muy escasa en la mayoría de los encapsulados. Y la otra alternativas es implementarlos en memoria del tipo BRAM (BlockRAM); que si bien no se tiene en abundancia, sí se pude administrar de una manera más eficiente. Una regla sencilla que puede seguirse para determinar el número

de pesos posibles que se pueden implementar en una FPGA se muestra en 3.18, derivada del trabajo de [Bonnici, 2006].

$$NumeroDePesos = \frac{TamanoBlockRAM \times NumeroDeBlockRAM}{TamanoDeLosPesos} \tag{3.18}$$

La precisión del valor de cada peso en la RNA, es un factor a considerar. Si se busca mayor precisión es necesario implementar la localidad de memoria con mayor número de bits. Sin embargo, existe una restricción impuesta desde el diseño de fabrica; los BRAM ya tiene un tamaño predefinido, un ancho de palabra y una profundidad. Aún cuando es posible concatenar BRAM para incrementar el ancho y/o profundidad de datos, existen restricciones físicas para unirlos. De manera que la estrategia de diseño de la RNA involucra hacer arreglos de BRAM que aproveche tanto el espacio utilizado como la latencia que existe por el solo hecho de concatenarlas.

También la forma de representar cada peso es sumamente importante. Dado que gran parte de las operaciones que se realizan para la operación de las RNA son sumas y multiplicaciones, es prioritario determinar desde el principio la precisión numérica de los pesos, ya que esto definirá la capacidad requerida de la FPGA. Estos pesos pueden ser representados como enteros o en punto flotante. La figura 3.13 muestra la comparación de los valores de salida de una RNA con pesos representados con enteros contra pesos representados en punto flotante; dichos resultados son un trabajo realizado por [Raeisi, 2006]. Para el caso de los pesos representados con enteros, se aprecia que el resultado es inaceptable, por que son sumamente imprecisos comparados con la referencia. Sin embargo la salida en punto flotante se apega más a la referencia, y prueba que los pesos implementados en punto flotante son la mejor opción . El análisis de [Shirazi, 1995] enfatiza que un algoritmo que utilice el estándar IEEE754 muestra resultados aceptables; que consisten en que cada número de punto flotante es representado por 32 bits distribuidos de la siguiente manera: 1 bit para el signo, 8 bits para el exponente y 23 bits para la fracción. El exponente tiene un desplazamiento de 127 bits.

RNA Fuzzy ARTMAP implementada en una FPGA

En el trabajo de [Azouaoui, 2002] se propone un modelo de una RNA Fuzzy ARTMAP implementado sobre una FPGA. En la figura 3.14 se muestra un esquema del concepto de un clasificador expresado en un diseño digital. Lo que proponen consiste en que las entradas

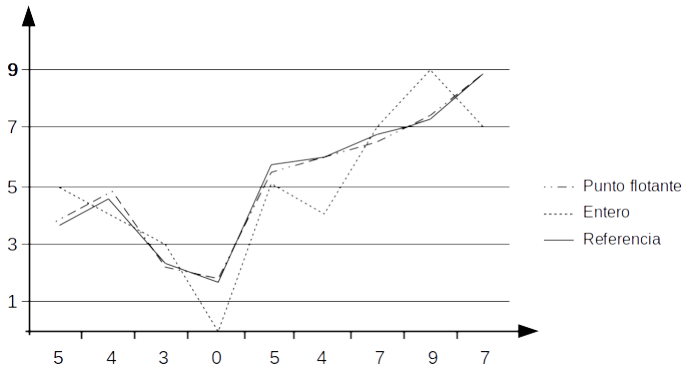

Figura 3.13: Comparación de una RNA con pesos representados por enteros y punto flotante (Raeisi y Kabir, 2006)

de la red, sean multiplexadas a través del componente MUX hacia un comparador, dónde es evaluado cada entrada con los pesos almacenados en una ROM y se determina cuál es el valor mpinimo. Dichos pesos fueron calculados previamente en un proceso fuera de línea. Lo obtenido en el comparado se acumula con el resultado anterior; la primera vez el acumulador vale cero. Cuando se ha iterado sobre todo el vector de entrada, es decir, después de n ciclos, se realiza una multiplicación por el factor $\frac{1}{\alpha+|w_j|}$, el cual es almacenado en un registro ROM.

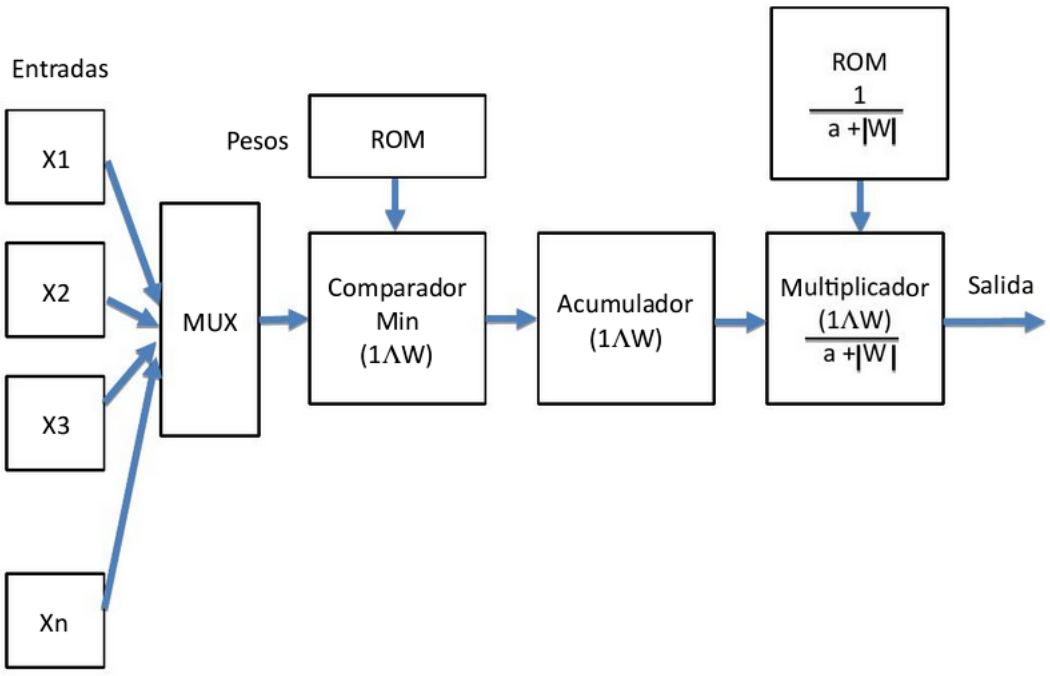

Figura 3.14: Clasificador fuzzyARTMAP implementado en FPGA)

En la propuesta de [Azouaoui, 2002] sólo se expone como calcular los T_j, sin embargo, es necesario determinar si la categoría ganadora j entra en resonancia o no, lo cual es fundamental para el proceso ARTMAP. Otro aspecto a resaltar, es el hecho de que en el diseño de [Azouaoui, 2002], se almacenan tanto los pesos como las entradas en registros del tipo ROM, espacio que habrá que considerar en el diseño en hardware reconfigurable. En el capítulo posterior se habla de las mejoras al diseño de [Azouaoui, 2002] y la arquitectura del resonador.

3.3. FPGA

En ésta sección se mencionan las características fundamentales de una FPGA y del lenguaje de alto nivel VHDL.

3.3.1. Dispositivos electrónicos reprogramables

En el diseño electrónico digital se han planteado diversas formas para implementar circuitos que sin hacerle muchos cambios a la estructura original, puedan efectuar otras operaciones. Con el fin de obtener un circuito digital maestro, es decir que pudiera implementar cualquier función lógica sin tener que re cablear, nació la lógica reprogramable.

Las FPGA son circuitos electrónicos reprogramables basados en un arreglo de compuertas lógicas y biestables (en inglés *flip-flops*) [Chu, 2008a, p. 11] las cuales son conectadas a través de buses de datos bidireccionales a módulos de entrada y salida. Su fama radica en el hecho de que cualquier circuito digital pueda ser implementado en ellas, no importando su complejidad. Desde luego existe una limitante de espacio que depende del número de bloques lógicos, del número de entradas y salidas y de la memoria RAM con el cual, el dispositivo fue fabricado.

La historia de las FPGA está precedida por otros circuitos reprogramables más austeros y de menor tamaño. Se puede considerar que los primeros circuitos reprogramables son las memorias PROM (del inglés *Programmable Read-Only Memory*) las cuales se construyeron con intención de almacenar información ya sea de manera permanente o bien borrar su contenido de acuerdo a su tecnología. Pero utilizando sus lineas de direccionamiento como entradas, y el contenido de la ROM como el resultado de una tabla de verdad, la salida es el resultado de un circuito combinacional. Sin embargo, está limitado por el ancho de palabra cómo por la

profundidad de datos a almacenar.

Un avance a las PROM, fue el conseguir la parte fundamental de una ecuación booleana en productos de sumas. De ésta manera, los PLA (del inglés *Programmable Logic Array*) implementan en su lógica interna dichas combinaciones. Un fusible es el encargado de conectar o desconectar todas y cada una de las combinaciones posibles. Fueron inventadas por Texas Instruments en 1970. [Andres, 1970]

En contraparte, se tienen los PAL (del inglés *Programmable Array Logic*) donde su estructura interna habilita o deshabilita las sumas de productos de una función booleana. El primer dispositivo PAL fue presentado por la empresa MMI en 1978. [Parnell, 2003, p. 12]

Posteriormente la empresa Lattice Semiconductor en 1985 [Lattice, 1998] inventó las GAL (del inglés *Generic Array Logic*). Cuya configuración, similar a las de las PAL, se puede borrar y volver a escribir ya que son de tecnología EE (del inglés *Electrically Erasable*).

Posterior a estos diseños, surgieron los CPLD (*Complex Programmable Logic Device*) que son circuitos que dieron origen al diseño de las FPGA. En la figura 3.15(a) se muestra el diagrama de una FPGA. El cual está compuesta fundamentalmente por bloques lógicos, bloques de entradas y salidas, BRAM y rutas para interconectarlos.

Los bloques lógicos son los elementos donde se implementa la lógica de la FPGA. Su diseño varía entre fabricantes e incluso entre familias del mismo fabricante. Sin embargo, en la figura 3.15(b) se muestra el diagrama clásico de un bloque lógico. El cual se compone de una LUT, un flip-flop y un multiplexor. Con estos tres elementos es posible instrumentar cualquier tipo de circuito combinacional o secuencial. En algunos casos es necesario unir más de un bloque lógico para encontrar la funcionalidad necesaria..

3.3.2. VHDL

VHDL es un lenguaje de descripción de hardware (del inglés *Very High-Speed integrated circuit Description Language*) que originalmente fue creado por el Departamento de Defensa de los Estados Unidos de América. Posteriormente, la IEEE (del inglés *Institute of Electrical and Electronics Engineers*) asumió su estandarización en 1987 [Chu, 2008b, p. 1] llamandolo IEEE Standard 1076 al cual se le refiere como "VHDL 87". La versión actual más ampliamente utilizada es la "VHDL 93".

Con VHDL se abarcan todas las posibilidades del diseño digital, pues por un lado se

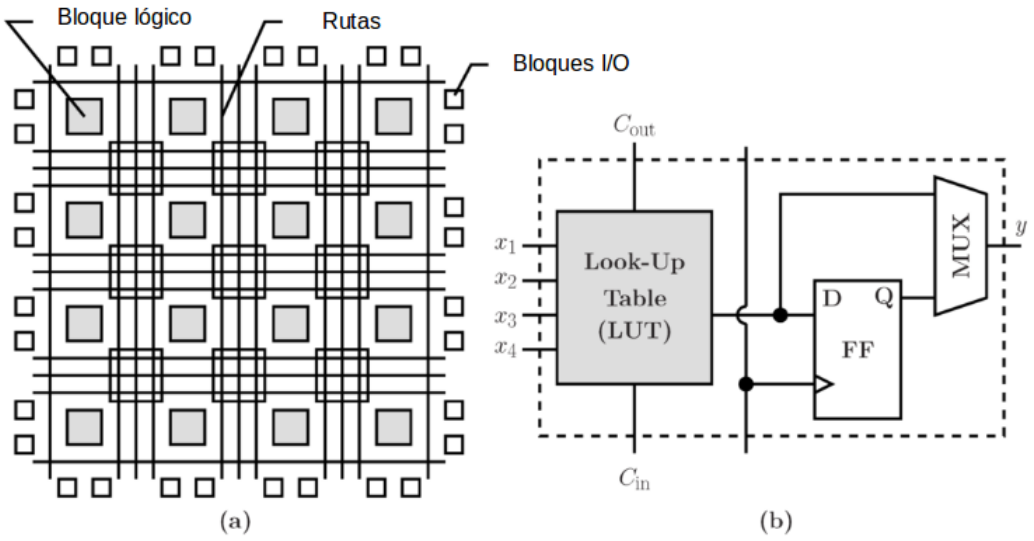

Figura 3.15: a) Diagrama FPGA, b)Bloque lógico

puede expresar un circuito como un conjunto de subcircuitos y sus conexiones y por el otro lado permite definir el funcionamiento del diseño utilizando un lenguaje de programación de alto nivel con similitudes a otros lenguajes de programación. [Ashender, 1990, p. 1]. La misma estructura de VHDL permite generar bancos de prueba para simular el funcionamiento de un hardware digital.

Existen otros lenguajes de descripción de hardware como son Verilog (IEEE 1364), ABEL (del inglés *Advanced Boolean Expression Language*), AHDL (del inglés *Altera Hardware Description Language*), Impulse C y muchos otros. Sin embargo, Verilog y VHDL cuentan con las simpatía de más número de usuarios, lo cual repercute en una mayor capacidad de soporte, más documentación disponible en la web y sobre todo, mayor oferta de módulos disponibles, tanto de pago como de distribución libe.

Aunque las herramientas de desarrollo de Xilinx Inc. soportan Verilog y VHDL en este trabajo se optó por usar VHDL. Esto obedece a varias razones, desde el planteamiento del diseño de la MDRO se observó que el requerimiento de usar punto flotante iba a ser un requisito fundamental. Por ejemplo, para obtener las distancias medias cuadráticas del centroide a cada una de los bordes, el resultado es un número real. De la misma forma, en el algoritmo de la RNA se utilizan varias ecuaciones cuyos resultados son fracciones. De tal manera que, una de las ventajas de usar VHDL es el hecho de que, varios paquetes disponibles en internet para la operación en punto flotante, están desarrollados en dicho lenguaje.

Otro punto favorable de VHDL es la posibilidad de la creación de procedimientos y fun-

ciones. Concepto que en Verilog no existe y que debe de ser resuelto de otra manera. Los procedimientos y funciones son necesario para la modificación del tipo de señales, activar o desactivar una señal debido a una alerta, realizar operaciones sobre variables, entre otros.

Existen tres tipos de descripción en VHDL por su

1. Comportamiento.
2. Flujo de datos.
3. Estructura.

Para los cuales la forma de expresar su funcionamiento cambia en función de las intenciones para con el componente. Es decir, habrá módulos dónde es más fácil describir su funcionamiento de acuerdo a su comportamiento, pero otros a partir de su estructura. No se prefuer un tipo sobre otro, son diferentes formas de describir la funcionalidad.

Existe una método para realizar el diseño en hardware reconfigurable. Las herramientas generalmente realizan estos pasos de forma autompática pero es importante saber el proceso. El cual se lista a continuación:

1. Se desarrolla el código HDL del componente, en términos de su comportamiento, por flujo de datos o por su estructura.
2. Síntesis del código. Se realiza una conversión de estructuras del lenguaje HDL a componentes digitales. Y se genera un *netlist*. El cual es un listado de todas las conexiones necesarias para implementar el diseño.
3. *Translate*. Coteja el *netlist* con las restricciones o reglas de implementación.
4. Mapeo. Colocación del diseño en una parte del hardware.
5. *Place and Route*. Valida rutas y posiciones del diseño de acuerdo a las restricciones dadas por las señales de tiempo.
6. Generación del *Bistream*. El *Bistream* es una serie de datos binarios para la configuración de del hardware reconfigruable.
7. Descarga del *Bistream* en el hardware reconfigurable, una vez configurado se inician el o los relojes del diseño.

Capítulo 4

Máquina Digital de Reconocimiento de Objetos y sistema embebido

En este capítulo se presentan los componentes básicos del sistema de reconocimiento de objetos y su interconexión con el SE. Por un lado se describe la arquitectura del sistema en general y por otro las características de funcionamiento (estructuras, código, abstracciones, etc.), tanto de la MDRO como del SE.

Para el caso de la MDRO, se describen los pasos del procesamiento de la imagen, extracción de la BOF y el cálculo de los T_j de la RNA Fuzzy ARTMAP. Para el SE se presenta la arquitectura, la interacción con la MDRO, tanto en el aspecto de control y configuración, así como el traslado de datos de video.

4.1. MDRO

La Máquina Digital de Reconocimiento de Objetos (MDRO), es un dispositivo digital capaz de realizar, a veces secuenciales y a veces en paralelo, operaciones lógicas, aritméticas y trigonométricas, desplazamiento de datos tanto a memoria BRAM y en registros. Esto con el firme propósito de procesar los datos de una imagen y discriminar objetos en ella. Sin embargo, es necesaria la intervención de un dispositivo para comandar dichas operaciones y que proporcione las herramientas necesarias para poder comunicar la MDRO con interfaces para el usuario o con otros dispositivos (brazo robótico, control de la celda de manufactura, control de fuentes de iluminación, etc.)

Antes de entrar a la descripción de la MDRO, la siguiente sección detalla la estrategia del uso de BRAM en la FPGA utilizada.

4.1.1. Análisis de uso de la Block RAM

El FPGA Zync XC7Z010-1CLG400C tiene 60 bloques de memoria tipo BRAM de 36 [kb] cada uno, puede almacenar alrededor de 240 [KB] [Xilinx, 2016, p. 3]. Cada bloque se puede configurar de acuerdo a las siguientes opciones 32K x 1, 16K x 2, 8K x 4, 4K x 9 (u 8), 2K x 8, 1K x 36 (o 32). Es posible concatenar las BRAM o bien acoplarlas para incrementar la capacidad de almacenamiento.

A manera de ejemplo se presenta la siguiente situación. Supongamos que se quiere almacenar las coordenadas de los puntos frontera de un objeto presente en una imagen de 640 x 480 pixeles. Se plantean dos configuraciones.

1.- Suponer que el mayor objeto que puede caber en la imagen tiene un perímetro que abarca toda la imagen en sí, sería $p = 2 * b + 2 * h = 2 * 640 + 2 * 480 = 2,240$. Por lo tanto se tendría un máximo de 2,240 coordenadas a almacenar. Cada coordenada x, y requiere de $numBitsX = \frac{log(640)}{log(2)} = 9{,}32$ para representar las coordenadas en x y $numBitsY = \frac{log(480)}{log(2)} = 8{,}9$ para representar las coordenadas en y. Redondeado hacia el entero más próximo se tiene, 10 bits para x y 9 bits para y, un total de 19 bits. Se almacenarían ambas coordenadas concatenadas, es decir x[18:9]:y[8:0]. Y se necesitarían 12 bits para direccionar esa cantidad de datos ya que $numBitsDir = \frac{log(2{,}240)}{log(2)} = 11{,}13$ redondeado al entero mas próximo.

Para éste caso, es necesario usar 4 bloques de los 60 que se tienen con una configuración de 1K x 36.

2.- Otra configuración es mapear todos los pixeles de la imagen en una BRAM y decir si forman parte del punto frontera o no. De manera que solo sería necesario 1 bit por cada localidad. Para representar $numPixelesTotales = 640 * 480 = 307,200$ se necesitan 307,200 localidades de memoria. Se requerirían 10 bloques de los 60 que se tienen con una configuración de 32K x 1.

Aún cuando en la primera configuración se desperdician 17 de 36 bits por cada localidad y se utiliza el 6.6 % del total de BRAM, resulta más favorable sobre la segunda opción, donde se sacrifica el 16 % del total de BRAM de este FPGA. Además es poco probable que los puntos frontera, de cualquiera que sea el objeto que se tenga en la imagen, abarque algo cercano a los 307,200 puntos.

4.1.2. Arquitectura general

La arquitectura del calculador de BOF de la Máquina Digital de Reconocimiento se muestra en la Figura 4.1.

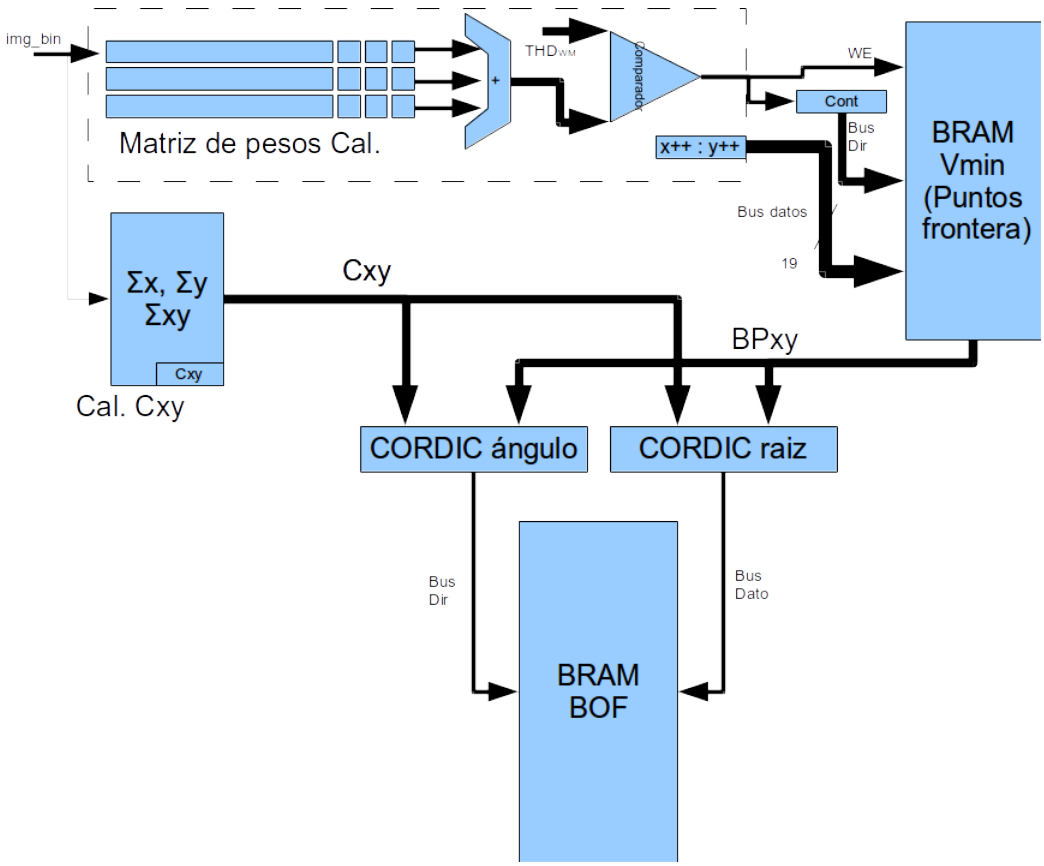

Figura 4.1: Arquitectura de la MDRO

la MDRO principalmente se compone de:

- el calculador de la Matriz de transformaciones de pesos,
- el calculador del centroide,
- los módulos CORDIC para cálculo del ángulo y de la raíz cuadrada,
- una BRAM para almacenar los puntos frontera,
- una BRAM donde se almacena la BOF y
- el clasificador Fuzzy ARTMAP (no se muestra en la figura).

A continuación se describen los pasos que procesan la imagen detalladamente.

4.1.3. BOF en la FPGA

Siguiendo el flujo del proceso para obtener la BOF, se explica a continuación los módulos desarrollados. Primero se menciona el cálculo y obtención del umbral, posteriormente el módulo de la matriz de pesos y centroide y para terminar la obtención de la BOF.

Histograma y umbral

Para binarizar la imagen donde se encuentra el objeto a reconocer, es necesario establecer el umbral, el cual se obtiene a partir del histograma de un cuadro del video. El proceso de su obtención es el siguiente

1. Buscar el primer máximo del histograma,
2. entonces, buscar el primer mínimo.

La forma en la que se calcula el umbral es a través del procesador del SE, es decir, se calcula por software. Al principio se optó por desarrollar un calculador de histogramas por hardware. De hecho se realizaron pruebas y se comprobó su eficiencia. Sin embargo, era necesario almacenar en un BRAM $2^8 \times 19$ = 4,864 bits, y dada la limitante de espacio se escogió que el histograma se alamcenara en la memoria RAM del SE.

En la Figura 4.2 se muestra el resultado de la binarización de una imagen a través del cálculo del umbral (THD) a partir del histograma de la imagen.

Matriz de transformaciones de pesos

En la sección 3.1.5 se aborda la forma de obtener el borde de un objeto a través del el cálculo de la Matriz de transformaciones de Pesos, a partir de una imagen binaria. En la figura 4.3 se muestra la arquitectura del calculador de la Matriz de transformaciones de pesos.

Para evitar almacenar toda la imagen binaria en memoria RAM se utiliza una memoria BRAM de 637 bits x 3 lineas y 9 registros ordenados en 3 x 3 bits. Al final se tiene una

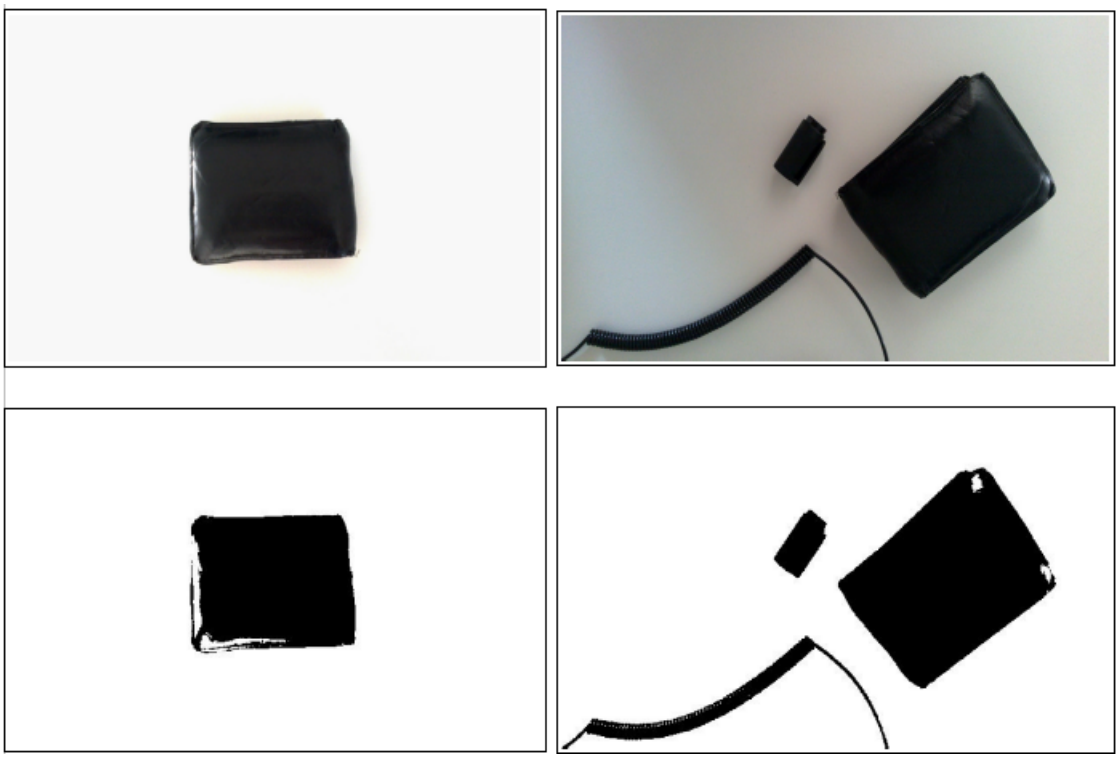

Figura 4.2: Binarizado de un objeto utilizando la MDRO

capacidad de almacenamiento de 637:3x3 bits. El propósito de este submódulo es calcular la Matriz de Pesos, para posteriormente obtener el centroide y el contorno del objeto.

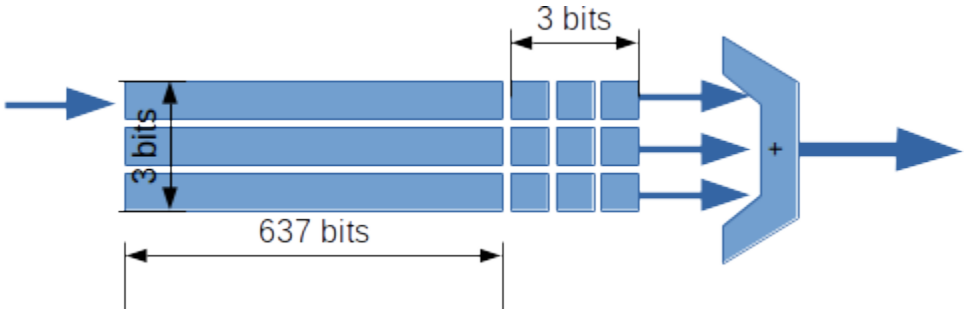

Figura 4.3: Calculador de la Matriz de transformaciones de pesos

Ésta configuración sustituye 9 memorias RAM de 307,200 posiciones x 1 bit. Con un proceso conocido como **memoria de ventana** se almacenan 3 líneas consecutivas de la imagen binaria. Y a través de un registro de corrimiento y memoria BRAM, se realizan corrimientos a la derecha por cada nuevo pixel que ingresa. Es decir, la memoria BRAM funciona como un dispositivo del tipo FIFO (First Input - First Output) de un bit de ancho por 637 bits de

profundidad. Al principio toda la memoria BRAM es inicializada con ceros.

En cada ciclo de reloj ingresa un bit proveniente de la señal img_bin a la primera localidad de la BRAM. En el segundo ciclo de reloj, ese dato es copiado a la localidad siguiente y un nuevo bit es ingresado a la localidad que ocupaba el dato original. Esto ocurre 637 veces, es decir, hasta que el primer bit de la imagen alcanza la última localidad de la memoria BRAM. Al siguiente ciclo de reloj, ese dato es copiado al primer registro del kernel 3 x 3. La siguiente vez, el valor del primer registro sr3x3_(0)(1) es copiado a sr3x3_(0)(2). Cuando el primer bit de la imagen alcanza el registro sr3x3_(0)(3), ese dato es copiado a la primer localidad de la BRAM del segundo renglón.

Así, después de llenar 2 lineas del registro de corrimiento, es decir al terminar 640 ciclos de reloj x 2, se empieza obtener la Matriz de Pesos sin necesidad de esperar toda la binarización de la imagen. Los 9 registros (3 x 3) realizan la suma del kernel circundante al pixel que se está explorando. Cada dato saliente es enviado a otro módulo que compara si se trata de un máximo o de un mínimo. En el código se muestra una ventana de tiempo implementada en VHDL. Cada registro sr3x3_n corresponde a un std_logic_vector con el valor temporal del kernel que barre todos los renglones y columnas de una imagen.

El código 1 muestra la implementación de la Matriz de Pesos (NWf) Ecuación 3.6. de todos los pixeles que conforman la imagen binaria.

```
sr3x3_0(0 to 1)<= sr3x3_0(1 to 2);
sr3x3_1(0 to 1)<= sr3x3_1(1 to 2);
sr3x3_2(0 to 1)<= sr3x3_2(1 to 2);
matriz_pesos_sign <= sl2int(sr3x3_0(0)) + sl2int(sr3x3_0(1))
                   + sl2int(sr3x3_0(2)) + sl2int(sr3x3_1(0))
                   + sl2int(sr3x3_1(1)) + sl2int(sr3x3_1(2))
                   + sl2int(sr3x3_2(0)) + sl2int(sr3x3_2(1))
                   + sl2int(sr3x3_2(2));
if(to_unsigned(matriz_pesos_sign,4)
            = to_unsigned(umbral_frontera,4)) then
    es_vmin <= '1';
else
        es_vmin <= '0';
end if;
```

Código 1: Obtención de la Matriz de Pesos en VHDL

Obtención del centroide

El cálculo del centroide se realiza al mismo tiempo en el que la imagen se explora pixel a pixel. Las ecuaciones 3.4 y 3.5 se utilizan para calcular el centroide en la FPGA. El código 2 muestra la forma de obtención del centroide. La señal de sincronía sinc_v anuncia la llegada de una nueva línea, la cual está activa en todo momento en que están ingresando pixeles binarios al módulo del calculo del centroide. Ax y Ay se auto incrementan i y j veces respectivamente. Esto solo ocurre si la bandera de imagen_binaria_in está activa, es decir que hay pixeles con presencia de objeto.

El área del objeto tanto en x como en y representada como u_b y se calcula con el auto-incremento en uno cada vez que hay un pixel perteneciente al objeto.

```
if(sinc_v = '1')then
        if(j < ancho_imagen) then
                if(imagen_binaria_in = '1' ) then
                        Ax := Ax + j;
                        Ay := Ay + i;
                        b := b + 1;
                        s_ax <= ax;
                        s_ay <= ay;
                        s_b <= b;
                end if;
                j := j + 1;
                s_j <= j;
        else
                j:= 1;
                i := i + 1;
                s_i <= i;
        end if;
else
        u_Ax := to_unsigned(Ax, u_Ax'length);
        u_Ay := to_unsigned(Ay, u_Ay'length);
        u_b := to_unsigned(b, u_b'length);
        s_cen_x <= divide (u_Ax, u_b);
        s_cen_y <= divide (u_Ay, u_b);
end if;
```

Código 2: Cálculo del centroide del objeto en VHDL

Una vez que es recorrida toda la imagen, se calcula la división de la suma de u_Ax entre

u_b para conocer el valor de la señal s_cen_x que será la que proporcione la coordenada X_c. Lo mismo ocurre para Y_c que es la señal s_cen_y. Obsérvese que no ha sido necesario almacenar ningún pixel binario de la imagen.

La construcción de este módulo es independiente del tamaño del rectángulo imaginario que contiene al objeto dentro de una imagen. De tal manera que pudiese ser replicado cuantas veces sea apropiado para que, en caso de encontrar más de un objeto en la imagen, se pueda calcular el centroide de cada uno de los objetos; y de ésta manera obtener el centroide de todos o algunos objetos de cierta forma en paralelo.

Obtención de los puntos frontera

A la salida del calculador de la Matriz de Pesos, es decir del sumador de 9 bits, está un comparador (Figura 4.4). El cual determina si a partir de cierto umbral (THDwd), se considera si el pixel forma parte de un punto frontera o bien de la parte sólida del objeto. De acuerdo a mediciones empíricas se considera que un THDwd = 5 discrimina pixeles sólidos de puntos fronteras.

El comparador activa la señal WE (activa escritura) de la BRAM llamada Vmin, dónde se almacenan temporalmente todos los puntos frontera. La misma salida del comparador incrementa un contador Cont que sirve como generador de direcciones de la memoria Vmin. Al principio Cont vale cero. Cada vez que un pixel frontera es encontrado se incrementa en uno la dirección. Y el dato que se almacena son las coordenadas x y y de dicho punto. La forma de obtener las coordenadas es a través de dos contadores X y Y que se autoincrmenentan en uno (X++; Y++) por cada pixel válido de la imagen.

Ambas coordenadas son almacenadas en una localidad de memoria de la siguiente forma [x(10 bits):y(9 bits)]. Esto es por que para representar el ancho de la imagen, en este caso 640, se necesitan 10 bits y para el alto de la imagen, en este caso 480, se necesitan 9 bits.

El tamaño de la BRAM queda 10 bits x 524,287. Con esto se garantiza que para el objeto más grande, es decir, que ocupe todo el ancho y todo el alto de la imagen pueda almacenarse sus puntos frontera.

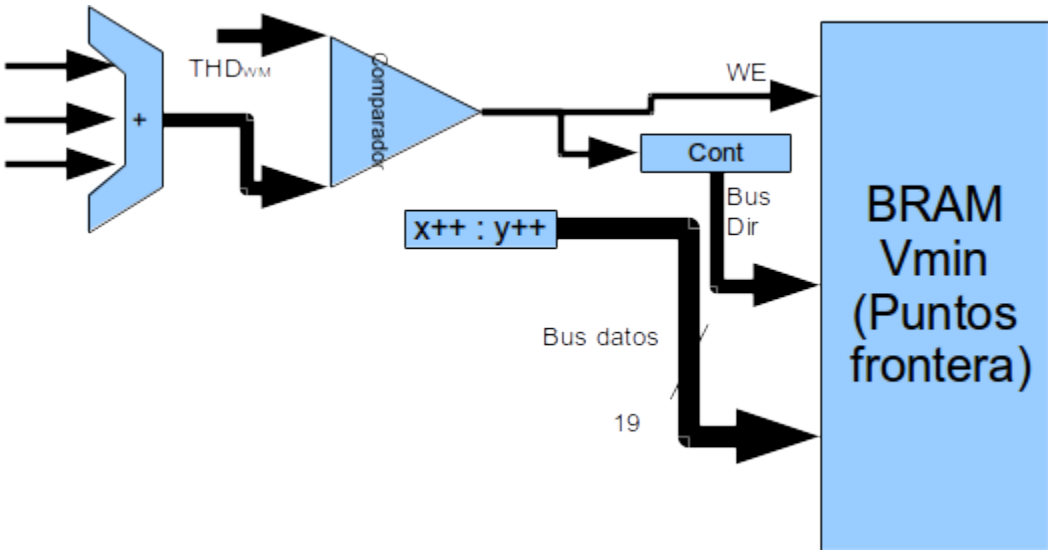

Figura 4.4: Calculador de los puntos frontera

Obtención de la BOF

Por último, para obtener la BOF de un objeto es necesario calcular las distancias entre el centroide y cada uno de los puntos frontera. En la figura 4.5 se muestran dos módulos CORDIC y una BRAM donde se almacenará la BOF.

El módulo CORDIC, construido a partir del Core Generator de Xilinx, realiza dos tareas simultáneas. Por un lado, permite calcular la raíz cuadrada de la suma de los catetos de la distancia del centroide a cada punto frontera y por el otro obtiene el ángulo que corresponde a dicho vector. El módulo denominado CORDIC ángulo tiene conectado a su salida el bus de direcciones de la BRAM BOF. De tal manera que al momento en que se obtienen los componentes de cada vector de Vmin, se obtiene tanto su distancia al centroide, como la dirección de memoria a la que corresponde.

Al final en BRAM BOF se obtiene el conjunto de distancias que componen el descriptor único del objeto que aparece en la escena.

4.1.4. Clasificador Fuzzy ARTMAP en la FPGA

Una vez obtenida la BOF, se realiza la comparación de ésta con la base de conocimientos de la RNA, actividad que realiza el clasificador FUZZY ARTMAP.

El clasificador FUZZY ARTMAP descrito en la sección 3.2.2 fue modelado en un arqui-

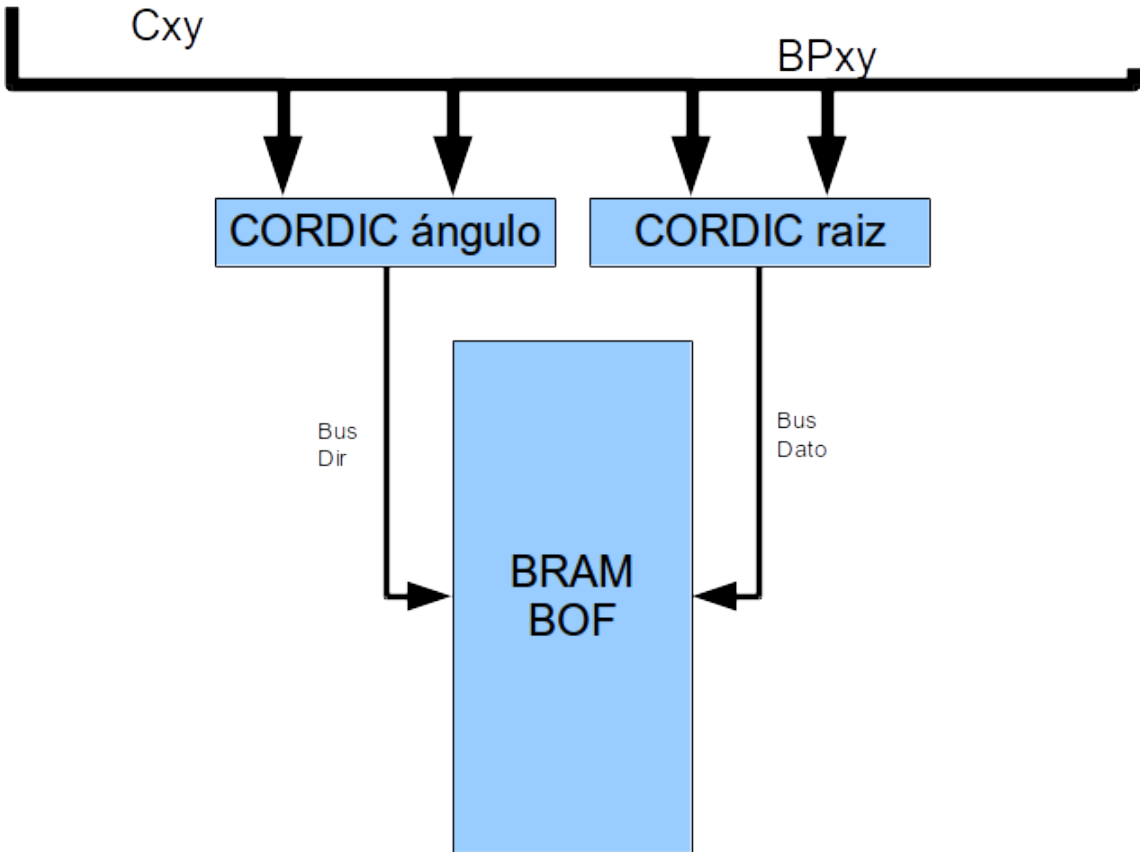

Figura 4.5: Calculador de la BOF

tectura para hardware reconfigurable utilizando VHDL. El esquema de funcionamiento del clasificador para j categorías se muestra en la figura 4.6. Se entiende que cada T_j categoría es una red neuronal entrenada para la BOF de cada objeto, o bien la misma BOF pero desde diferentes puntos de vista. Por lo tanto, el número de T_j estará en función del número de objetos con los que se entrenó la RNA fuera de línea. Obsérvese que cada T_j es idéntico a los otros.

El núcleo de funcionamiento del clasificador FUZZY ARTMAP se basa en el esquema de una sola categoría que se muestra en la Figura 4.7. Las entradas corresponden a la BOF obtenida en el proceso anterior $X_{i,j}$ y los pesos $W_{i,j}$ obtenidos al entrenar la red.

El segundo paso del algoritmo del clasificador indica que es necesario calcular el complemento de 1 tanto de la señal de entrada, X, como el de los pesos W. Como ya se conoce el número total de las entradas, es decir el número de elementos de la BOF, y en consecuencia el número total de los pesos, es posible dividir ambos conjuntos en dos partes: la primera con

los pesos y entradas, y la segunda como el complemento de 1 tanto de los pesos como de las entradas. De esta manera, que el total de ciclos de reloj para calcular el mínimo de ambos conjuntos se realiza en la mitad de tiempo.

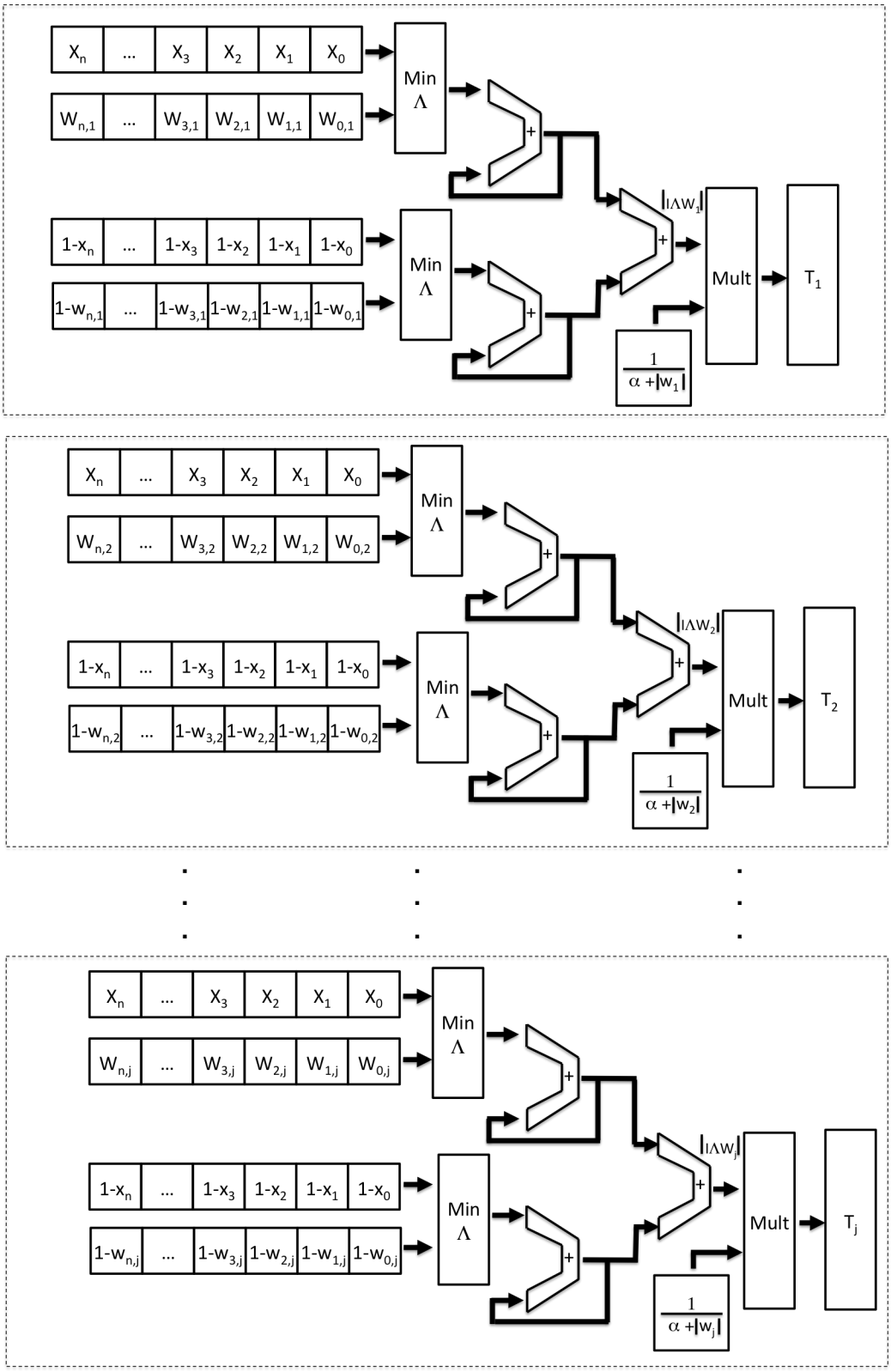

Figura 4.6: Clasificador T_j.

Para el tercer paso, ambas partes de los datos son procesadas simultáneamente en 2 secciones que simplemente calculan el mínimo entre el conjunto de datos de entrada y el conjunto de datos de pesos. Esta operación se realiza en un ciclo de reloj para cada elemento de la BOF. A la salida de esta operación se tiene un acumulador que suma el valor presente con el total de los valores anteriores. Al principio el acumulador vale cero. El registro donde se van acumulando los datos tiene una amplitud máxima igual al total de número de datos de entrada. Esto es porque en el peor de los casos al ser una señal normalizada la suma de todos los conjuntos de entrada con su correspondiente complemento a 1 pueden sumar como máximo dicha cantidad. En el mejor de los casos la suma acumulada será cero.

Para este punto el término $|I \wedge W_j|$ de la ecuación 3.11 ya ha sido calculado. No solo para una categoría j, sino para todas las neuronas entrenadas. Después, ese término es multiplicado por $\frac{1}{\alpha+|w_j|}$, término calculado fuera de línea y que es almacenado en un registro std_vector_logic. Al final se tiene el T_j calculado al mismo tiempo que el resto de las categorías.

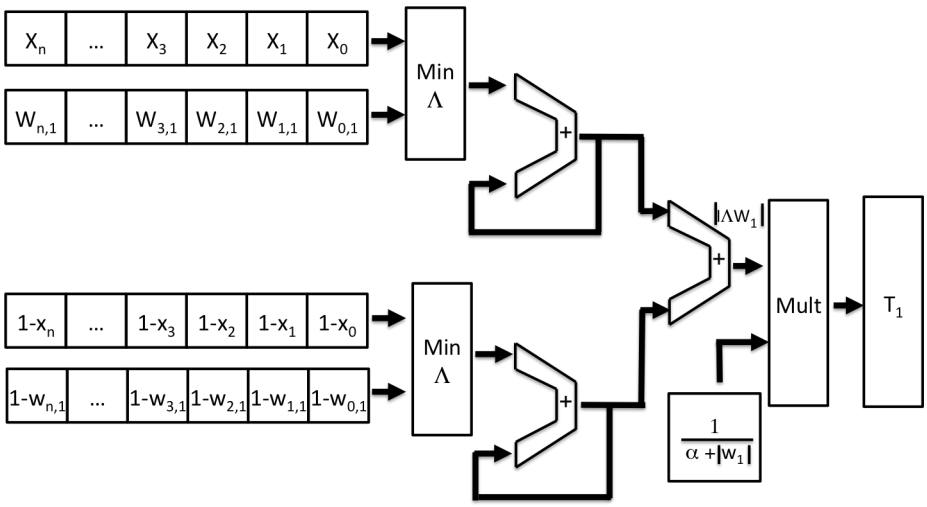

Figura 4.7: Clasificador T_j. Con j = 1

Posteriormente, se ordena de mayor a menor las categorías activadas y se itera sobre ellas para comprobar si supera el parámetro de vigilancia ρ de la ecuación 3.15. Dado que ya se calculó $|I \wedge W_j|$ solo es necesario dividirlo entre $|I|$. El cual es idéntico al número de datos de entrada, es decir el número de elementos de la BOF.

El ordenamiento de los T_j se realizan por software en el procesador del sistema embebido. Esto es por qué la ordenación de datos en hardware no es apropiada para ésta aplicación,

debido al consumo excesivo de recursos. De manera que los T_j son almacenados en registros accesibles desde el bus de datos del sistema embebido. Posteriormente, al activarse una bandera, el software los lee y los ordena. Con la categoría activada se comprueba si supera al factor de vigilancia. Para eso los términos $|I \wedge W_j|$ de cada categoría también son accesibles para el bus de datos. La división entre $|I|$ se realiza por software. Un extracto del código en C se muestra en el código 3.

```
int TJGanadora(int NumCategorias, int DirT0, int LongitudBOF,
                                                    float PVigilancia){
    float T[NumCategorias];
    float IminW;
    float rho;
    int indicesOrdenados[NumCategorias];
    for(i = 0; i <= NumCategorias; i++)
        Xil_In32(DirT0 + i, T[i]);
    indicesOrdenados = ordenaCategorias(T);
    Xil_In32(IminW + indicesOrdenados[0], IminW);
    rho = IminW / LongitudBOF;
    if (rho >= PVigilancia )
        return indicesOrdenados[0];
    else
        return 0;
}
```

Código 3: Código C del ordenamiento de Tj y obtención del parámetro de vigilancia

La función TJGanadora() regresa el índice de la categoría ganadora.

4.2. Sistema embebido de reconocimiento de objetos

En esta sección se describe el proceso de diseño de la arquitectura del sistema embebido y sus componentes. Además de exponer las razones de la necesidad de un sistema embebido.

La MDRO podría funcionar de forma autónoma para reconocer objetos, incluso algunas partes del proceso para reconocer objetos que han sido implementadas para resolverse por software, podrían haberse implementado totalmente en hardware. Aunque por razones de espacio y eficiencia en la FPGA se prefirió desarrollar en software.

Sin embargo, existen aspectos tanto de configuración, comunicaciones y flexibilidad ha-

cen es más complicado resolver sólo con máquinas digitales y en cambio, por software resulta más versátil. Además, existe la posibilidad de tener toda la funcionalidad, desde la captura de video hasta el reconocimiento del objeto, en un solo chip, SOC, por lo que resulta muy alentadora. Como ya se vió en el estado del arte de éste trabajo, un sistema de bajo consumo de potencia eléctrica y qué además este reducido en espacio resulta muy conveniente para desarrollar celdas de manufactura a menores costos.

Por lo tanto, el diseño del SOC se contempla como un sistema embebido (SE) que comanda, comunica y configura a la MDRO.

Una razón importante para embeber la MDRO en un sistema de cómputo es el hecho de poder comunicar todo el sistema a través de redes de datos TCP. Las cuales son un estándar en los subsistemas que componen una celda de manufactura. Al integrar en un SOC la MDRO se amplía la posibilidad de conectividad con otros subsistemas. Si bien ésta capacidad de interconexión está fuera del alcance de este trabajo, se deja esa disponibilidad en torno a dos aspectos: el primero tiene que ver con la configuración y adaptación del proceso de reconocimiento, es decir, la actualización de los W_j precomputados, el cambio de resolución de la cámara, la ecualización de los colores del sensor, la velocidad de captura, etc. Todo esto a través de ya sea comando sobre la red o bien la publicación de un servicio (p. ej. *webservices*) El otro aspecto tiene que ver con la posibilidad de transmitir el video a través de cualquier medio. En esta sección se verá que el *framebuffer* de video se almacena y actualiza dentro de la memoria RAM del sistema embebido. Y que al colocarla ahí se tienen las capacidades de velocidad y de disponibilidad para transmitirla a donde sea necesario. Por ejemplo, para el caso de este trabajo, se utiliza para proyectar la imagen original con capacidades de realidad aumentada en un monitor VGA. Aunque también pudo haber sido sobre HDMI. Sin embargo, en el sistema embebido podría incluirse un módulo Ethernet para las transmisión o publicación de video a través de una red. Aunque sólo alambrica por el momento; esto debido a las características propias de la tarjeta de desarrollo.

4.2.1. Aspectos generales de la tarjeta ZYBO y el circuito Zynq

Al principio del diseño del sistema embebido y la MDRO, se utilizó la tarjeta ML605 del fabricante Xilinx. La cual cuenta con un FPGA de la familia Virtex-6, memoria RAM (512 MB), conectividad PCIe, UART, 10/100/1000 Tri-Speed Ethernet entre otras características. Si bien no tiene un *hardprocessor* como la tarjeta Zybo, utilizada finalmente, sí se puede implementar un procesador RISC de 32 bits, el MicroBlaze.

Ahí se implementaron los primeros diseños de la MDRO, el capturador de video, el binarizador, el calculador de centroide y puntos frontera. El Microblaze controlaba esa versión primigenia de la MDRO. Un controlador de video (software-hardware) enviaba el *buffer* desde el capturador a la memoria. Y una rutina de software copiaba el *buffer* a una memoria BRAM. Esta BRAM se usaba para enviar los pixeles directamente al monitor VGA.

Uno de los principales problemas que se tenían con esa configuración, era la resolución limitada de la imagen, dado que la BRAM almacenaba todo el *buffer*. Al estar limitado el tamaño de la BRAM en función de la utilización y capacidades de la FPGA, el eventual incremento de la resolución de la imagen se vería sumamente limitado. Y peor aún, le resta espacio al resto de la lógica reprogramable.

Otro factor en contra es el hecho de que el software de control del SE comparte tiempo y recursos con el envío de video hacia el monitor. Ya que comparten el bus de datos, la memoria RAM y sobre todo el mismo procesador.

Por éstas razones se decidió utilizar otra tarjeta de desarrollo, la tarjeta Zybo del fabricante Digilent Inc. En el apéndice "Tarjeta Zybo"se muestran las especificaciones de éste dispositivo, sin embargo, en esta sección se abordan temas referidos de la tarjeta y el SE. La tarjeta Zybo integra un circuito de la familia Zynq-7000, el cual contiene un procesador ARM Cortex A9 y una FPGA de la familia 7 de Xillinx. En la figura C.1 se muestra la arquitectura del circuito integrado Zynq-7000 (XC7Z010-1CLG400C)

El Zynq está dividido en dos partes, el sistema de procesamiento (SP), ARM Cortex A9 y la lógica programable (LP), FPGA 7-series. Ambas partes están conectadas entre sí y con componentes de la Zybo, y la mayoría lo es a través del *bus* AMBA (*Advanced Microcontroller Bus Architecture*).

La MDRO está implementada en la parte de lógica programable. Y el software de control opera sobre el procesador Cortex A9. La figura 4.8 muestra el diagrama de conexión del SE con el sensor CCD y el monitor. Y al interior de la Zybo se observa el flujo de datos, a grandes rasgos, de todo el sistema.

A través del conector Pmod se unen los pines de control y datos del sensor de video OV7670 (para más información consultar el Apéndice "Sensor CCD OV7670"). Del lado de la LP se conecta a un módulo de configuración y adaptación de datos de video y de ahí tanto a la MDRO como al bloque VDMA (*Video Direct Memory Access* IN, datos de video de acceso directo a memoria) Es la forma en que tanto la MDRO como el SP reciben el video proveniente del sensor CCD.

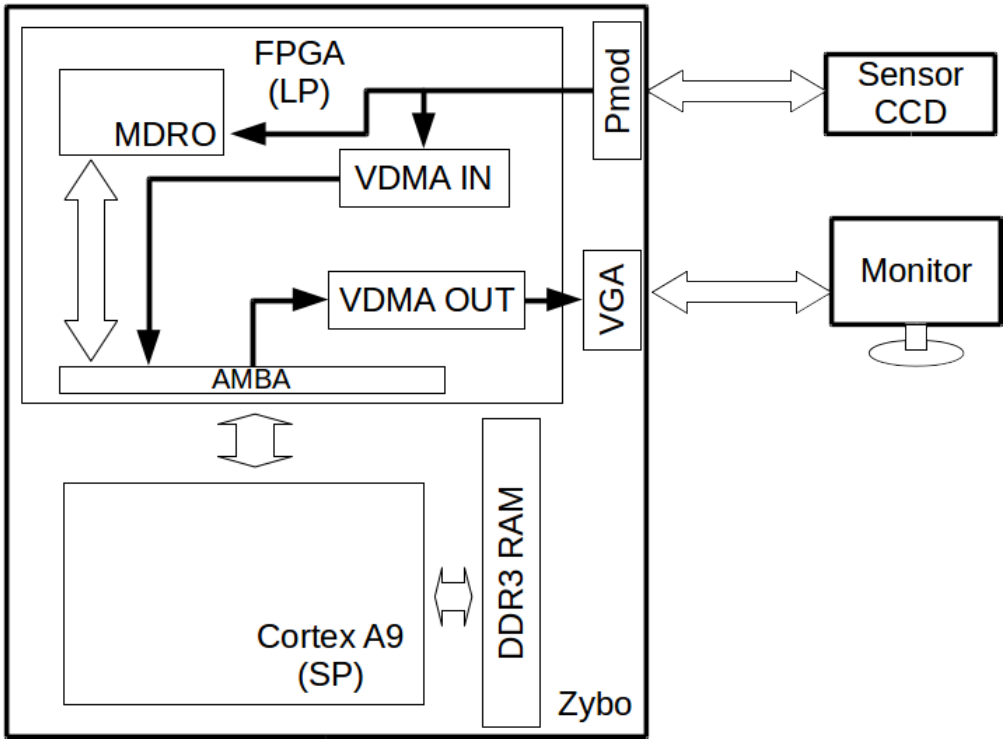

Figura 4.8: Diagrama del SE.

Por otro lado, el SP envía el *framebuffer* de video, almacenado en RAM, hacia el módulo VDMA OUT. Y de ahí al monitor conectado a través de una interfaz VGA de 15 pines.

De ésta manera, el VDMA es el encargado de gestionar el flujo de video desde el CCD hacia la memoria y a su vez, de la memoria hacia el monitor. El procesador queda libre para realizar otras funciones mientras se realiza ese proceso. Aunque al principio el SP configura los parámetros de las máquinas de estado de los módulos VDMA. Con esto, el hilo principal de ejecución del SP está destinado a comunicar, configurar y procesar datos relativos al proceso de reconocimiento y la MDRO.

Una ventaja significativa entre un ARM Cortex A9 sobre el Microblaze es que el primero es de mayor velocidad y mayor capacidad de procesamiento que el segundo.

4.2.2. Arquitectura del sistema embebido

En la figura 4.9 se muestra la arquitectura del SE. Se aprecia que todos los componentes, memoria RAM, MDRO, periféricos, VDMA, etc. están conectados al bus AXI, que es el encargado de comunicar los dispositivos con el procesador. A su vez lo hace también con la

memoria RAM.

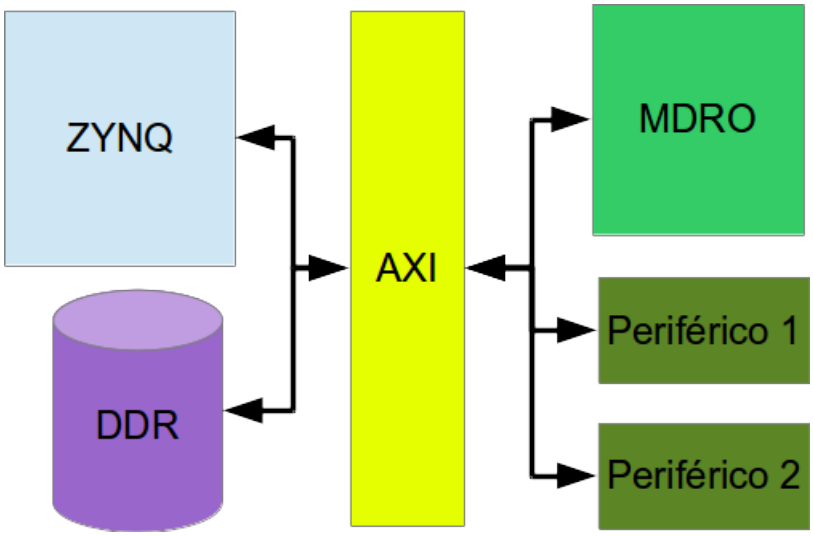

Figura 4.9: Arquitectura del SE

El procesador Cortex A9, un procesador de 32 bits de arquitectura tipo RISC, es el que comanda la operación del SE; tiene la función de configurar, controlar, administrar la memoria RAM para video y comunicar los resultados con el exterior del sistema.

El diagrama de conexión de todo el SE se muestra en la figura E.2 (Apéndice Sistema Embebido). Para este diagrama es irrelevante que partes corresponden a la LP o al SP dado que lo importante es mostrar la forma física de conexión entre componentes.

En la figura 4.10 está el inventario de módulos que componen al SE, así como la relación de dependencia con sus módulos maestros o esclavos de acuerdo al caso.

El módulo principal, processing_system7_0, es el procesador Cortex A9 que está integrado a la tarjeta ZYBO. Por defecto, utiliza una configuración estandar en la cual busca tener un ahorro de energía, por lo tanto, es necesario modificar los parámetros a fin de lograr los objetivos. Al principio del desarrollo del SE, el controlador del sensor OV76700, era un módulo desarrollado en VHDL, que se encargaba de configurar sus parámetros. Sin embargo, a medida que avanzaba el proyecto se prescindió de dicho módulo, en aras de disminuir el espacio en la FPGA, y fue necesario usar el módulo I2C para comunicarse con la cámara. De manera que fue necesario activar los registros de configuración del Cortex A. en las figuras C.2 y C.3 se muestra el esquema de configuración.

El procesador Cortex A9, realiza la comunicación DMA a través del módulo axi_mem_intercon (figura 4.11). El uso de DMA es prioritario para el funcionamiento del SE. Es a

ID	Nombre	Tipo	Maestro	Esclavo	Descripción
1	processing_system7_0	ZYNQ7 Processing System		2, 5	Procesador del SE. ARM
2	axi_mem_intercon	AXI Interconnect	1	3, 4	Interconexión AXI con RAM
3	axi_vdma_0	AXI Video Direct Memory Access	2, 6	8	VDMA de salida a VGA
4	axi_vdma_2	AXI Video Direct Memory Access	2, 6	7	VDMA de entrad de OV7670
5	axi_protocol_converter_0	AXI_Protocol_Converter	1	6	Covertidor de protocolo AXI 3 a AXI 4
6	processing_system7_0_axi_periph	AXI Interconnect	6	3, 4, 8, 9, 10, 11	Interconexión AXI con periféricos
7	ov7670_capture_0	OV7670_capture_v1	4	-	Módulo de captura de 0V7670
8	axi_dispctrl_0	AXI Display Controller	3	-	Módulo AXI a VGA
9	BTNs_4Bits	AXI_GPIO	6	-	Control de botones de ZYBO
10	LEDs_4Bits	AXI_GPIO	6	-	Control de leds de ZYBO
11	Sws_4Bits	AXI_GPIO	6	-	Control de interruptores de ZYBO
12	mdro_0	MDRO_V1	6	-	MDRO

Figura 4.10: Inventarios de módulos e interconexión del SE

través de éste, que el *framebuffer*, proveniente del sensor 0V7670, se almacena en la memoria RAM. El bus de datos M00_AXI proviene del procesador y tanto S00_AXI como S01_AXI, son los buses en modo esclavo que van hacia los módulos VDMA.

En la figura 4.12 se muestra la configuración del módulo axi_mem_intercon, el cual tiene 2 interfaces en modo esclavo y una intefaz en modo maestro.

Todos los módulos del SE son vistos como esclavos desde el punto de vista del procesador. Es por ello que todos los módulos deben estar conectados al bus de datos y direcciones del Cortex A9, incluido el controlador de DMA. Sin embargo, debido al cambio de versiones en el estandar AXI, de la versión 3 a la versión 4, algunos componentes de Xilinx no han sido actualizados a la nueva versión; es por ello que se utiliza un convertidor de protocolo. En la figura 4.13 se muestra el controlador de periféricos y su respectivo convertidor de protocolo.

El intermediario axi_protocol_converter_0 enlaza al procesador con el controlador pro-

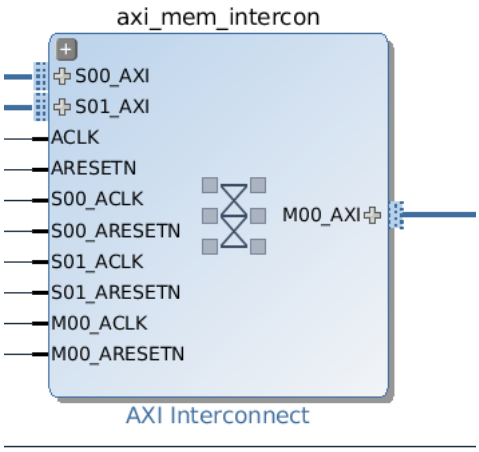

Figura 4.11: Memoria RAM a bus AXI

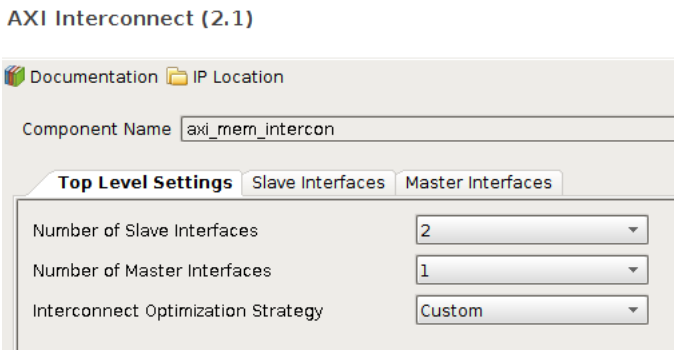

Figura 4.12: Configuración del bloque de interconexión de memoria RAM a bus AXI

cessing_system7_0_axi_periph, a través de un bus esclavo. Las salidas de éste último, van conectadas a cada periférico. Por ejemplo, el bus etiquetado como M00_AXI envía señales de datos, direcciones y control al módulo LEDs_4Bits, el cual consiste de un componente del tipo GPIO (*General Purpose Input/Output*) y que en este caso se trata de los leds de la tarjeta Zybo.

La configuración de processing_system7_0_axi_periph se muestra en la figura 4.14. Se muestra que se compone de una interfaz en modo esclavo, la proveniente de axi_protocol_converter_0, que a su vez es la misma del procesador, y 7 interfaces en modo maestro dirigidas a comandar todos los periféricos del SE.

Una opción muy importante que proporciona la herramienta Vivado (herramienta de Xilinx para el ensamblado de sistemas embebidos sobre FPGA) en el proceso de diseño, es definir la estrategia en el trazado de circuitos, se debe recordar que todo esto se realiza en

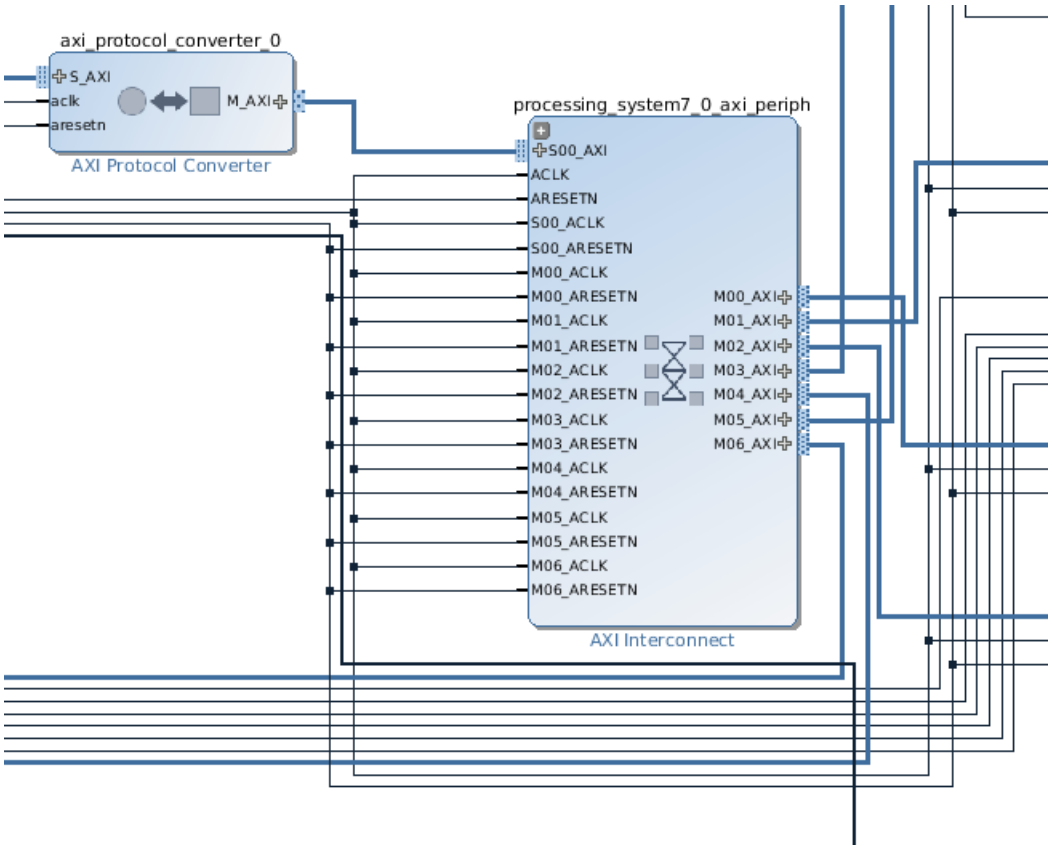

Figura 4.13: controlado de periféricos y el convertidor de protocolo

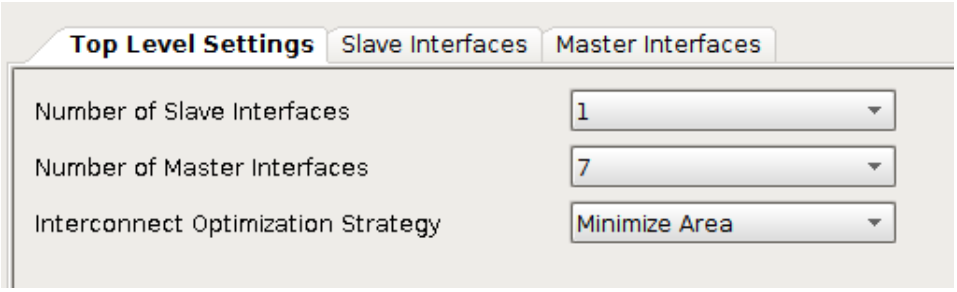

Figura 4.14: Configuración bus AXI del procesador con el resto de los periféricos

la parte configurable de la FPGA, la LP. Dicha estrategia busca reducir al mínimo, el área necesaria para implementar estos circuitos en la lógica interna de la FPGA.

Dicho lo anterior, conviene hacer mención del tamaño del SE que ocupa dentro de la FPGA, sin la MDRO. En la tabla 4.1 y la figura 4.15 se muestra el uso de los recursos de la FPGA para la implementación lógica de los componentes mencionados anteriormente. Claro, a excepción del procesador.

Recurso	Estimación	Disponible	% utilización
FF	7,869	35,200	22.36
LUT	5,975	17,600	33.95
Memoria LUT	454	6,000	7.57
I/O	57	100	57.00
BRAM	6	60	9.17
BUFG	7	32	21.88
MMCM	2	2	100.00

Tabla 4.1: Tabla del porcentaje de utilización del SE en la Zybo

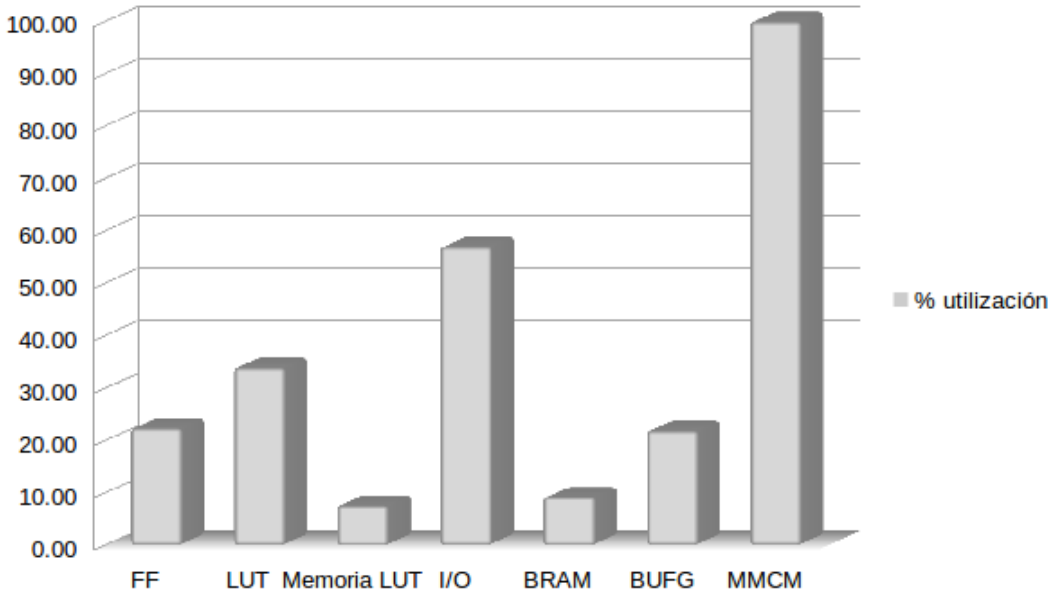

Figura 4.15: Gráfica del porcentaje de utilización del sistema embebido en la Zybo

En la figura se puede apreciar que hay disponibilidad de BRAM para incluir la MDRO. Lo mismo ocurre para los flip-flops (FF) y los LUT (*Lookup tables*).

Una vez ensamblados todos los módulos y realizadas las conexiones entre ellos, es necesario realizar la asignación de direcciones de memoria. Vivado lo realiza de forma automática buscando optimizar los recursos del chip Zynq, sin embargo, es posible modificar la dirección de acuerdo al diseño. En la figura 4.16 se muestra el mapa de las direcciones de acceso de cada módulo. Que serán de mucha importancia al momento del desarrollo del controlador de software que deberá configurar a cada componente.

Por último, se muestra el árbol de conexiones del procesador y sus módulos (figura 4.17),

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
axi_vdma_0					
Data_MM2S (32 address bits : 4G)					
processing_system7_0	S_AXI_HP0	HP0_DDR_LO...	0x0000_0000	512M	0x1FFF_FFFF
Data_S2MM (32 address bits : 4G)					
Data_SG (32 address bits : 4G)					
axi_vdma_2					
Data_MM2S (32 address bits : 4G)					
Data_S2MM (32 address bits : 4G)					
processing_system7_0	S_AXI_HP0	HP0_DDR_LO...	0x0000_0000	512M	0x1FFF_FFFF
Data_SG (32 address bits : 4G)					
processing_system7_0					
Data (32 address bits : 4G)					
axi_dispctrl_0	S_AXI	S_AXI_reg	0x43C0_0000	64K	0x43C0_FFFF
BTNs_4Bits	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
LEDs_4Bits	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF
SWs_4Bits	S_AXI	Reg	0x4122_0000	64K	0x4122_FFFF
axi_vdma_0	S_AXI_LITE	Reg	0x4300_0000	64K	0x4300_FFFF
axi_vdma_2	S_AXI_LITE	Reg	0x4301_0000	64K	0x4301_FFFF
axi_i2s_adi_1	S_AXI	S_AXI_reg	0x43C1_0000	64K	0x43C1_FFFF

Figura 4.16: Dirección de periféricos y memoria RAM

haciendo referencia al listado de puertos (*Port*), de redes (*Nets*), de interfaces externas (*External Interfaces*) y de conexiones (*Interface Connections*).

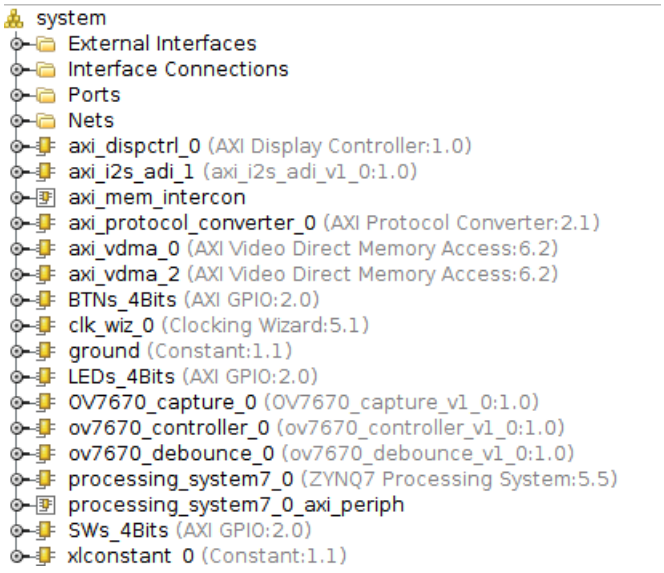

Figura 4.17: Listado de los componentes del sistema

4.2.3. Subsistema de video

El susbsistema de video tiene como objetivos principales recibir la señal de video proveniente del sensor, transmitirla a la MDRO y almacenar el **framebuffer** en RAM, para que posteriormente sea enviado a un puerto VGA y de ahí a un monitor. La última parte solo es

necesaria en caso de querer visualizar el video de entrada y algunas etapas del procesamiento de la señal.

Durante el proceso de construcción del SE-MDRO fue indispensable tener la salida de video conectada a un monitor, y con los botones tener interacción para cambiar entre el video original, el preprocesamiento, la imagen del histograma, la imagen de la BOF, entre otras vistas. Y es que para probar el funcionamiento del sistema, es necesario visualizar el procesamiento por etapas y en tiempo real. Sin embargo, una vez construido, no hace falta operar el video de salida y se podrían eliminar el bloque VDRAM y su correspondiente módulo de salida VGA.

En la figura 4.18 se muestra un esquema del fujo de datos que propone Xilinx y que se realizaron algunas modificaciones por parte de [Võsandi, 2015]. Para este diseño, se propone que la gestión de los recursos se lleven a cabo mediante el sistema operativo GNU/Linux, sin embargo, por el momento; no es del interés de este desarrollo incorporar tal sistema. Por lo que la configuración y gestón de recursos se hace a través del propio controlador de DMA y estableciendo las premisas de configuración a través de un **firmware** compilado en C.

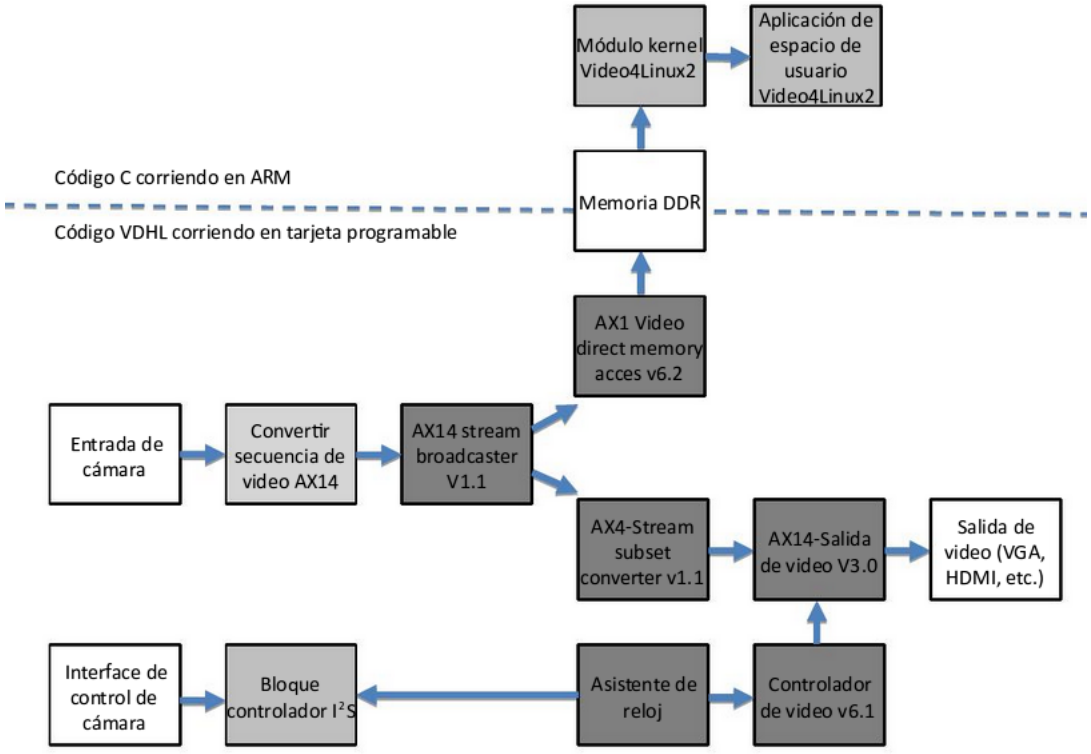

Figura 4.18: Diagrama a bloques de VDMA [Võsandi, 2015]

Lo que se propone es mantener en comunicación constante a los módulos VDMA con la memoria RAM, esto ocurre desde que termina la configuración de cada módulo en el arranque, hasta que el procesador decida terminar su operación. Cada VDMA recibe un *streaming* de datos de video, con cierto formato, ya sea desde la RAM, para el caso de la salida a VGA, o desde el sensor OV7670 (figura 4.19). El destino, para el primer caso, es el acondicionador de señal de VGA, el axi_dispctrl_0 y para el segundo caso, es la memoria RAM.

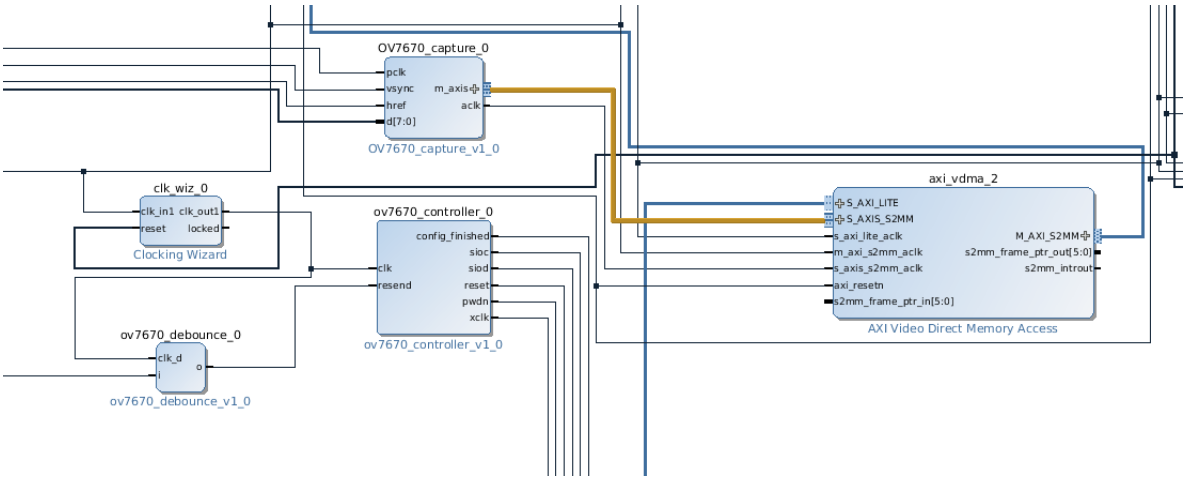

Figura 4.19: Cámara OV7670 a VDMA 2

En la figura 4.20 se muestra el módulo axi_vdma_0, el cual recibe el control desde processing_system7_0_axi a través del bus S_AXI_LITE. Pero, la comunicación de datos de video hacia la RAM se realiza por el bus M_AXI_S2MM que conecta con el módulo axi_mem_intercon. Es axi_vdma_0 el que recibe los datos de la RAM y los envía a VGA, por el contrario, axi_vdma_2 lo toma de módulo OV7670_capture_0 y lo escribe en RAM.

La configuración del módulo axi_vdma_0 se muestra en las figuras 4.21 y 4.22. En ellas se establecen parámetros relacionados con el formato del dato y del tipo de sincronización.

El formato de datos es muy importante, existe un protocolo de comunicación para el envío de *streaming* de video. El cual lleva, entre otras señales, las de sincronía horizontal, sincronía vertical y los valores de cada pixel. Pero también establece el tamaño, ancho y la tasa de envío de datos. Esto es por que independientemente de las funciones que realice el procesador, VDMA copia o extrae datos de la RAM por ráfagas.

Lo primero que se establece es si se trata de un canal de lectura o escritura. El axi_vdma_0 es de lectura. El parámetro *Frame Buffers* establece el número de veces que está replicado un cuadro completo en la memoria RAM. A mayor número, menos posibilidades de copiar o

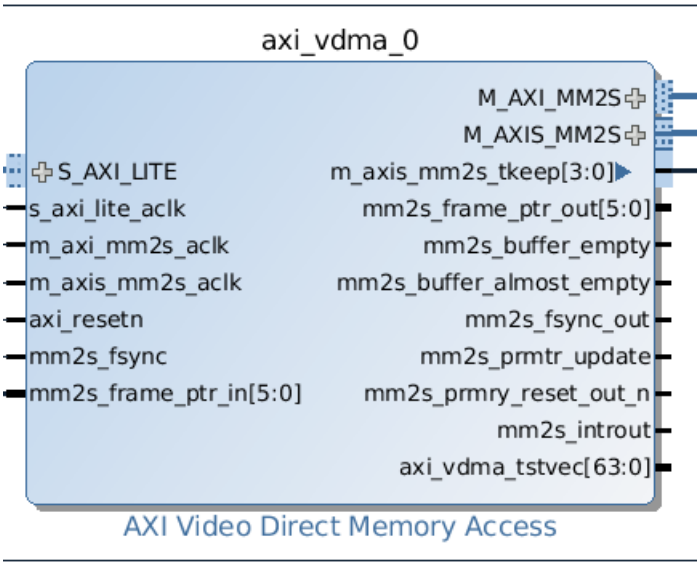

Figura 4.20: AXI VDMA bloque

pegar, según sea el caso, un pixel con el valor desactualizado. Sin embargo, se consume más RAM. Los otros parámetros están relacionados con el tamaño y profundidad de la imagen adquirida por el sensor OV7670.

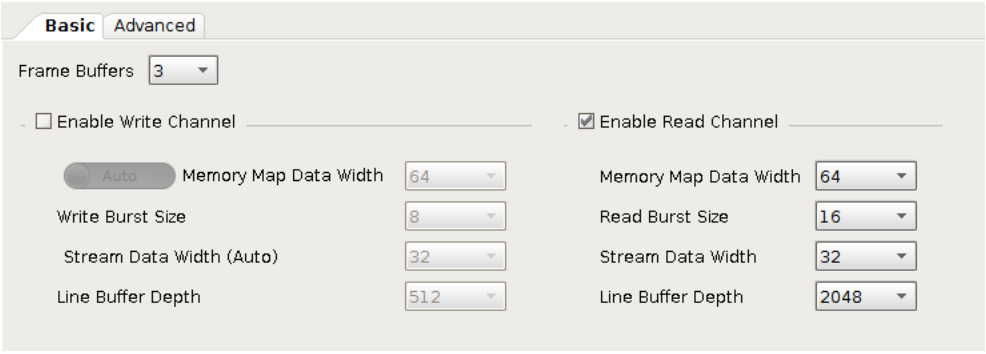

Figura 4.21: AXI VDMA configuración 1

La sincronización del bus VDMA la lleva el controlador mm2s y su modo de configuración es esclavo.

Por último se tiene el diagrama a bloques del módulo ov7670_capture_0 en la figura 4.23. Se trata de una modificación al diseño de [Field, 2013]. Éste módulo se encarga de obtener la imagen proveniente del sensor OV7670 a través de sus pines de salida de datos.

En el apéndice CCD OV7670 se pueden observar las características del dispositivo, así como el diagrama de tiempos para obtener un *frame*. Las señales HREF y VSYNC, provie-

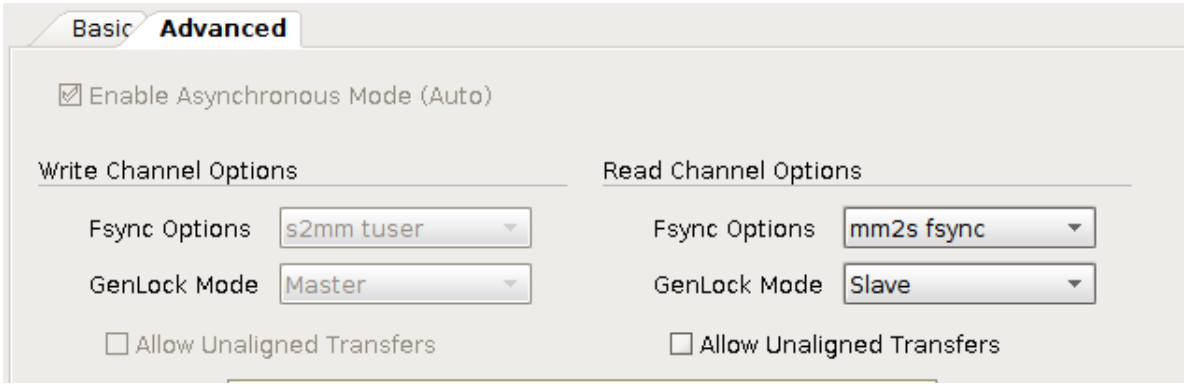

Figura 4.22: AXI VDMA configuración 2

nen del sensor, una a la sincronía horizontal, es decir cada línea de la imagen y la otra la sincronía del cuadro. En el módulo se acondicionan estas señales para hacerlas compatibles con AXI_4_STREAM.

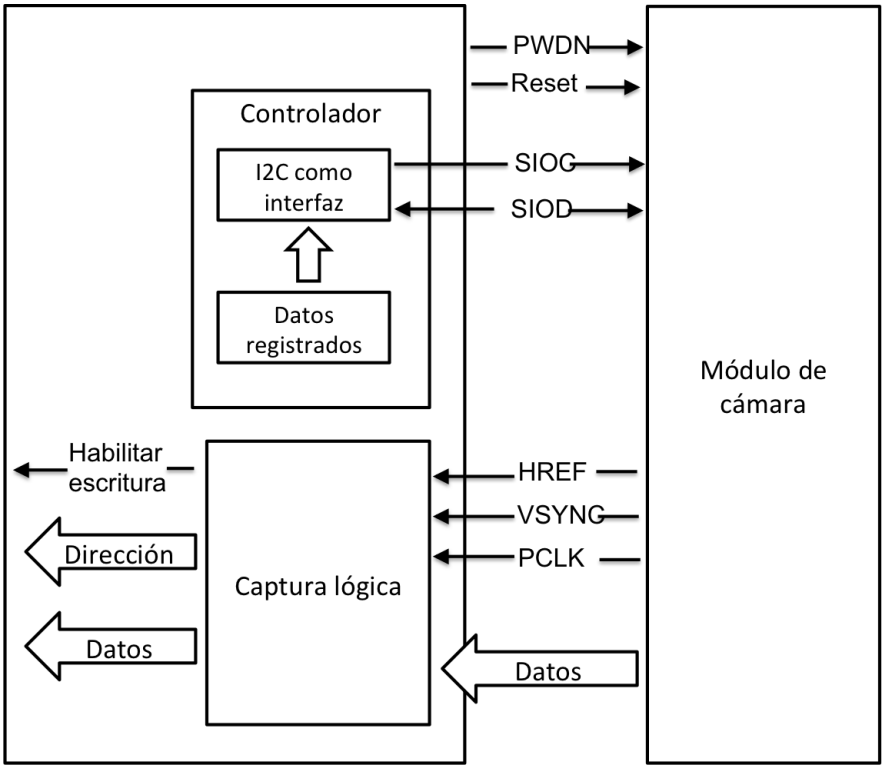

Figura 4.23: Capturador de video a bloques

Por otro lado, cada pixel es enviado a través de un puerto de 8 bits, y dependiendo de la configuración escogida envía primero una parte del pixel y después otra. En la modali-

dad RGB (5:6:5), se completan los bits suficientes para hacerlos compatibles con AXI_4_ STREAM, RGBA (8:8:8:8). Donde R es para rojo, G para verde, B para azul y A para el canal alfa (transparencia).

El diseño original tiene una máquina de estados para configurar la cámara, sin embargo, para ahorrar espacio, la comunicación se hace a través de un puerto I2C directamente desde el procesador CortexA9.

Capítulo 5

Pruebas y resultados

En éste capítulo se presentan las pruebas y los resultados que se realizaron para validar los objetivos establecidos. Como se planteó en la metodología, las pruebas y sus resultados están en función de realizar los procesos intermedios para el reconocimiento de objetos en diversas CPG y con diversos lenguajes de programación a fin de comparar la eficiencia de la MDRO.

Las diferentes pruebas se realizaron por subprocesos y al final para el proceso completo de reconocimiento. En las siguientes secciones se presentan los resultados, y su metodología para obtenerlos, de acuerdo al proceso de reconocimiento: cálculo de la BOF y posteriormente su clasificación a través de la RNA.

Para determinar el número de repeticiones de la prueba se optó por realizar 1000 veces dado que en la literatura [Vanhoucke, 2011] se observó que el promedio de tiempo obtenido en este tipo de pruebas no varia considerablemente entre 5 y mas repeticiones.

Las pruebas se realizaron ejecutando el algoritmo en tres diferentes plataformas, descritas en el cuadro 5.1. En las configuraciones CPU A y CPU B se desactivaron la mayoría de los procesos que son ajenos al funcionamiento del sistema operativo o a las pruebas para el reconocimiento de objetos.

***Nota**: Los datos mostrados para el SE y la MDRO son sólo para referencias, recordar que no es en el procesador Cortex A9 donde se realizan los cálculos.

El proceso de reconocimiento para las pruebas en CPU A y CPU B se ejecutó en 2 diferentes lenguajes de programación, C++ y Matlab. Para C++ el compilador es **gcc** versión 4.9. y para Matlab se usó la versión 17.

Características/Plataforma	CPU A	CPU B	S.E. MDRO*
Procesador	Core i7 3610QM	Core i7 2600S	Cortex A9
# núcleos	8	8	2
Velocidad	2.3 [GHz]	2.8 [GHz]	650 MHZ
Memoria	8 [GB]	8 [GB]	512 [MB]
Ancho de bus	64 [bits]	64 [bits]	32 [bits]
Sistema operativo	GNU/Linux 3.13	GNU/Linux 3.13	Firmware

Tabla 5.1: Tiempo de procesamiento para cálculo de frontera

En general, para ambas pruebas en CPU A y CPU B, el procesamiento de la imagen se resume a cargar la imagen en memoria RAM y hacer algunas visualizaciones. El resto del procesamiento se hace sobre arreglos. Para C++ se utilizó OpenCV 3.1 y para Matlab la biblioteca Graphics.

En todos los casos siempre se uso la misma imagen, la cual tiene una resolución de 640 x 480 pixeles en escala de grises, es decir un solo canal. La imagen se convierte a escala de grises en cada iteración (no se toma en cuenta el tiempo utilizado) para las pruebas en los CPU A y B. Para las pruebas en el SE-MDRO se utilizó el código 4 para crear un archivo de texto con el contenido de la imagen binaria. El formato del archivo de texto es el siguiente: cada pixel binario está representado por **0** o **1** en forma de caracter. Cada renglón es un pixel binario, de manera que se tienen 307,200 lineas. El simulador Isim de Xilinx lee la imagen binaria, presente en el archivo de texto, y la introduce a la MDRO.

```
...
ofstream archivo;
archivo.open ("imagenBinaria.txt");
for(i=0; i < imagenArregloBinario.size(); i++ ){
 if(imagenArregloBinario[i]==255){
  archivo << 0 << "\n";
 }else{
  archivo << 1 << "\n";
 }
}
archivo.close();
```

Código 4: Segmento del código en C++ para crear un archivo de texto a partir de una imagen binaria

5.1. Tiempo promedio de ejecución

El número de iteraciones realizadas en cada prueba fue de 1,000 veces. Salvo en el caso del SE-MDRO donde sólo se realizó una iteración, ya que el procesamiento en la MDRO no sufre ningún tipo de interrupción mientras el *framebuffer* proporcione datos. Por tanto, no hay lapsos de tiempo sin operación. A diferencia de las pruebas con CPU A y CPU B, que pueden ver interrumpida su ejecución por cualquier llamada al sistema proveniente de algún periférico o incluso del mismo sistema operativo.

El código 5 es un extracto de la forma en que se obtuvo el tiempo que transcurre entre el inicio y fin de un proceso.

```
clock_t tInicio, tDiferencia;
double tTotal;
...
tInicio = clock();
...
...
tDiferencia = clock() - tInicio;
tTotal = (double)(0.001 * ( (double)tDiferencia
 / (double)CLOCKS_PER_SEC ));
```

Código 5: Segmento de código en C++ para calcular el tiempo transcurrido de un proceso

5.2. Captura de un frame

El tiempo requerido para la carga de un *frame* de 640 X 480 pixeles se muestra en la figura 5.1.

CPU A		**CPU B**		**MDRO**
C++	**Matlab**	**C++**	**Matlab**	
9.83	15	9.6	14	6.144

Tabla 5.2: Tiempo de captura de video de un *frame* de resolución de 640 x480 pixeles

El código empleado en C++ para la carga reiterativa de *frames* se muestra en el listado 6.

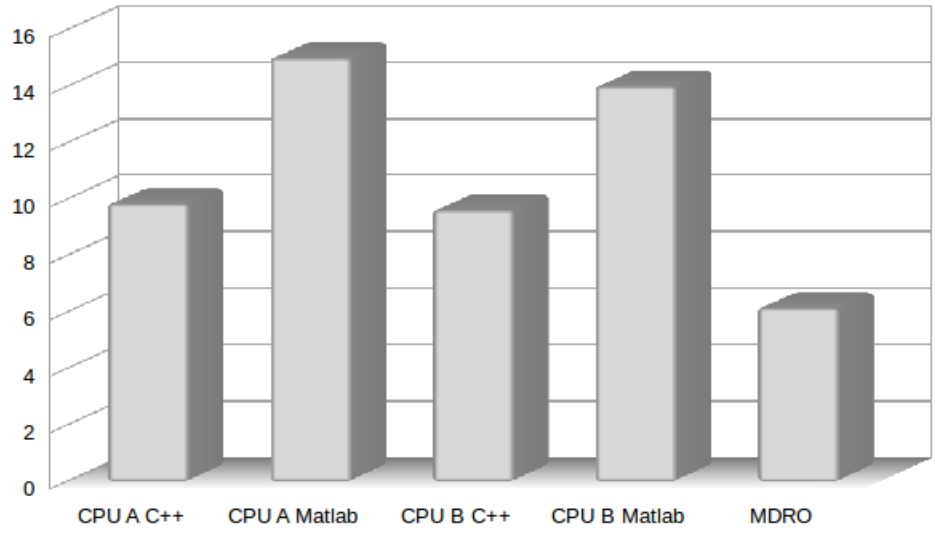

Figura 5.1: Comparación de tiempo de captura de un *frame* de 640 x 480 pixeles en [ms]

5.3. Cálculo de BOF

Es necesario recalcar que uno de los criterios importantes que se usaron para éste diseño fue el de optimizar el consumo de recursos físicos, en este caso se trata del espacio utilizado en la memoria RAM, donde finalmente se almacena temporalmente la imagen o el cuadro a procesar. La biblioteca OpenCV lee las imágenes y las convierte en un objeto llamado **Mat**. El cual administra eficientemente el alojamiento de los datos de la imagen en memoria [OpenCV, 2015b]. Dejando en manos del compilador el apartado o liberación de memoria según sea requerido.

La forma en que OpenCV realiza el alojamiento de una imagen en memoria depende de su tamaño, compresión, resolución y cantidad de canales [OpenCV, 2015a]. Generalmente almacena los renglones de forma continua, de manera que el acceso es unidimensional. En el código 7 se muestra la conversión del objeto Mat en un vector unidimensional, ya sea que el almacenamiento haya sido continuo o no. Al transformar la imagen a color a una imagen simplemente binaria se requieren ancho x alto localidades de memoria. Y sólo se usa un bit de cada localidad para representar si el pixel forma parte del objeto o no.

```
...
VideoCapture cap(0);
if(!cap.isOpened())
 return -1;
cap.set(CV_CAP_PROP_FRAME_WIDTH, 640);
cap.set(CV_CAP_PROP_FRAME_HEIGHT, 480);
Mat imagen_GS;
Mat imagen_BIN;
for(;;){
 Mat frame;
 cap >> frame;
 cvtColor(frame, imagen_GS, CV_BGR2GRAY);
 imagen_BIN = imagen_GS > umbral;
}
```

Código 6: Segmento del código en C++ para captura de video.

```
char* imageName = argv[1];
Mat imagen_BIN;
std::vector<uchar> imagenArregloBinario;
...
imagen_GS = imread( imageName, CV_LOAD_IMAGE_GRAYSCALE );
imagen_BIN = imagen_GS > umbral;
...
if (imagen_BIN.isContinuous()) {  //Es continua
    imagenArregloBinario.assign(imagen_BIN.datastart,
     imagen_BIN.dataend);
} else {
    for (i = 0; i < imagen_BIN.rows; ++i) {  //No es continua
        imagenArregloBinario.insert(imagenArregloBinario.end(),
         imagen_BIN.ptr<uchar>(i),
         imagen_BIN.ptr<uchar>(i)+imagen_BIN.cols);
    }
}
```

Código 7: Segmento del código en C++ para cargar una imagen en memoria RAM

5.3.1. Obtención de los puntos frontera y centroide

En ésta sección se presenta el código en C++ y los resultados de comparación con la MDRO para la extracción de puntos frontera, cálculo del centroide y obtención de la BOF.

En el código 8 se muestra la inicialización del arreglo donde se almacenará la imagen

convertida a pesos, destacando con 0 el fondo del objeto, mayor que 0 la frontera y 9 la parte solida del objeto. El direccionamiento del vector imagenTransformacionPesos se realiza a través de una transformación de coordenadas, pues no hay que olvidar que el almacenamiento es unidimensional. La dirección de los 9 pixeles circundantes se calcula en un proceso similar.

```
...
int Ax = 0, Ay = 0, b = 0, xc, yc;  //variables centroide
int puntosFrontera[2000][2];
int contadorPuntosFrontera = 0;
std::vector<int>::size_type sz =
 imagenArregloBinario.size();
std::vector<int> imagenTransformacionPesos
 (imagenArregloBinario.size());
...
for( i = 0; i < imagen_BIN.rows; i++ ){
 for(  j = 0; j < imagen_BIN.cols; j++ ){  //Matriz de pesos
  imagenTransformacionPesos[j+1+(i+1)*imagen_BIN.cols]=
   imagenArregloBinario[(j+0)+(i+0)*imagen_BIN.cols]+
   imagenArregloBinario[(j+1)+(i+0)*imagen_BIN.cols]+
   imagenArregloBinario[(j+2)+(i+0)*imagen_BIN.cols]+
   imagenArregloBinario[(j+0)+(i+1)*imagen_BIN.cols]+
   imagenArregloBinario[(j+1)+(i+1)*imagen_BIN.cols]+
   imagenArregloBinario[(j+2)+(i+1)*imagen_BIN.cols]+
   imagenArregloBinario[(j+0)+(i+2)*imagen_BIN.cols]+
   imagenArregloBinario[(j+1)+(i+2)*imagen_BIN.cols]+
   imagenArregloBinario[(j+2)+(i+2)*imagen_BIN.cols];
  if(imagenTransformacionPesos[j+1+(i+1)*imagen_BIN.cols]>0){
   Ax = Ax + j;
   Ay = Ay + i;
   b = b + 1;
  }
  if(imagenTransformacionPesos[j+1+(i+1)*imagen_BIN.cols] == 3){
        puntosFrontera[contadorPuntosFrontera][0] = j;
        puntosFrontera[contadorPuntosFrontera][1] = i;
        contadorPuntosFrontera++;
  }
 }
}
xc= floor(Ax / b);        //calcula centroide
yc= floor(Ay / b);
```

Código 8: Segmento del código en C++ para la obtención de puntos frontera

Obsérvese que es necesario realizar el mapeo de 9 direcciones por cada pixel que se calcula. Si la imagen es de 640 x 480 pixeles, se realizan 2,764,800 accesos a memoria.

Resultados

La figura 5.2 muestra el resultado de una simulación del cálculo de centroide y puntos frontera en la MDRO. El tiempo de procesamiento fue de 6.144 [ms]

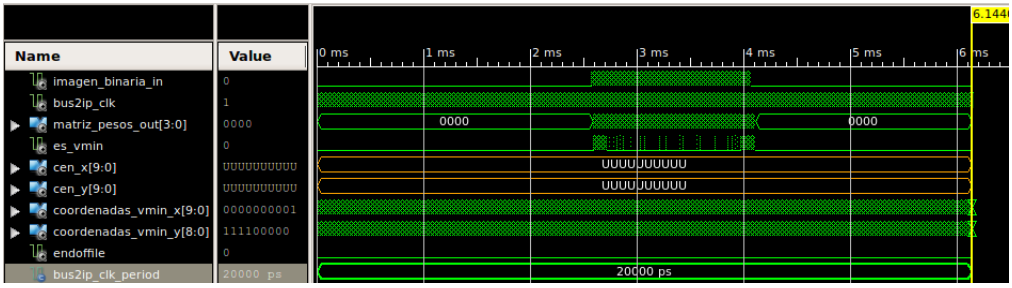

Figura 5.2: Diagrama de tiempo para obtener el centroide y la Matriz de tiempos de la MDRO

En la figura 5.3 están los resultados de las pruebas para el cálculo del centroide y la extracción de los puntos frontera. Contrasta el resultado de la MDRO contra el tiempo obtenido en los CPU A y CPU B.

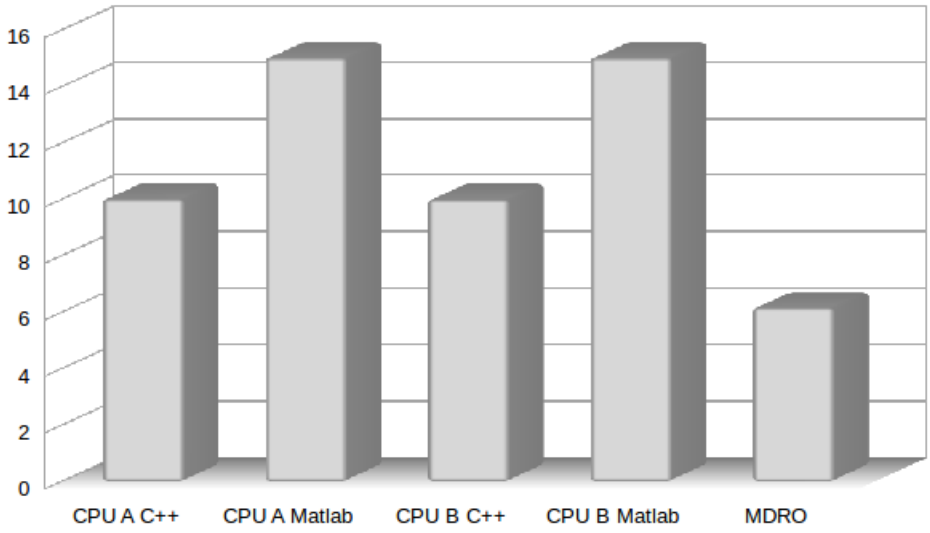

Figura 5.3: Comparación de tiempo de procesamiento para el cálculo de frontera y centroide en [ms]

CPU A		CPU B		MDRO
C++	Matlab	C++	Matlab	
10	15	9.98	15	6.144

Tabla 5.3: Tiempo de procesamiento para cálculo de frontera y centroide en [ms]

5.3.2. Cálculo de distancia y ángulo de la BOF

Tanto en la MDRO como en la implementación del algoritmo en C++ y Matlab es necesario realizar la obtención de la BOF en dos partes. Ya que se requiere el centroide para calcular el ángulo de cada punto frontera con respecto al centroide y su distancia. El código 9 realiza iteraciones sobre el vector puntosFrontera para obtener uno a uno el ángulo y su distancia. El ángulo es la dirección de un arreglo de 180 elementos y la distancia el valor de cada registro. Dicho arreglo se trata del almacenamiento de la BOF.

```
...
double bof[180] = {};
double maxDistanciaBof = 0;
double angBOF, distanciaBOF;
...
for( i = 0; i < contadorPuntosFrontera; i++ ){
 angBOF = atan2((double)(puntosFrontera[i][1] - yc) ,
      (double)(puntosFrontera[i][0] - xc)) * 180 / PI;
 distanciaBOF = sqrt( pow( puntosFrontera[i][0] - xc, 2) +
      pow( puntosFrontera[i][1] - yc, 2));
 if(angBOF < 0)
  angBOF = angBOF + 360;
 bof[(int)round(angBOF/2)] = distanciaBOF;
 if(maxDistanciaBof < distanciaBOF)
  maxDistanciaBof = distanciaBOF;
}

for(i = 0; i < 180; i++)
 bof[i] = bof[i]/maxDistanciaBof;
```

Código 9: Segmento del código en C++ para el cálculo de la distancia y ángulo de la BOF

Posteriormente es necesario normalizar la BOF por lo que se utiliza un bucle *for*.

Resultados

La simulación mostrada en la figura 5.4 muestra el diagrama de tiempo para obtener el ángulo y la distancia de los vectores que componen la BOF. El proceso se realiza dentro de la MDRO y tarda 2.010 [ms].

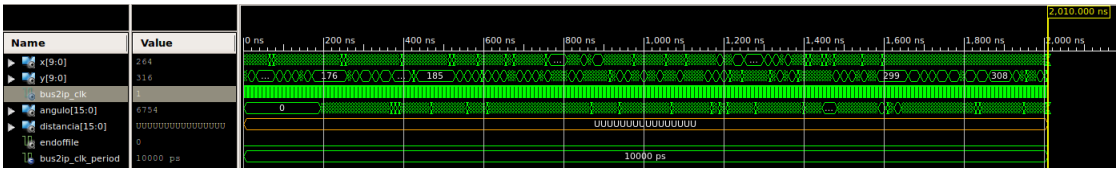

Figura 5.4: Diagrama de tiempo para calcular el ángulo y la distancia en la MDRO

Finamente tenemos la figura 5.5 de la comparación de tiempos para ésta etapa.

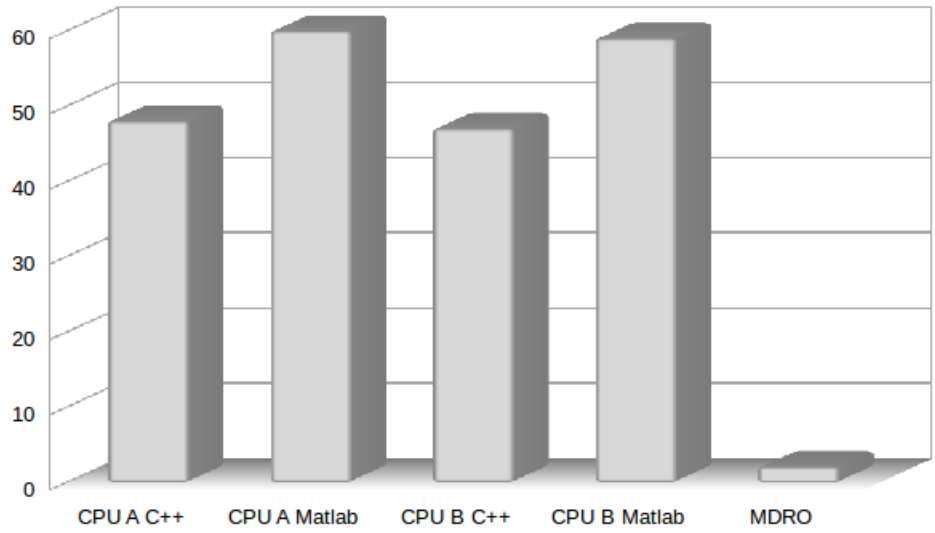

Figura 5.5: Comparación de tiempo de cálculo de distancia y ángulo de la BOF en [us]

CPU A		CPU B		MDRO
C++	Matlab	C++	Matlab	
48	60	47	59	2.01

Tabla 5.4: Tiempo de cálculo de distancia y ángulo de la BOF en [us]

5.4. Clasificador Fuzzy ARTMAP

A continuación se presentan los resultados de tiempo de procesamiento tanto para la obtención de las categorías T_j y la validación de la categoría ganadora a través del proceso de resonancia.

5.4.1. Obtención de los T_j

El código 10 y el código 11 muestran el proceso en C++ y Matlab para llevar a cabo el cómputo de las categorías T_j respectivamente.

```
...
int numeroCategorias = 4;
int longitudPesos = 360;
double pesos[numeroCategorias][longitudPesos];
double BOFClasificar[longitudPesos];
double suma=0;
double alfa = 0.001;
double T_j[numeroCategorias];
...
for (j=0; j < numeroCategorias; j++){
 suma = 0;
 for (i = 0; i <  longitudPesos; i++)
  suma = suma + min(BOFClasificar[i], pesos[j][i]);
 T_j[j]  = suma/(alfa + longitudPesos/2);
}
sort(T_j, T_j + numeroCategorias);
```

Código 10: Segmento del código en C++ para la obtención de los T_j

```
...
vActivados = ones(1,length(red.pesos));
for i=1:length(red.pesos)
 vActivados(i)  = sum(min(entrada,red.pesos{i}))/...
                    (red.alfa + sum(red.pesos{i}));
end
[vActivadosOrdenados, indicesOrdenados] =
            ...sort(vActivados,'descend');
```

Código 11: Segmento del código en Matlab para la obtención de los T_j

Resultados

En la figura 5.6 se muestra la comparación de tiempos para la obtención de los T_j en [ms].

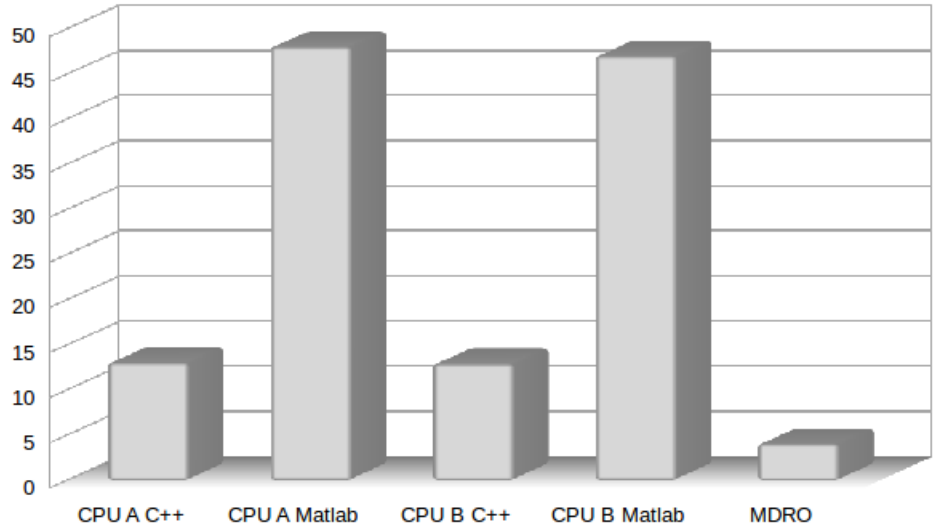

Figura 5.6: Comparación de tiempo de obtención de los T_j [ms]

CPU A		CPU B		MDRO
C++	Matlab	C++	Matlab	
13	48	12.88	47	4

Tabla 5.5: Tiempo de obtención de los T_j [ms]

5.4.2. Resonancia

El código 10 muestra el proceso en C++ para llevar a cabo el cómputo de las categorías T_j.

```
...
int numeroCategorias = 4;
int longitudPesos = 360;
double pesos[numeroCategorias][longitudPesos];
double BOFClasificar[longitudPesos];
double suma=0;
double alfa = 0.001;
double T_j[numeroCategorias];
...
for (j=0; j < numeroCategorias; j++){
 suma = 0;
 for (i = 0; i <  longitudPesos; i++)
  suma = suma + min(BOFClasificar[i], pesos[j][i]);
 T_j[j]  = suma/(alfa + longitudPesos/2);
}
```

Código 12: Segmento del código en C++ para la obtención de los T_j

```
clasificacion(s)=-1;
for p=indicesOrdenados
 match = sum(min(entrada,red.pesos{p}))/red.D;
 if match>=red.fVigilancia
  clasificacion(s) = red.etiquetas(p);
  if ~isempty(etiquetas)
  if etiquetas(s)==clasificacion(s), hits = hits + 1; end;
 end
end
```

Código 13: Segmento del código en Matlab para verificar si hay resonancia

Resultados

En la figura 5.7 se muestra la comparación de tiempos para verificar si hubo resonancia, el tiempo está en [ms].

CPU A		CPU B		MDRO
C++	Matlab	C++	Matlab	
39.5	50	39	49	4

Tabla 5.6: Tiempo empleado para verificar si hubo resonancia [us]

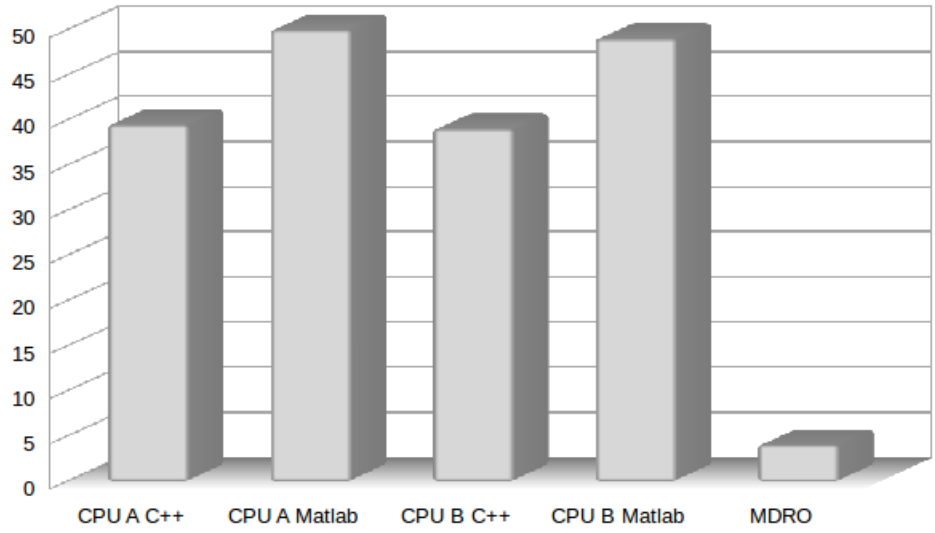

Figura 5.7: Comparación de tiempo empleado para verificar si hubo resonancia [us]

5.5. Tiempo total de reconocimiento

En la figura 5.8 se muestra la comparación de tiempos totales de reconocimiento con 4 categorías.

En la tabla 5.5 se muestran los tiempos por etapa y el tiempo de reconocimiento total. Cabe destacar que para el caso de la MDRO, el tiempo total no se considera el tiempo de captura, pues ocurre simultáneamente con la etapa de obtención de puntos frontera y cálculo de centroide.

	Captura	**Puntos frontera y Cxy**	**BOF**	**Tj**	**Resonancia**	**Total**
CPU A C++	9.83	10	0.048	13	39.5	72.378
CPU A Matlab	15	15	0.06	48	50	128.06
CPU B C++	9.6	9.98	0.047	12.88	39	71.507
CPU B Matlab	14	15	0.059	47	49	125.059
MDRO	6.144	6.144	0.00201	4	4	14.14601

Tabla 5.7: Tiempos por etapa y tiempo total de reconocimiento [ms]

Y por último, en la figura 5.9 se muestra el tiempo total empleado en cada máquina en el reconocimiento de un objeto con aprendizaje de 4 categorías.

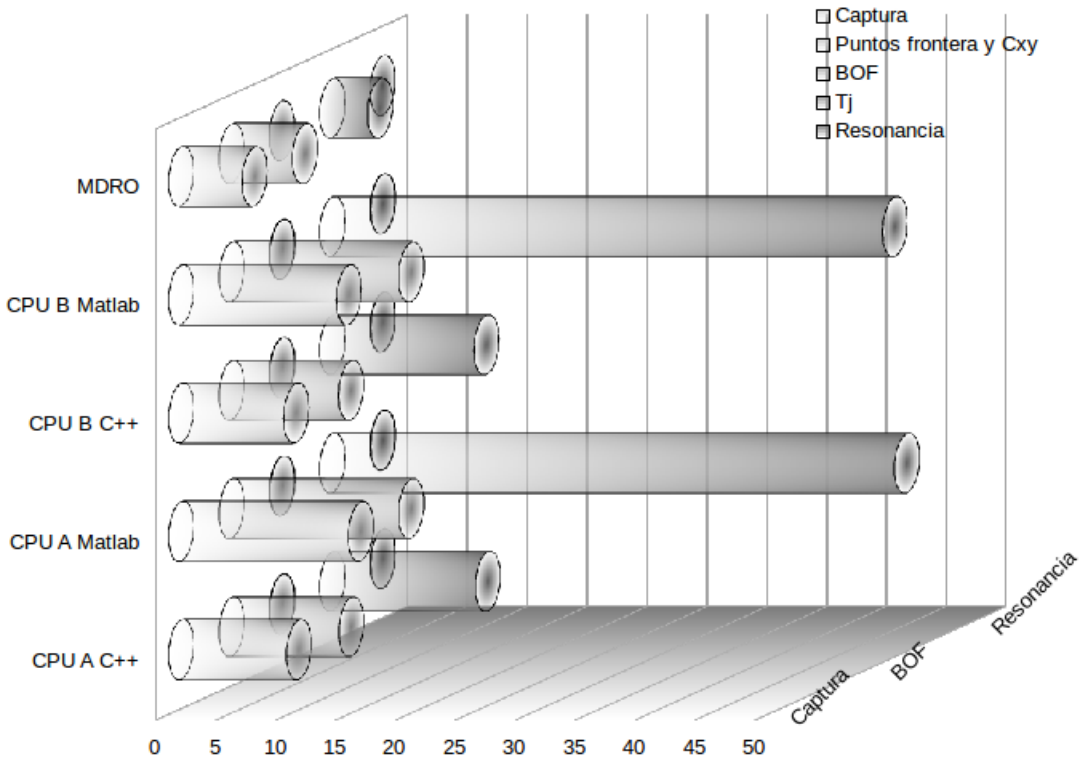

Figura 5.8: Comparación de tiempos por etapas con 4 categorías [ms]

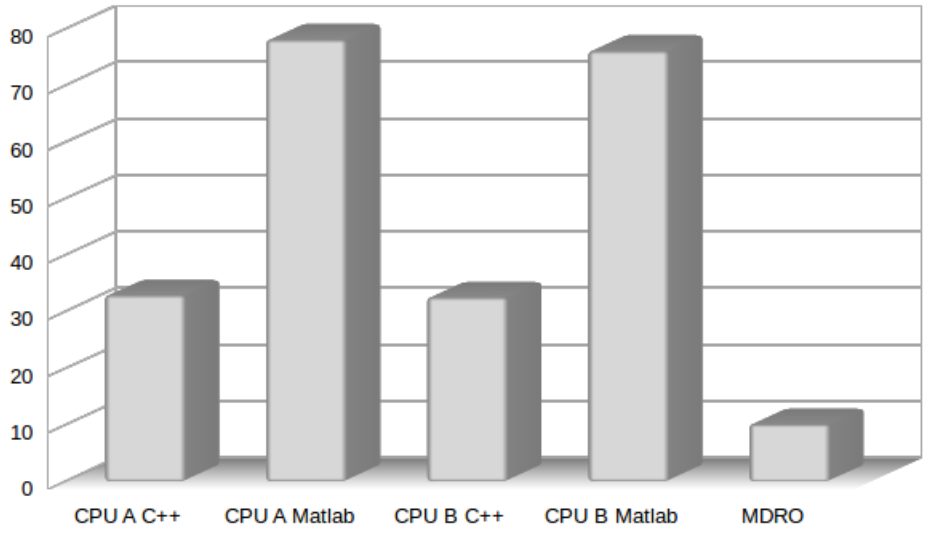

Figura 5.9: Comparación de tiempos totales de reconocimiento con 4 categorías [ms]

Capítulo 6

Conclusiones

Se logró realizar el diseño de la arquitectura del SE-MDRO, de tal manera que se obtuvieron resultados muy favorables de acuerdo a la hipótesis planteada.

Como resultado del diseño de la arquitectura de este sistema embebido se aprecian tres características fundamentales para resolver el problema planteado en este trabajo. La primera tiene que ver con el cálculo en linea de la BOF, la segunda el diseño de la red Fuzzy ART MAP como máquina digital y la tercera, no menos importante, la integración de todos sus componentes como sistema embebido de reconocimiento de objetos para ensamble en una celda de manufactura.

6.1. BOF en la FPGA

En ésta sección se mencionan las conclusiones relativas al preprocesamiento de la imagen y posteriormente a la obtención de la BOF.

6.1.1. Captura de video

En el campo de la automatización, la mayoría de las cámaras para procesamiento de vídeo se conectan a través de un puerto/protocolo hacia la unidad de procesamiento de una CPG. La información es transmitida desde un controlador hacia la memoria RAM y de ahí al tratamiento de la información en específico. Una ventaja importante del SE aquí presentado sobre estos sistemas es el reconocimiento en línea, es decir, no se almacena la imagen en

memoria RAM para procesarla.

Como ya se ha mencionado antes, este sistema embebido obtiene el video de forma cruda proveniente de la cámara. La primer ventaja con respecto a un CPG es el hecho de que sin intermediarios, la cámara se comunica directamente al SE, tal cual el valor de los píxeles fluye a través de un puerto hacia la MDRO donde empiezan a ser procesados. En el capítulo de resultados se observa que el SE-MDRO en casi dos tercios del tiempo que tarda en capturar una CPG usando OpenCV; y que al almacenar la imagen en la CPG se utilizan alrededor 19 [Mb] a diferencia del SE-MDRO.

6.1.2. Etapa de binarización

La binarización se realiza en línea, en cada ciclo de reloj el píxel es evaluado y de acuerdo al umbral se considera si forma parte del objeto o no. No es necesario almacenar toda la imagen, en RAM o en disco duro, para realizar este proceso. En la CPG, el algoritmo recorre toda la imagen almacenada en memoria RAM para discretizarla.

Al estar binarizada la imagen, solo es necesario un bit por píxel para representarla. Otra ventaja del FPGA sobre la CPG es que con un registro de 1x1 se almacena el valor de un píxel. En la CPG, se puede utilizar una localidad de memoria (de 64 bits) para representar cada píxel binarizado. Lo cual resulta en un desperdicio del 98 % de la memoria ya que para una imagen de 640 x 480 píxeles se requieren 307,200 direcciones.

No sólo la memoria es mal administrada, el tiempo de cómputo también lo es. Si la imagen original es almacenada en memoria RAM, se requieren 307,200 ciclos de instrucción (para el caso de la imagen de 640 x 480 píxeles) para comparar píxel a píxel con el umbral. Lo anterior es únicamente válido si se considera que tiene una estructura de *pipeline*, es decir que en el mismo ciclo de reloj se compara y almacena el nuevo valor en la misma dirección. De lo contrario requerirá al menos de dos ciclos de instrucción por cada píxel.

En la arquitectura de la MDRO también es necesario recorrer la imagen entera, sin embargo, lo que la hace más veloz en esta etapa del procesamiento, con respecto al CPG, es el hecho de que lo hace una sola vez y desde el principio.

6.1.3. Matriz de transformaciones de pesos y centroide

El cálculo de la Matriz de transformaciones de pesos y la obtención del centroide en la MDRO también se realiza en línea. No es necesario almacenar toda la imagen original ni mucho menos la binaria, solo se requiere rellenar un registro FIFO de 637 x 3 bits más un registro del tipo std_logic_vector de 3 x 3 bits. Una vez transcurridos 1,920 ciclos de reloj, es decir el procesamiento de (637 x 3) pixeles, al tener el 0.625 % de la imagen, ya se empieza la obtener la primer suma del *kernel* de 9x9; y por consiguiente se determina si ese píxel es frontera o no.

El cálculo de la Matriz de transformaciones de pesos es sin duda más eficiente en la MDRO que en la CPG. Dado que al estar la imagen almacenada en memoria RAM, en la CPG, es necesario obtener el valor de cada pixel 9 veces en diferente tiempos, ya que no es posible hacer lecturas paralelas en la memoria DRAM. Así que el número de ciclos de reloj para llevar a cabo esta operación es de 9 x 640 x 480 = 2,764,800. Ésta cifra contrasta con el tiempo requerido para calcular la misma matriz de pesos en la MDRO, que es de 307,200 ciclos de reloj (el 11.11 % del tiempo requerido en la CPG). Incluso este dato es optimista, ya que se considera que en la CPG se lleva un *pipeline* para acceder al dato en RAM y después de nueve ciclos empezar a sumarlos.

Lo mismo ocurre para el centroide. Al terminar de procesar la imagen más unos ciclos de reloj después (tiempo que tarda en calcularse la división) se obtienen las coordenadas del centroide.

Tanto para la captura como el cálculo de la Matriz de transformaciones de pesos y la obtención del centroide, el SE-MDRO se tarda 6.144 [ms] en comparación con los 19.83 [ms] que tarda en procesar la CPG. Éste último dato es la suma de la captura del *frame* y la obtención de los puntos frontera junto con el centroide, usando el CPU A con C++.

6.1.4. BOF

Para obtener la BOF es necesario esperar a que se tengan los puntos frontera del objeto y el centroide. En la MDRO se obtienen los valores de distancia y ángulo simultáneamente para los 180 componentes del vector BOF. El ángulo representa la dirección, en tanto la distancia el valor del registro direccionado. Es casi 24 veces más rápido obtener la BOF en el SE-MDRO que en la CPG.

Este tiempo es independiente del tamaño de la imagen y del tamaño relativo del objeto. Más bien depende del tamaño de la BOF. Esa medida se obtuvo entre el balance de elementos mínimos que debiese tener la BOF para que la RNA FuzzyARTMAP pudiera reconocer el objeto.

6.2. RNA FUZZY ARTMAP implementada en FPGA

El alcance de éste trabajo, se limitó a que únicamente la clasificación del objeto se realiza en la FPGA, por medio de la RNA Fuzzy ARTMAP. No así la parte de entrenamiento. Como ya se ha mencionado, la obtención de los pesos se hace en MATLAB y sólo son transferidos a la MDRO a través del software de control del sistema embebido, de manera que son compilados como una variable de tipo arreglo en C.

En la MDRO la obtención de los T_j y su posterior resonancia es casi 3 veces más rápida que en la CPG; dato proveniente del CPU B usando C++ que es el que mejor desempeño tuvo.

Éste resultado se obtuvo, como ya se ha mencionado, sólo usando 4 categorías. Éste proceso es dependiente del número de categorías en caso de usar una CPG. Debido a que la obtención de los T_j radica en operar sobre cada elemento de la BOF tantos números de categorías se tenga. En el caso de la MDRO, el cálculo se realiza de manera simultánea, siempre y cuando, el FPGA tenga la capacidad de albergar varios módulos de clasificación de la RNA FuzzyARTMAP. Cuando llegue a su límite, se implementaría un sistema de colas, pero de cualquier manera, el clasificador del SE-MDRO será más rápido que en la CPG.

6.3. Trabajo futuro

En el trabajo posterior a este diseño se plantea se pueda continuar en 4 vías: una referente al mejoramiento del desempeño en cada etapa del procesamiento y de la red neuronal, segunda de aprovechar las capacidades de reconfiguración parcial de la FPGA, tercera de utilizar la arquitectura del clasificador Fuzzy ARTMAP también para realizar el proceso de aprendizaje de nuevas categorías, y por último, en términos teóricos, aprovechar las ventajas de la extracción rápida del vector descriptivo para incorporarla en otros algoritmos de visión. A continuación se describen éstas vías.

Se identificaron procesos que limitan el ancho de banda de procesamiento de la imagen en ésta arquitectura de la MDRO. Para algunos cuellos de botella se puede encontrar una solución simplemente aumentando la velocidad del reloj del hardware de la MDRO, ya que es de 50 Mhz. Principalmente esta limitada por la velocidad del sensor OV7670. Éste chip es muy barato y entre otras razones es por eso que se escogió. Sin embargo, al paso del tiempo, ya existen otros sensores más rápidos, con más resolución y mejor óptica. Se plantea sustituir el sensor por uno más adecuado. La consecuencia radica en que al aumentar la resolución del cuadro, aumenta el número de ciclos de reloj necesarios para procesarla. Pero se gana en muchos otros aspectos.

La determinación del umbral de la imagen cruda, con el cual se binariza la imagen, es muy limitada. La estrategia para obtener el umbral consiste en buscar el mínimo valor después de haber encontrado el primer máximo de las ordenadas del histograma. Pero no siempre, y no en todos los cuadros, ese umbral es el más adecuado; pues depende en gran medida de las condiciones de ilumniación. Por lo que, hallar el umbral óptimo para un conjunto de cuadros consecutivos es prioritario para poder segmentar la imagen. Usar una red neuronal artificial que determine el mejor umbral, de acuerdo al entrenamiento previo en varios escenarios con variaciones de iluminación resultaría favorable. Bien se podría usar la estructura de la RNA Fuzzy ARTMAP en la FPGA para realizar ese proceso cognitivo.

Por otro lado, el clasificador de la RNA Fuzzy ARTMAP está compuesto por solo 4 categorías. La FPGA utilizada, Zynq-7000, tiene capacidad para que se puedan implementar más categorías. Pero siempre tendrá un límite debido al número de componentes que puede integrar. Sin embargo, no se puede restringir al número de objetos que es capaz de reconocer. Por lo que se puede construir un módulo que previamente administre una cola de varios BOF, con el objetivo de poder reconocer un objeto de un sinfín de categorías ya aprendidas. El cuello de botella estará en función del número de categorías implementadas. Sin embargo, puede ocurrir lo contrario, tener pocas categorías y que estructuralmente se tengan instrumentadas el máximo número de clasificadores en la FPGA, lo cual resulta inconveniente debido al consumo de potencia que prácticamente se desperdicia. Es por eso que se plantea como trabajo futuro aprovechar las capacidades de reconfiguración parcial de la FPGA. Con esto, dinámicamente el sistema se adapta a las condiciones de operación. Y de acuerdo a las necesidades de operación, ella misma decide cuantos clasificadores debe activar o bien desactivar.

La velocidad con la que la MDRO extrae el descriptor único de un objeto, alienta a aprovechar su arquitectura en otros algoritmos exhaustivos de visión robótica, sobre todo en aquellos dónde es importante extraer puntos característicos de una escena (*feature points*). Con

el cálculo rápido de la BOF, se puede aprovechar para obtener características de forma de dichos puntos y así eficientar aquellos algoritmos construidos en harwdare reconfigurable.

Bibliografía

[Anagnostopoulos, 2002] Anagnostopoulos, Georgios C. y Georgiopoulos, M. (2002). Category regions as new geometrical concepts in Fuzzy-ART and Fuzzy-ARTMAP. *Neural Networks*, 15(10):1205–1221.

[Andres, 1970] Andres, K. (1970). A Texas Instruments Application Report: MOS programmable logic arrays. Technical report, Texas instruments.

[Ashender, 1990] Ashender, P. J. (1990). *The VHDL Cookbook*. Dept. Computer Science University of Adelaide South Australia.

[Azouaoui, 2002] Azouaoui, Ouahiba y Chohra, A. (2002). Soft Computing Based Pattern Classifiers for the Obstacle Avoidance Behavior of Intelligent Autonomous Vehicles (IAV). *Appl. Intell.*, 16(3):249–272.

[Bonnici, 2006] Bonnici, Mark y Gatt, E. y. M. J. y. G. I. (2006). Artificial Neural Network Optimization for FPGA. In *ICECS*, page 1340–1343. IEEE.

[Bribiesca, 1980] Bribiesca, Ernesto y Guzmán, A. (1980). How to describe pure form and how to measure differences in shapes using shape numbers. *Pattern Recognition*, 12(2):101–112.

[Bryson, 1969] Bryson, A. E. y Ho, Y. C. (1969). *Applied Optimal Control*. Blaisdell, New York.

[Bullock, 1991] Bullock, D. y Grossberg, S. (1991). Adaptive neural networks for control of movement trajectories invariant under speed and force rescaling. *Human Movement Science,*.

[Capson, 2002] Capson, D y Fortuna, J. y. S. D. (2002). A comparison of PCA and ICA for object recognition under varying illumination . *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, 3:11–15.

[Carpenter, 2003] Carpenter, G. A. y Grossberg, S. (2003). *Adaptive resonance theory*, chapter Adaptive resonance theory, pages 87–90. MIT Press, Cambridge, MA, second edition edition.

[Carpenter, 1991] Carpenter, G.A. y Grossberg, S. y. R. J. (1991). ARTMAP: Supervised Real-Time Learning and Classification of Nonstationary Data by a Self-Organizing Neural Network. *Neural Networks*, 4:565–588.

[Chu, 2008a] Chu, P. (2008a). *FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version*. Wiley.

[Chu, 2008b] Chu, P. P. (2008b). *Fpga Prototyping by vhdl exambles*. Wiley.

[Cullinan, 2012] Cullinan, C. y Wyant, C. y. F. T. y. H. X. (2012). Computing Performance Benchmarks among CPU, GPU, and FPGA. *MathWorks*.

[Field, 2013] Field, M. (2013). Zedboard ov7670.

[Francisco, 2011] Francisco, C.-M. (2011). Reconocimiento de objetos empleando RNA, iluminación variable y caracterísiticas combinadas. Master's thesis, Unidad saltillo.

[Gaudiano, 1992] Gaudiano, P. y Grossberg, S. (1992). Adaptive vector integration to end-point: Self-organizing neural circuits for control of planned movement trajectories. *Human Movement Science*, 11.

[Gonzalez, 1977] Gonzalez, R.C. y Wintz, P. (1977). *Digital Image Processing*. Addison & Wesley.

[Grossberg, 1992] Grossberg, S. y Carpenter, G. y. M. N. y. R. J. y. R. D. (1992). Fuzzy ART-MAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps. *IEEE Transactions on Neural Networks*, 3(5):698–713.

[Grozea, 2010] Grozea, Cristian y Bankovic, Z. y. L. P. (2010). FPGA vs. Multi-core CPUs vs. GPUs: Hands-On Experience with a Sorting Application. In Keller, R., Kramer, D., and Weiss, J.-P., editors, *Facing the Multicore-Challenge*, volume 6310 of *Lecture Notes in Computer Science*, page 105–117. Springer.

[Guzman, 1969] Guzman, A. (1969). Decomposition of a visual scene into three dimensional bodies. *Automatic Interpretation and Classification of Images*, pages 243–276.

[Hebb, 1949] Hebb, D. O. (1949). *The Organization of Behavior*. Wiley, New York.

[Herakovic, 2010] Herakovic, N. (2010). *Robot Vision in Industrial Assembly and Quality Control Processes*. AlesUde (Ed.).

[Hopfield, 1982] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proc Natl Acad Sci U S A*, 79(8):2554–2558.

[Horn, 1986] Horn, B. K. P. (1986). *Robot vision.* MIT electrical engineering and computer science series. MIT Press.

[Hung, 1995] Hung, Cheng-An y Lin, S.-F. (1995). Adaptive hamming net: A fast-learning ART 1 model without searching. *Neural Networks*, 8(4):605–618.

[Kohonen, 1990] Kohonen, T. (1990). The Self-Organizing Map. *Proceedings of the IEEE*, 78:1464–1480.

[Kuon, 2007] Kuon, Ian y Rose, J. (2007). Measuring the Gap Between FPGAs and ASICs. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 26(2):203–215.

[Lattice, 1998] Lattice (1998). *Introduction to GAL Device Architectures*.

[Lomas-Barrie et al., 2015] Lomas-Barrie, V., Peña-Cabrera, M., and Durán-Ortega, J. (2015). Determining humanoid soccer player position on Goal detection. *Proceedings on the International Conference on Artificial Intelligence (ICAI)*, page 61.

[Marr, 1982] Marr, D. (1982). *Vision : a computational investigation into the human representation and processing of visual information / David Marr*. MIT Press.

[Martín, 2007] Martín, Bonifacio y Sanz, A. (2007). *Redes Neuronales y Sistemas Borrosos*. Alfaomega.

[Mcculloch, 1943] Mcculloch, Warren S. y Pitts, W. H. (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, 5:115–133.

[Nageswaran, 2009] Nageswaran, Jayram Moorkanikara y Dutt, N. y. K. J. L. y. N. A. y. V. A. V. (2009). 2009 Special Issue: A Configurable Simulation Environment for the Efficient Simulation of Large-scale Spiking Neural Networks on Graphics Processors. *Neural Netw.*, 22(5-6):791–800.

[OpenCV, 2015a] OpenCV, E. (2015a). How to scan images, lookup tables and time measurement with OpenCV. http://docs.opencv.org/2.4/doc/tutorials/core/how_to_scan_images/how_to_scan_images.html. Ingresado: 20-07-2015.

[OpenCV, 2015b] OpenCV, E. (2015b). Mat - The Basic Image Container. http://docs.opencv.org/2.4/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_contai Ingresado: 20-07-2015.

[Parnell, 2003] Parnell, Karen y Mehta, N. (2003). *Programmable Logic Design Quick Start Hand Book* . Xilinx, 4 edition.

[Peña-Cabrera, 2005] Peña-Cabrera, M. (2005). *Aprendizaje y reconocimiento invariante de objetos en ensamble con robots empleado redes neuronales artificiales*. PhD thesis, Centro de Ingeniería y Desarrollo Industrial.

[Peña-Cabrera et al., 2012] Peña-Cabrera, M., Lomas-Barrie, V., López-Juárez, I., Osorio, R., and Gómez, H. (2012). Contour Object Generation in Object Recognition Manufacturing Tasks. *Proceedings on the International Conference on Artificial Intelligence (ICAI)*, page 1.

[Peña-Cabrera et al., 2013] Peña-Cabrera, M., Lomas-Barrie, V., López-Juárez, I., and Osorio-Comparán, R. (2013). Contour Object Generation Method for Object Recognition using FPGA. *International Journal of Automation Technology*, 7(22):182–189.

[Raeisi, 2006] Raeisi, R. y Kabir, A. (2006). Implementation of Artificial Neural Network on FPGA. In *Proceedings Illinois-Indiana and North Central Joint Section Conference*. American Society for Engineering Education.

[Reyes-Acosta, 2009] Reyes-Acosta, A. (2009). Método utilizando silueta y curvatura local de objetos para reconocimiento invariante de objetos empleando redes neuronales artificiales. Master's thesis, Unidad saltillo.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6):386–408.

[Shirazi, 1995] Shirazi, Nabeel y Walters, A. y. A. P. M. (1995). Quantitative analysis of floating point arithmetic on FPGA based custom computing machines. In *FCCM*, page 155–163. IEEE Computer Society.

[Vanhoucke, 2011] Vanhoucke, Vincent y Senior, A. y. M. M. Z. (2011). Improving the speed of neural networks on CPUs. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*.

[Võsandi, 2015] Võsandi, L. (2015). Video capture with vdma.

[Wang, 2005] Wang, W. (2005). Place and Route for FPGAs.

[Widrow, 1960] Widrow, B. y Hoff, M. E. (1960). Adaptive switching circuits. *Institute of Radio Engineers, Western Electronics Show and Convention*, Part 4:96–104.

[Wray, 2010] Wray, Stephen y Luk, W. y. P. P. (2010). Exploring algorithmic trading in reconfigurable hardware. In Charot, F., Hannig, F., Teich, J., and Wolinski, C., editors, *ASAP*, page 325–328. IEEE Computer Society.

[Xilinx, 2016] Xilinx, E. (2016). Zynq-7000 All Programmable SoC Overview. PDF. v1.9.

[Zhu, 2003] Zhu, Jihan y Sutton, P. (2003). FPGA Implementations of Neural Networks - A Survey of a Decade of Progress. In Cheung, P. Y. K., Constantinides, G. A., and de Sousa, J. T., editors, *FPL*, volume 2778 of *Lecture Notes in Computer Science*, page 1062–1066. Springer.

Apéndice A

Lista de abreviaturas

- **BOF** *Boundary Object Function* (Función de frontera de un objeto).
- **CPG** Computadora de propósito general.
- **SO** Sistema operativo.
- **MDRO** Máquina Digital de Reconocimiento de Objetos.
- **RNA** Red neuronal artificial.
- **SE** Sistema embebido.
- **SP** Sistema de procesamiento del chip Zynq-7000.
- **LP** Sección de de lógica programable del chip Zynq-7000.
- **VDMA** Datos de video con acceso directo a memoria.
- **FPGA** Field Programmable Gate Array.

Apéndice B

Descripción de hardware

B.1. Histograma.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity cal_hst is
    Port ( n_pix_img : in  std_logic_vector(7 downto 0);
        Bus2IP_Clk : in std_logic;
        slv_reg0 : in std_logic_vector(7 downto 0)
                        );
end cal_hst;

architecture Behavioral of cal_hst is

component myRAM is
        port ( Clk_a : in std_logic;
                read_enable         : in std_logic;
                write_enable : in std_logic;
                addr_in : in std_logic_vector(7 downto 0);
                addr_out : in std_logic_vector(7 downto 0);
                data_in : in std_logic_vector (18 downto 0);
                data_out:out std_logic_vector(18 downto 0));
end component;

signal read_enable : std_logic:='1';
signal write_enable : std_logic:='0';
signal data_in : std_logic_vector (18 downto 0);
```

```
signal data_out : std_logic_vector (18 downto 0);
signal addr_in : std_logic_vector(7 downto 0);
signal addr_out : std_logic_vector(7 downto 0);
signal dato_retardado :
   std_logic_vector(18 downto 0):="0000000000000000000";
signal selector : std_logic:='0';


begin

ram: myRAM port map (
        Clk_a => Bus2IP_Clk,
        read_enable => read_enable,
        write_enable => write_enable,
        addr_in => addr_in,
        addr_out =>addr_out ,
        data_in         => data_in,
        data_out => data_out
        );

addr_out <= n_pix_img;
data_in <= dato_retardado;


histograma: process (Bus2IP_Clk, slv_reg0)
        variable retardar_w : boolean := true;
   begin
                if (rising_edge(Bus2IP_CLK)) then
                 if slv_reg0(0) = '1' then
                  if retardar_w = true then
                   write_enable <= '1';
                   retardar_w := false;
                  end if;
                  addr_in <= addr_out;
                  if (addr_in = addr_out) then
                   selector <= '1';
                   dato_retardado <= std_logic_vector(
                        unsigned(dato_retardado) + 1);
                  else
                   selector <= '0';
                   dato_retardado <= std_logic_vector(
                        unsigned(data_out) + 1);
                  end if;
```

```
                elsif  slv_reg0(1) = '1' then
                       read_enable <= '0';
                              write_enable <= '0';
                       end if;
                end if;
        end process histograma;

end Behavioral;
```

B.2. Centroide.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity centroide is
         generic(ancho_imagen : natural:=640;
                 alto_imagen : natural:=360);
    Port ( imagen_binaria_in : in  STD_LOGIC;
           sinc_h : in  STD_LOGIC;
           sinc_v : in  STD_LOGIC;
           Bus2IP_Clk : in  STD_LOGIC;
           cen_x : out  STD_LOGIC_VECTOR (9 downto 0);
           cen_y : out  STD_LOGIC_VECTOR (9 downto 0));
end centroide;

architecture Behavioral of centroide is

function  divide  (a : UNSIGNED; b : UNSIGNED) return
                        UNSIGNED is
        variable a1 : unsigned(a'length-1 downto 0):=a;
        variable b1 : unsigned(b'length-1 downto 0):=b;
        variable p1 : unsigned(b'length downto 0):=
                        (others => '0');
        variable i : integer:=0;

begin
        for i in 0 to b'length-1 loop
                p1(b'length-1 downto 1) := p1
                                (b'length-2 downto 0);
                p1(0) := a1(a'length-1);
```

```
                a1(a'length-1 downto 1) := a1
                                (a'length-2 downto 0);
                p1 := p1-b1;
                if(p1(b'length-1) ='1') then
                a1(0) :='0';
                p1 := p1+b1;
                else
                a1(0) :='1';
                end if;
        end loop;
return a1;

end divide;

signal s_ax, s_ay, s_b, s_i, s_j : integer:=0;
signal s_cen_x, s_cen_y : unsigned (35 downto 0)
                                :=(others => '0');



begin

centroide:process(Bus2IP_Clk, sinc_v)

        variable Ax : integer:=0;
        variable Ay : integer:=0;
        variable b  : integer:=0;
        variable i,j : integer:=1;

        variable u_Ax, u_Ay, u_b: unsigned (35 downto 0)
                                :=(others => '0');
                        -- maximo numero 47185766912


        begin
                if (rising_edge(Bus2IP_CLK) ) then
                 if(sinc_v = '1')then
                  if(j < ancho_imagen) then
                   if(imagen_binaria_in = '1' ) then
                    Ax := Ax + j;
                    Ay := Ay + i;
                    b := b + 1;
                    s_ax <= ax;
```

```
                    s_ay <= ay;
                    s_b <= b;
                   end if;
                    j := j + 1;
                    s_j <= j;
                   else
                    j:= 1;
                    i := i + 1;
                    s_i <= i;
                   end if;
                   else
                    u_Ax := to_unsigned(Ax, u_Ax'length);
                    u_Ay := to_unsigned(Ay, u_Ay'length);
                    u_b := to_unsigned(b, u_b'length);
                    s_cen_x <= divide (u_Ax, u_b);
                    s_cen_y <= divide (u_Ay, u_b);
                      end if;
                end if;

end process;

end Behavioral;
```

B.3. CORDIC angulo.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synthesis translate_off
LIBRARY XilinxCoreLib;
-- synthesis translate_on
ENTITY cordic_angulo IS
  PORT (
    x_in : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
    y_in : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
    nd : IN STD_LOGIC;
    phase_out : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
    rdy : OUT STD_LOGIC;
    clk : IN STD_LOGIC
  );
END cordic_angulo;
```

```
ARCHITECTURE cordic_angulo_a OF cordic_angulo IS
-- synthesis translate_off
COMPONENT wrapped_cordic_angulo
  PORT (
    x_in : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
    y_in : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
    nd : IN STD_LOGIC;
    phase_out : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
    rdy : OUT STD_LOGIC;
    clk : IN STD_LOGIC
  );
END COMPONENT;

-- Configuration specification
  FOR ALL : wrapped_cordic_angulo USE ENTITY
              XilinxCoreLib.cordic_v4_0(behavioral)
    GENERIC MAP (
      c_architecture => 2,
      c_coarse_rotate => 1,
      c_cordic_function => 3,
      c_data_format => 0,
      c_family => "virtex6",
      c_has_ce => 0,
      c_has_clk => 1,
      c_has_nd => 1,
      c_has_phase_in => 0,
      c_has_phase_out => 1,
      c_has_rdy => 1,
      c_has_rfd => 0,
      c_has_sclr => 0,
      c_has_x_in => 1,
      c_has_x_out => 0,
      c_has_y_in => 1,
      c_has_y_out => 0,
      c_input_width => 12,
      c_iterations => 0,
      c_output_width => 12,
      c_phase_format => 0,
      c_pipeline_mode => -2,
      c_precision => 0,
      c_reg_inputs => 1,
      c_reg_outputs => 1,
      c_round_mode => 1,
```

```
      c_scale_comp => 0,
      c_xdevicefamily => "virtex6"
    );
-- synthesis translate_on
BEGIN
-- synthesis translate_off
U0 : wrapped_cordic_angulo
  PORT MAP (
    x_in => x_in,
    y_in => y_in,
    nd => nd,
    phase_out => phase_out,
    rdy => rdy,
    clk => clk
  );
-- synthesis translate_on

END cordic_angulo_a;
```

B.4. matriz pesos.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity matriz_pesos is
        Generic (
               ancho_imagen : INTEGER := 640;
               umbral_frontera : INTEGER := 1
        );
    Port ( imagen_binaria_in : in STD_LOGIC;
                sinc_h : in STD_LOGIC;
                sinc_v : in STD_LOGIC;
    matriz_pesos_out : out STD_LOGIC_VECTOR (3 downto 0);
           Bus2IP_Clk : in STD_LOGIC;
           es_vmin : out STD_LOGIC := '0';
           coordenadas_vmin_x : out UNSIGNED (9 downto 0);
           coordenadas_vmin_y : out UNSIGNED (8 downto 0)
                       );
end matriz_pesos;
```

```
architecture Behavioral of matriz_pesos is

component fifo_1x640 is port (
    clk : IN STD_LOGIC;
    rst : IN STD_LOGIC;
    din : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    wr_en : IN STD_LOGIC;
    rd_en : IN STD_LOGIC;
    dout : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
    full : OUT STD_LOGIC;
    empty : OUT STD_LOGIC;
    data_count : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)
  );
end component;

function sl2int (x: STD_LOGIC) return INTEGER is
begin
        if x='1' then
                return 1;
        else
                return 0;
        end if;
end;

signal sr3x3_0 : std_logic_vector(0 to 2):=(others=>'0');
signal sr3x3_1 : std_logic_vector(0 to 2):=(others=>'0');
signal sr3x3_2 : std_logic_vector(0 to 2):=(others=>'0');

signal dout : STD_LOGIC_VECTOR(0 DOWNTO 0);
signal wr_en : STD_LOGIC_VECTOR(2 DOWNTO 0):=
                                        (others=>'0') ;
signal rd_en : STD_LOGIC_VECTOR(2 DOWNTO 0):=
                                       (others=>'0') ;
signal full : STD_LOGIC_VECTOR(2 DOWNTO 0);
signal empty : STD_LOGIC_VECTOR(2 DOWNTO 0);
signal data_count0 : STD_LOGIC_VECTOR(9 DOWNTO 0);
signal data_count1 : STD_LOGIC_VECTOR(9 DOWNTO 0);
signal data_count2 : STD_LOGIC_VECTOR(9 DOWNTO 0);
signal matriz_pesos_sign: integer:= 0;

signal din_0 : STD_LOGIC_VECTOR(0 TO 0):="0";
signal dout_0 : STD_LOGIC_VECTOR(0 TO 0):="0";
```

```
signal din_1 : STD_LOGIC_VECTOR(0 TO 0):="0";
signal dout_1 : STD_LOGIC_VECTOR(0 TO 0):="0";
signal din_2 : STD_LOGIC_VECTOR(0 TO 0):="0";
signal dout_2 : STD_LOGIC_VECTOR(0 TO 0):="0";

signal x : unsigned(9 DOWNTO 0) := "0000000000";
signal y : unsigned(8 DOWNTO 0) := "000000000";

begin

fifo_1x640_0: fifo_1x640 port map (
        clk => Bus2IP_Clk,
   rst => '0',
   din => din_0,
   wr_en => wr_en(0), --'1',
   rd_en => rd_en(0),
   dout => dout_0,
   full => full(0),
   empty => empty(0),
   data_count => data_count0
);

fifo_1x640_1: fifo_1x640 port map (
        clk => Bus2IP_Clk,
   rst => '0',
   din => din_1,
   wr_en => wr_en(1),
   rd_en => rd_en(1),
   dout => dout_1,
   full => full(1),
   empty => empty(1),
   data_count => data_count1
);

fifo_1x640_2: fifo_1x640 port map (
        clk => Bus2IP_Clk,
   rst => '0',
   din => din_2,
   wr_en => wr_en(2),
   rd_en => rd_en(2),
   dout => dout_2,
   full => full(2),
   empty => empty(2),
```

```
   data_count => data_count2
);

matriz_pesos:process(Bus2IP_Clk, sinc_v, sinc_h)

        begin
                if(sinc_v = '1') then
                 wr_en <= "111";
                 if (rising_edge(Bus2IP_CLK) ) then
                  if(sinc_h = '0') then
                   x <= "0000000000";
                   y <= y + 1;
                  else
                   x <= x + 1;
                  end if;
                  if(data_count0 > std_logic_vector(
                        to_unsigned(ancho_imagen - 5,
                                data_count0'length)))
                                                then
                   rd_en(0) <= '1';
                  end if;
                   if(data_count1 > std_logic_vector(
                        to_unsigned(ancho_imagen - 5,
                                data_count1'length)))
                                                then
                    rd_en(1) <= '1';
                   end if;
                    if(data_count2 > std_logic_vector(
                         to_unsigned(ancho_imagen - 5,
                                data_count2'length)))
                                                then
                     rd_en(2) <= '1';
                    end if;

                     sr3x3_0(0 to 1)<= sr3x3_0(1 to 2);
                     sr3x3_1(0 to 1)<= sr3x3_1(1 to 2);
                     sr3x3_2(0 to 1)<= sr3x3_2(1 to 2);
                     matriz_pesos_sign <=
                        sl2int(sr3x3_0(0)) +
                        sl2int(sr3x3_0(1)) +
                        sl2int(sr3x3_0(2)) +
                        sl2int(sr3x3_1(0)) +
                        sl2int(sr3x3_1(1)) +
```

```
                        sl2int(sr3x3_1(2)) +
                        sl2int(sr3x3_2(0)) +
                        sl2int(sr3x3_2(1)) +
                        sl2int(sr3x3_2(2));
                    if(to_unsigned(matriz_pesos_sign,4) =
                        to_unsigned(umbral_frontera,4))
                                                then
                     es_vmin <= '1';
                    else
                     es_vmin <= '0';
                    end if;
                  end if;
                end if;
        end process matriz_pesos;

din_0(0) <= imagen_binaria_in;
sr3x3_0(2) <= dout_0(0);
din_1(0) <=  sr3x3_0(0);
sr3x3_1(2) <= dout_1(0);
din_2(0) <=  sr3x3_1(0);
sr3x3_2(2) <= dout_2(0);
matriz_pesos_out <= std_logic_vector(to_unsigned(
                                matriz_pesos_sign,4));
coordenadas_vmin_x <= x;
coordenadas_vmin_y <= y;

end Behavioral;
```

Apéndice C

Tarjeta Zybo

Las Zynq-7000 son una familia de circuitos electrónicos conocidos como *All Programmable System-on-Chip (AP SoC)*, (Sistema en un solo chip, todo programable). La cual integra un procesador de doble núcleo de la arquitectura ARM del modelo Cortex A9 y una FPGA de la familia 7 de Xilinx. Algunas de sus características son:

- FPGA de la familia Zynq-7000 (XC7Z010-1CLG400C).
- 512 MB de memoria DDR3 RAM.
- 128 Mb de memoria Flash.
- Conector microSD, soporta al sistema de archivos de Linux.
- Puerto 1G Ethernet.
- Puerto USB 2.0
- Puertos UART, SPI e I2C.
- Puerto HDMI bidireccional.
- Puerto VGA.
- Puerto USB OTG 2.0
- Codificador de audio. Entrada de micrófono, audífonos y auxiliares.
- Puertos digitales de propósito general conectados a 6 *pushbuttons*, 4 apagadores y 5 LED.
- Conectores Pmod.
- Puerto JTAG para reprogramación.

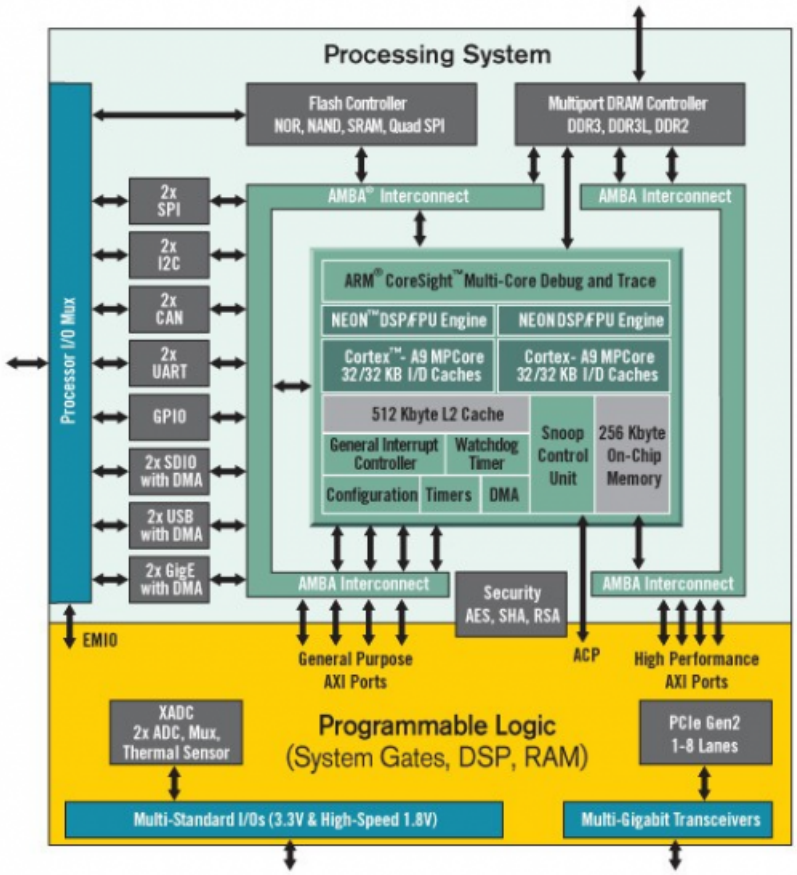

Figura C.1: Arquitectura del Zynq

C.1. Configuración Cortex A9

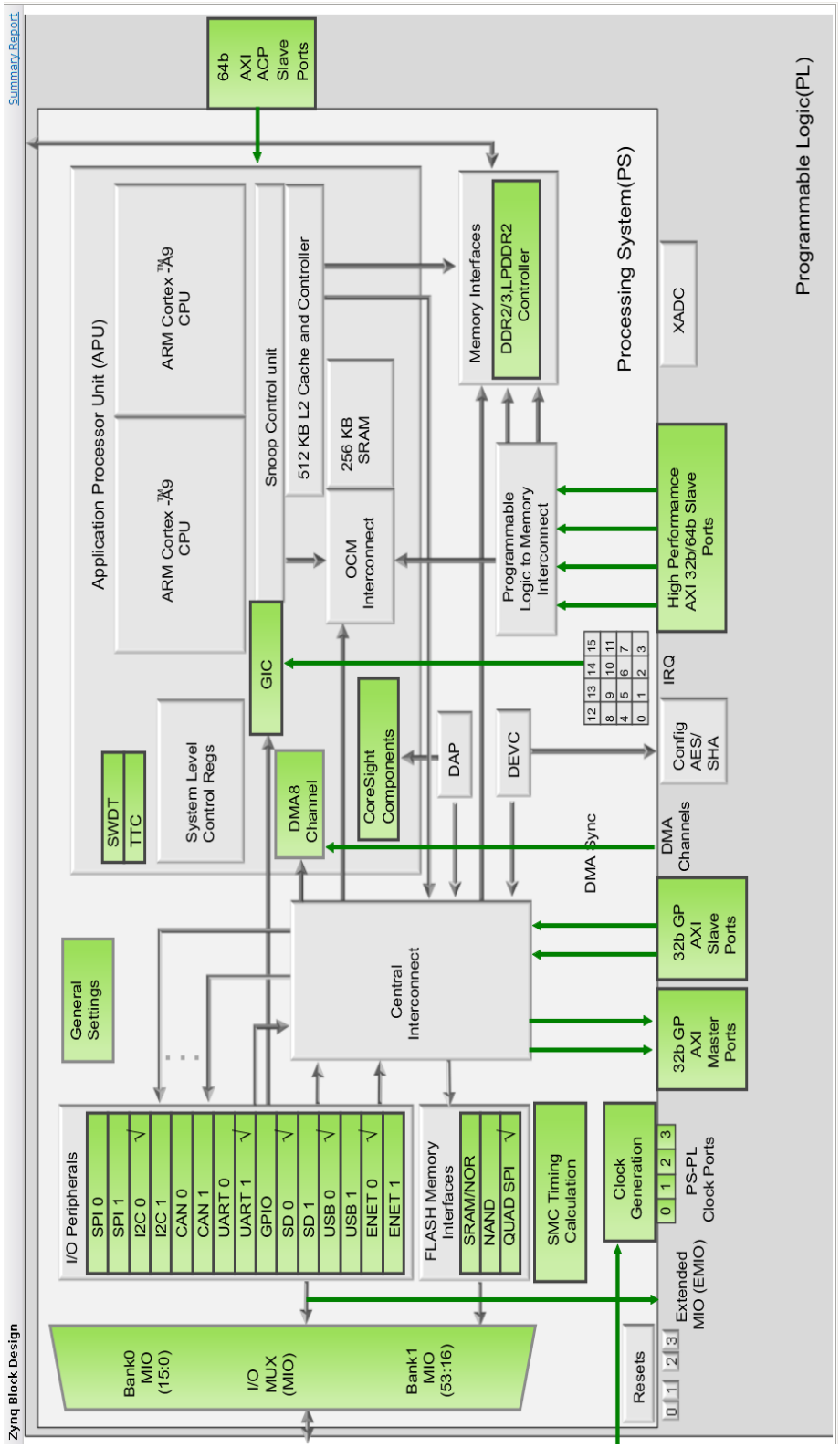

Figura C.2: Cortex A9 configuración interna

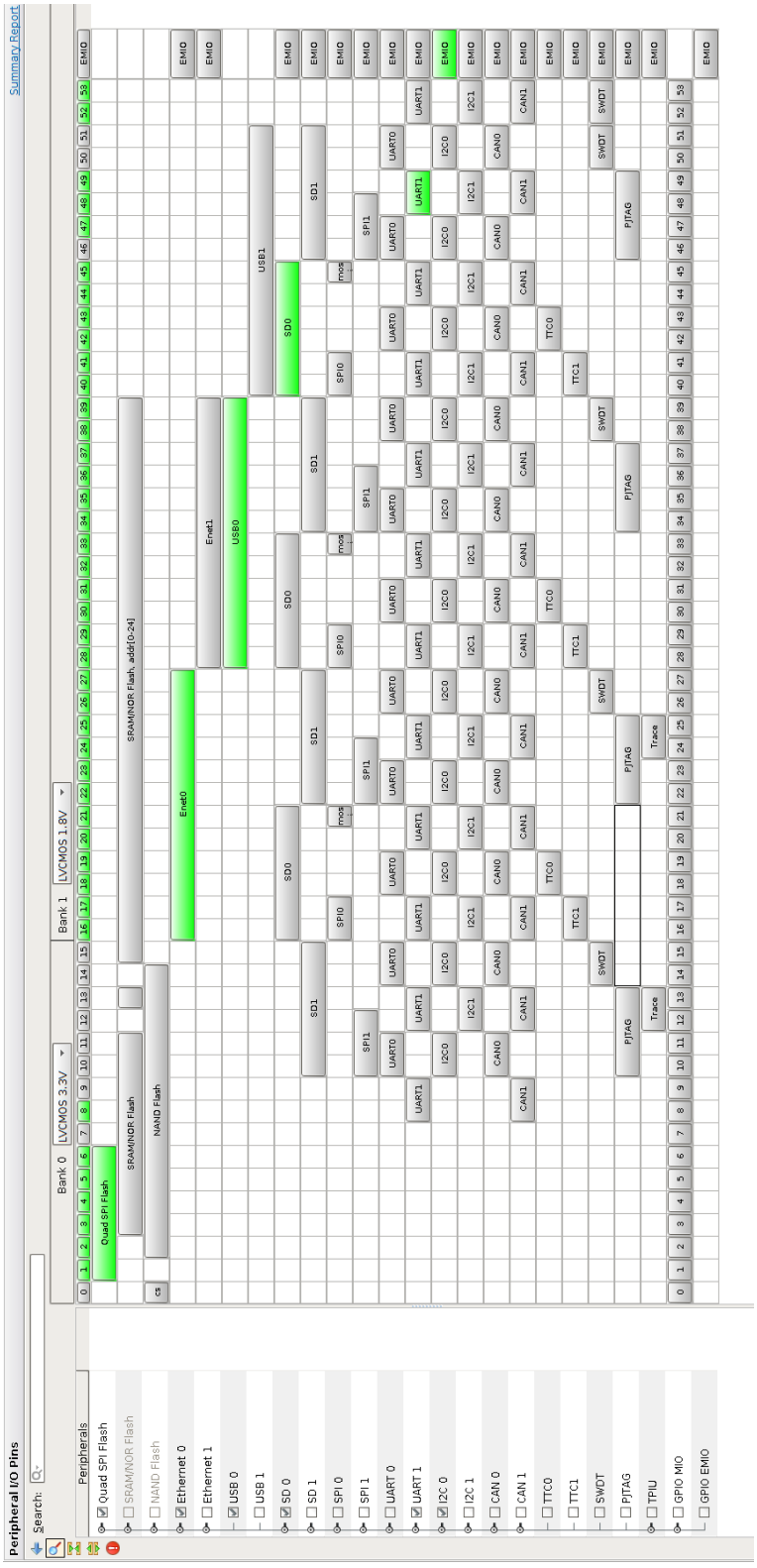

Figura C.3: Configuración de periféricos del Cortex A9

Apéndice D

Sistema Embebido

A continuación se muestra el diagrama de conexiones del sistema embebido (página siguiente).

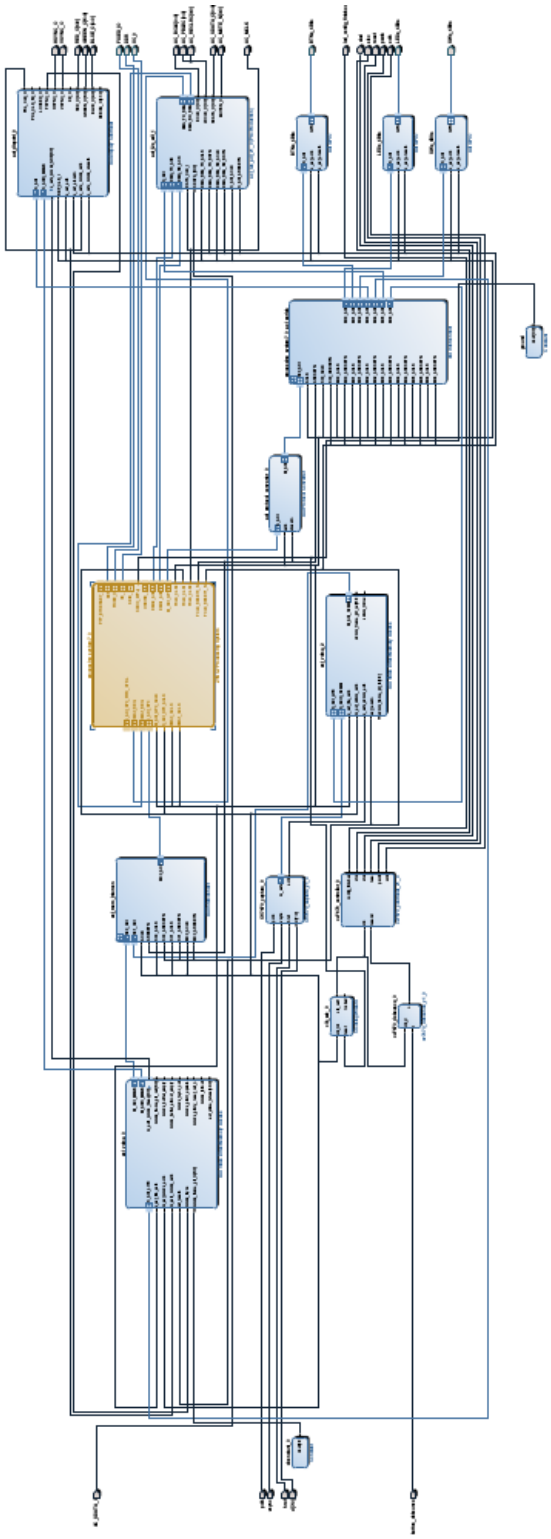

Figura D.1: Diagrama de conexión del sistema embebido

Apéndice E

Sensor CCD OV7670

Array Element (VGA)		640 x 480
Power Supply	**Digital Core**	1.8VDC ±10%
	Analog	2.45V to 3.0V
	I/O	1.7V to 3.0V
Power Requirements	**Active**	TBD
	Standby	< 20 µA
Temperature Range	**Operation**	-30°C to 70°C
	Stable Image	0°C to 50°C
Output Formats (8-bit)		• YUV/YCbCr 4:2:2 • RGB565/555 • GRB 4:2:2 • Raw RGB Data
Lens Size		1/6"
Chief Ray Angle		24°
Maximum Image Transfer Rate		30 fps for VGA
Sensitivity		1.1 V/Lux-sec
S/N Ratio		40 dB
Dynamic Range		TBD
Scan Mode		Progressive
Electronics Exposure		Up to 510:1 (for selected fps)
Pixel Size		3.6 µm x 3.6 µm
Dark Current		12 mV/s at 60°C
Well Capacity		17 K e
Image Area		2.36 mm x 1.76 mm
Package Dimensions		3785 µm x 4235 µm

Figura E.1: Especificaciones del sensor OV7670

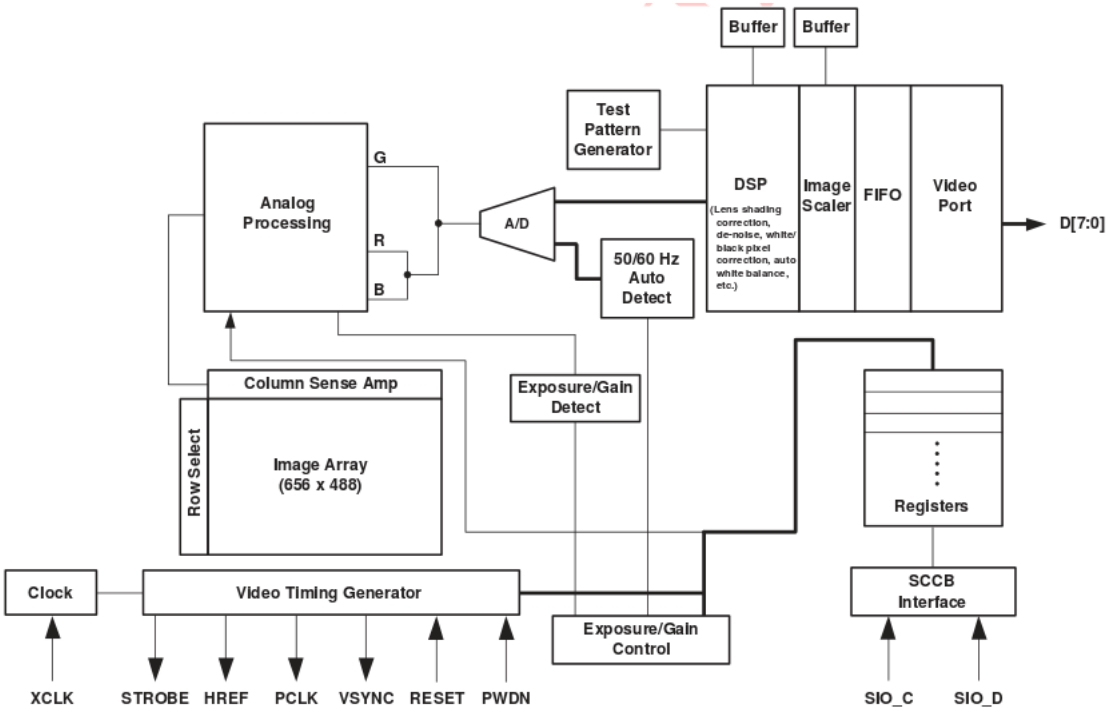

Figura E.2: Diagrama de bloques del sensor OV7670

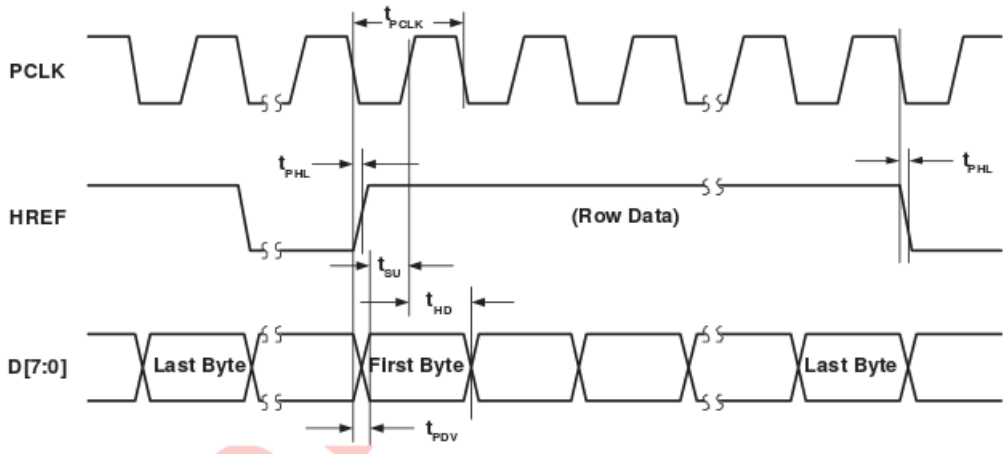

Figura E.3: Diagrama de tiempo horizontal del sensor OV7670

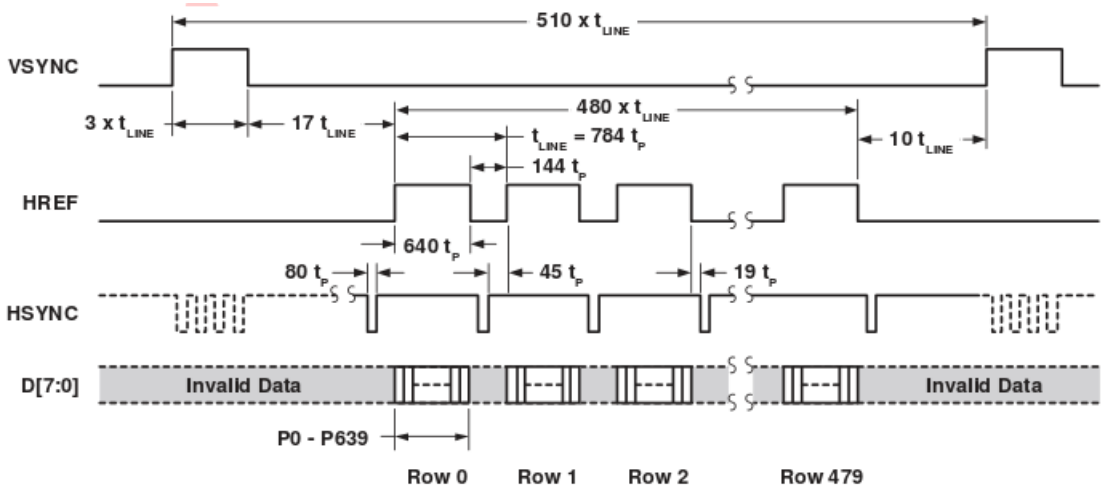

Figura E.4: Diagrama de tiempo de un *frame* del sensor OV7670