UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
ELÉCTRICA – SISTEMAS ELECTRÓNICOS

IMPLEMENTACIÓN EN UN FPGA DE LA FUNCIÓN DE FRONTERA PARA EL RECONOCIMIENTO DE OBJETOS

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

PRESENTA:
RAÚL ALVARO ROMERO AYALA

TUTOR PRINCIPAL
JUAN MARIO, PEÑA, CABRERA, IIMAS-UNAM

MÉXICO, D. F. ENERO 2014

ÍNDICE

Capítulo VI

Capítulo VII

RESUMEN

En el campo de la visión por computadora, se emplean redes neuronales artificiales para el reconocimiento invariante de objetos. Estas redes aprenden las características de dichos objetos y obtienen patrones que los identifican y permiten diferenciar unos de otros.

Esto se realiza capturando imágenes de los objetos por medio de cámaras de video para luego obtener us características, como lo son: la diferencia del objeto con el fondo, el borde o contorno, el área y el perímetro. Gracias a estas características se tiene un patrón diferente por cada objeto.

Una de estas características importantes se da mediante el cálculo de la función de frontera BOF(Boundary Object Function de sus siglas en inglés); así se conforma un vector descriptor de cada objeto observado. De este vector, es que el sistema por medio de sus redes neuronales aprende a reconocer posteriormente objetos similares y también puede diferenciarlos.

Este trabajo se enfoca en el desarrollo del algoritmo para el cálculo de la función de frontera, así como aquellos que son necesarios para otras características importantes; como el contorno del objeto, el centroide y la distancia de un punto del contorno hasta el centroide; los cuales en conjunto permiten alcanzar el objetivo final, obtener el BOF.

Con el empleo de lenguaje descriptor de hardware, se hará uso de una tarjeta de desarrollo con un arreglo de compuertas programables en campo FPGA(Field Programmable Gate Array) integrado; para la implementación final de los algoritmos, mencionados anteriormente. Esto nos permite hacer todas las pruebas de laboratorio requeridas, economizando en recursos de hardware, de software, de espacio y de tiempo de diseño.

.

ABSTRACT

In the field of computer vision, artificial neural networks are used for invariant object recognition. These networks learn the characteristics of those objects and obtain patterns that identify and differentiate from each other.

This is done by capturing images from the objects using video cameras to then get features, such as; the difference of the object with the background, border or contour, area and perimeter. Thanks to these characteristics we have a different pattern for each object.

One of the important features is given by calculating the boundary function BOF(Boundary Object Function its acronym); thus a descriptor vector of each observed object is formed. From this vector, the system by their neural network, learns to recognize similar objects and subsequently can also differentiate them.

This work focuses on the development of the algorithm for calculating the boundary function, as well as those necessary for other important characteristics, such as the contour of the object, the centroid and the distance from one point of the contour to the centroid, which together allow us to reach the ultimate goal, to obtain the BOF.

Through hardware description language, will be using a development board with an integrated FPGA(Field Programmable Gate Array); to get the final implementation of the algorithms mentioned above. This allows us to do all the required laboratory tests, saving on hardware, software, space and design time resources.

AGRADECIMIENTOS

Agradezco a mis profesores, sin los cuales no hubiera llegado

al buen término de esta tesis,

al Programa de Posgrado por todo su apoyo y sobretodo

a la Universidad Nacional Autónoma de México por abrirme las puertas

de su digna institución.

Un agradecimiento especial para mi tutor,

el Doctor Juan Mario Peña Cabrera

por el tiempo y los consejos brindados.

DEDICATORIA

A mis padres, que me enseñaron a valorar la vida y

que siempre mi objetivo sea; mejorar como persona.

A mis dos hermanas y a mi hermano

por la confianza que me tuvieron.

Y a mi familia, por cada palabra de aliento,

que me dio la fuerza para seguir adelante.

LISTA DE FIGURAS

LISTA DE IMÁGENES

LISTA DE TABLAS

LISTA DE SIGLAS Y ACRÓNIMOS

BOF	Boundary Object Function
FPGA	Field Programmable Gate Array
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
2D	Two Dimensions
LCD	Liquid Crystal Display
VGA	Video Graphic Adapter
RS-232	Recommended Standard 232
ROI	Region Of Interest
PLD	Programmable Logic Device
CPLD	Complex Programmable Logic Device
CLB	Configurable Logic Block
LC	Logic Cell
MAC	Multiply Accumulate
FFT	Fast Fourier Transform
FWT	Fast Wavelet Transform
PCB	Printed Circuit Board
IP	Intellectual Property
RF	Radio Frequency
LUT	Look Up Table
DB9	Data Bit 9
UART	Universal Asynchronous Receiver-Transmitter
Pixel	Picture Element
ESD	Electro Static Discharge
JPEG	Joint Photographic Expert Group
RAM	Random Access Memory
SR	Shift Register
ISE	Integrated Software Environment

CAPÍTULO I

INTRODUCCIÓN

En el campo de desarrollo de celdas de manufactura en procesos industriales se tienen componentes bien definidos, cada uno de los cuales se encarga de hacer un trabajo específico dentro del sistema del que forman parte.

Es así como podemos nombrar algunos como el control y manejo de un brazo robótico empleado generalmente en procesos reiterativos y que requieren de la mayor velocidad de trabajo posible, sin comprometer la repetitividad y confiabilidad del mismo.

Por otro lado tenemos los sensores que obtienen la información del medio físico en el que se está inmerso para poder tomar decisiones de operación. Dentro de estos, uno de ellos es el de procesos de visión industrial para procesamiento digital de señales y su posterior manipulación a nivel electrónico para el reconocimiento de posiciones o reconocimiento de objetos.

Todos estos elementos en conjunto hacen de una celda de manufactura un sistema eficaz, confiable y robusto que pueda cumplir con los requerimientos del proceso en cuestión.

En lo referente a reconocimiento de objetos, se tienen varias técnicas y varios procedimientos ya probados, que consiguen una buena respuesta ya en funcionamiento. Una de las características que se busca obtener de un objeto, es la función de frontera BOF(Boundary Object Function), para lo cual se han ido desarrollando algoritmos diversos. Esto permite al sistema tener una caracterización de los objetos capturados por una cámara de video que permite diferenciar unos de otros en cuanto a su figura en 2D.

Ya en la práctica, se busca hacer este tipo de caracterización en plataformas veloces y de fácil mantenimiento y el poder modificar su configuración de hardware en el menor tiempo que sea posible. Esto nos brinda ahorro de tiempo y recursos a la hora del diseño, desarrollo y puesta en marcha.

Una plataforma que cumple con todos estos requisitos, es aquella basada en FPGA. Este tipo de dispositivos deben ser configurados a partir de algún tipo de nivel de abstracción, esto con la ayuda de algún paquete de aplicación específico. Es en este punto donde se centra esta investigación, el de trabajar en el reconocimiento de objetos utilizando un FPGA.

En lo que respecta a la configuración de estos dispositivos, existen varios lenguajes y diversas maneras para realizarlo. Un lenguaje muy utilizado es el VHDL(VHSIC Hardware Description

Language; donde VHSIC viene de Very High Speed Integrated Circuit). Este es un lenguaje descriptor de hardware donde podemos definir la estructura y que funciones debe cumplir nuestra arquitectura. Luego se puede configurar el FPGA con la arquitectura diseñada.

1.1 PLANTEAMIENTO DEL PROBLEMA

En sistemas automáticos, dentro de celdas de manufactura, las cuales hacen uso de robots para manipulación de objetos, se busca siempre la mayor eficiencia posible. Se utilizan computadoras que necesitan de sistemas operativos para funcionar, y lenguajes de programación donde los códigos desarrollados deben ser interpretados antes de ejecutarse; todo esto se va sumando al tiempo total que lleva realizar el proceso. Por otro lado el mantenimiento de los equipos, modificaciones de software, o cambio de hadware, implican un mayor uso de recursos.

1.2 HIPÓTESIS

Se busca demostrar que el cálculo de la función de frontera BOF(Boundary Object Function), configurada en una arquitectura dentro de un arreglo de compuertas programables en campo FPGA(Field Programmable Gate Array); a partir del algoritmo creado para este propósito, además de brindar una etapa de desarrollo menor y facilidad para realizar cambios en la funcionalidad de los circuitos, tiene un tiempo de ejecución de procesamiento más rápido y usa menos recursos de hardware y software; que uno basado en una computadora, que cuenta con sistema operativo y el algoritmo es codificado en algún lenguaje de programación residente.

1.3 OBJETIVOS

El objetivo principal de este trabajo de investigación es el desarrollo de un algoritmo para el cálculo de la función de frontera que denominamos BOF(Boundary Object Function); y su posterior implementación en una plataforma FPGA(Field Programmable Gate Array).

Esta función de frontera está definida como: la distancia de varios de los puntos del contorno de la figura de un objeto, de la imagen capturada en 2D, hasta su centroide. Sobreentendiendo que es necesario tener primero las coordenadas de dichos puntos, así como las del centroide.

Como objetivos específicos a cumplir, están: el desarrollo del algoritmo para la obtención de las coordenadas del centroide y las coordenadas de algunos puntos del contorno. También el algoritmo encargado de realizar la interconexión de los anteriormente mencionados.

Desarrollar en una plataforma FPGA implica otro objetivo específico, la codificación de los algoritmos en un lenguaje descriptor de hardware, como lo es el VHDL(VHSIC Hardware Description Language). Este tipo de lenguaje de configuración de hardware es de naturaleza paralela y no secuencial.

La implementación de esta configuración en una plataforma basada en FPGA y la puesta en funcionamiento para observar la respuesta del circuito diseñado, por medio de un paquete de

aplicación como lo es el de Xilinx, es otro de los objetivos a alcanzar. Con estos elementos a disposición se realizarán varias pruebas experimentales.

1.4 JUSTIFICACIÓN

El desarrollo de algoritmos, para la obtención de la función de frontera de objetos capturados en imagen por una cámara de video, se hacen en computadoras personales, así como el procesamiento digital de las imágenes en plataformas con sistemas operativos y en lenguajes de alto nivel como lo son el C++ y el Java .

Estos procedimientos implican el uso de recursos de hardware y software poco eficiente. Ya que se tienen sistemas operativos que interpretan primero las instrucciones de los programas que se realizan en los lenguajes utilizados, y recién ejecutan físicamente lo que se requiere que hagan.

También el tener un espacio dedicado para la computadora y la consecuente necesidad de instalar software para poner en funcionamiento el equipo, así como la incomodidad que genera el trasladar todo a otra ubicación, son imprácticos.

Por otro lado, la velocidad de ejecución de un sistema desarrollado en una computadora de escritorio en comparación con una plataforma basada en un FPGA es más lenta. Aquí se puede ver que es obvio el escoger esta última como una opción mejor ya que en el procesamiento digital de señales provenientes de imágenes de video es muy importante la velocidad de procesamiento de información en tiempo real.

Pero la ventaja más grande es la capacidad que se tiene, al utilizar dispositivos con FPGA integrados en ellos, es la de poder modificar los circuitos configurados con un lenguaje descriptor de hardware en un nivel de abstracción tal que, al cambiar la configuración a nivel de software, se consigue cambiar también la configuración a nivel de hardware dentro del FPGA. Esto es más complicado al hacerlo en computadoras que funcionan con microprocesador y circuitos integrados soldados, y dedicados a una función específica desde fábrica.

1.5 ALCANCE

Implementar dentro de un FPGA, por medio de configuración de hardware, el algoritmo desarrollado para la obtención de la función de frontera BOF(Boundary Object Function). Esto para tener un circuito funcional que al ingresar la imagen de un objeto en formato binario(unos y ceros), nos devuelva como resultado la función de frontera; visible en algún medio adecuado, un visualizador de cristal líquido LCD(Liquid Cristal Display) por ejemplo, esto para comprobar el funcionamiento del diseño propuesto y desarrollado inicialmente.

No está dentro del alcance de esta tesis el hacer todo un sistema de reconocimiento de objetos, ni tampoco implementar una celda de manufactura. Pero si se tienen objetos físicos bien definidos (un cubo y un cilindro) que son el punto de partida para las pruebas experimentales.

También implementar algunos de los elementos de una estación de trabajo para propósitos de experimentación. Dentro de estos mencionar; una cámara de video, un brazo robótico pequeño con su circuito de control y sensores respectivos, y finalmente, integrar los elementos mínimos requeridos para la parte de comunicación.

Será necesario el uso de una computadora para poder comparar los datos obtenidos. Es en esta y en el FPGA que se comprueba el funcionamiento y la velocidad de ejecución del algoritmo creado para la obtención de la función de frontera.

1.6 METODOLOGÍA

Primero se adquiere una imagen 2D de un objeto en 3D con una cámara fotográfica a colores o a blanco y negro. Previamente se define una región de interés ROI (Region Of Interest) y varios parámetros, como la resolución de la imagen (cantidad de filas y cantidad de columnas) y el tipo de iluminación (direccional o difusa), a utilizar en conjunto con la cámara.

Enseguida se obtiene la imagen en escala de grises a partir de la original; que luego mediante un proceso de binarización queda en formato de unos y ceros, donde un uno representa el objeto y un cero el fondo. De ser necesario se pueden usar métodos de filtrado para eliminar puntos no deseados y ruido.

Esa imagen final, es alimentada a la tarjeta de desarrollo con FPGA integrado, donde están implementados los algoritmos; para el cálculo de la función de frontera y los relacionados a la obtención del centroide y varios puntos del contorno del objeto, los de control de dispositivos para visualización de resultados, y los de control para comunicaciones.

Finalmente se realizan varios experimentos con el sistema funcionando, recopilando las mediciones de los valores calculados y visualizados por el FPGA. De estos datos conformamos la información final para presentar nuestros resultados.

CAPÍTULO II

ANTECEDENTES

2.1 ESTADO DEL ARTE

El sentido de la vista es el que permite al hombre conocer el medio que lo rodea. Las imágenes dentro del espectro visible proporcionan a través del ojo, información sobre el color, la forma, la distancia, posición y movimiento de los objetos [10]. Es el sentido humano más perfecto y evolucionado.

De acuerdo con la teoría de la Gestalt, la forma o configuración de cualquier cosa está compuesta de una figura y un fondo y los elementos son organizados en figuras lo más simples que sea posible. (simétricas, regulares y estables) [6]

La visión artificial es la que busca imitar el funcionamiento de la visión humana [6], esto lo realiza obteniendo características de la imagen y mediante la identificación de patrones [10]. Está muy relacionada con el reconocimiento de objetos, tema de investigación en automatización y robótica.

Las aplicaciones de visión artificial industrial integran sistemas de captura de imágenes digitales, dispositivos de entrada/salida y redes de computadoras para el control de equipos destinados a la fabricación, tales como brazos robóticos [4].

Se dispone de un sensor como lo es una cámara digital, una serie de algoritmos que realizan el reconocimiento de patrones y finalmente se recurre a algún tipo de clasificación, como por ejemplo, el uso de redes neuronales para el proceso de aprendizaje y posterior caracterización de los objetos vistos [1,2].

Redes neuronales: Se basan en la analogía que existe en el comportamiento y función del cerebro humano. Las redes neuronales artificiales son dispositivos o software programado de manera tal que funcionen como las neuronas biológicas de los seres vivos.

La integración de robots en celdas flexibles de manufactura para realizar tareas tediosas, repetitivas y peligrosas, a motivado mucho la investigación en el campo de visión artificial a través del reconocimiento de patrones.

Como en la inspección de partes de manufactura, reconocimiento y clasificación de objetos y operaciones de ensamble[4], los robots deben realizar el reconocimiento de imágenes e identificación para ensamblarlos de manera rápida y sin importar su posición u orientación[5].

Con la función de frontera podemos caracterizar diferentes objetos de las imágenes digitalizadas que vayamos obteniendo, para posteriormente tener una referencia, para comparar con otros similares, y hacer el reconocimiento de patrones respectivo [6].

Varios algoritmos para el cálculo de la función de frontera fueron desarrollados anteriormente en lenguajes de alto nivel como por ejemplo visual C++ y java, y programados en computadoras personales de escritorio [8][9].

Surge entonces la necesidad de mejorar la eficiencia de estos sistemas, y tener un tiempo de ejecución menor. El empleo de un FPGA(Field Programmable Gate Array) es una elección acertada para la implementación de dichos algoritmos, para mejorar el cálculo de la función de frontera (BOF) en tiempo real. Y ese es el objetivo principal de este estudio.

2.2 MARCO TEÓRICO

2.2.1 VISIÓN ARTIFICIAL

La visión artificial por computadora, es una línea de investigación con muchas aplicaciones, como inspección, mediciones y reconocimiento de objetos o asistencia en celdas de manufactura con brazos robóticos. [17]

La captura de una imagen es el conjunto de operaciones que se efectúan para transformar la iluminación de una escena en una señal digital. Las imagen no siempre se presenta en un formato adecuado para su análisis, por lo que el siguiente paso es el pre procesamiento; en el cual, se utilizan técnicas encaminadas a realizar la mejora de la imagen, como son el nivel de gris, contraste, eliminación de ruido o el realce de algunas características de interés.

Una vez que esta imagen está en condiciones de ser procesada, se tienen que hallar los objetos dentro de la misma de forma independiente, y esto se hace a través de la segmentación, que divide la escena en objetos individuales. Cada uno de ellos puede ser caracterizado y clasificado. La siguiente tarea es la extracción de características para el reconocimiento. El reconocimiento es la identificación de cada objeto dentro de la escena mediante una etiqueta.

Las imágenes pueden ser monocromáticas (de niveles de gris) o a colores, pueden provenir de una o varias cámaras e incluso cada cámara, puede ser estacionaria o móvil. Las estructuras y propiedades del mundo tridimensional que queremos deducir en visión artificial incluyen: no sólo sus propiedades geométricas; sino también sus propiedades materiales. Ejemplos de propiedades geométricas son: la forma, tamaño y localización de los objetos. Ejemplos de propiedades de los materiales son: su color, iluminación, textura y composición.

2.2.2 ETAPAS DE LA VISION ARTIFICIAL

Se incluyen diversas técnicas; tales como el procesamiento de imágenes (captura, transformación, codificación) o como el reconocimiento de formas (teoría estadística de decisiones, enfoques

sintácticos y neuronales, aplicados a la clasificación de patrones). En este tipo de sistemas además se incluyen modelado geométrico y procesos para conocimiento.

El objetivo es transformar la imagen del medio ambiente, proporcionada por una cámara, en una descripción de los elementos presentes. Dicha descripción deberá contener la información necesaria para hacerlo. Se deben realizar las siguientes funciones:

a. Iluminación de la escena a capturar mediante la cámara.

b. Captación de la imagen del entorno significativo.

c. Mejoramiento de la imagen y realzado de las características geométricas y topológicas.

d. Segmentación de la imagen.

e. Descripción de la imagen y extracción de características.

f. Reconocimiento de los objetos.

2.2.3 APLICACIONES DE VISIÓN ARTIFICIAL

La mayoría de las aplicaciones de visión artificial se pueden clasificar por el tipo de tarea:

La medición o calibración, se refiere a la correlación cuantitativa con los datos del diseño; asegurando que las mediciones cumplan con las especificaciones del mismo. Por ejemplo, el comprobar que un cable tenga el espesor recomendado.

La detección de fallas, es un análisis cualitativo que involucra la detección de defectos o artefactos no deseados, con forma desconocida o en una posición desconocida. Por ejemplo, encontrar defectos en la pintura de un auto nuevo, o agujeros en hojas de papel.

La verificación, es una revisión cualitativa de que una operación de ensamble ha sido llevada a cabo correctamente. Por ejemplo, que no falte ninguna tecla en un teclado, o que no falten componentes en un circuito impreso.

El reconocimiento, involucra la identificación de un objeto en base a descriptores asociados con este. Por ejemplo, la clasificación de frutas por color y tamaño.

La identificación, es el proceso de identificar un objeto por el uso de símbolos en el mismo. Por ejemplo, el código de barras.

El análisis de localización, es la evaluación de la posición de un objeto. Por ejemplo, determinar la posición donde debe insertarse un circuito integrado.

Guía, significa proporcionar adaptivamente información posicional de retroalimentación para dirigir una actividad. Por ejemplo, guiar un brazo robótico mientras suelda o manipula partes, o la navegación en vehículos autónomos.

2.2.4 DESCRIPCION Y ESTRACCION DE CARACTERÍSTICAS

Para reconocer un objeto es necesario tener una descripción de él (descriptor o modelo). Los descriptores deben ser independientes del tamaño, localización u orientación; y deben ser suficientes para discriminarlos entre sí. Se basan en la evaluación de alguna característica. Descriptores unidimensionales: códigos de cadena, perímetro, forma del perímetro; descriptores bidimensionales: área, momentos de inercia; descriptores específicos: número de agujeros, área de agujeros, posición relativa de agujeros, rasgos diferenciadores.

Uno de los principales problemas en el reconocimiento de patrones es encontrar una manera óptima de representar la información original que describe a cada uno, basado en los descriptores mencionados inicialmente. Este problema es conocido como extracción de características, que trata de reducir la cantidad de información (reducción de dimensionalidad) que representa a cada uno de los patrones, obteniendo de esta forma, un vector de características que represente de la mejor manera posible al patrón original

CAPÍTULO III

BOF(BOUNDARY OBJECT FUNCTION)

3.1 INTRODUCCIÓN

Con el objetivo de reconocer las formas de figuras en 2D, a partir de un objeto en 3D, se utilizan dispositivos de captura, como lo es una cámara de video ó una cámara fotográfica, para pasar de la escena tridimensional del mundo real a una bidimensional; actualmente en forma de imagen digitalizada. Esta depende del punto de vista de la toma, en nuestro caso la cámara estará en un punto fijo.

Luego de la digitalización, se busca como siguiente resultado una imagen binarizada, esto se consigue eliminando el ruido con algún tipo de filtro y una comparación de umbrales. Este procedimiento permite segmentarla en dos regiones: figura y fondo. Como consecuencia tenemos solamente dos niveles de gris.

3.2 MODELO DE UNA IMAGEN

Se define una imagen: $Im = f(x, y)$

Para un rango de niveles de gris de: $0 <= Gl <= L - 1$

Donde Gl(Gray Level) es el nivel de gris de cada punto en la imagen y L la cantidad de niveles:

$$\exists Gl(i, j) \in \mathrm{Im}(xi, yj) \text{ donde } Gl(i, j) = 0, L-1, \forall i, j = 1, n$$

Podemos modelarla con una matriz M pixeles por N pixeles, elementos $p(i, j)$ de valores escalares, que indica el flujo de intensidad de luz en el elemento (x, y) de la imagen representado por el pixel. [20]

Donde: N es el número máximo de pixeles en una columna y M es el número máximo de pixeles en una fila.

Un pixel específico en su forma básica, es identificado por las coordenadas en el arreglo $M\ xN$ que representa la imagen. Cada pixel tiene un valor numérico codificado como un uno o un cero, para la representación binaria. (Figura 3.1)

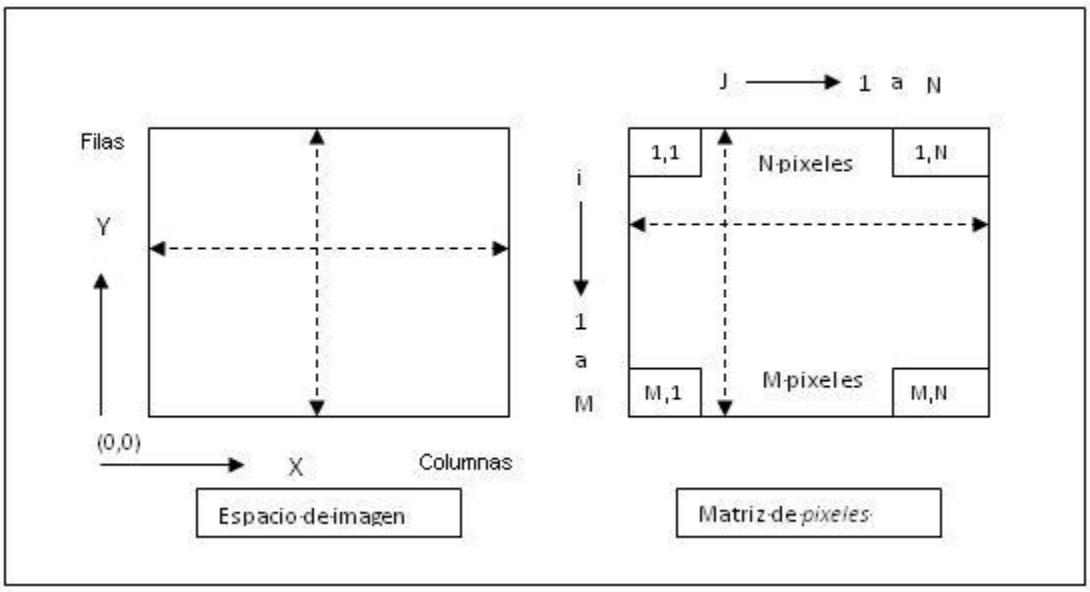

Figura 3.1 Modelado matemático de una imagen.
Apreciamos la correspondencia de los elementos de la imagen digitalizada con los pixeles del modelo. [20]

3.3 CARACTERÍSTICAS PARAMÉTRICAS 2D

Para caracterizar una forma podemos utilizar sus propiedades métricas para obtener un vector descriptivo para identificarla. Las características métricas se basan en una medida de la distancia $d(V1, V2)$ entre dos puntos $V1$ y $V2$ del plano de la imagen. Para calcularla usamos la distancia Euclidiana, deducida a partir del teorema de Pitágoras como sigue:

Distancia euclidiana: $dE(V1, V2) = \sqrt{(V2x - V1x)^2 + (V2y - V1y)^2}$

Donde: $(V1x, V1y)$ son las coordenadas del punto $V1$; y $(V2x, V2y)$ son las coordenadas del punto $V2$, dentro de un mismo plano.

Al trabajar con imágenes binarias, que son representadas por pixeles, todos estos valores estrían cuantificados en pixeles.

Algunas de estas propiedades métricas son: el área, que se calcula sumando la cantidad de pixeles que pertenecen a la forma:

$$\hat{A} = \sum_m \sum_n Pixels,(m,n) \in forma$$

Otra el perímetro, que es la cantidad de pixeles que pertenecen al contorno:

$$\hat{P} = \sum_m \sum_n Pixels,(m,n) \in Contorno$$

La excentricidad, que es la relación entre el eje mayor y el eje menor de la forma, también llamada elongación:

$$E=R/S$$

Donde: *R* es la cantidad de pixeles del eje mayor y *S* es la cantidad de pixeles del eje menor.

Otras características utilizadas son, el centroide:

$$X_c = \frac{\sum_{x,y} j}{A}, Y_c = \frac{\sum_{x,y} i}{A}$$

Donde *A* es el área de la forma expresada en pixeles; y (X, Y) las coordenadas del centroide obtenido.

El BOF(Boundary Object Function), que está formado por las distancias del centroide a los puntos del contorno:

Para un elemento del contorno, con coordenadas (Px, Py)

Y el centroide con coordenadas (Cx, Cy):

$$BOF(n) = \sqrt{(Px - Cx)^2 + (Py - Cy)^2}$$

Donde $BOF(n)$ es uno de los valores de la función BOF, los demás se calculan de manera similar.

En este trabajo vamos a realizar el cálculo de dieciséis valores, para un BOF de dieciséis elementos, para realizar la caracterización de dos objetos, uno cúbico y otro cilíndrico. (*n* = 1..16)

3.4 FUNCIÓN DE FRONTERA BOF

Para obtener el BOF ó función de frontera de un objeto, se tiene originalmente un objeto real en 3D, de este se captura una imagen estática en 2D a través de algún medio visual que generalmente es una cámara fotográfica o de video, así al tener la figura del objeto, que deberá estar en contraste con el fondo, tomamos una parte de la imagen que lo contenga, a esto lo denominamos región de interés ROI(Región of interest).

Entonces se extrae el contorno de la figura (puntos externos que separan dicha figura del fondo), y se procede a calcular las coordenadas del centroide, que es su centro geométrico. Ya con estos valores podemos conformar nuestro BOF con las magnitudes de las distancias desde el centro mencionado hasta determinados puntos del contorno.

En síntesis, el cálculo de la función de frontera BOF (Boundary Object Function) implica obtener el centroide de la figura en 2D del objeto, cuya imagen fue capturada por una cámara, la obtención de varios puntos del contorno de la figura y el cálculo de la distancia de cada uno de esos puntos

hasta el centroide. Todas las magnitudes de esas distancias conforman la función de frontera o BOF.

La función de frontera es un parámetro muy útil para poder diferenciar las formas de las figuras resultantes de objetos 3D capturados en imagen 2D.

Veamos dos figuras para observar las diferencias, un cuadrado y un círculo: (Figura 3.2)

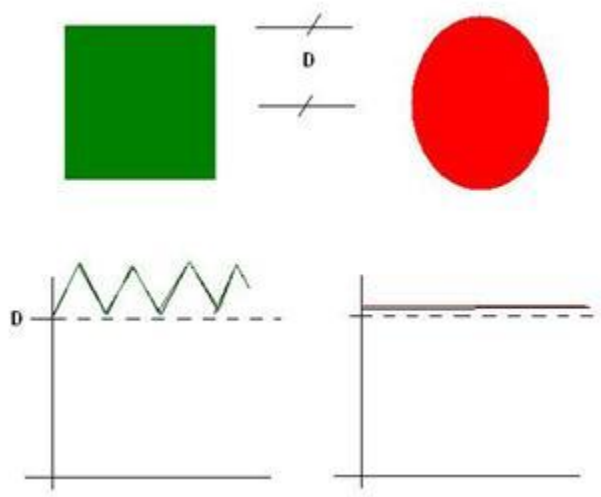

Figura 3.2 Función de frontera de un cuadrado y un círculo.
Comparación del BOF de dos figuras regulares de dimensiones aproximadas.
[Elaboración propia]

La función de frontera nos permite diferenciar una figura de otra, al ser los patrones característicos de cada una diferentes entre sí. Es por esta razón el interés de realizar algún tipo de método para su cálculo.

3.4.1 FUNCIÓN DE FRONTERA DE UN CUADRADO

Si tomamos un cuadrado con un valor de lado igual a L unidades y queremos hallar su función de frontera, debemos primero definir un sistema de coordenadas y su origen. Conseguimos el centroide de la figura y luego dividimos esta en secciones iguales, tomemos por ejemplo dieciséis divisiones espaciadas por ángulos iguales (22.5 grados por cada una) y partiendo del norte recorremos en dirección de las manecillas del reloj, obteniendo las coordenadas de los puntos del contorno que cortan con los lados de sección. Una vez hecho esto y por medio de la distancia entre dos puntos conocidos calculamos las distancias entre estos y el centroide. Al final tenemos un vector de dieciséis valores que nos permite caracterizar al cuadrado, esto lo podemos graficar, y observar el patrón característico del mismo. (Figura 3.3)

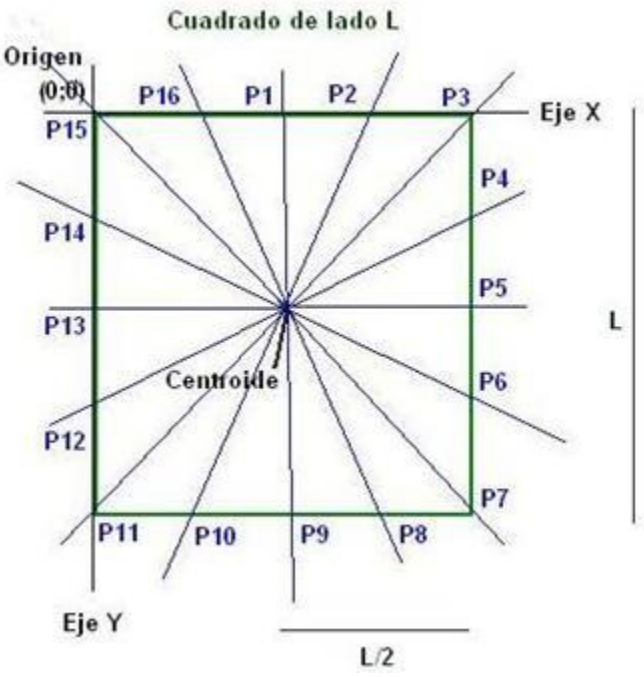

Figura 3.3 BOF de un cuadrado.
Descripción gráfica para el cálculo de la función de frontera de un cuadrado de lado L.
[Elaboración propia]

Tenemos definido nuestro sistema de coordenadas de ejes cartesianos con el origen $O(0;0)$ en la esquina superior izquierda del cuadrado. Definiendo valores positivos hacia abajo en el eje Y de las ordenadas y valores positivos hacia la derecha en el eje X de las abscisas y debido a la simetría de la figura, calculamos las siguientes coordenadas:

Coordenadas del centroide: $C(Cx;Cy) = \left(\frac{L}{2};\frac{L}{2}\right)$

Coordenada del punto 1 : $P1(P1x;P1y) = (L/2;0)$
Coordenada del punto 2 : $P2(P2x;P2y) = (3L/4;0)$
Coordenada del punto 3 : $P3(P3x;P3y) = (L;0)$
Coordenada del punto 4 : $P4(P4x;P4y) = (L;L/4)$
Coordenada del punto 5 : $P5(P5x;P5y) = (L;L/2)$
Coordenada del punto 6 : $P6(P6x;P6y) = (L;3L/4)$
Coordenada del punto 7 : $P7(P7x;P7y) = (L;L)$
Coordenada del punto 8 : $P8(P8x;P8y) = (3L/4;L)$
Coordenada del punto 9 : $P9(P9x;P9y) = (L/2;L)$
Coordenada del punto 10: $P10(P10x;P10y) = (L/4;L)$
Coordenada del punto 11: $P11(P11x;P11y) = (0;L)$
Coordenada del punto 12: $P12(P12x;P12y) = (0;3L/4)$
Coordenada del punto 13: $P13(P13x;P13y) = (0;L/2)$
Coordenada del punto 14: $P14(P14x;P14y) = (0;L/4)$

Coordenada del punto 15: $P15(P15x; P15y) = (0; 0)$
Coordenada del punto 16: $P16(P16x; P16y) = (L/4; 0)$

Con esta información procedemos a calcular las distancias de cada uno de estos puntos al centroide.

Si definimos como: $D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14, D15, D16$ a estas distancias Tenemos estos resultados:

$$D1 = \sqrt{(\mathrm{Cx} - \mathrm{P1x})^2 + (\mathrm{Cy} - \mathrm{P1y})^2} = \sqrt{(\mathrm{L}/2 - \mathrm{L}/2)^2 + (\mathrm{L}/2 - 0)^2} = \sqrt{0 + L^2/4} = L/2$$

$$D2 = \sqrt{(\mathrm{Cx} - \mathrm{P2x})^2 + (\mathrm{Cy} - \mathrm{P2y})^2} = \sqrt{(\mathrm{L}/2 - 3\mathrm{L}/4)^2 + (\mathrm{L}/2 - 0)^2} = \sqrt{L^2/16 + L^2/4} = L/4\sqrt{5}$$

$$D3 = \sqrt{(\mathrm{Cx} - \mathrm{P3x})^2 + (\mathrm{Cy} - \mathrm{P3y})^2} = \sqrt{(\mathrm{L}/2 - \mathrm{L})^2 + (\mathrm{L}/2 - 0)^2} = \sqrt{L^2/4 + L^2/4} = L\sqrt{1/2}$$

Las demás distancias se van repitiendo en el caso del cuadrado y quedan así:

$$D1 = D5 = D9 = D13 \ = \ 0.5\, L$$

$$D2 = D4 = D6 = D8 = D10 = D12 = D14 = D16 \ = \ 0.559\, L$$

$$D3 = D7 = D11 = D15 \ = \ 0.707\, L$$

Y nuestra función de frontera BOF del cuadrado será: (Figura 3.4)

$$BOF = [D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14, D15, D16]$$

3.4.2 FUNCIÓN DE FRONTERA DE UN CÍRCULO

Si tomamos un círculo de radio igual a R unidades y queremos hallar su función de frontera, debemos primero definir un sistema de coordenadas y su origen. Conseguimos el centroide de la figura y luego dividimos esta en secciones iguales, tomemos por ejemplo dieciséis divisiones espaciadas por ángulos iguales (22.5 grados por sección) y partiendo del norte recorremos en dirección de las manecillas del reloj, obteniendo las coordenadas de los puntos del contorno en cada sección. Una vez hecho esto y por medio de la distancia entre dos puntos conocidos, calculamos las distancias entre estos y el centroide. Al final tenemos un vector de dieciséis valores que nos permite caracterizar al círculo, esto lo podemos graficar, y observar el patrón característico del mismo. (Figura 3.5)

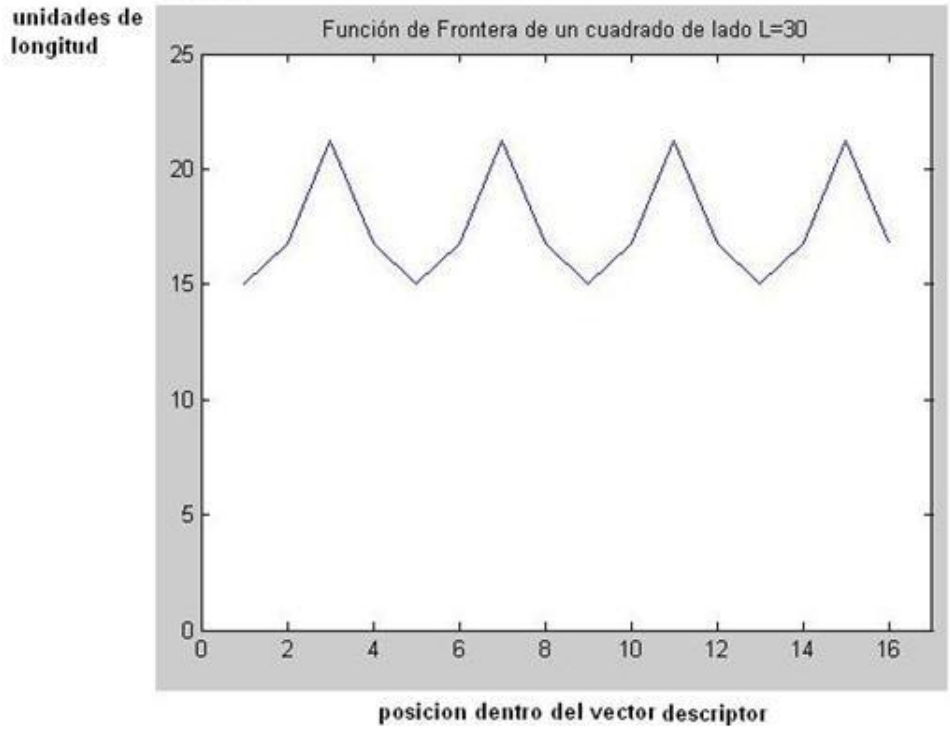

Figura 3.4 Función de frontera de un cuadrado.
Función característica de un cuadrado de lado L=30 unidades de longitud.
[Elaboración propia]

Tenemos definido nuestro sistema de coordenadas de ejes cartesianos con el origen $O(0;0)$ en el centro del círculo y debido a la simetría de la figura calculamos las siguientes coordenadas:

Coordenadas del centroide: $C(Cx;Cy) = (0;0)$

En cuanto a las coordenadas de los puntos del contorno del círculo no son necesarias por ahora, pero en la práctica si se deben calcular. En este caso como lo que queremos es la distancia al centroide, todas las distancias serían iguales al radio R.

Si definimos como: $D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14, D15, D16$ a estas distancias Tenemos estos resultados:

$$D1 = D2 = D3 = D4 = D5 = D6 = D7 = D8 = D9 = D10 = D11 = D12 = D13 = D14 = D15 = D16 = R$$

Y nuestra función de frontera BOF del círculo será: (Figura 3.6)

$$BOF = [D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14, D15, D16]$$

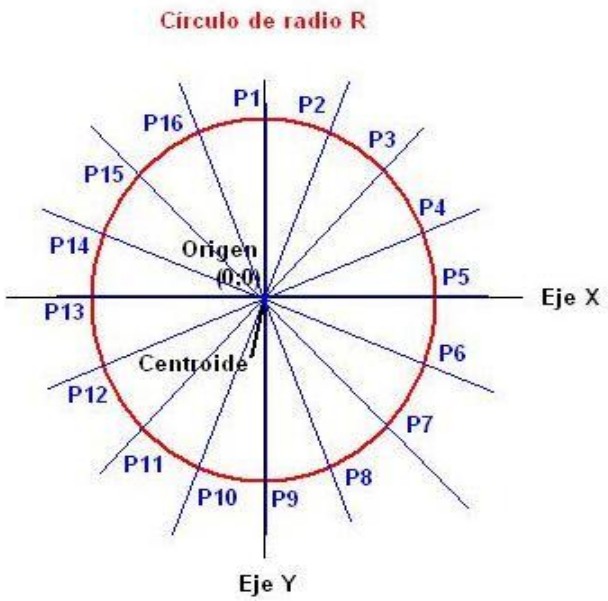

Figura 3.5 BOF de un círculo.
Descripción gráfica para el cálculo de la función de frontera de un círculo de radio R.
[Elaboración propia]

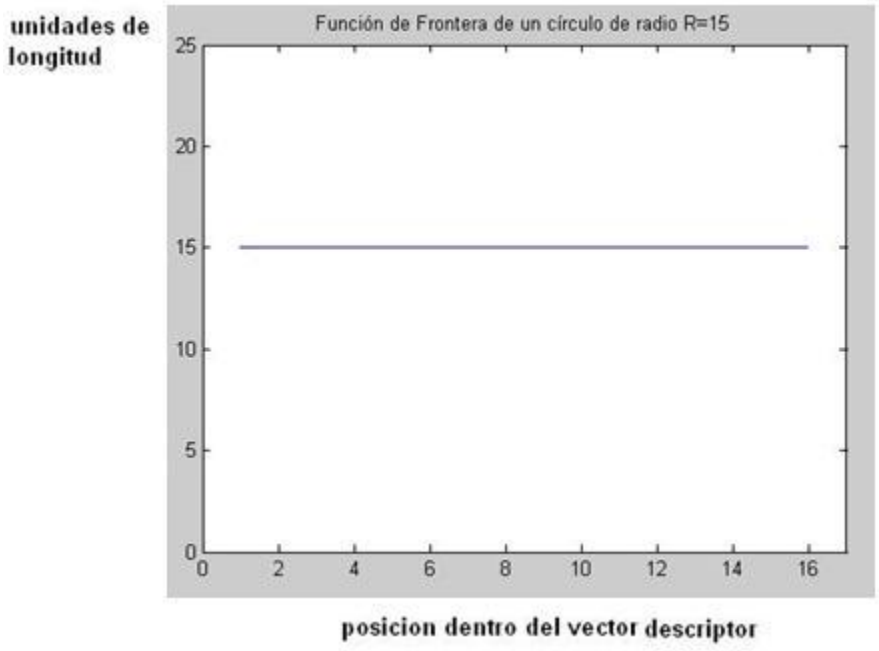

Figura 3.6. Función de frontera de un círculo.
Función característica de un círculo del radio R=15 unidades de longitud.
[Elaboración propia]

CAPÍTULO IV

FPGA (FIELD PROGRAMMABLE GATE ARRAY)

4.1 INTRODUCCIÓN

Los arreglos de compuertas programables en campo o FPGA, fueron inventados en el año 1984 por Ross Freeman y Bernard Vonderschmitt, co-fundadores de Xilinx, y surgen como una evolución de los CPLD(Complex Programmable Logic Device).

Al principio de los sesentas se tenían circuitos lógicos discretos (AND, OR, Flip-Flop). Los sistemas eran construidos a partir de inmensas cantidades de estos, con cableados complejos. Era difícil su modificación o mantenimiento y su fabricación tomaba bastante tiempo, y además, generalmente requerían de una nueva tarjeta de circuito impreso.

Surgen entonces los PLD(Programmable Logic Device) que contienen un arreglo de fusibles que pueden dejarse intactos o quemarse para interconectar varias compuertas AND-OR. Con esto se evitaba el recableado y la fabricación de otra tarjeta impresa tan solo con el intercambio de un nuevo PLD. El inconveniente era que no podían manejar diseños lógicos muy grandes.

Los fabricantes de circuitos integrados nos dan entonces una solución con los CPLD y los FPGA(Field Programmable Gate Array), en los cuales puedes tener esencialmente un sistema completo en un solo chip.

Un CPLD contiene varios PLD cuyas entradas y salidas están relacionadas entre sí por una matriz de interconexión global. Cada PLD así como las interconexiones entre ellos pueden ser programados. (Figura 4.1)

Un FPGA tiene un enfoque diferente. Tiene varios bloques lógicos configurables CLB(Configurable Logic Blocks) entrelazados en una configuración de matriz, pudiendo reacomodar las interconexiones entre ellos. Cada bloque es programado individualmente para realizar una función lógica determinada y entonces las interconexiones son programadas, para conectar estos últimos, de manera tal que todo el sistema lógico quede implementado.[19] (Figura 4.2)

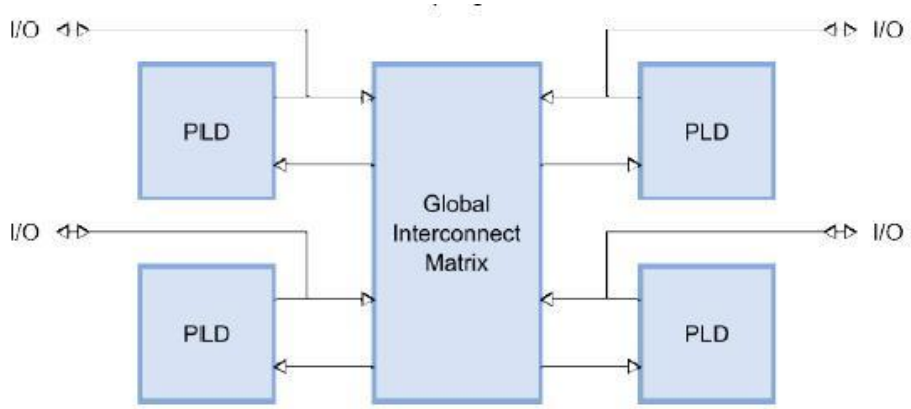

Figura 4.1 Configuración básica de un CPLD.[19]

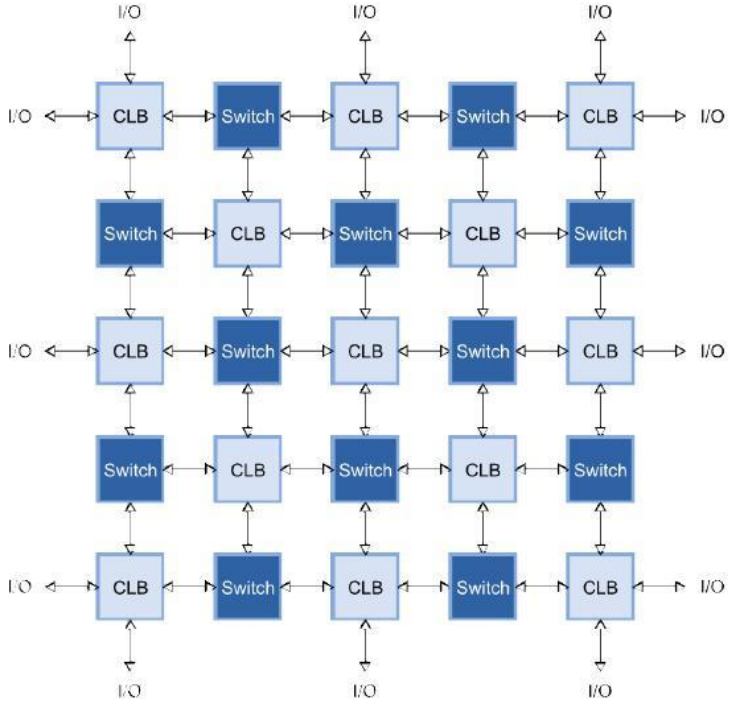

Figura 4.2 Configuración básica de un FPGA.[19]

4.2 ARQUITECTURA DE UN FPGA

4.2.1 BLOQUES LÓGICOS CONFIGURABLES

Un FPGA está formado por una matriz de bloques lógicos configurables CLB(Configurable Logic Block), que a su vez están formados por secciones(Slices) y cada sección está formada por celdas lógicas LC(Logic Cell). (Figura 4.3)

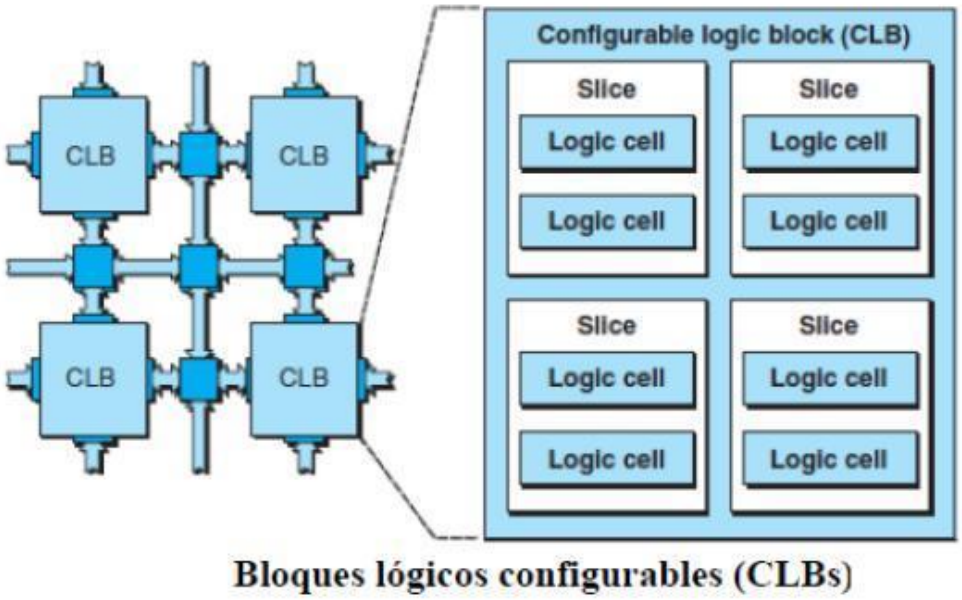

Figura 4.3 Configuración de un bloque lógico configurable. [19]

4.2.2 SECCIONES

Una sección está formada por dos o más celdas lógicas individuales, que comparten la misma señal de reloj. (Figura 4.4)

Algunos FPGA de Xilinx tienen dos secciones por cada bloque y otros tienen cuatro. La razón para tener este tipo de arquitectura jerárquica es que es complementada por una equivalente en la conexión. De tal forma que se tienen conexiones muy rápidas con retardos de propagación muy cortos entre elementos lógicos dentro de una sección, luego un poco más lentas entre secciones dentro de un bloque y más lentas entre estos. La finalidad es lograr una óptima compensación entre hacer que las conexiones sean fáciles, sin incurrir mucho en los retardos.

4.2.3 CELDA LÓGICA

Es la unidad más pequeña de un FPGA. Contiene principalmente una tabla de referencia LTU(Look Up Table) de cuatro entradas, un multiplexor y un registro. Cada tabla se puede usar como una RAM de 16x1bits o como un registro de corrimiento de 16bits. (Figura 4.5)

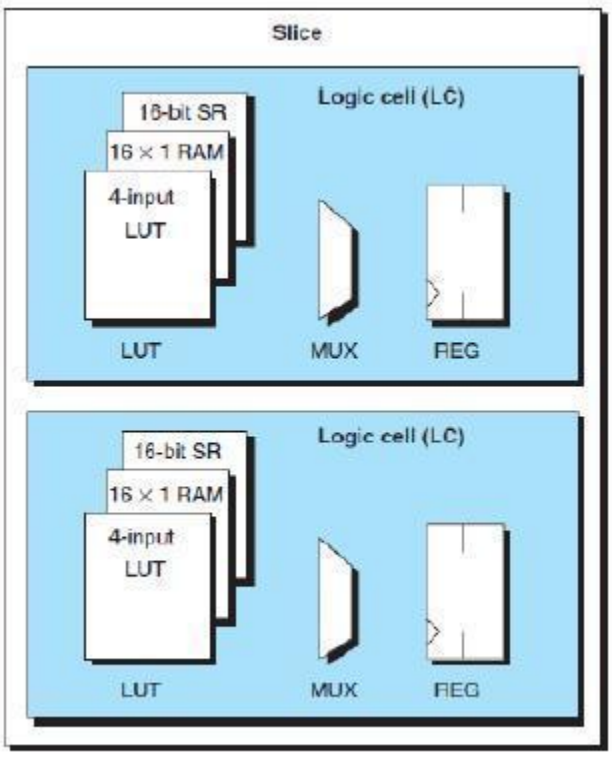

Figura 4.4 Configuración de una sección. [19]

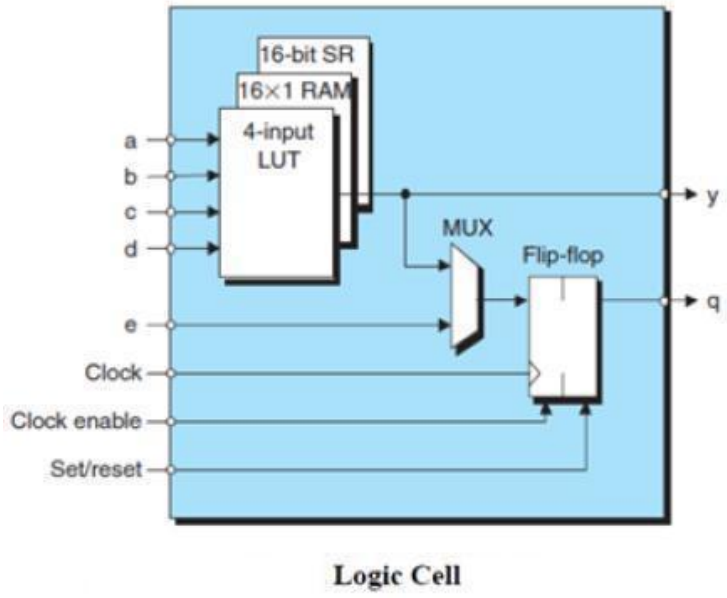

Figura 4.5 Configuración de una celda lógica [19].

4.2.4 MEMORIA RAM DISTRIBUIDA Y REGISTROS DE CORRIMIENTO

Cada tabla de referencia puede ser usada como una RAM(Random Access Memory) de 16x1bits y asumiendo cuatro secciones por bloque, todas las tablas dentro de un bloque pueden ser configuradas juntas para implementar una RAM de 16x8bits, de 32x4bits, de 64x2bits, de 128x1bits, de 16x4bits, de 32x2bits, o de 64x1bits. Además cada tabla de referencia de 4bits puede ser usada como un registro de desplazamiento SR(Shift Register) de 16bits. En este caso, existen conexiones especiales entre celdas lógicas dentro de una sección y también entre estas para dejar que el último bit se conecte al primero de otro, permitiendo implementar registros de corrimiento de 128bits por bloque.

Cada celda lógica también contiene una lógica especial para manejo de acarreos, lo cual permite realizar circuitos aritméticos como contadores. Juntando esta lógica de acarreos con los registros de desplazamiento y multiplicadores embebidos, permite a los FPGA realizar operaciones de procesamiento digital de señales.

La tabla 4.1 muestra la cantidad de bloques lógicos configurables(CLB), secciones(Slices), tablas de referencia(LUT) y memoria RAM(Random Access Memory) distribuida, de la familia Spartan-3 de Xilinx.

4.2.5 MEMORIA RAM EMBEBIDA

Muchas aplicaciones requieren el uso de memoria, de tal forma que los FPGA poseen largos bloques de memoria RAM(Random Access Memory). Dependiendo de la arquitectura estos pueden ir posicionados alrededor de la matriz de bloques o estar organizada en columnas. (Figura 4.6)

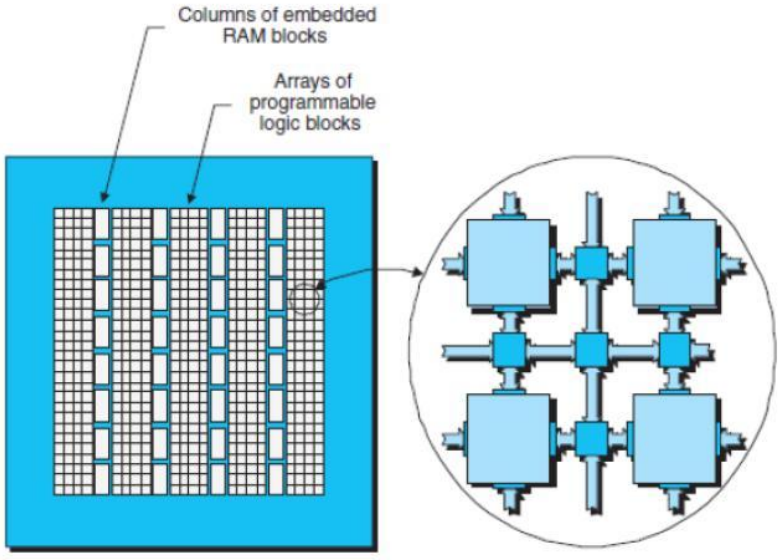

Figura 4.6 Memoria RAM embebida.[19]

Tabla 4.1. Familia Spartan-3 de Xilinx[19]

Device	CLB Rows	CLB Columns	CLB Total	Slices	LUTs / Flip-Flops	Equivalent Logic Cells	RAM16 / SRL16	Distributed RAM Bits
Spartan®-3A DSP FPGA CLB Resources								
XC3SD1800A	88	48	4,160	16,640	33,280	37,440	16,640	266,240
XC3SD3400A	104	58	5,968	23,872	47,744	53,712	23,872	381,952
Spartan-3A/3AN FPGA CLB Resources								
XC3S50A/AN	16	12	176	704	1,408	1,584	704	11,264
XC3S200A/AN	32	16	448	1,792	3,584	4,032	1,792	28,672
XC3S400A/AN	40	24	896	3,584	7,168	8,064	3,584	57,344
XC3S700A/AN	48	32	1,472	5,888	11,776	13,248	5,888	94,208
XC3S1400A/AN	72	40	2,816	11,264	22,528	25,344	11,264	180,224
Spartan-3E FPGA CLB Resources								
XC3S100E	22	16	240	960	1,920	2,160	960	15,360
XC3S250E	34	26	612	2,448	4,896	5,508	2,448	39,168
XC3S500E	46	34	1,164	4,656	9,312	10,476	4,656	74,496
XC3S1200E	60	46	2,168	8,672	17,344	19,512	8,672	138,752
XC3S1600E	76	58	3,688	14,752	29,504	33,192	14,752	236,032
Spartan-3 FPGA CLB Resources								
XC3S50	16	12	192	768	1,536	1,728	768	12,288
XC3S200	24	20	480	1,920	3,840	4,320	1,920	30,720
XC3S400	32	28	896	3,584	7,168	8,064	3,584	57,344
XC3S1000	48	40	1,920	7,680	15,360	17,280	7,680	122,880
XC3S1500	64	52	3,328	13,312	26,624	29,952	13,312	212,992
XC3S2000	80	64	5,120	20,480	40,960	46,080	20,480	327,680
XC3S4000	96	72	6,912	27,648	55,296	62,208	27,648	442,368
XC3S5000	104	80	8,320	33,280	66,560	74,880	33,280	532,480

4.2.6 SUMADORES, MULTIPLICADORES Y MAC EMBEBIDOS

Los circuitos sumadores y multiplicadores son lentos ya que generan largos retrasos de propagación al conectar un gran número de bloques juntos. Debido a esto y a que son operaciones muy comunes, muchos FPGA incorporan bloques sumadores y multiplicadores. Estos se encuentran localizados cerca de la memoria RAM embebida debido a que comúnmente se usan juntos. (Figura 4.7)

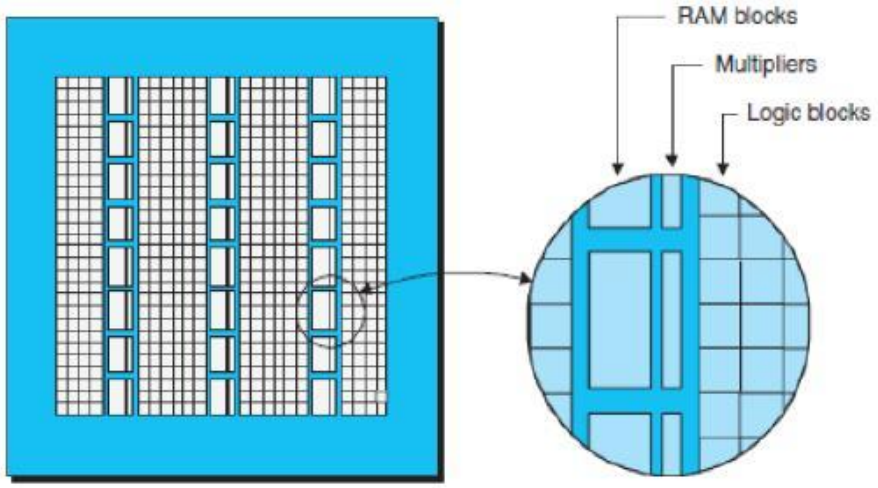

Figura 4.7 Sumadores y multiplicadores embebidos.[19]

Una operación muy común en las aplicaciones de procesamiento digital de señales DSP (Digital Signal Processing) es la llamada multiplica y acumula MAC(Multiply–ACcumulate). Como su nombre lo indica, esta función multiplica dos números y luego envía el resultado a un acumulador. Esta operación es usada para el cálculo de la convolución, transformada rápida de Fourier FFT(Fast Fourier Transform), transformada rápida de wavelet FWT(Fast Wavelet Transform), entre otras.

4.2.7 PROCESADORES EMBEBIDOS

Los FPGA nos brindan la posibilidad de diseñar sistemas digitales de muy alta velocidad (picosegundos y nanosegundos) y un alto grado de paralelismo. Sin embargo la mayoría de las aplicaciones requieren realizar gran número de operaciones por programa, por tanto, casi todos los diseños digitales hacen uso de microprocesadores o microcontroladores.

Actualmente los FPGA de alta capacidad, poseen uno o más microprocesadores embebidos denominados núcleos(Microprocessor Cores), de tal forma que un mismo FPGA puede hacer las tareas tanto por software como por hadware, implementando lo que se llaman Soc(System on Chip) brindando importantes ventajas como son: disminución de costo (un solo circuito integrado), disminución de un gran número de pistas, pines y pads en el PCB(Printed Circuit Board) y circuitos más pequeños y livianos.

4.2.8 PROPIEDAD INTELECTUAL IP(INTELLECTUAL PROPERTY)

Los diseños actuales con FPGA son tan grandes y complejos que sería impráctico crear cada porción del diseño desde cero. Una solución a esto es reutilizar bloques funcionales existentes y dedicar la mayor parte del tiempo en crear las nuevas porciones del diseño que lo hacen diferente de otros.

A los bloques funcionales existentes se les llama propiedad intelectual o IP. Existen tres fuentes principales de IP: bloques creados anteriormente, reutilizados de otros diseños, fabricantes de FPGA, o proveedores de IP externos.

Cada fabricante de FPGA tiene su propia colección de hard, firm y soft IP.

Los hard IP vienen en forma de bloques pre implementados, tales como núcleos de microprocesador, interfaces, multiplicadores, sumadores, y otros. Estos bloques están diseñados para ser lo más eficiente posible en términos de consumo de potencia, espacio y rendimiento.

Los soft IP vienen en forma de bibliotecas en lenguaje de alto nivel que pueden ser incluidas en los diseños.

Los firm IP se encuentran en un punto medio, los cuales también vienen en forma de funciones de bibliotecas, pero a diferencia de los soft IP, estas funciones ya están óptimamente mapeadas, colocadas y ruteadas en un grupo de bloques lógicos programables.

4.3 ÁREAS DE APLICACIÓN

Cualquier circuito de aplicación específica puede ser configurado dentro de un FPGA, siempre y cuando, este último tenga los recursos necesarios.

En procesamiento digital de señales, sistemas aeroespaciales y de defensa, tratamiento de imágenes para medicina, visión para computadoras, reconocimiento de voz, bioinformática, emulación de hardware de computadora y prototipos de ASIC entre otras.

Su uso en otras áreas es cada vez mayor, sobre todo en aquellas aplicaciones que requieren un alto grado de paralelismo, sistemas en tiempo real o una velocidad de procesamiento muy rápida.

CAPÍTULO V

LENGUAJE DESCRIPTOR DE HARDWARE

Para poder implementar un diseño lógico en una tarjeta con FPGA, se recurre al uso de un lenguaje descriptor de hardware HDL(Hardware Description Language), como el VHDL o el Verilog que son comúnmente empleados. En este trabajo de investigación hago uso del lenguaje VHDL.

Conforme los FPGA fueron creciendo en potencia y velocidad, aparecieron más versiones de lenguajes de alto nivel conocidos como C y C++ para FPGA. Otras opciones incluyen Simulink y LabVIEW.

5.1 LENGUAJE VHDL

El VHDL originalmente surgió como un lenguaje para simulación de circuitos digitales, definido por el Instituto de Ingenieros Eléctricos y Electrónicos IEEE(Institute of Electrical and Electronics Engineers) (ANSI/IEEE 1076-1993). Para el diseño se usaban otras herramientas como esquemáticos y lista de conexiones (Netlist).

Conforme los circuitos digitales fueron haciéndose más complejos surgió la necesidad de poder describirlos con un alto grado de abstracción, no desde el punto de vista estructural, más bien del funcional.

El VHDL es un lenguaje de alto nivel de abstracción que nos permite el uso de código estructurado mediante palabras reservadas y sentencias de control. Este código se crea y modifica dentro de cualquier editor de texto simple, incluido generalmente en los paquetes de desarrollo para tal fin.

5.1.1 PARTES DEL CÓDIGO

La ventaja del VHDL es que nos permite organizar jerárquicamente los diseños, de manera que cada elemento en conjunto con otros de nivel inferior, es en sí mismo un diseño auto contenido.[18]

5.1.1.1 BIBLIOTECA

La palabra reservada usada para esta parte es LIBRARY(Biblioteca), es una parte muy importante. Dentro de estas se pueden definir componentes, funciones y procedimientos, es decir, unidades del lenguaje que pueden ser útiles en múltiples aplicaciones, siendo un elemento de reutilización

del lenguaje. Las bibliotecas se estructuran en packages(paquetes) que permiten una fácil clasificación de los elementos que la componen.

Sintaxis:

```
LIBRARY Nombre_biblioteca;
use Nombre_biblioteca.Paquete1.ALL;
use Nombre_biblioteca.Paquete2.ALL;
. . .
```

5.1.1.2 ENTIDAD

Un elemento del diseño es designado por la palabra reservada ENTITY(Entidad), tiene asociado un nombre identificador, usualmente relativo a la función que realiza. Cada vez que hagamos uso de este circuito utilizaremos el nombre asociado. Está relacionada por señales de enlace con el exterior y una arquitectura funcional. Es donde se declaran las entradas y salidas para la interfaz del módulo de hardware dejando escondidos los detalles internos.

Sintaxis:

```
ENTITY Nombre_entidad IS
PORT ( Nombre_señal: modo tipo_de_señal;
. . .
Nombre de señal: modo tipo_de_señal ) ;

END Nombre_entidad;
```

5.1.1.3 ARQUITECTURA

Describe la funcionalidad y consta de dos partes: la parte declarativa, donde se especifican los elementos de tipo estructural que van la van a componer, y la parte descriptiva, donde se define la funcionalidad que concretamente se le asigna a la arquitectura. Es la descripción detallada de la estructura interna del diseño, su palabra reservada es ARCHITECTURE(Arquitectura).

Sintaxis:

```
ARCHITECTURE Nombre_arquitectura OF Nombre_entidad IS
Declaración de tipos
Declaración de señales
Declaración de constantes
Declaración de componentes
Definición de funciones
Definición de procedimientos

BEGIN
Enunciado concurrente
. . .
Enunciado concurrente

END Nombre_arquitectura;
```

Ejemplo de diseño de una compuerta AND de dos entradas; con las tres partes del código descritas anteriormente:

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

ENTITY ejemplo1 IS
PORT ( a, b : IN bit;
        F : OUT bit );
END ejemplo1;

ARCHITECTURE and_2 OF ejemplo1 IS
BEGIN
F <= a AND b;
END and_2;
```

5.2 TIPOS DE PROGRAMACIÓN

Podemos crear nuestros circuitos de varias maneras, eso depende de cada diseñador. Fundamentalmente tres: comportamiento, flujo de datos y estructural. También es válida una combinación de todas o algunas de ellas.

5.2.1 PROGRAMACIÓN POR COMPORTAMIENTO

Describe el comportamiento del circuito mediante sentencias típicas en los lenguajes de programación de software, como: IF(Si), ELSE(Si no), THEN(Entonces) entre otras. Al ser un estilo de programación secuencial, necesita obligatoriamente estar dentro de un PROCESS(Proceso).

Ejemplo de un multiplexor de dos entradas con señal de habilite; por comportamiento:

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

ENTITY ejemplo2 IS
PORT ( a, b    : IN bit;
       sel,en : IN bit;
       F : OUT bit );
END ejemplo2;

ARCHITECTURE mux_2 OF ejemplo2 IS
BEGIN
       PROCESS(a,b,sel)
       BEGIN
              IF en = '1' THEN
                     F <= '1';
              ELSE
                     IF sel = '0' THEN
                            F <= b;
                     ELSE
                            F <= a;
                     END IF;
              END IF;
```

```
        END PROCESS;
END emux_2;
```

5.2.2 PROGRAMACIÓN POR FLUJO DE DATOS

Describe como es transferida la información desde la entrada a la salida, sin el uso de sentencias secuenciales, sino concurrentes. Veamos el mismo diseño con este tipo de programación.

Ejemplo de un multiplexor de dos entradas con señal de habilite; por flujo de datos:

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

ENTITY ejemplo3 IS
PORT ( a, b    : IN bit;
        sel,en : IN bit;
        F : OUT bit );
END ejemplo3;

ARCHITECTURE mux_2 OF ejemplo3 IS
        SIGNAL aux,ax,bx,ux: bit;
BEGIN
        aux <= NOT se;
        ax <= a AND sel;
        bx <= b AND aux;
        ux <= ax OR bx;
        F <= ux OR en;
END mux_2;
```

5.2.3 PROGRAMACIÓN ESTRUCTURAL

Describe los componentes internos como si fueran netlist(listas de conexionado), las cuales se evalúan mediante señales. Permite mayor control sobre la configuración interna de los dispositivos programables, sin embargo el tamaño del programa es más largo y complejo.

Ejemplo de un multiplexor de dos entradas con señal de habilite; estilo estructurado:

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

ENTITY ejemplo4 IS
PORT ( a, b    : IN bit;
        sel,en : IN bit;
        F : OUT bit );
END ejemplo4;

ARCHITECTURE mux_2 OF ejemplo4 IS

        COMPONENT and2
                PORT(e1,e2: IN bit; y: OUT bit);
        END COMPONENT;
```

```
        COMPONENT or2
                PORT(e1,e2: IN bit; y: OUT bit);
        END COMPONENT;

        COMPONENT inv
                PORT(e: IN bit; y: OUT bit);
        END COMPONENT;

        SIGNAL aux,ax,bx,ux: bit;

BEGIN
        u0: inv PORTMAP(e=>sel,y=>aux);
        u1: and2 PORTMAP(e1=>b, e2=>aux, y=>bx);
        u2: and2 PORTMAP(e1=>a, e2=>sel, y=>ax);
        u3: or2 PORTMAP(e1=>ax, e2=>bx, y=>ux);
        u4: or2 PORTMAP(e1=>ux, e2=>en, y=>sal);
END mux_2;
```

5.3. PALABRAS RESERVADAS DEL VHDL

Existen, como en cualquier lenguaje de programación, ciertas palabras que no podemos usar para identificar nuestras señales y variables, ya que son propias del paquete para realizar sus propios procesos. De esta versión del ISE 10.1 mostramos una lista, recordando que estas pueden ser modificadas o diferentes en otras versiones. (Figura 5.1)

5.4 CONFIGURACION FÍSICA DEL FPGA

El proceso de determinar la función que deben realizar los bloques lógicos configurables y la matriz de interconexión en un FPGA, con el fin de crear un circuito lógico, lo desarrollamos gracias al software que nos facilitan los fabricantes; que compila una descripción lógica de alto nivel del diseño lógico, y nos da como resultado, una secuencia de bits(Bitstream) con la que podemos configurar apropiadamente el circuito integrado FPGA.

Los procesos que seguimos dentro del paquete de desarrollo para compilar un diseño y obtener esta secuencia de bits son los siguientes:

1) Introducimos una descripción del circuito lógico usando un lenguaje HDL (Hardware Description Language) como lo son VHDL o Verilog. O simplemente dibujamos el circuito usando un editor esquemático. Puede ser también una combinación de ambos.

2) Un sintetizador lógico transforma el código HDL en una lista de conexiones(netlist). Esta es simplemente una descripción de diferentes compuertas lógicas de nuestro diseño y como están interconectadas.

abs	disconnect	is	out	sli
access	downto	label	package	sra
after	else	library	port	srl
alias	elsif	linkage	postponed	subtype
all	end	literal	procedure	then
and	entity	loop	process	to
architecture	exit	map	pure	transport
array	file	mod	range	type
assert	for	nand	record	unaffected
attribute	function	new	register	units
begin	generate	next	reject	until
block	generic	nor	return	use
body	group	not	rol	variable
buffer	guarded	null	ror	wait
bus	if	of	select	when
case	impure	on	severity	while
component	in	open	signal	with
configuration	inertial	or	shared	xnor
constant	inout	others	sla	xor

Figura 5.1 Palabras reservadas para programación en VHDL. [Elaboración propia]

3) La fase de implementación emplea tres herramientas diferentes. Un traductor que junta varias listas de conexión, luego ocurre un mapeo que combina las compuertas dentro de estas, en grupos que quepan eficientemente en los bloques lógicos del FPGA. Por último, la herramienta de ubicación y ruteo asigna estos grupos en varias locaciones y determina como deben ser conectados en la matriz de interconexión.

4) Luego un generador toma el resultado de la fase de implementación y con algunos parámetros de configuración nos da la secuencia de bits(bitstream) final, que contiene las tablas de verdad de los bloques lógicos y los parámetros para la matriz de interconexión.

Xilinx distribuye y vende sus herramientas ISE FPGA, y también una versión gratuita llamada que denominan WebPACK. Esta versión no genera secuencias de bis para FPGA realmente grandes, y carece de algunas herramientas de diseño con funciones especiales.

CAPÍTULO VI

DESARROLLO DEL SISTEMA

6.1 DIAGRAMA ESQUEMÁTICO

El sistema consta de tres partes bien definidas, que interactúan entre sí; una para realizar el cálculo de la función de frontera BOF(Boundary Object Function), otra para la visualización de resultados y otra para recoger alguno de los objetos observados. Cada una puede funcionar independientemente y contienen sus propios elementos. (Figura 6.1)

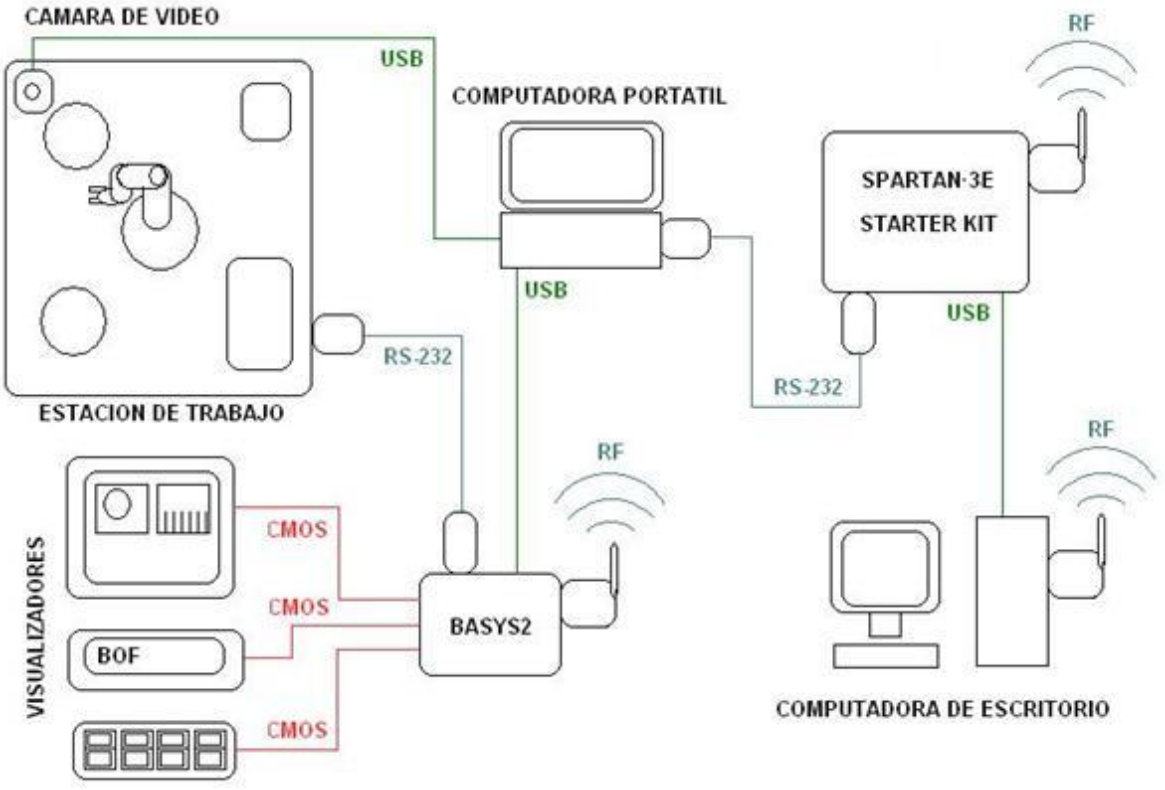

Figura 6.1 Diagrama del sistema desarrollado.
Se muestran los elementos que componen la parte física de la implementación.
[Elaboración:propia]

La etapa que adquiere la imagen y la procesa, está constituida por la cámara de video (Imagen 6.1), la computadora personal y la tarjeta de desarrollo Spartan3-E Starter Kit. En la computadora se tiene un programa en MatLab versión 7.12(Ver Apéndice F), que permite el tomar una

fotografía, procesarla y enviar una imagen binarizada vía RS-232, de manera alámbrica a la Starter Kit. El FPGA de la tarjeta ejecuta los algoritmos internos para calcular: el contorno, el centroide y la función de frontera BOF.

Imagen 6.1 Cámara de video.
Modelo: WEB CAMERA FI#2.0 f:4.8mm
[Fotografía tomada]

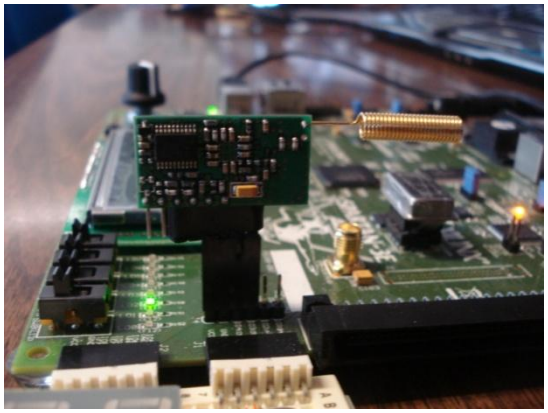

Imagen 6.2 Módulo RF en la Starter Kit.
Modelo: KYL-500S 433MHz 600m TTL
[Fotografía tomada]

Entonces transmite estos valores, a través de un módulo de radio frecuencia RF(Radio Frequency) al exterior (Imagen 6.2). Esta señal puede ser captada por cualquier dispositivo que tenga un receptor que trabaje en la misma frecuencia. En nuestro caso la tarjeta BASYS2 y la computadora de escritorio.

La etapa de visualización es controlada por la tarjeta de desarrollo BASYS2, que al recibir los valores del transmisor de RF, hace uso de tres elementos para que podamos ver los resultados. En una pantalla de cristal líquido, en un visualizador de cristal líquido de dos líneas por dieciséis caracteres y en un visualizador de siete segmentos de cuatro dígitos. Además, esta tarjeta puede enviar comandos a la estación de trabajo para que el brazo robótico vaya a recoger el objeto observado en ese momento. Todas estas tareas funcionan de manera paralela e independiente, pudiendo desconectar alguna en cualquier momento, sin afectar al resto.

La tercera etapa es la estación de trabajo, compuesta por un brazo robótico, dos bases giratorias, su fuente de alimentación y su circuito de control. Es explicada con detalle en el Anexo 2.

6.2 ELEMENTOS EMPLEADOS

A continuación se describen los más importantes.

6.2.1 TARJETA DE DESARROLLO STARTER KIT

Es una tarjeta de Xilinx con un FPGA XC3S500E de la familia Spartan-3E de 16cm. por 18cm. (Imagen 6.3); con hasta 232 puertos de entrada y salida para usuario y unas 10000(diez mil) celdas lógicas, aproximadamente 500000 (quinientas mil) compuertas. Con un reloj oscilador de 50 MHz de frecuencia. Se configura vía USB(Universal Serial Bus). (Ver Apéndice B)

Imagen 6.3 Tarjeta de desarrollo Spartan-3E Starter Kit.
[Fotografía tomada]

6.2.2 TARJETA DE DESARROLLO BASYS2

También de Xilinx, tiene un FPGA XC3S100E,de 7.5cm por 11 cm. (Imagen 6.4); con aproximadamente100000 (cien mil) compuertas. Con tres frecuencias de reloj seleccionables por el usuario, 100MHz, 50MHz y 25MHz. Se configura vía USB(Universal Serial Bus) (Ver Apéndice B). Tiene protección contra corto circuito y descarga electrostática ESD(ElectroStatic Discharge).

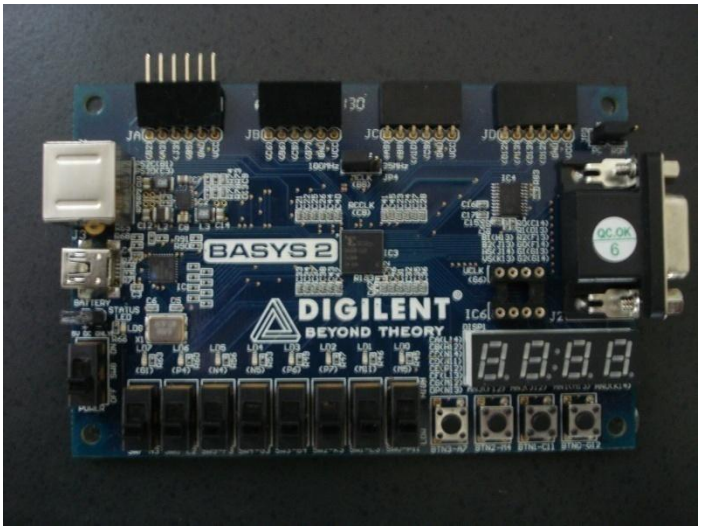

Imagen 6.4 Tarjeta de desarrollo BASYS2.
[Fotografía tomada]

6.2.3 ESTACION DE TRABAJO CON BRAZO ROBÓTICO

Es explicada con detalle en el Anexo 2. Sus elementos son: un brazo robótico articulado de cinco grados de libertad, dos bases giratorias, una fuente de alimentación, dos fuentes de luz, y un circuito de control con comunicación serial RS-232. (Ver Anexo 2)

6.2.4 COMPUTADORA PORTÁTIL CON MATLAB

Tiene instalado MatLab 7.12 donde se hizo el programa para la captura de imágenes y comunicación con la Starter Kit. El paquete ISE 8.2i para la codificación en VHDL, síntesis, mapeo y ruteo de la BASYS2; y la aplicación Adept de Digilent para su configuración. También el lenguaje de programación PIC C 4.120, en este se codifico el integrado PIC18F4550 del circuito controlador de la estación de trabajo. El sistema operativo es Windows Vista Ultimate. (Imagen 6.5)

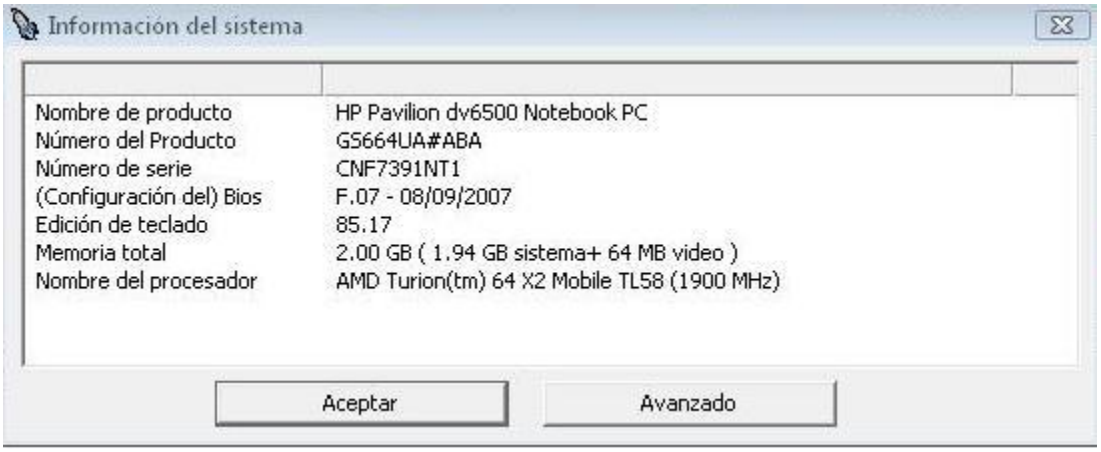

Imagen 6.5 Características de la computadora portátil.
[Captura de pantalla]

6.2.5 COMPUTADORA DE ESCRITORIO CON ISE 10.1

Esta computadora es la encargada de la programación de la tarjeta Starter Kit. El paquete instalado es el ISE versión 10.1, dentro del que se realizan; la codificación en lenguaje VHDL, los procesos de síntesis, mapeo y ruteo de nuestro diseño. El sistema operativo es Windows XP versión 2002. Como algo extra, puede recibir la señal RF con los resultados que envía el FPGA donde se calculó el BOF. (Imagen 6.6)

6.3 MÓDULOS PRINCIPALES DESARROLLADOS

El objetivo principal de este trabajo es poder realizar el cálculo de la función de frontera en un FPGA, por lo que se desarrollaron varios módulos principales, que contribuyen de manera coherente a hacerlo.

Imagen 6.6 Características de la computadora de escritorio.
[Captura de pantalla]

6.3.1 MÓDULO CONTORNO

Dentro de este módulo se procede a identificar los bordes de la figura, dentro de una imagen binaria en 2D con dimensiones conocidas, de determinado número de filas por determinado número de columnas. Recorremos la imagen tanto por filas como por columnas, dentro de los límites de la misma. Es decir, partiendo de la primera fila, recorremos la imagen hasta la última fila, en busca de la figura y, por otro lado, partiendo de la última fila hasta la primera fila en sentido contrario. Así obtenemos un borde superior y otro inferior. De manera análoga se procede con las columnas, para el borde derecho e izquierdo. Los cuatro juntos nos permiten conocer las coordenadas de todo el contorno, almacenadas en cuatro vectores y son de tipo entero.

6.3.2 MÓDULO CENTROIDE

En esta parte se realiza la evaluación del promedio entre valores para conseguir, de manera separada, la coordenada entera Cx, entre los vectores del borde izquierdo y derecho; y la

coordenada entera Cy, entre los vectores del borde superior e inferior; coordenadas del centroide.

También es posible hacer una suma total y dividir entre el número de sumandos, dependiendo de como se codifique el algoritmo. Este módulo está muy relacionado con el anterior, por lo que pueden estar dentro de un solo bloque de código.

6.3.3 MÓDULO BOF

Este necesita como entradas para proceso: el contorno y el centroide; que serían cuatro vectores con los valores de los bordes superior, inferior, derecho e izquierdo y las dos coordenadas Cx y Cy.

Lo que hace es que, desde el centroide, recorre la figura hasta llegar a su contorno, y almacena las coordenadas de ese punto, repite la tarea dieciséis veces. Siguiendo trayectorias definidas como sigue: tomando como vértice al centroide, dividimos la figura en dieciséis ángulos iguales, partiendo de un eje vertical y en sentido de las manecillas del reloj, cada lado de los ángulos que se forman es una de las dieciseis trayectorias. Al final el vector bof_x contiene las coordenadas sobre el eje x y el vector bof y las coordenadas sobre el eje y.

En seguida se calculan las distancias de esos puntos hasta el centroide, estas magnitudes en valor entero, se almacenan dentro de un vector, que representa al BOF.

6.4 MÓDULOS SECUNDARIOS DESARROLLADOS

Son aquellos que realizan tareas complementarias dentro del sistema, los describimos a continuación. Con excepción del que separa dígitos de un número entero con rango de 0 hasta 9999, los módulos contadores, divisores de reloj y otros, que por su simplicidad se explican solos al consultar el código. (Ver Apéndice E)

6.4.1 MÓDULO VISUALIZADOR SIETE SEGMENTOS

Se encarga de controlar un visualizador siete segmentos de cuatro dígitos, donde se despliegan las unidades, decenas, centenas, y unidades de mil (están desactivadas), de un número entero. Para seleccionar que valor será mostrado recurrimos a los interruptores del FPGA. Pudiendo ver Cx, Cy, 16 valores de BOF y 32 valores de las coordenadas bof_x y bof_y. (La tarjeta BASYS2 tiene este elemento, pero la Starter Kit no lo incluye) (Imagen 6.7)

6.4.2 MÓDULO VISUALIZADOR DE CRISTAL LÍQUIDO

Usamos un visualizador de cristal líquido de dos líneas por dieciséis caracteres, en este caso es posible ver los valores de Cx, Cy, BOF(pos), bof_x(pos), bof_y(pos) al mismo tiempo, donde pos significa la posición dentro de cada vector, esta es seleccionable con los interruptores del FPGA. Es una manera más amigable para recopilar los datos, al momento de realizar las pruebas experimentales. (La tarjeta BASYS2 no tiene este elemento, la Spartan-3E Starter Kit si lo incluye). (Imagen 6.8)

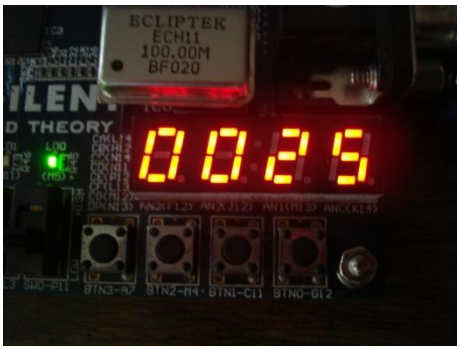

Imagen 6.7 Visualizador siete segmentos.
[Fotografía tomada]

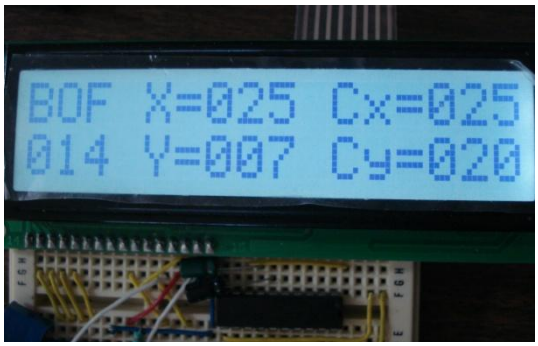

Imagen 6.8 Visualizador de cristal líquido
[Fotografía tomada]

6.4.3 MÓDULO PANTALLA DE CRISTAL LÍQUIDO

Tenemos conectado un monitor ViewSonic a nuestra tarjeta BASYS2, y enviamos las señales de sincronismo vertical, sincronismo horizontal y tres de color; rojo, verde y azul. Muestra de manera gráfica los valores Cx, Cy, BOF, bof_x y bof_y, en cantidad de pixeles, gracias a las coordenadas calculadas, que podemos relacionar con la pantalla a partir de su posición inicial de imagen (1,1), que se ubica en la parte superior izquierda. En el monitor no se visualizan números ni palabras. (Ambas tarjetas poseen puerto de conexión para conectar un monitor). (Imagen 6.9)

Imagen 6.9 Pantalla de cristal líquido. ViewSonic VE7105de 17 pulgadas.
[Fotografía tomada]

6.4.4 MÓDULO COMUNICACIÓN SERIAL

Sirve para comunicaciones con las tarjetas, con las computadoras y con la estación de trabajo. Entre la computadora portátil y la tarjeta Spartan-3E Starter Kit usamos un receptor-transmisor

asíncrono UART(Universal Asynchronous Receiver-Transmitter) para RS-232 alámbrico (Imagen 6.10); a 19200Baudios, 1bit de inicio, 8bits de datos, 1bit de parada y con paridad impar. Con las mismas características realizamos la comunicación entre ambas tarjetas, pero de manera inalámbrica (Imagen 6.11); se adquirieron varios módulos de radio frecuencia para cada plataforma.

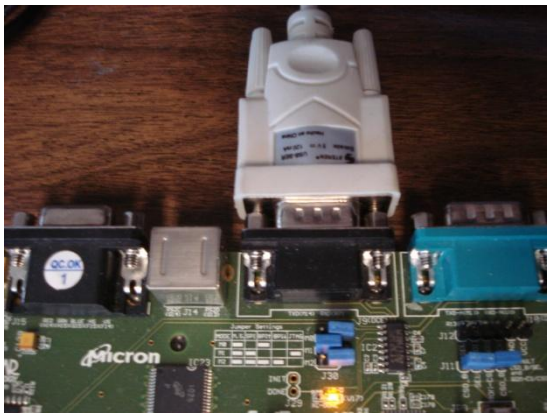

Imagen 6.10 Cable RS-232 a USB.
Comunicación Starter Kit con computador portátil.
[Fotografía tomada]

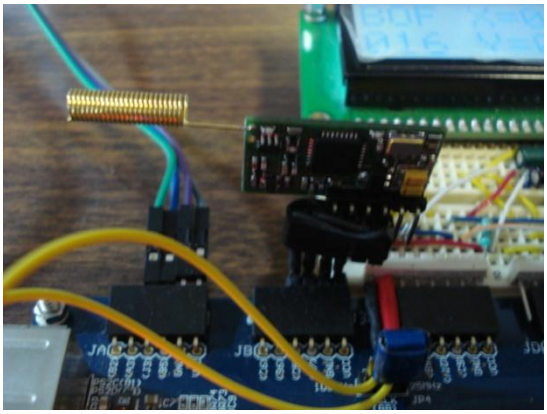

Imagen 6.11 Módulo RF en la BASYS2.
Modelo: KYL-500S 433MHz 600m TTL.
[Fotografía tomada]

6.4.5 MÓDULO BRAZO ROBÓTICO

Encargado de enviar comandos a la estación de trabajo, para mover el brazo robótico o las bases giratorias, con la finalidad de recoger el objeto observado por la cámara. Su comunicación es otra UART pero a 9600Baudios, 1 bit de inicio, 8 bits de datos, 1bit de parada y sin paridad, con conexión alámbrica desde la tarjeta BASYS2. (Imagen 6.12)

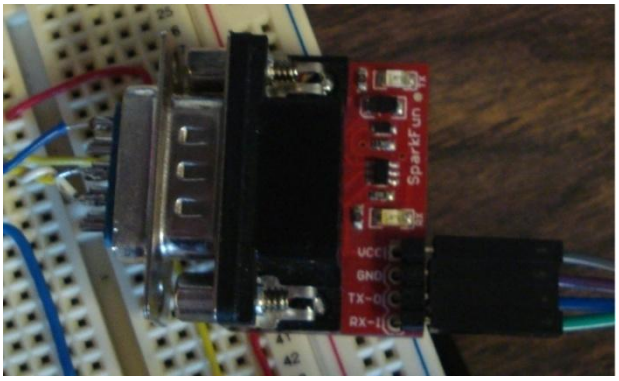

Imagen 6.12 Convertidor de voltaje TTL a RS-232.
Comunicación BASYS2 con la estación de trabajo.
[Fotografía tomada]

6.5 SISTEMA COMPLETO

El sistema ya implementado se pone en funcionamiento. Conectamos la fuente de poder de la estación de trabajo a la red eléctrica, encendemos las dos tarjetas de desarrollo, y en la computadora portátil ponemos en funcionamiento el programa creado para la captura de imágenes hecho en MatLab.

Tomamos una fotografía, binarizamos la imagen, se la envía a la Starter Kit, que la procesa y calcula los valores Cx, Cy, BOF, bof_x y bof_y. Los transmite a la BASYS2, más un valor uno o cero, para indicarle si se trata de un cuadrilátero o una figura de forma circular. Una vez que llegan todos los valores, son visualizados; paralelamente el brazo recoge el objeto de la estación de trabajo y lo lleva de la base giratoria 1 a la base giratoria 2, siempre que el último dato recibido sea un uno, es decir, recoge objetos cúbicos. Para el siguiente objeto, enviamos el comando para que gire la base 1 donde se encuentra el siguientes objeto, y se repite el proceso. Aclaramos en este momento que en ambas tarjetas disponemos de un botón de re inicialización, por lo que debemos presionarlos para la siguiente toma.

Todos y cada uno de los módulos y partes del sistema funcionan independientemente, si desconectamos alguno de ellos, los restantes siguen activos. Lo presentamos en las imágenes 6.13 y 6.14.

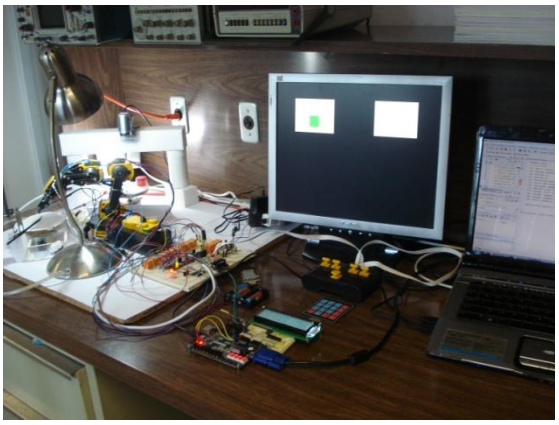

Imagen 6.13 BASYS2 y Estación de trabajo.
[Fotografía tomada]

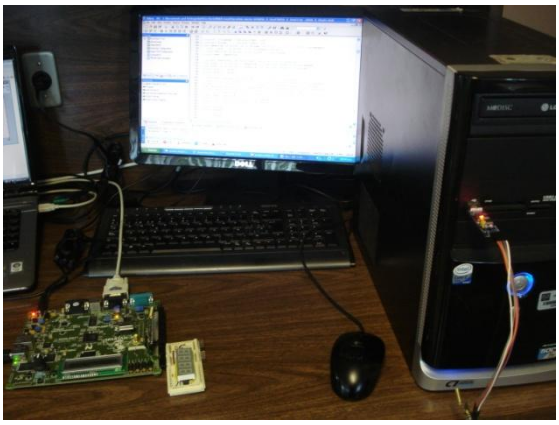

Imagen 6.14 Spartan-3E Starter Kit.
[Fotografía tomada]

Nota: La tarjeta de desarrollo Spartan-3E Starter Kit, es la encargada de obtener la función de frontera.

CAPÍTULO VII

RESULTADOS EXPERIMENTALES

Se realizaron varias pruebas; con imágenes binarias creadas en MatLab a partir de matrices de unos y ceros; otras cargadas desde un archivo de imagen comprimida JPEG(Joint Photographic Expert Group) y de las capturas de imagen hechas por una cámara de video.

Los datos recopilados son los valores de: Cx y Cy que son las coordenadas del centroide, los vectores BOF, bof_x y bof_y, de dimensión 16, por lo que cada uno almacena dieciséis valores, donde los dos últimos representan las coordenadas de los puntos del contorno y BOF contiene las distancias de estos al centroide. Todos dentro de una región de interés de 40x40 pixeles de la imagen 2D. Los programas codificados nos dan estos valores en variables de tipo entero positivo. Por ejemplo si el proceso nos dice que Cx=7, esto indica que tiene un valor equivalente de 7 pixeles, esto se cumple para todos los demás. También se anotaron los tiempos de procesamiento, tanto en MatLab como en el FPGA. Ambas plataformas con sus propios resultados.

En los experimentos, se muestran: la imagen original; la imagen binarizada y dentro de la región de interés, en una gráfica de MatLab; y los del FPGA en un monitor para compararlos; También una tabla con los datos calculados por ambos, con los tiempos de ejecución en milisegundos.

7.1 PRUEBAS EXPERIMENTALES CON IMÁGENES CREADAS EN MATLAB

7.1.1 PRUEBAS CON UN CÍRCULO

```
imagenMEM =[
%                   1
%1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7       6 7 8 9 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0       0 0 0 0 0;%1
 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0       0 0 0 0 0;%2
 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0       0 0 0 0 0;%3
 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0       0 0 0 0 0;%4
 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0   --  0 0 0 0 0;%5
 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0       0 0 0 0 0;%6
 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0       0 0 0 0 0;%7
 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0       0 0 0 0 0;%8
 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0       0 0 0 0 0;%9
 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0   --  0 0 0 0 0;%10
 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0       0 0 0 0 0;%11
 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0       0 0 0 0 0;%12
 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0       0 0 0 0 0;%13
 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0       0 0 0 0 0;%14
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0       0 0 0 0 0;%15
```

Figura 7.1 Matriz de un círculo creado en MatLab. [Captura de pantalla]

Tabla 7.1 Prueba experimental 1. [Elaboración propia]

DATO	VALORES OBTENIDOS															
pos =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
BOF(pos) [pixeles]	7	8	7	8	7	8	7	8	7	8	7	8	7	8	7	8
bof_x(pos)[pixeles]	8	11	12	14	14	14	12	11	8	5	4	2	2	2	4	5
bof_Y(pos)[pixeles]	2	2	4	5	8	11	12	14	14	14	12	11	8	5	4	2
Cx [pixeles]	8			Tiempo de ejecución en MatLab [ms]									86.4			
Cy [pixeles]	8			Tiempo de ejecución en el FPGA [ms]									0.13272			

Resultados similares tanto en MatLab como en el FPGA. (Excepto el Tiempo de ejecución)

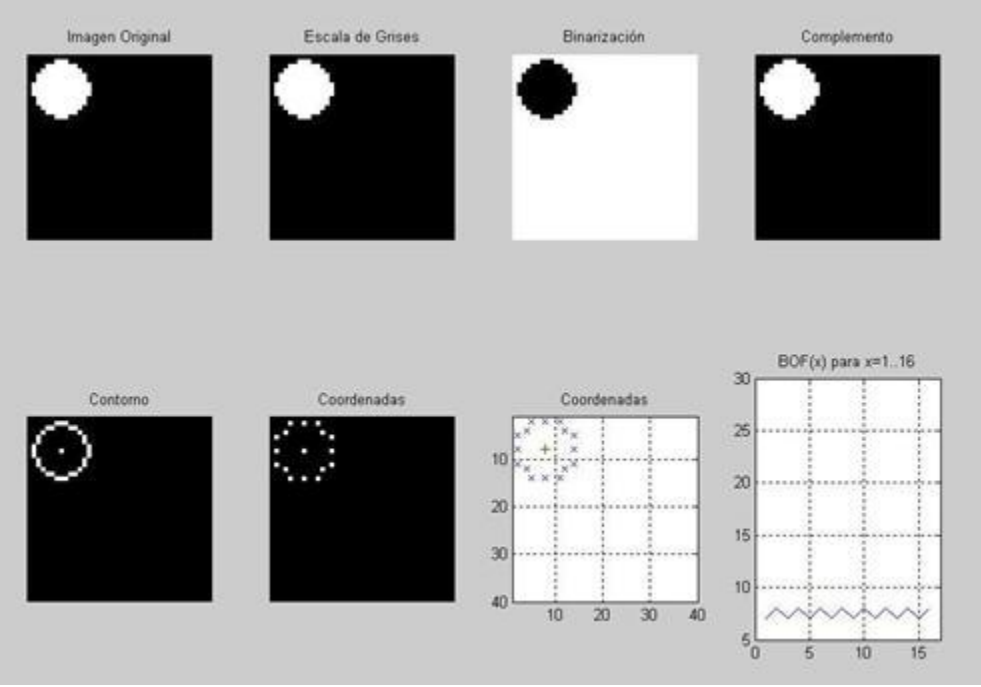

Imagen 7.1.a Representación gráfica de resultados en MatLab. [Captura de pantalla]

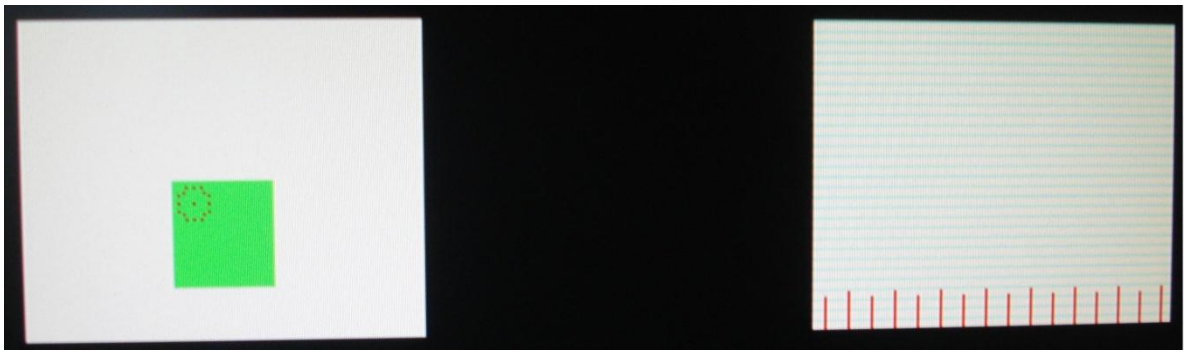

Imagen 7.1.b Representación gráfica de resultados del FPGA en Monitor. [Fotografía tomada]

7.1.2 PRUEBAS CON UN CUADRADO

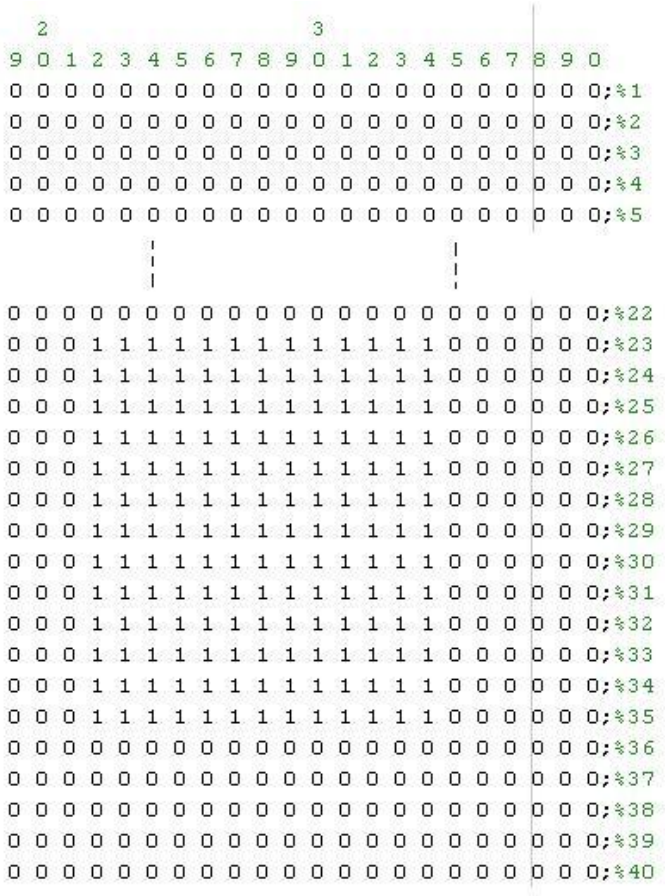

Figura 7.2 Matriz de un cuadrado creado en MatLab. [Captura de pantalla]

Tabla 7.2 Prueba experimental 2. [Elaboración propia]

DATO	VALORES OBTENIDOS															
pos =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
BOF(pos) [pixeles]	7	8	10	8	7	8	10	8	7	8	10	8	7	8	10	8
bof_x(pos)[pixeles]	28	31	34	34	34	34	34	31	28	25	22	22	22	22	22	25
bof_Y(pos)[pixeles]	23	23	23	26	29	32	35	35	35	35	35	32	29	26	23	23
Cx [pixeles]			28	Tiempo de ejecución en MatLab [ms]									91.8			
Cy [pixeles]			29	Tiempo de ejecución en el FPGA [ms]									0.13402			

Resultados similares tanto en MatLab como en el FPGA. (Excepto el Tiempo de ejecución)

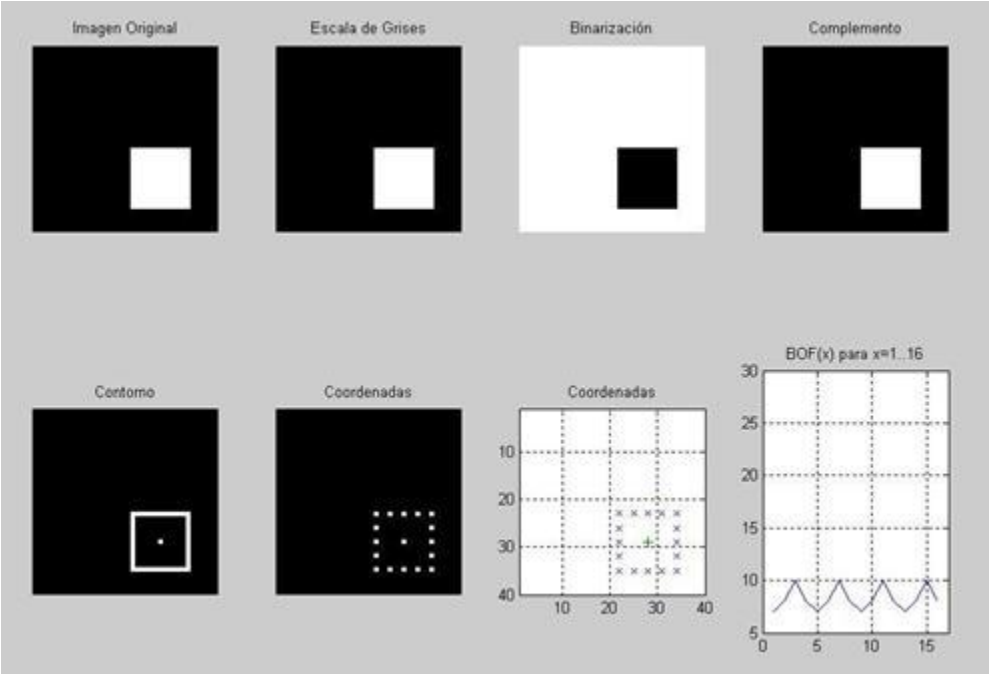

Imagen 7.2.a Representación gráfica de resultados en MatLab. [Captura de pantalla]

Imagen 7.2.b Representación gráfica de resultados del FPGA en Monitor. [Fotografía tomada]

7.2 PRUEBAS EXPERIMENTALES CON IMÁGENES EN FORMATO JPEG

7.2.1 PRUEBAS CON CUADRADO GRANDE AL CENTRO

Tabla 7.3 Prueba experimental 3. [Elaboración propia]

DATO	VALORES OBTENIDOS															
pos =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
BOF(pos) [pixeles]	16	19	23	19	16	19	23	19	16	19	23	19	16	19	23	19
bof_x(pos)[pixeles]	20	28	35	36	35	36	35	28	20	12	5	4	5	4	5	12
bof_Y(pos)[pixeles]	5	4	5	12	20	28	35	36	35	36	35	28	20	12	5	4
Cx [pixeles]			20	Tiempo de ejecución en MatLab [ms]									8.3			
Cy [pixeles]			20	Tiempo de ejecución en el FPGA [ms]									0.07016			

Resultados similares tanto en MatLab como en el FPGA. (Excepto el Tiempo de ejecución)

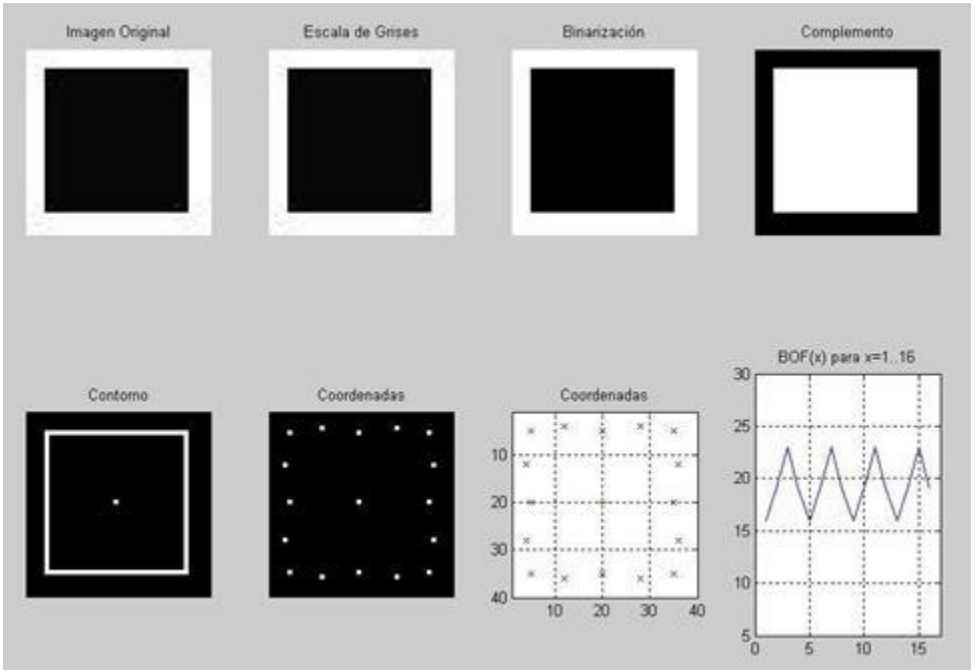

Imagen 7.3.a Representación gráfica de resultados en MatLab. [Captura de pantalla]

Imagen 7.3.b Representación gráfica de resultados del FPGA en Monitor. [Fotografía tomada]

7.2.2 PRUEBAS CON CUADRADO CHICO AL CENTRO

Tabla 7.4 Prueba experimental 4. [Elaboración propia]

DATO	VALORES OBTENIDOS															
pos =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
BOF(pos) [pixeles]	7	8	10	8	7	8	10	8	7	8	10	8	7	8	10	8
bof_x(pos)[pixeles]	20	23	26	26	26	26	26	23	20	17	14	14	14	14	14	17
bof_Y(pos)[pixeles]	14	14	14	17	20	23	26	26	26	26	26	23	20	17	14	14
Cx [pixeles]			20	Tiempo de ejecución en MatLab [ms]									8.4			
Cy [pixeles]			20	Tiempo de ejecución en el FPGA [ms]									0.13368			

Resultados similares tanto en MatLab como en el FPGA. (Excepto el Tiempo de ejecución)

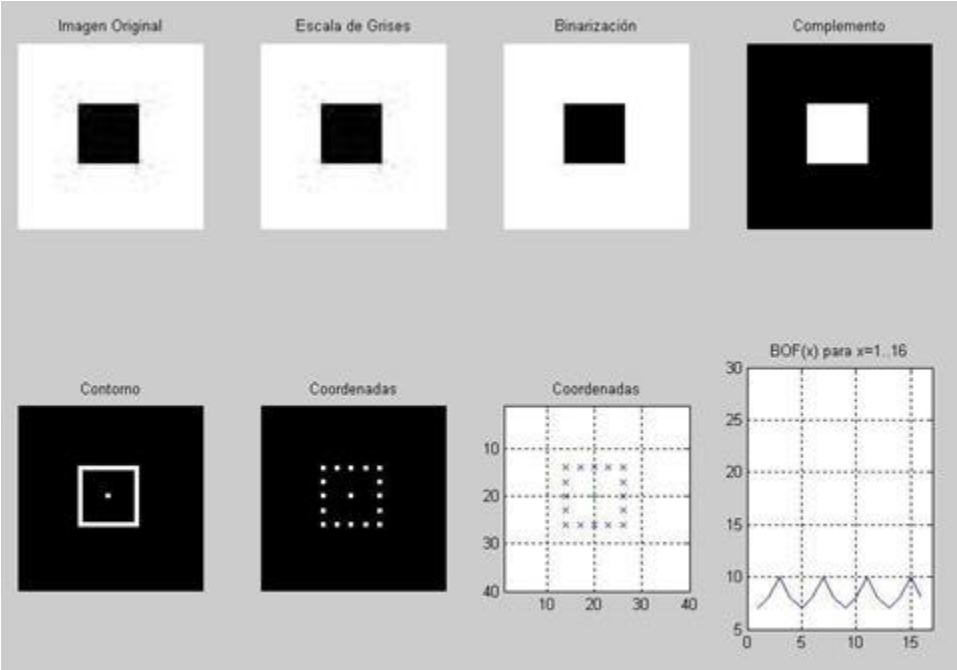

Imagen 7.4.a Representación gráfica de resultados en MatLab. [Captura de pantalla]

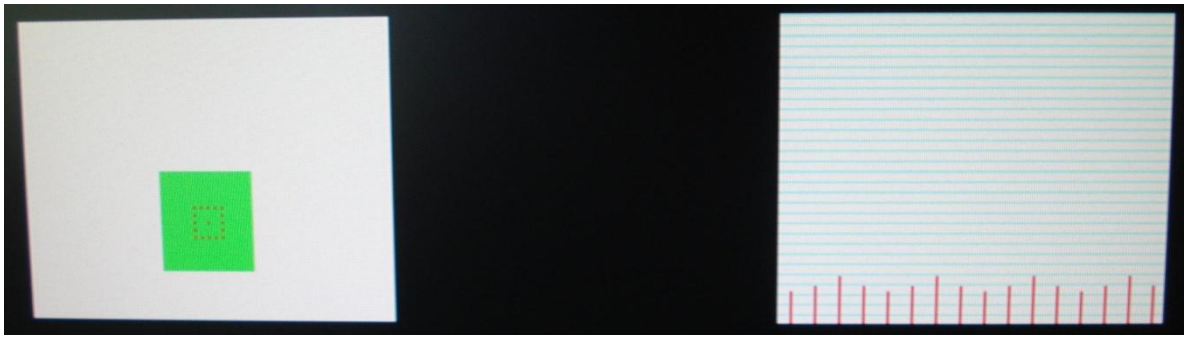

Imagen 7.4.b Representación gráfica de resultados del FPGA en Monitor. [Fotografía tomada]

7.2.3 PRUEBAS CON CUADRADO CHICO ABAJO Y A LA IZQUIERDA

Tabla 7.5 Prueba experimental 5. [Elaboración propia]

DATO	VALORES OBTENIDOS															
pos =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
BOF(pos) [pixeles]	7	8	10	8	7	8	10	8	7	8	10	8	7	8	10	8
bof_x(pos)[pixeles]	10	13	16	16	16	16	16	13	10	7	4	4	4	4	4	7
bof_Y(pos)[pixeles]	25	25	25	28	31	34	37	37	37	37	37	34	31	28	25	25
Cx [pixeles]			10	Tiempo de ejecución en MatLab [ms]									8.1			
Cy [pixeles]			31	Tiempo de ejecución en el FPGA [ms]									0.1337			

Resultados similares tanto en MatLab como en el FPGA. (Excepto el Tiempo de ejecución)

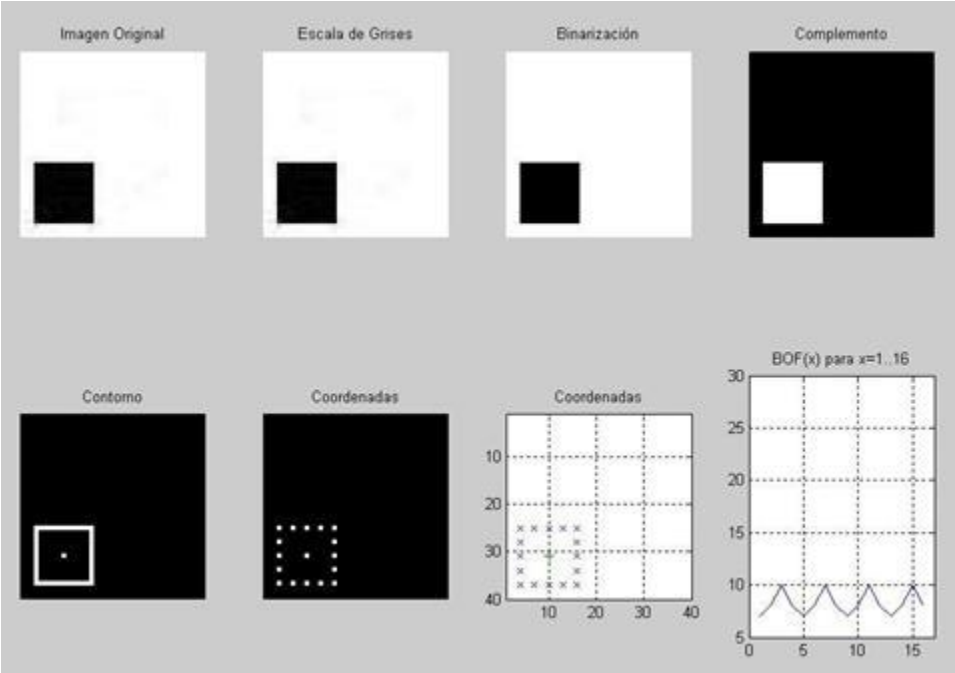

Imagen 7.5.a Representación gráfica de resultados en MatLab. [Captura de pantalla]

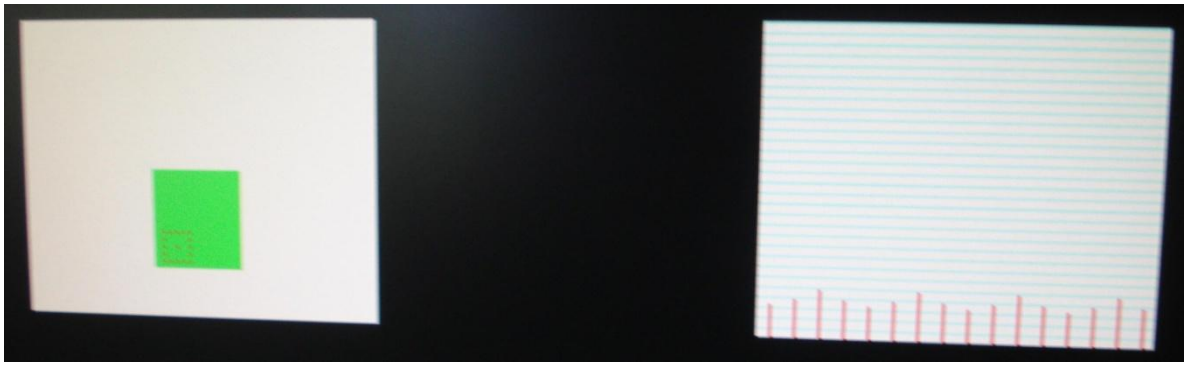

Imagen 7.5.b Representación gráfica de resultados del FPGA en Monitor. [Fotografía tomada]

7.2.4 PRUEBAS CON CÍRCULO GRANDE AL CENTRO

Tabla 7.6 Prueba experimental 6. [Elaboración propia]

DATO	VALORES OBTENIDOS															
pos =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
BOF(pos) [pixeles]	16	17	16	17	16	17	16	17	16	17	16	17	16	17	16	17
bof_x(pos)[pixeles]	20	27	30	34	35	34	30	27	20	13	10	6	5	6	10	13
bof_Y(pos)[pixeles]	5	6	10	13	20	27	30	34	35	34	30	27	20	13	10	6
Cx [pixeles]			20	Tiempo de ejecución en MatLab [ms]									8.4			
Cy [pixeles]			20	Tiempo de ejecución en el FPGA [ms]									0.13784			

Resultados similares tanto en MatLab como en el FPGA. (Excepto el Tiempo de ejecución)

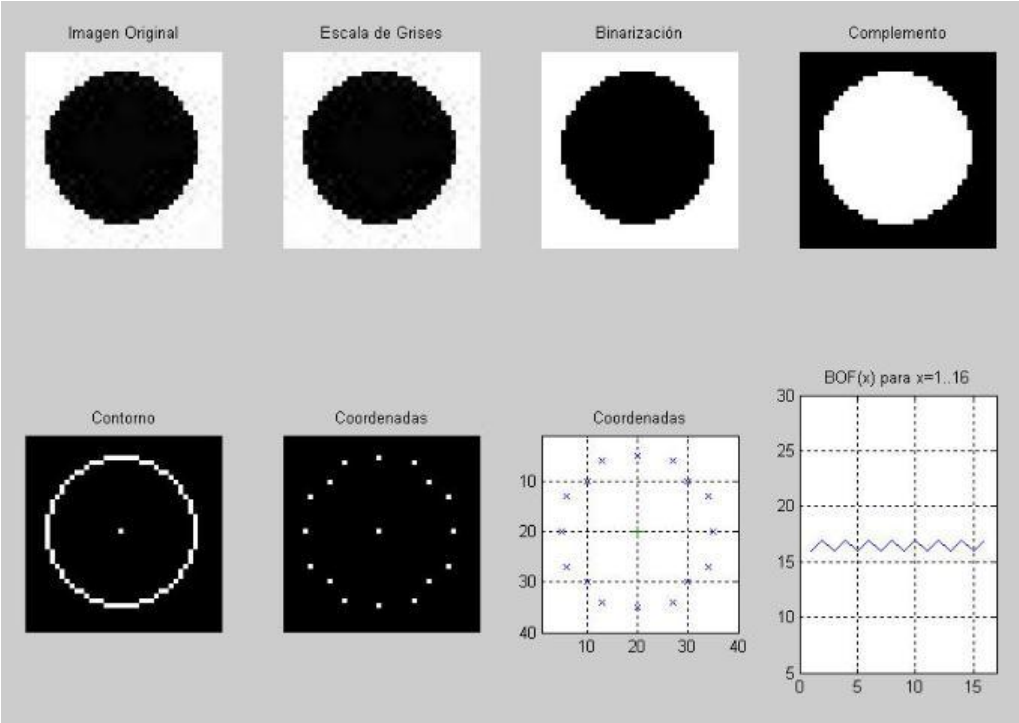

Imagen 7.6.a Representación gráfica de resultados en MatLab. [Captura de pantalla]

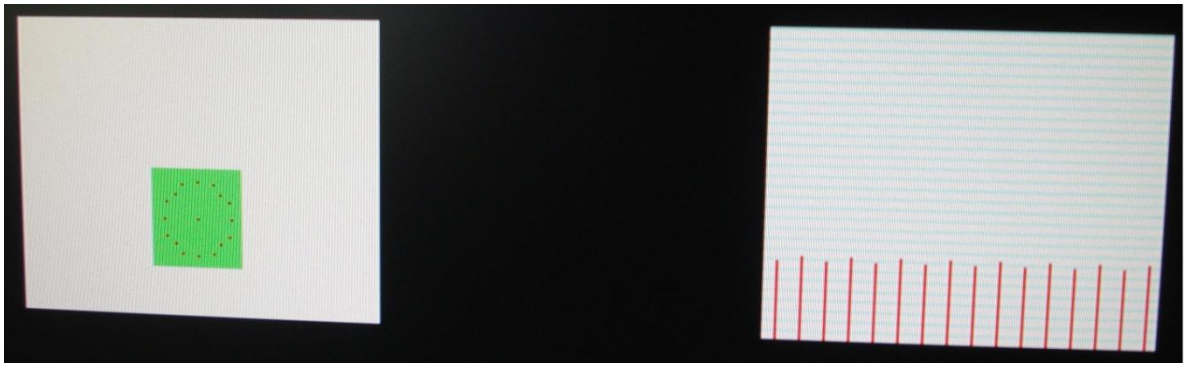

Imagen 7.6.b Representación gráfica de resultados del FPGA en Monitor. [Fotografía tomada]

7.2.5 PRUEBAS CONCÍRULO GRANDE ABAJO Y A LA DERECHA

Tabla 7.7 Prueba experimental 7. [Elaboración propia]

DATO	VALORES OBTENIDOS															
pos =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
BOF(pos) [pixeles]	16	17	16	17	16	17	16	17	16	17	16	17	16	17	16	17
bof_x(pos)[pixeles]	23	30	33	37	38	37	33	30	23	16	13	9	8	9	13	16
bof_Y(pos)[pixeles]	9	10	14	17	24	31	34	38	39	38	34	31	24	17	14	10
Cx [pixeles]			23	Tiempo de ejecución en MatLab [ms]									8.3			
Cy [pixeles]			24	Tiempo de ejecución en el FPGA [ms]									0.13798			

Resultados similares tanto en MatLab como en el FPGA. (Excepto el Tiempo de ejecución)

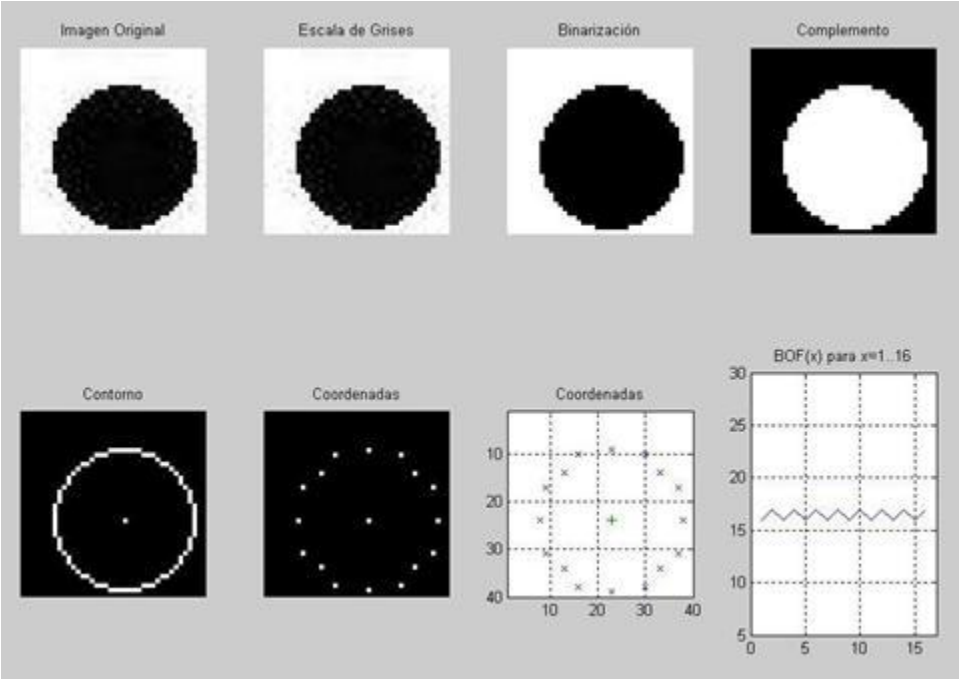

Imagen 7.7.a Representación gráfica de resultados en MatLab. [Captura de pantalla]

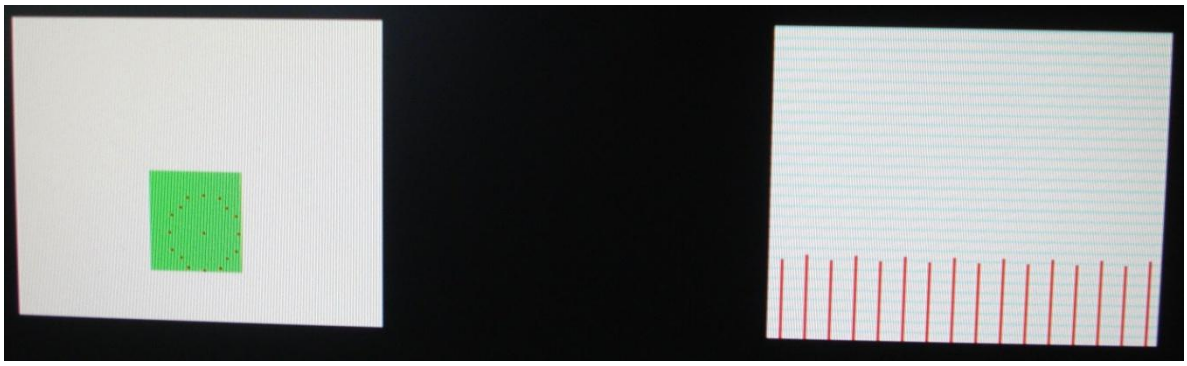

Imagen 7.7.b Representación gráfica de resultados del FPGA en Monitor. [Fotografía tomada]

7.2.6 PRUEBAS CON FIGURA COMPUESTA

Tabla 7.8 Prueba experimental 8. [Elaboración propia]

DATO	VALORES OBTENIDOS															
pos =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
BOF(pos) [pixeles]	17	17	17	17	16	19	21	17	15	17	21	19	16	17	17	17
bof_x(pos)[pixeles]	20	27	31	34	35	36	34	27	20	13	6	4	5	6	9	13
bof_Y(pos)[pixeles]	5	7	10	14	21	29	35	35	35	35	35	29	21	14	10	7
Cx [pixeles]			20	Tiempo de ejecución en MatLab [ms]									8.6			
Cy [pixeles]			21	Tiempo de ejecución en el FPGA [ms]									0.13858			

Resultados similares tanto en MatLab como en el FPGA. (Excepto el Tiempo de ejecución)

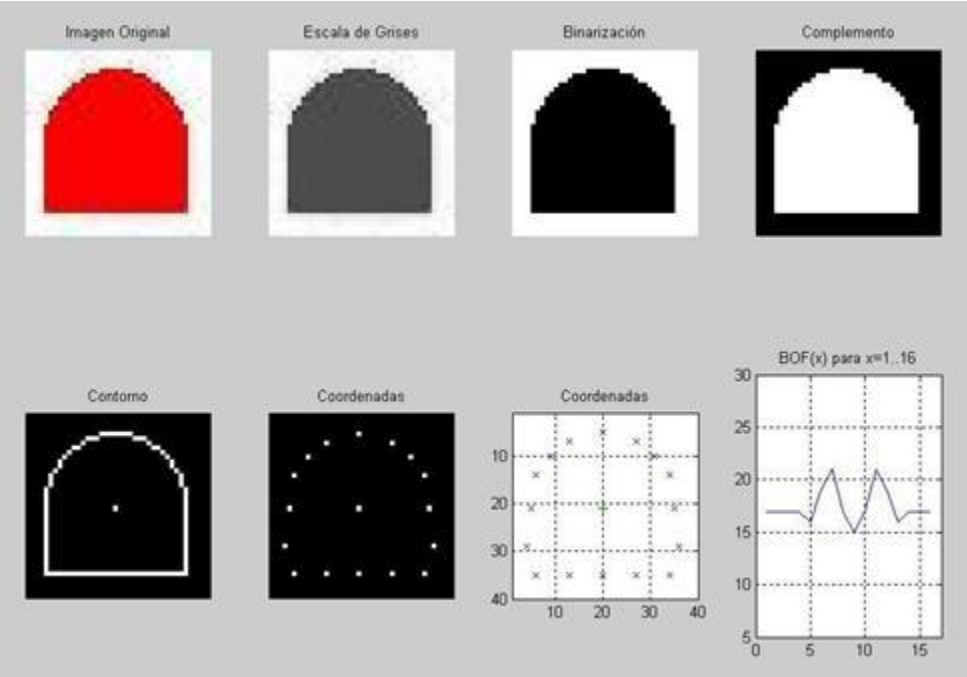

Imagen 7.8.a Representación gráfica de resultados en MatLab. [Captura de pantalla]

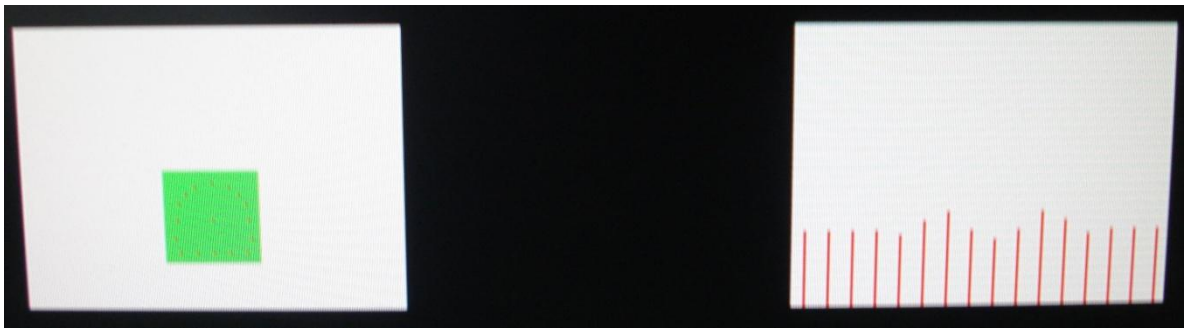

Imagen 7.8.b Representación gráfica de resultados del FPGA en Monitor. [Fotografía tomada]

7.3 PRUEBAS EXPERIMENTALES CON IMÁGENES DESDE CÁMARA DE VIDEO

7.3.1 PRUEBAS CON OBJETO CILÍNDRICO

Tabla 7.9 Prueba experimental 9. [Elaboración propia]

DATO	VALORES OBTENIDOS															
pos =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
BOF(pos) [pixeles]	14	15	16	15	15	17	16	17	15	15	14	15	14	15	14	15
bof_x(pos)[pixeles]	25	31	35	37	39	39	35	32	25	19	16	13	12	13	16	19
bof_Y(pos)[pixeles]	7	8	10	14	20	27	30	34	34	32	29	26	20	14	11	8
Cx [pixeles]			25	Tiempo de ejecución en MatLab [ms]									8.4			
Cy [pixeles]			20	Tiempo de ejecución en el FPGA [ms]									0.13722			

Resultados similares tanto en MatLab como en el FPGA. (Excepto el Tiempo de ejecución)

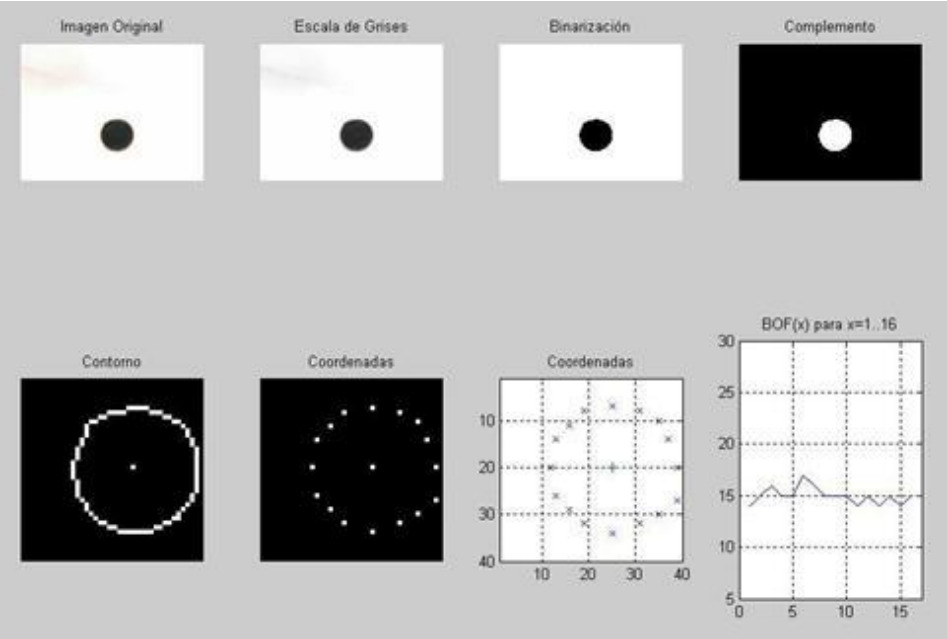

Imagen 7.9.a Representación gráfica de resultados en MatLab. [Captura de pantalla]

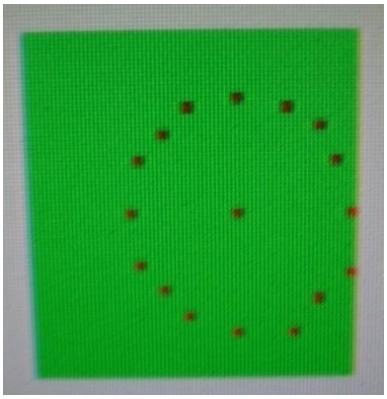

Imagen 7.9.b Resultado gráfico en Monitor con FPGA. [Fotografía tomada]

Imagen 7.9.c Objeto cilíndrico real. [Fotografía tomada]

7.3.2 PRUEBAS CON OBJETO CÚBICO

Tabla 7.10 Prueba experimental 10. [Elaboración propia]

DATO	VALORES OBTENIDOS															
pos =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
BOF(pos) [pixeles]	16	19	20	17	15	17	21	19	16	19	17	15	12	15	17	19
bof_x(pos)[pixeles]	24	32	37	38	38	38	38	32	24	16	13	12	13	12	13	16
bof_Y(pos)[pixeles]	3	2	5	11	18	25	32	34	33	34	29	24	18	12	7	2
Cx [pixeles]			24	Tiempo de ejecución en MatLab [ms]									8.2			
Cy [pixeles]			18	Tiempo de ejecución en el FPGA [ms]									0.06908			

Resultados similares tanto en MatLab como en el FPGA. (Excepto el Tiempo de ejecución)

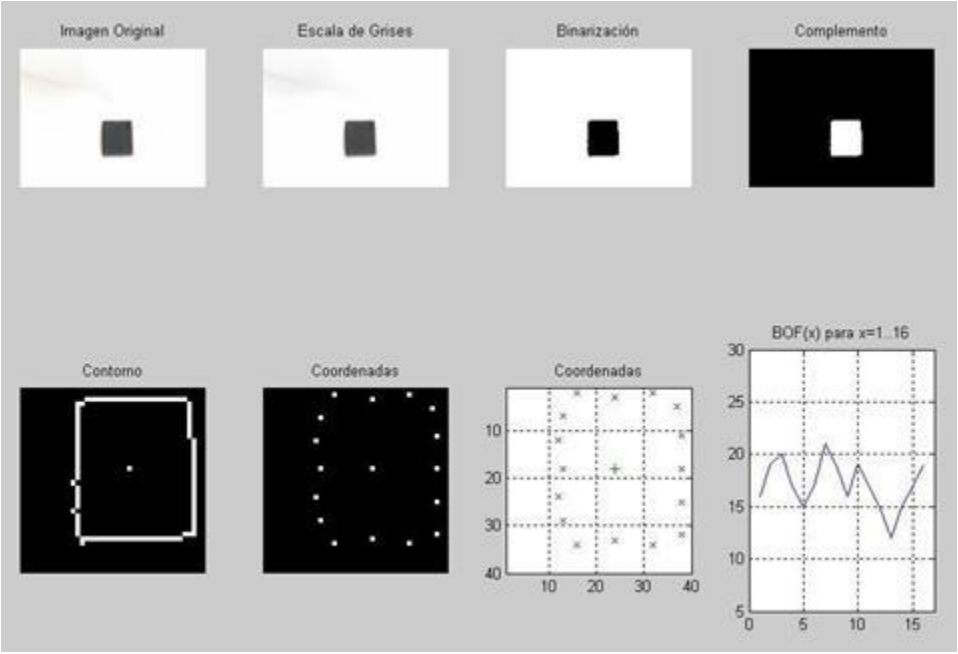

Imagen 7.10.a Representación gráfica de resultados en MatLab. [Captura de pantalla]

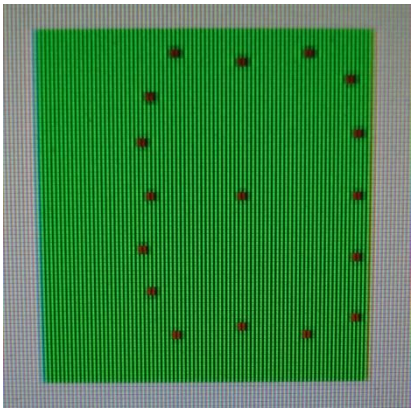

Imagen 7.10.b Resultado gráfico en Monitor con FPGA. [Fotografía tomada]

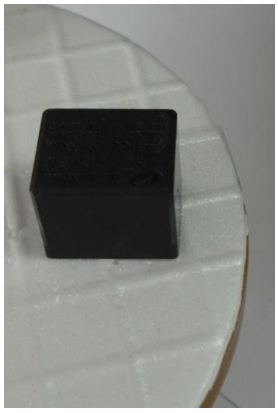

Imagen 7.10.c Objeto cúbiico real. [Fotografía tomada]

7.4 COMPARACIÓN DE TIEMPOS ENTRE MATLAB Y FPGA

Los tiempos de ejecución son medidos desde que se inician hasta que se terminan los procesos para el cálculo del centroide, del contorno, y la función de frontera BOF. (Tabla 7.11)

Para la estimación de tiempos, se usaron instrucciones de inicio y parada de conteo tanto en MatLab como en el FPGA, por medio de contadores dedicados a esta tarea, posteriormente se anotaron los valores obtenidos.

TABLA 7.11 Comparativo de los tiempos de ejecución. [Elaboración propia]

PRUEBA EXPERIMENTAL	**TIEMPO DE EJECUCIÓN [ms]**	
(Número de experimento)	MatLab	FPGA
1	86.4	0.13272
2	91.8	0.13402
3	8.3	0.07016
4	8.4	0.13368
5	8.1	0.1337
6	8.4	0.13784
7	8.3	0.13798
8	8.6	0.13858
9	8.4	0.137226
10	8.2	0.06908
PROMEDIO [ms]	**24.49**	**0.1224986**

Tiempos medidos se muestran expresados en milisegundos.

CAPÍTULO VIII

CONCLUSIONES Y TRABAJO FUTURO

8.1 CONCLUSIONES

Se comprobó que el algoritmo para obtener la función de frontera o BOF está funcionando correctamente. Tanto con figuras regulares como lo son un círculo o un cuadrado, así como, con figuras irregulares. Dependiendo de si la figura tiene número par o impar de filas o columnas, el centroide dará un error de un pixel en algunos valores, al calcular los puntos del contorno y distancias de estos al centroide,

Esto resulta obvio, ya que a menor resolución mayor error y a mayor resolución menor error, y aunque se tenga el mismo error, en los casos con mayor resolución, este no será significativo en comparación a los valores que se esperan obtener. Todo esto refiriéndonos a la cantidad de pixeles por fila y columna de la imagen binarizada y procesada.

A pesar de que no se mencionó la parte de la iluminación en este trabajo, es muy importante tener esa parte controlada, porque los cambios de intensidad luminosa al momento de la captura de imagen con la cámara, afectan al resultado final en lo que se refiere a tener un buen contraste en la representación en 2D de la figura del objeto, y por consiguiente un buen contorno. La empleada para captura de imágenes fue direccional, con dos fuentes de luz, una posterior y otra elevada 30 centímetros y en una dirección a 45 grados hacia el objeto.

Al ser un diseño modular, es factible cambiar cualquier etapa o elemento del mismo, por otro equivalente y que cumpla la misma función. También podemos aumentar, quitar o modificar todos y cada uno de sus elementos. Con esto demostramos la adaptabilidad inherente del sistema.

Las velocidades en los tiempos de ejecución del FPGA en comparación con la computadora de escritorio fueron mucho más rápidas; y se emplearon cronómetros integrados en los códigos para su cálculo.

8.2 TRABAJO FUTURO

Es necesario hacer más experimentos con diferentes resoluciones y con más figuras para poder llegar a una conclusión definitiva en lo que respecta a la aplicación en el área de reconocimiento de objetos y más específicamente, en la obtención de la función de frontera como se propuso en este trabajo.

En lo referente a trabajo futuro, hay muchas cosas que se verán; como hacer algoritmos diferentes que consigan la misma finalidad pero sin usar memoria para imágenes, trabajar con mejores cámaras para adquirir las imágenes, y también depurar el sistema hasta llegar a un punto aceptable para tener algo que se pueda aplicar en una estación de trabajo real.

8.3 COMENTARIO FINAL

Estas plataformas en comparación con otras, nos brindan muchas ventajas en lo que respecta a tiempos de desarrollo, depuración y adaptabilidad, mencionando también mayor velocidad de procesamiento y reconfiguración de los circuitos muy rápida.

La posibilidad de adquirir una tarjeta de desarrollo y empezar, casi en el momento, a configurar el FPGA de la misma, pudiendo ver nuestros diseños funcionando, es algo que motiva a profundizar la investigación y estudio de estos dispositivos.

APÉNDICE A

ALGORITMO PARA LA OBTENCIÓN DEL BOF

A.1 INTRODUCCIÓN

Primero partimos de una imagen en 2D binarizada, es decir que contiene solamente ceros o unos; con resolución igual a la cantidad de filas por la cantidad de columnas que la conforman. Cada uno de estos equivalen a los pixeles(Picture Element) de la misma, donde un cero significa que es parte del fondo y un uno es parte del objeto.

Por tanto definimos la imagen como sigue:

$Imagen(fil, col) = 1$; el pixel es parte del objeto

0 ; el pixel es parte del fondo

Donde : $fil = 1, 2, 3 \ldots\ldots, filas$ ($filas$=cantidad de filas de la imagen)

$col = 1,2,3, \ldots\ldots, columnas$ ($columnas$= cantidad de columnas de la imagen)

Acontinuación definimos los puntos que pertenecen al contorno:

$Contorno(fil, col) = 1$, si $Imagen(fil, col) = 1$ y NO todos sus pixeles vecinos son 1

0, si $imagen\ (fil, col) = 1$ y todos sus pixeles vecinos son 1

Donde los pixeles vecinos de un punto determinado de la imagen se muestran en la Figura A.1.

Un caso particular se da cuando el objeto está en los límites exteriores de la imagen, en tal situación :

$Contorno\ (fil, col) = 1$, si $imagen(fil, col) = 1$ y $(fil = filas\ o\ col = columnas)$, con un o inclusivo

Ya con estas definiciones y sabiendo que el centroide es el centro geométrico, podemos elaborar un algoritmo eficaz para el cálculo del BOF.

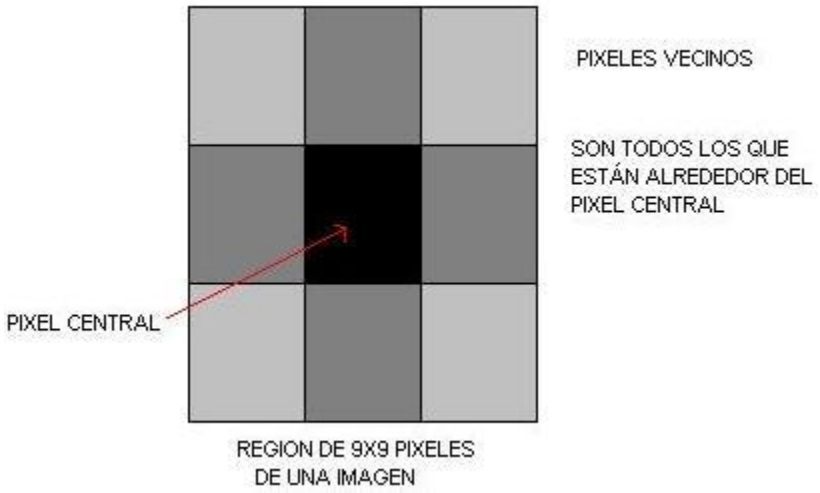

Figura A.1 Pixeles vecinos dentro de una imagen binaria.

A.2 ALGORITMO DESARROLLADO

Algoritmo para obtener el centroide y luego el BOF de una figura en 2D.

1) Se busca el contorno del objeto tanto por filas como por columnas.

2) Se obtienen los puntos medios entre contornos de cada fila.

3) Se obtienen los puntos medios entre contornos de cada columna.

4) Se calcula el promedio de todos los puntos medios.

5) La coordenada Cx del centroide es la media respecto a las columnas.

6) La coordenada Cy del centroide es la media respecto a las filas.

7) Ya tenemos el centroide

8) Ahora, se divide la figura en 16 ángulos iguales tomando el centroide como el vértice y partiendo de un eje vertical con dirección norte, en sentido de las manecillas del reloj.

9) A partir del centroide se recorre la figura hacia afuera por los lados de esos ángulos, hasta encontrar el contorno.

10) Con el paso anterior se obtienen 16 coordenadas que forman parte del contorno.

11) Se calculan 16 distancias de cada una de estas coordenadas hacia el centroide.

12) La función de frontera es el vector conformado por esas 16 distancias.

A.3 EJEMPLO PRÁCTICO

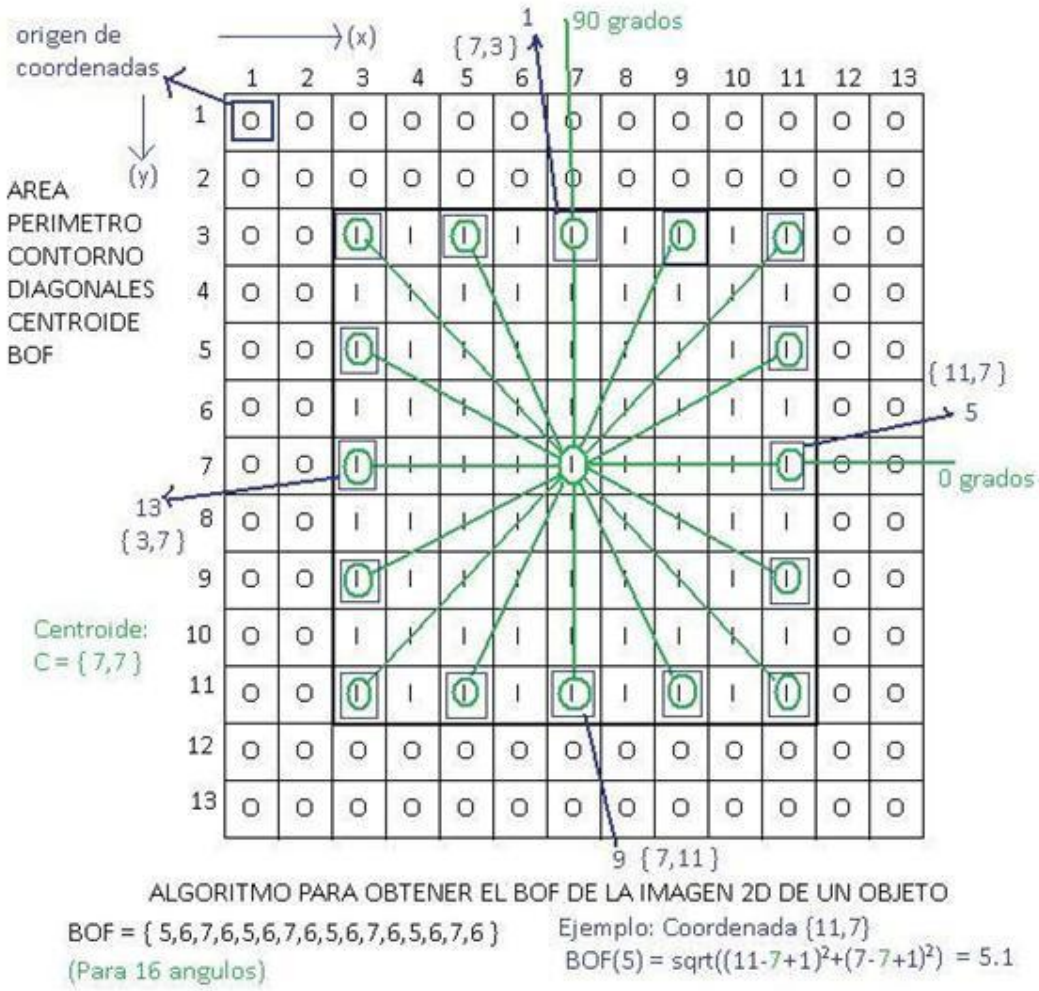

Figura A.2 Ejemplo demostrativo del algoritmo desarrollado.
[Elaboración propia]

Tenemos una imagen binarizada, cuyo origen de coordenadas lo definimos en la posición $(1,1)$, eje "y" positivo hacia abajo y eje "x" positivo hacia la derecha sobre el mismo plano. La imagen contiene una figura cuadrada de 9x9 pixeles y con centroide de coordenadas $Cx = 7\ y\ Cy = 7$. Fue ubicada por nosotros como se muestra en la gráfica, con fines demostrativos.

Con nuestro algoritmo obtenemos el centroide, con valores de $Cx = 7\ y\ Cy = 7$; luego el contorno y después las 16 coordenadas de los pixeles que están encerrados con un círculo verde.(Gráfica x) Ahora para el cálculo del BOF, tomamos cada uno de estos puntos y hallamos sus distancias hasta el centroide. Para el quinto punto $(11,7)$ el BOF sería

$$\sqrt{(11-7+1)^2+7-7+1^2} = 5.1 \cong 5$$

Repitiendo el proceso, el resultado es: $BOF = [5,6,7,6,5,6,7,6,5,6,7,6,5,6,7,6]$.

APÉNDICE B

CONFIGURACIÓN DE FPGA CON ISE de Xilinx

B.1 PAQUETE DE DESARROLLO ISE DE XILINX

Para tener las dos tarjetas de desarrollo funcionando correctamente, se tienen que configurar bajando una secuencia de bits (Bitstream) desde una computadora conectada vía usb. Debemos realizar algunos procesos dentro del paquete de programación para tal propósito, que en nuestro caso es el ISE(Integrated Software Environment) versión 10.1.Puede descargarse, de la página de internet de Xilinx.

B.2 PASOS BÁSICOS PARA CONFIGURAR UN FPGA USANDO ISE 10.1

Lasos mínimos a seguir para configurar e implementar un diseño hecho en VHDL utilizando el paquete de desarrollo de Xilinx, se describen a continuación. Usar la Imagen B.1 como referencia.

a) Crear el código VHDL en el editor de texto y grabarlo. Es posible utilizar cualquier editor externo, pero luego debemos incluir el archivo a nuestra carpeta de proyectos, y asignarle una extensión .VHD para que sea reconocido.

b) Utilizar la opción de revisión de sintaxis dentro del modulo de síntesis. Esto es para corregir errores de sintaxis en nuestro código. El paquete nos indicará los errores encontrados, pero depende de la habilidad del desarrollador para solucionarlos rápidamente.

c) Utilizar la opción de síntesis y esperar confirmación de que no se tenga ningún error. Esta parte es realizada automáticamente. La corrección de errores de esta tapa requiere de mayor experiencia.

d) Utilizar la opción de implementación del diseño. Tomar en cuenta que dentro de este módulo, están contenidos los módulos de traducción, mapeo, ubicación y ruteo. Para poder tener un resultado correcto, no debemos olvidar crear el archivo con extensión .UCF; que nos permite definir nuestros puertos encargados de interactuar con el exterior, asignando nuestras señales a los puertos físicos del FPGA que ya tienen identificadores compuestos de una letra y un número.

e) Utilizar la opción de generar archivo de programación, esto nos genera una secuencia de bits (Bitstream) que se graba en un archivo con extensión .BIT, que finalmente podemos bajar a la tarjeta de desarrollo con el FPGA. Simplemente con este archivo, es posible programar cualquier otra tarjeta con características similares.

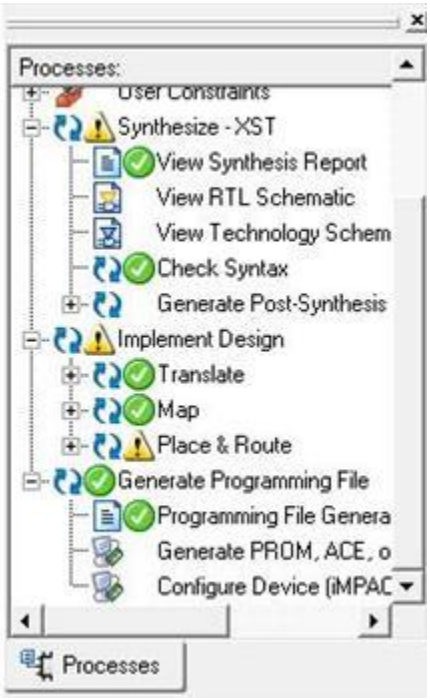

Figura B.1 Ubicación de las opciones para compilar un diseño en VHDL.
[Captura de pantalla]

B.3 CONEXIÓN FÍSICA DE LA TARJETA CON EL PAQUETE ISE

En el caso de la tarjeta Sparta3-E Starter kit, se debe estar conectado por USB con la computadora donde se instaló el paquete ISE de Xilinx, y se utiliza la opción configurar dispositivo dentro del módulo programar dispositivo. Luego de una serie de ventanas de opciones varias, se puede bajar la secuencia de bits a la tarjeta.

En la BASYS2 se configura el FPGA con otro programa, el ADEPT de Digilent. Esto es muy sencillo, simplemente se abre la carpeta donde se guardo el archivo .BIT, se lo selecciona y finalmente utilizamos la opción programar FPGA. Esta aplicación detecta automáticamente el dispositivo conectado vía USB.

Figura B.2 Programa de aplicación Adept de Digilent.
[Captura de pantalla]

APÉNDICE C

RECURSOS UTILIZADOS EN FPGA

Las tarjetas de desarrollo utilizadas tienen determinadas capacidades para contener nuestros circuitos desarrollados, por lo que se presentan a continuación los recursos empleados por el sistema.

Al principio se muestran los módulos independientes: para poder ver los resultados; control de monitor VGA(Video Graphics Adapter), control de visualizador de dos líneas por dieciseis caracteres LCD(Liquid Crystal Display), y control de 4 visualizadores de siete segmentos LED(Light Emitting Diode); así como también el de comunicación serial RS-232(Recommended Standard 232) y el de control del brazo robótico, también serial.

Al último están los dos módulos principales que se usaron para las pruebas experimentales; el de la BASYS2 que integra los controles de visualización y el de la Starter kit que realiza el cálculo del centroide, contorno y la función de frontera BOF(Boundary Object Function).

C.1 MÓDULO VGA

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	24	1,920	1%	
Number of 4 input LUTs	37	1,920	1%	
Logic Distribution				
Number of occupied Slices	30	960	3%	
Number of Slices containing only related logic	30	30	100%	
Number of Slices containing unrelated logic	0	30	0%	
Total Number 4 input LUTs	55	1,920	2%	
Number used as logic	37			
Number used as a route-thru	18			
Number of bonded IOBs	11	83	13%	
IOB Latches	2			
Number of GCLKs	2	24	8%	
Total equivalent gate count for design	541			
Additional JTAG gate count for IOBs	528			

Imagen C.1 Uso de recursos del módulo VGA. [Captura de pantalla]

C.2 MÓDULO LCD

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	53	1,920	2%	
Number of 4 input LUTs	190	1,920	9%	
Logic Distribution				
Number of occupied Slices	109	960	11%	
Number of Slices containing only related logic	109	109	100%	
Number of Slices containing unrelated logic	0	109	0%	
Total Number 4 input LUTs	209	1,920	10%	
Number used as logic	190			
Number used as a route-thru	19			
Number of bonded IOBs	7	83	8%	
Number of GCLKs	1	24	4%	
Total equivalent gate count for design	1,705			
Additional JTAG gate count for IOBs	336			

Imagen C.2 Uso de recursos del módulo LCD. [Captura de pantalla]

C.3 MÓDULO 7SEG

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	18	1,920	1%	
Number of 4 input LUTs	81	1,920	4%	
Logic Distribution				
Number of occupied Slices	59	960	6%	
Number of Slices containing only related logic	59	59	100%	
Number of Slices containing unrelated logic	0	59	0%	
Total Number 4 input LUTs	111	1,920	5%	
Number used as logic	81			
Number used as a route-thru	30			
Number of bonded IOBs	29	83	34%	
IOB Flip Flops	12			
Number of GCLKs	1	24	4%	
Total equivalent gate count for design	939			
Additional JTAG gate count for IOBs	1,392			

Imagen C.3 Uso de recursos del módulo 7SEG. [Captura de pantalla]

C.4 MÓDULO RS-232

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	68	1,920	3%	
Number used as Flip Flops	59			
Number used as Latches	9			
Number of 4 input LUTs	48	1,920	2%	
Logic Distribution				
Number of occupied Slices	49	960	5%	
Number of Slices containing only related logic	49	49	100%	
Number of Slices containing unrelated logic	0	49	0%	
Total Number 4 input LUTs	58	1,920	3%	
Number used as logic	48			
Number used as a route-thru	9			
Number used as Shift registers	1			
Number of bonded IOBs	12	83	14%	
IOB Flip Flops	2			
Number of GCLKs	2	24	8%	
Total equivalent gate count for design	942			
Additional JTAG gate count for IOBs	576			

Imagen C.4 Uso de recursos del módulo RS-232. [Captura de pantalla]

C.5 MÓDULO5 ROBOT

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	66	1,920	3%	
Number used as Flip Flops	65			
Number used as Latches	1			
Number of 4 input LUTs	120	1,920	6%	
Logic Distribution				
Number of occupied Slices	78	960	8%	
Number of Slices containing only related logic	78	78	100%	
Number of Slices containing unrelated logic	0	78	0%	
Total Number 4 input LUTs	129	1,920	6%	
Number used as logic	120			
Number used as a route-thru	9			
Number of bonded IOBs	16	83	19%	
IOB Flip Flops	4			
Number of GCLKs	1	24	4%	
Total equivalent gate count for design	1,349			
Additional JTAG gate count for IOBs	768			

Imagen C.5 Uso de recursos del módulo ROBOT. [Captura de pantalla]

C.6 MÓDULO BASYS2

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	610	1,920	31%	
Number used as Flip Flops	608			
Number used as Latches	2			
Number of 4 input LUTs	1,520	1,920	79%	
Logic Distribution				
Number of occupied Slices	958	960	99%	
Number of Slices containing only related logic	847	958	88%	
Number of Slices containing unrelated logic	111	958	11%	
Total Number 4 input LUTs	1,616	1,920	84%	
Number used as logic	1,520			
Number used as a route-thru	95			
Number used as Shift registers	1			
Number of bonded IOBs	55	83	66%	
IOB Flip Flops	33			
Number of GCLKs	4	24	16%	
Total equivalent gate count for design	16,479			
Additional JTAG gate count for IOBs	2,640			

Imagen C.6 Uso de recursos del módulo BASYS2. [Captura de pantalla]

C.7 MÓDULO7 STARTER KIT

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	2,425	9,312	26%	
Number used as Flip Flops	2,423			
Number used as Latches	2			
Number of 4 input LUTs	7,063	9,312	75%	
Logic Distribution				
Number of occupied Slices	3,923	4,656	84%	
Number of Slices containing only related logic	3,923	3,923	100%	
Number of Slices containing unrelated logic	0	3,923	0%	
Total Number of 4 input LUTs	7,168	9,312	76%	
Number used as logic	7,052			
Number used as a route-thru	105			
Number used for Dual Port RAMs	10			
Number used as Shift registers	1			
Number of bonded IOBs	19	232	8%	
Number of BUFGMUXs	2	24	8%	
Number of MULT18X18SIOs	4	20	20%	

Imagen C.7 Uso de recursos del módulo Starter Kit. [Captura de pantalla]

APÉNDICE D

ESPECIFICACIONES DE LAS TARJETAS DE DESARROLLO

Imagen D.1 Tarjetas utilizadas en este trabajo.
[Capturas de pantalla]

En este trabajo se utilizarón dos tarjetas de desarrollo para la implementación del sistema, una es la "*BASYS2"*, la otra es la "*Spartan-3E Starter Kit"*(imagen D.1); ambas plataformas para el diseño e implementación de nuestros circuitos. Las características y especificaciones son mostradas en la Imagen D.2 y en la Imagen D.3.

Basys2 board features

- *100,000-gate Xilinx Spartan 3E FPGA*
- *Atmel AT90USB2 Full-speed USB2 port providing board power and programming/data transfer interface*
- *Xilinx Platform Flash ROM to store FPGA configurations*
- *8 LEDs, 4-digit 7-segment display, 4 buttons, 8 slide switches*
- *PS/2 port and 8-bit VGA port*
- *User-settable clock (25/50/100MHz), plus socket for 2nd clock*
- *Four 6-pin header expansion connectors*
- *ESD and short-circuit protection on all I/O signals.*

Imagen D.2 Especificaciones de la tarjeta BASYS2.
[Captura de pantalla]

Spartan-3E Starter Kit

Key Components and Features

The key features of the Spartan-3E Starter Kit board are:

- Xilinx XC3S500E Spartan-3E FPGA
  - Up to 232 user-I/O pins
  - 320-pin FBGA package
  - Over 10,000 logic cells
- Xilinx 4 Mbit Platform Flash configuration PROM
- Xilinx 64-macrocell XC2C64A CoolRunner CPLD
- 64 MByte (512 Mbit) of DDR SDRAM, x16 data interface, 100+ MHz
- 16 MByte (128 Mbit) of parallel NOR Flash (Intel StrataFlash)
  - FPGA configuration storage
  - MicroBlaze code storage/shadowing
- 16 Mbits of SPI serial Flash (STMicro)
  - FPGA configuration storage
  - MicroBlaze code shadowing
- 2-line, 16-character LCD screen
- PS/2 mouse or keyboard port
- VGA display port
- 10/100 Ethernet PHY (requires Ethernet MAC in FPGA)
- Two 9-pin RS-232 ports (DTE- and DCE-style)
- On-board USB-based FPGA/CPLD download/debug interface
- 50 MHz clock oscillator
- SHA-1 1-wire serial EEPROM for bitstream copy protection
- Hirose FX2 expansion connector
- Three Digilent 6-pin expansion connectors
- Four-output, SPI-based Digital-to-Analog Converter (DAC)
- Two-input, SPI-based Analog-to-Digital Converter (ADC) with programmable-gain pre-amplifier
- ChipScope™ SoftTouch debugging port
- Rotary-encoder with push-button shaft
- Eight discrete LEDs
- Four slide switches
- Four push-button switches
- SMA clock input
- 8-pin DIP socket for auxiliary clock oscillator

Imagen D.3 Especificaciones de la tarjeta Spartan-3E Starter Kit.
[Captura de pantalla]

APÉNDICE E

CÓDIGO VHDL

A continuación presentamos el código hecho en lenguaje descriptor de hardware VHDL. Es el que contiene los módulos para el cálculo del contorno, el centroide, las cordenadas de dieciséis puntos del contorno; y la distancia entre esos puntos y el centroide, estos valores finales forman parte de la función de frontera BOF(Boundary Object Function).

Este se hizo para la tarjeta de desarrollo Spartan-3E Starter Kit, y contiene también dos controladores de comunicaciones seriales RS-232, uno para conectarse con la computadora portátil de manera alámbrica, que envía la imagen binarizada; y el otro inalámbrico para comunicarse con la tarjeta de desarrollo BASYS2, a la que se le envían los datos para visualización.

```
-- Cálculo de la Función de Frontera BOF(Boundary Object Function)
-- Creado por Raúl Alvaro Romero Ayala
-- Correo electrónico: raulromero2mil1@yahoo.com
-- Universidad Nacional Autónoma de México
-- México - Distrito Federal - 2013
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
----------------------------------------------------------------------------
entity MEM_X_final2 is
Port (RST   : in  std_logic;
      MCLK3 : in  std_logic;
      TXD3  : out std_logic := '1';
      RXD3  : in  std_logic := '1';
      RST3  : in  std_logic := '0';
      TXD9  : out std_logic := '1';
      RXD9  : in  std_logic := '1';
      RST9  : in  std_logic := '0';
      LED1  : out std_logic := '0';
      led0  : out std_logic := '0';
      led2  : out std_logic := '0';
      LED   : out STD_LOGIC_VECTOR (7 DOWNTO 4):="0000";
      SWT   :  in STD_LOGIC_VECTOR (3 DOWNTO 0)
            );
end MEM_X_final2;
----------------------------------------------------------------------------
architecture Behavioral of MEM_X_final2 is

component RS232RefComp3
Port (TXD3  : out   std_logic := '1';
      RXD3  : in    std_logic;
```

```
      CLK3  : in    std_logic;
      DBIN3 : in    std_logic_vector (7 downto 0);
      DBOUT3: out   std_logic_vector (7 downto 0);
      RDA3  : inout std_logic;
      TBE3  : inout std_logic := '1';
      RD3   : in    std_logic;
      WR3   : in    std_logic;
      PE3   : out   std_logic;
      FE3   : out   std_logic;
      OE3   : out   std_logic;
      RST3  : in    std_logic := '0');
end component;

signal dbInSig3    : std_logic_vector(7 downto 0);
signal dbOutSig3   : std_logic_vector(7 downto 0);
signal rdaSig3     : std_logic;
signal tbeSig3     : std_logic;
signal rdSig3      : std_logic;
signal wrSig3      : std_logic;
signal peSig3      : std_logic;
signal feSig3      : std_logic;
signal oeSig3      : std_logic;

constant columnas : natural:=40;
constant filas    : natural:=40;
type memoria is array (1 to filas) of std_logic_vector (1 to columnas);
--type linea   is array (1 to columnas) of bit;
--type memoria is array (1 to filas   ) of linea;
signal mem : memoria;

constant angulos : natural:=15;--De 0 a 15 = 16 angulos
type posx is array (0 to angulos) of natural range 0 to columnas;
type posy is array (0 to angulos) of natural range 0 to filas;
signal bof_x : posx;
signal bof_y : posy;

--Usar para guardar valores de los contornos en vectores
--type vec_fil is array (1 to filas)    of natural range 0 to columnas;
--type vec_col is array (1 to columnas) of natural range 0 to filas;
type vec_bof is array (0 to angulos) of natural range 0 to 30;--valor y
--si x=(mayor entre filas y columnas)/2 => y=sqr(x^2+x^2)+2
--eje: para imagen de 16x16pixeles => y=sqr(8^2+8^2)+2=13
signal BOF : vec_bof;
signal Cx  : natural range 0 to columnas:=0;
signal Cy  : natural range 0 to filas:=0;

signal sigue0 : std_logic:='0';
signal sigue1 : std_logic:='0';
signal sigue3 : std_logic:='0';
------------------------------------------------------------------------
component RS232RefComp9
   Port (TXD9     : out    std_logic   := '1';
         RXD9     : in     std_logic;
         CLK9     : in     std_logic;
         DBIN9    : in     std_logic_vector (7 downto 0);
         DBOUT9   : out    std_logic_vector (7 downto 0);
         RDA9     : inout  std_logic;
```

```
         TBE9     : inout     std_logic   := '1';
         RD9      : in        std_logic;
         WR9      : in        std_logic;
         PE9      : out       std_logic;
         FE9      : out       std_logic;
         OE9      : out       std_logic;
         RST9     : in        std_logic   := '0');
end component;

      signal dbInSig9     :   std_logic_vector(7 downto 0);
      signal dbOutSig9    :   std_logic_vector(7 downto 0);
      signal rdaSig9      :   std_logic;
      signal tbeSig9      :   std_logic;
      signal rdSig9       :   std_logic;
      signal wrSig9       :   std_logic;
      signal peSig9       :   std_logic;
      signal feSig9       :   std_logic;
      signal oeSig9       :   std_logic;
-------------------------------------------------------------------------
signal mayor  : natural range 0 to 31:= 0;--De 0 (al mismo de raiz)
signal menor  : natural range 0 to 31:=31;--De 0 (al mismo de raiz)

begin

UART3: RS232RefComp3 port map (
            TXD3  => TXD3,
            RXD3  => RXD3,
            CLK3  => MCLK3,
            DBIN3 => dbInSig3,
            DBOUT3=> dbOutSig3,
            RDA3  => rdaSig3,
            TBE3  => tbeSig3,
            RD3   => rdSig3,
            WR3   => wrSig3,
            PE3   => peSig3,
            FE3   => feSig3,
            OE3   => oeSig3,
            RST3  => RST3);
-------------------------------------------------------------------------
process (MCLK3)
variable sm : natural range 0 to 7 :=0;
variable i : natural range 0 to filas+1 := 0;
variable j : natural range 0 to columnas+1 := 0;
begin
if rising_edge(MCLK3) then
if RST='1' then
      sm:=0;sigue0<='0';
else
      case sm is
      when 0 => i:=1;j:=1;sm:=sm+1;
      when 1 => rdSig3 <= '0';wrSig3 <= '0';sm:=sm+1;
      when 2 => if rdaSig3 = '1' then dbInSig3 <= dbOutSig3;sm:=sm+1;end
if;
      when 3 => rdSig3 <= '1';wrSig3 <= '1';sm:=sm+1;
      when 4 => rdSig3 <= '1';wrSig3 <= '0';sm:=sm+1;
      when 5 => mem(i)(j to j+7)<=dbInSig3;
            j:=j+8;sm:=1;
```

```
            if j>columnas then j:=1;i:=i+1;if i> filas then
i:=1;j:=1;sm:=6;end if;end if;
      when 6 =>--null;
            sigue0<='1';
      when others => null;
      end case;
end if;
end if;
end process;
-------------------------------------------------------------------------
process (MCLK3,sigue0)
--Usar para guardar valores de los contornos en vectores
--variable izquierda : vec_fil:=(others=>0);
--variable derecha   : vec_fil:=(others=>0);
--variable arriba    : vec_col:=(others=>0);
--variable abajo     : vec_col:=(others=>0);
variable izquierda : natural range 0 to columnas:=0;--De 0 a columnas
variable derecha   : natural range 0 to columnas:=0;--De 0 a columnas
variable arriba    : natural range 0 to filas:=0;--De 0 a filas
variable abajo     : natural range 0 to filas:=0;--De 0 a filas

variable Cx_suma   : natural range 0 to 1600:=0;--De 0 a (columnas*filas)
variable Cy_suma   : natural range 0 to 1600:=0;--De 0 a (filas*columnas)
variable Cx_inc    : natural range 0 to 80:=0;--De 0 a 2*filas
variable Cy_inc    : natural range 0 to 80:=0;--De 0 a 2*columnas
variable Cx_aux    : natural range 0 to columnas:=0;--De 0 a columnas
variable Cy_aux    : natural range 0 to filas:=0;--De 0 a filas

variable sm : natural range 0 to 11:=0;
variable i : natural range 0 to filas+1:=0;
variable j : natural range 0 to columnas+1:=0;
begin
if rising_edge(MCLK3) then
if RST='1' then
      sm:=0;sigue1<='0';
      Cx_suma:=0;Cy_suma:=0;Cx_aux:=0;Cy_aux:=0;
      Cx<=0;Cy<=0;
else

if sigue0='1' then
case sm is
--Calculo de Cx
when 0 => i:=1;j:=0;sm:=1;derecha:=0;izquierda:=0;Cx_inc:=0;
when 1 =>
      j:=j+1;
      if j>columnas then
            --Usar para guardar valores de los contornos en vectores
            --Cx_suma:=Cx_suma+izquierda(i)+derecha(i);
            Cx_suma:=Cx_suma+izquierda+derecha;izquierda:=0;derecha:=0;
            j:=1;i:=i+1;
            if i>filas then i:=1;sm:=3;else sm:=2;end if;
      else sm:=2;
      end if;
when 2 =>
      if mem(i)(j)='1' then
            derecha:=j;
            --Usar para guardar valores de los contornos en vectores
```

```
            --derecha(i):=j;
            --if izquierda(i)=0 then izquierda(i):=j;Cx_inc:=Cx_inc+2;end
if;
            if izquierda=0 then izquierda:=j;Cx_inc:=Cx_inc+2;end if;
      end if;
      sm:=1;
when 3 =>
      if Cx_suma<Cx_inc then sm:=4;
      else
            Cx_suma:=Cx_suma-Cx_inc;Cx_aux:=Cx_aux+1;
      end if;
when 4 =>Cx <= Cx_aux;sm:=5;
--Calculo de Cy
when 5 => j:=1;i:=0;sm:=6;arriba:=0;abajo:=0;Cy_inc:=0;
when 6 =>
      i:=i+1;
      if i>=filas then
            --Usar para guardar valores de los contornos en vectores
            --Cy_suma:=Cy_suma+arriba(j)+abajo(j);
            Cy_suma:=Cy_suma+arriba+abajo;arriba:=0;abajo:=0;
            i:=1;j:=j+1;
            if j>columnas then j:=1;sm:=8;else sm:=7;end if;
      else sm:=7;
      end if;
when 7 =>
      if mem(i)(j)='1' then
            abajo:=i;
            --Usar para guardar valores de los contornos en vectores
            --abajo(j):=i;
            --if arriba(j)=0 then arriba(j):=i;Cy_inc:=Cy_inc+2;end if;
            if arriba=0 then arriba:=i;Cy_inc:=Cy_inc+2;end if;
      end if;
      sm:=6;
when 8 =>
      if Cy_suma<Cy_inc then sm:=9;
      else
            Cy_suma:=Cy_suma-Cy_inc;Cy_aux:=Cy_aux+1;
      end if;
when 9 =>Cy <= Cy_aux;sm:=10;
--Terminado
when 10 => sigue1<='1';--null;
when others => null;
end case;
end if;
end if;
end if;
end process;
-------------------------------------------------------------------------
process (MCLK3,sigue1)
variable Cx_pos : posx:=(others=>0);
variable Cy_pos : posy:=(others=>0);

variable Cx_aux : natural range 0 to columnas:=0;--De 0 a columnas
variable Cy_aux : natural range 0 to filas:=0;--De 0 a filas
variable XX     : natural range 0 to 21:=0;--De 0 a (columnas/2+1)=x
variable YY     : natural range 0 to 21:=0;--De 0 a (filas/2+1)=y
variable MM     : natural range 0 to 882:=0;--De 0 a (x^2+y^2)
```

```
--variable mayor  : natural range 0 to 31:=0;--De 0 (al mismo de raiz)
--variable menor  : natural range 0 to 31:=31;--De 0 (al mismo de raiz)
variable raiz   : natural range 0 to 31:=0;--De 0 (al mismo de bof+1)
--si x=(mayor entre filas y columnas)/2 => y=sqr(x^2+x^2)+2
--eje: para imagen de 16x16pixeles => y=sqr(8^2+8^2)+2=13
variable pot1   : natural range 0 to 961:=0;--De 0 a raiz^raiz
variable pot2   : natural range 0 to 900:=0;--De 0 a (raiz-1)^(raiz-1)

variable sm     : natural range 0 to 24:=0;
variable i      : natural range 0 to filas+1:=0;
variable j      : natural range 0 to columnas+1:=0;

variable indice : natural range 0 to angulos+1:=0;

begin
if rising_edge(MCLK3) then
if RST='1' then
      sm:=0;sigue3<='0';
      Cx_pos :=(others=>0);Cy_pos :=(others=>0);
      mayor<=0;menor<=31;
else
--Calculo de coordenadas de 16 puntos del contorno
if sigue1='1' then
case sm is
when 0 => --inicializacion
      Cx_aux:=Cx;Cy_aux:=Cy;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
when 1 =>--90 grados
      if mem(i)(j)='1' then i:=i-1;
      else Cx_pos(0):=j;Cy_pos(0):=i+1;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
      end if;
when 2 =>--67.5 grados
      if mem(i)(j)='1' then j:=j+1;i:=i-2;
      else if mem(i+1)(j)='0' then j:=j-1;i:=i+2;end if;
            Cx_pos(1):=j;Cy_pos(1):=i;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
      end if;
when 3 =>--45 grados
      if mem(i)(j)='1' then j:=j+1;i:=i-1;
      else Cx_pos(2):=j-1;Cy_pos(2):=i+1;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
      end if;
when 4 =>--22.5 grados
      if mem(i)(j)='1' then j:=j+2;i:=i-1;
      else if mem(i)(j-1)='0' then j:=j-2;i:=i+1;end if;
            Cx_pos(3):=j;Cy_pos(3):=i;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
      end if;
when 5 =>--0/360 grados
      if mem(i)(j)='1' then j:=j+1;
      else Cx_pos(4):=j-1;Cy_pos(4):=i;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
      end if;
when 6 =>--337.5 grados
      if mem(i)(j)='1' then j:=j+2;i:=i+1;
      else if mem(i)(j-1)='0' then j:=j-2;i:=i-1;end if;
            Cx_pos(5):=j;Cy_pos(5):=i;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
      end if;
when 7 =>--315 grados
      if mem(i)(j)='1' then j:=j+1;i:=i+1;
      else Cx_pos(6):=j-1;Cy_pos(6):=i-1;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
```

```
      end if;
when 8 =>--292.5 grados
      if mem(i)(j)='1' then j:=j+1;i:=i+2;
      else if mem(i-1)(j)='0' then j:=j-1;i:=i-2;end if;
            Cx_pos(7):=j;Cy_pos(7):=i;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
      end if;
when 9 =>--270 grados
      if mem(i)(j)='1' then i:=i+1;
      else Cx_pos(8):=j;Cy_pos(8):=i-1;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
      end if;
when 10=>--247.5 grados
      if mem(i)(j)='1' then j:=j-1;i:=i+2;
      else if mem(i-1)(j)='0' then j:=j+1;i:=i-2;end if;
            Cx_pos(9):=j;Cy_pos(9):=i;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
      end if;
when 11=>--225 grados
      if mem(i)(j)='1' then j:=j-1;i:=i+1;
      else Cx_pos(10):=j+1;Cy_pos(10):=i-1;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
      end if;
when 12=>--202.5 grados
      if mem(i)(j)='1' then j:=j-2;i:=i+1;
      else if mem(i)(j+1)='0' then j:=j+2;i:=i-1;end if;
            Cx_pos(11):=j;Cy_pos(11):=i;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
      end if;
when 13=>--180 grados
      if mem(i)(j)='1' then j:=j-1;
      else Cx_pos(12):=j+1;Cy_pos(12):=i;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
      end if;
when 14=>--157.5 grados
      if mem(i)(j)='1' then j:=j-2;i:=i-1;
      else if mem(i)(j+1)='0' then j:=j+2;i:=i+1;end if;
            Cx_pos(13):=j;Cy_pos(13):=i;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
      end if;
when 15=>--135 grados
      if mem(i)(j)='1' then j:=j-1;i:=i-1;
      else Cx_pos(14):=j+1;Cy_pos(14):=i+1;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
      end if;
when 16=>--112.5 grados
      if mem(i)(j)='1' then j:=j-1;i:=i-2;
      else if mem(i+1)(j)='0' then j:=j+1;i:=i+2;end if;
            Cx_pos(15):=j;Cy_pos(15):=i;i:=Cy_aux;j:=Cx_aux;sm:=sm+1;
      end if;

when 17=>indice:=0;sm:=sm+1;
when 18=>--calculo de distancias para conformar el BOF
      if Cx_aux>Cx_pos(indice) then XX:=Cx_aux-Cx_pos(indice)+1;
      else XX:=Cx_pos(indice)-Cx_aux+1;end if;
      if Cy_aux>Cy_pos(indice) then YY:=Cy_aux-Cy_pos(indice)+1;
      else YY:=Cy_pos(indice)-Cy_aux+1;end if;
      MM:=XX*XX+YY*YY;sm:=sm+1;
when 19=>raiz:=0;pot1:=0;Pot2:=0;sm:=sm+1;
when 20=>raiz:=raiz+1;pot1:=raiz*raiz;if pot1>MM then sm:=sm+1;end if;
when 21=>pot2:=(raiz-1)*(raiz-1);
      if (pot1-MM)>(MM-pot2) then raiz:=raiz-1;end if;
      sm:=sm+1;
when 22=> --BOF_aux(indice):=raiz;
      BOF(indice)<=raiz;--Conformación del vector BOF
```

```
      if raiz>mayor then mayor<=raiz;end if;
      if raiz<menor then menor<=raiz;end if;
      indice:=indice+1;if indice>15 then sm:=23;else sm:=18;end if;
when 23=>--null;
--    if (mayor-menor)>4 then led0<='1';else led2<='1';end if;
-- comparación para diferenciar entre círculo y cuadrado
      bof_x <= Cx_pos;
      bof_y <= Cy_pos;
      sigue3<='1';
      sm:=sm+1;
  when 24=>null;
when others => null;
end case;
end if;
end if;
end if;
end process;
-------------------------------------------------------------------------
      UART9: RS232RefComp9 port map (
                                                    TXD9  => TXD9,
                                                    RXD9  => RXD9,
                                                    CLK9  => MCLK3,
                                                    DBIN9 => dbInSig9,
                                                    DBOUT9=> dbOutSig9,
                                                    RDA9  => rdaSig9,
                                                    TBE9  => tbeSig9,
                                                    RD9   => rdSig9,
                                                    WR9   => wrSig9,
                                                    PE9   => peSig9,
                                                    FE9   => feSig9,
                                                    OE9   => oeSig9,
                                                    RST9  => RST9);
-------------------------------------------------------------------------
process (MCLK3)
variable sm  : natural range 0 to 32:=0;
variable pos : integer range 0 to angulos+1:=0;
begin
if rising_edge(MCLK3) then
if RST='1' then
      sm:=0;--sigue3<='0';
      led0<='0';led2<='0';
else

if sigue3='1' then
      case sm is
      when  0 => sm:=sm+1;pos:=0;rdSig9 <= '1';wrSig9 <='0';

      when  1 => sm:=sm+1;dbInSig9 <= conv_std_logic_vector(Cx,8);
      when  2 => sm:=sm+1;rdSig9 <= '1';wrSig9 <='1';
      when  3 => sm:=sm+1;rdSig9 <= '0';wrSig9 <='0';
      when  4 => if rdaSig9 = '1' then --LED9 <= dbOutSig9;
            sm:=sm+1;end if;
      when  5 => sm:=sm+1;rdSig9 <= '1';wrSig9 <='0';

      when  6 => sm:=sm+1;dbInSig9 <= conv_std_logic_vector(Cy,8);
      when  7 => sm:=sm+1;rdSig9 <= '1';wrSig9 <='1';
      when  8 => sm:=sm+1;rdSig9 <= '0';wrSig9 <='0';
```

```
      when  9 => if rdaSig9 = '1' then --LED9 <= dbOutSig9;
            sm:=sm+1;end if;
      when 10 => sm:=sm+1;rdSig9 <= '1';wrSig9 <='0';

      when 11 => sm:=sm+1;dbInSig9 <= conv_std_logic_vector(BOF(pos),8);
      when 12 => sm:=sm+1;rdSig9 <= '1';wrSig9 <='1';
      when 13 => sm:=sm+1;rdSig9 <= '0';wrSig9 <='0';
      when 14 => if rdaSig9 = '1' then --LED9 <= dbOutSig9;
            sm:=sm+1;end if;
      when 15 => sm:=sm+1;rdSig9 <= '1';wrSig9 <='0';

      when 16 => sm:=sm+1;dbInSig9 <=
conv_std_logic_vector(bof_x(pos),8);
      when 17 => sm:=sm+1;rdSig9 <= '1';wrSig9 <='1';
      when 18 => sm:=sm+1;rdSig9 <= '0';wrSig9 <='0';
      when 19 => if rdaSig9 = '1' then --LED9 <= dbOutSig9;
            sm:=sm+1;end if;
      when 20 => sm:=sm+1;rdSig9 <= '1';wrSig9 <='0';

      when 21 => sm:=sm+1;dbInSig9 <=
conv_std_logic_vector(bof_y(pos),8);
      when 22 => sm:=sm+1;rdSig9 <= '1';wrSig9 <='1';
      when 23 => sm:=sm+1;rdSig9 <= '0';wrSig9 <='0';
      when 24 => if rdaSig9 = '1' then --LED9 <= dbOutSig9;
            sm:=sm+1;end if;
      when 25 => sm:=sm+1;rdSig9 <= '1';wrSig9 <='0';

      when 26 => sm:=11;pos:=pos+1;if pos>15 then sm:=27;end if;

      when 27 =>
            if (mayor-menor)>4 then led0<='1';dbInSig9 <= "00000001";
            else led2<='1';dbInSig9 <= "00000000";
            end if;
            sm:=sm+1;
      --when 27 => sm:=sm+1;dbInSig9 <=
conv_std_logic_vector(bof_y(pos),8);
      when 28 => sm:=sm+1;rdSig9 <= '1';wrSig9 <='1';
      when 29 => sm:=sm+1;rdSig9 <= '0';wrSig9 <='0';
      when 30 => if rdaSig9 = '1' then --LED9 <= dbOutSig9;
            sm:=sm+1;end if;
      when 31 => sm:=sm+1;rdSig9 <= '1';wrSig9 <='0';
      when 32 => null;
      when others => null;
      end case;
end if;
end if;
end if;
end process;
-------------------------------------------------------------------------
process (MCLK3)--led testigo UCLK=100MHz
variable cuenta:natural range 0 to 50000000:=0;
variable testigo:std_logic:='0';
begin
if rising_edge(MCLK3) then
      cuenta:=cuenta+1;
      if cuenta = 50000000 then
            cuenta:=0;
```

```
            testigo:= not testigo;
      end if;
end if;
LED1 <= testigo;
end process;
--------------------------------------------------------------------------
--1011 1110 1011 1100 0010 0000 0000=200000000(200millones)
process (MCLK3)--led testigo UCLK=100MHz
variable cuenta : std_logic_vector(27 downto 0):=x"0000000";
--variable cuenta:natural range 0 to 200000000:=0;
variable testigo:std_logic:='0';
begin
if rising_edge(MCLK3) then
if RST='1' then
      LED <= "0000";
      cuenta:=x"0000000";
else
      if sigue0='1' and sigue3='0' then
            cuenta:=cuenta+1;
            --if cuenta = 50000000 then
            --    cuenta:=0;
            --    testigo:= not testigo;
            --end if;
      elsif sigue0='1' and sigue3='1' then
                  case SWT is
                  when "0000" => LED <= cuenta( 3 downto  0);
                  when "0001" => LED <= cuenta( 7 downto  4);
                  when "0010" => LED <= cuenta(11 downto  8);
                  when "0011" => LED <= cuenta(15 downto 12);
                  when "0100" => LED <= cuenta(19 downto 16);
                  when "0101" => LED <= cuenta(23 downto 20);
                  when "0110" => LED <= cuenta(27 downto 24);
                  when others => LED <= "0000";
                  end case;
      end if;
end if;
end if;
end process;
--------------------------------------------------------------------------
end Behavioral;
--------------------------------------------------------------------------
```

APÉNDICE F

CÓDIGO MATLAB

Aquí tenemos el programa codificado en MatLab7.12, creado para pruebas experimentales y de comparación con el FPGA. Tiene un menu de usuario con opciones de previsualización con la cámara de video, captura de una imagen, procesamiento local, y envío de una región de interés a la tarjeta de desarrollo Spartan-3E Starter Kit, con conexión alámbrica serial Rs-232.

```
% Programa para el Sistema Desarrollado (BOF)
% ============================================
% Codificado por : Raúl Alvaro Romero Ayala (raulromero2mil1@yahoo.com)
% Tutor de tesis : Dr. Juan Mario Peña Cabrera
% Universidad Nacional Autónoma de México
% Distrito Federal
% Versión Actual : Enero 2013
%
% ENVIA! imagen via RS-232 a una tarjeta basada en FPGA
% La imagen es enviada a partir de tres fuentes:
% 1)Creada manualmente en matriz de unos y ceros(40filas x 40columnas)
% 2)Cargada desde un archivo JPG(40x40 pixeles)
% 3)Desde una foto tomada por una camara (Región de Interés=40x40 pixeles)
%
% El proceso se puede hacer de dos maneras:
% A) El FPGA se encarga de procesar la imagen recibida y calcula:
%    Cx,Cy,BOF,bof_x,bof_y
% B) En matlab se calculan Cx,Cy,BOF,bof_x,bof_y
%    (no es necesario enviar imagen)
%
% Los tiempos de ejecución se calculan Matlab y en el FPGA
% Mencionar que las velocidades de procesamiento son diferentes
% -MatLab7.12: Procesando en computadora portátil,
%              Windows Vista Ultimate, procesador de 2x1.9GHz
% -FPGA      : Procesando en tarjeta de desarrollo Spartan-3E Starter Kit,
%              velocidad de reloj interno de 50MHz
%
%% Inicializar
clc%Limpiar pantalla
clear%Borrar memoria
filas=40;columnas=40;
colorfondo1='Blanco';
%% Imagen de 40x40 pixeles, creada para prueba experimental
imagenMEM =[
%                  1                   2                   3
%1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%2
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%3
```

```
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%4
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%5
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%6
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%7
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%8
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%9
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%10
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%11
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%12
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%13
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%14
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%15
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%16
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%17
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%18
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%19
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%20
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%21
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%22
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0;%23
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0;%24
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0;%25
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0;%26
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0;%27
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0;%28
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0;%29
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0;%30
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0;%31
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0;%32
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0;%33
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0;%34
```

```
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0;%35
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%36
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%37
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%38
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%39
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0;%40
];%Cuadrado en la parte inferior derecha de la imagen,Cx=28,Cy=29,13x13pixeles

%% Inicializar puerto serie
clc%Limpiar pantalla
disp('Estableciendo comunicación...')
Puerto6 = serial('COM6');
set(Puerto6,'BaudRate',19200);
set(Puerto6,'DataBits',8);
set(Puerto6,'Parity','odd');
set(Puerto6,'StopBits',1);
set(Puerto6,'FlowControl','none');
pause(3);
%% Inicializar variables
op=0;%Variable de menu de opciones
origen_imagen=0;% 1=camara/2=JPG/3=memoria
%% Realizar el proceso
while op == 0
    clc;
    disp('PRUEBA FPGA')%Mostrar titulo
    disp('============')
    %disp(' ')
    disp('[1] Abtir puerto.')
    disp('[2] Cerrar puerto.')
    disp('[a] Inicializar camara.')
    disp('[b] Enviar solamente un dato(1 byte).')
    disp('[3] Previsualización de la camara.')
    colorfondo2=sprintf('[4] Color del fondo: %s',  colorfondo1);
    disp(colorfondo2)
    disp('[5] Capturar imagen (Identificar objeto).')
    disp('[6] ENVIAR toda la imagen(por bytes).')
    disp('[7] Cargar Foto tomada por CAMARA.')
    disp('[8] Cargar imagen de la MEMORIA.')
    disp('[9] Cargar Imagen desde JPG.')
    disp(' ')
    disp('[e] Calculo de Cx,Cy y BOF.')
    disp('[t] Ver resultados.')
    disp(' ')
    disp('[0] Salir.(Numero 0)')
    disp(' ')
    resp=input('Seleccione opcion y presione <enter>: ','s');
%% Ejecutar la opción elegida
    switch lower(resp)
        case '0'
           op=1;
        case '1'
           %% Abrir puerto serie
           fopen(Puerto6);
           disp(' ')
           disp('Abriendo puerto...')
           pause(2)
        case '2'
```

```
            %% Cerrar puerto serie
           fclose(Puerto6);
           disp(' ')
           disp('Cerrando puerto...')
           pause(3)
        case 'a'
            closepreview
            vid=videoinput('winvideo',2,'RGB24_160x120');%Identifica dispositivo
            preview(vid)%Realizar un preview (caso normal)
            closepreview
        case 'b'%utilizado para depuración
            fprintf(Puerto6,'b');%Enviar b
            disp(' ')
            disp('Enviando dato...b')
            %pause(2)
        case '3'
            closepreview
            preview(vid)%Realizar un preview (caso normal)
        case '4'
            if colorfondo1=='Blanco'
                colorfondo1='Negro ';
            else
                colorfondo1='Blanco';
            end
        case '5'
            disp(' ')
            ok=input('Presione <enter> para tomar foto e identificar objeto...');
            %closepreview(no borrar)
            %preview(vid)%para un preview(caso rapido)
            disp(' ')
            foto=getsnapshot(vid);%para tomar una foto
            image(foto)%para visualizar la foto
            imagen=foto;
            %closepreview (no borrar)
            disp(' ')
            ok=input('Presione enter para continuar...');
            %pause(3)
            close
            imshow(imagen)%Mostrar figura, reemplaza figura actual
            %figure, imshow(imagen)%Mostrar figura en nueva ventana
            pause on
            disp(' ')
            ok=input('Presione enter para continuar...');
            %pause(3)
            close
            %%Procesar imagen
            imagen_grises=rgb2gray(imagen); %convertir a escala de grises
            umbral=graythresh(imagen_grises);%calcular threshold y
            %umbral=umbral/2;%calibra umbral hacia objetos claros en fondo blanco
            umbral=umbral;%/1.3;%calibra umbral hacia objetos obscuros en fondo
blanco
            bw=im2bw(imagen_grises,umbral); %binarizar
            imshow(bw)%mostrar imagen
            disp(' ')
            ok=input('Presione enter para continuar...');
           %pause(3)
            close

            if colorfondo1 == 'Blanco'
                bw2=not(bw);%invertir imagen para fondos claros
                imshow(bw2)%mostrar imagen
                disp(' ')
                ok=input('Presione enter para continuar...');
```

```
                %pause(3)
                close
            end

            [L Ne]=bwlabel(bw2);%Etiquetar elementos conectados
            %propied= regionprops(L,'all');%Calcular propiedades de los objetos
de la imagen
            propied=
regionprops(L,'Area','BoundingBox','Centroid','Perimeter','Eccentricity');
            EE=propied.Eccentricity;
            if EE<0.4%utilizado para depuración
                %disp('Es un Circulo')
                forma='circulo';
            else
                %disp('Es un Cuadrado')
                forma='cuadrado';
            end
            disp(' ')
            ok=input('Presione enter para continuar...');
        case '6'
            ms=0.05;%tiempo de  50 milisegundos
            %ms=0.1;%tiempo de 100 milisegundos
            tecla='';num=0;
            disp(' ')
            disp('Enviando imagen....')
            disp(' ')
            for i=1:filas
             for j=1:8:columnas-(8-1)
               mem1 = mem(i,j:j+(8-1));
               mem2 = '00000000';
               for jj=1:8; mem2(jj)=int2str(mem1(jj));end
               mem3=bin2dec(mem2);
               mem4=[char(mem3)];
               fwrite(Puerto6,mem4); %ok1?
             end
            end
            disp(' ')
            disp('Imagen enviada OK....')
            disp(' ')
            pause(1)
        case '7'%Cargar imagen de la foto tomada por la camara de video
            imagenX=zeros (40,40);
            filsX=61;colsX=60;
            imagenX=bw2(filsX:filsX+40-1,colsX:colsX+40-1);
            imshow(imagenX)
            disp(' ')
            ok=input('Presione enter para continuar...');
            imagenX
            mem=imagenX;
            r_imagen_original=foto;
        case '8'%Cargar imagen  de memoria local
            mem=imagenMEM;
            imshow(mem);
            disp(' ')
            ok=input('Presione enter para continuar...');
            r_imagen_original=imagenMEM;
            imagen_grises=imagenMEM;
            bw=not(imagenMEM);
            bw2=imagenMEM;
        case '9'%Cargar imagen  de archivo JPG
            imagenJPG=imread('E:\pruebas ultimas bof\secuencial\prueba
matlab\imagenJPG\imagenJPG.jpg');%Ubicación del archivo
            r_imagen_original=imagenJPG;
```

```
            disp(' ')
            fot=imagenJPG;
            image(fot)
            imagen=imagenJPG;
            %closepreview (no borrar)
            disp(' ')
            ok=input('Presione enter para continuar...');
            %pause(3)
            close
            imshow(imagen)%Mostrar figura, reemplaza figura actual
            %figure, imshow(imagen)%Mostrar figura en nueva ventana
            disp(' ')
            ok=input('Presione enter para continuar...');
            %pause on
            %pause(3)
            close
            %%Procesar imagen
            imagen_grises=rgb2gray(imagen); %convertir a escala de grises
            umbral=graythresh(imagen_grises);%calcular threshold y
            %umbral=umbral/2;%calibra umbral hacia objetos claros en fondo blanco
            umbral=umbral;%/1.3;%calibra umbral hacia objetos obscuros en fondo
blanco
            bw=im2bw(imagen_grises,umbral); %binarizar
            imshow(bw)%mostrar imagen
            %pause(3)
            disp(' ')
            ok=input('Presione enter para continuar...');
            close

            if colorfondo1 == 'Blanco'
                bw2=not(bw);%invertir imagen para fondos claros
                imshow(bw2)     %mostrar imagen
                %disp('bbbbbb') %breakpoint para depurar
                disp(' ')
                ok=input('Presione enter para continuar...');
                %pause(3)
                close
            end

            [L Ne]=bwlabel(bw2);    %Etiquetar elementos conectados
            %propied= regionprops(L,'all');%Calcular propiedades de los objetos
de la imagen
            propied=
regionprops(L,'Area','BoundingBox','Centroid','Perimeter','Eccentricity');
            EE=propied.Eccentricity;
            if EE<0.1;%0.4%utilizado para depuración
                %disp('Es un Circulo')
                forma='circulo';
            else
                %disp('Es un Cuadrado')
                forma='cuadrado';
            end
            disp(' ')
            ok=input('Presione enter para continuar...');

            mem=bw2;%carga la imagen binarizada
        case 'e'%Calculon del Centroide
            r_tiempo1=tic;
            r_filas=filas;r_columnas=columnas;
            Cx_suma=0;Cy_suma=0;Cx_aux=0;Cy_aux=0;Cx_inc=0;Cy_inc=0;
            r_Cx=0;r_Cy=0;r_imagen=mem;
            r_arriba   = zeros (1 , r_columnas);%vector de enteros del contorno
arriba
```

```
            r_abajo    = zeros (1 , r_columnas);%vector de enteros del contorno
abajo
            r_izquierda= zeros (r_filas , 1);%vector de enteros del contorno
izquierda
            r_derecha  = zeros (r_filas , 1);%vector de enteros del contorno
derecha
            r_ciclo=0;r_sm=0;r_i=0;r_j=0;
            while r_ciclo==0
                switch r_sm
                    case 0 %calculo de Cx
                        r_i=1;r_j=0;Cx_inc=0;r_sm=1;
                        r_izquierda= zeros (r_filas , 1);%vector de enteros del
contorno izquierda
                        r_derecha  = zeros (r_filas , 1);%vector de enteros del
contorno derecha
                    case 1
                        r_j=r_j+1;
                        if r_j>r_columnas
                            Cx_suma=Cx_suma+r_izquierda(r_i)+r_derecha(r_i);
                            %r_derecha(r_i)=0;r_izquierda(r_i)=0;
                            r_j=1;r_i=r_i+1;
                            if r_i>r_filas r_i=1;r_sm=3;
                            else r_sm=2;end;
                        else r_sm=2;end
                    case 2
                        if r_imagen(r_i,r_j)==1
                            r_derecha(r_i)=r_j;
                            if r_izquierda(r_i)==0
                                r_izquierda(r_i)=r_j;Cx_inc=Cx_inc+2;end
                        end
                        r_sm=1;
                    case 3
                        if Cx_suma<Cx_inc r_sm=4;
                        else Cx_suma=Cx_suma-Cx_inc;Cx_aux=Cx_aux+1;end
                    case 4
                        r_Cx=Cx_aux;r_sm=5;
                    case 5 %calculo de Cy
                        r_j=1;r_i=0;Cy_inc=0;r_sm=6;
                        r_arriba   = zeros (1 , r_columnas);%vector de enteros
del contorno arriba
                        r_abajo    = zeros (1 , r_columnas);%vector de enteros
del contorno abajo
                    case 6
                        r_i=r_i+1;
                        if r_i>r_filas
                            Cy_suma=Cy_suma+r_arriba(r_j)+r_abajo(r_j);
                            %r_abajo(r_j)=0;r_arriba(r_j)=0;
                            r_i=1;r_j=r_j+1;
                            if r_j>r_columnas r_j=1;r_sm=8;
                            else r_sm=7;end;
                        else r_sm=7;end
                    case 7
                        if r_imagen(r_i,r_j)==1
                            r_abajo(r_j)=r_i;
                            if r_arriba(r_j)==0
                                r_arriba(r_j)=r_i;Cy_inc=Cy_inc+2;end
                        end
                        r_sm=6;
                    case 8
                        if Cy_suma<Cy_inc r_sm=9;
                        else Cy_suma=Cy_suma-Cy_inc;Cy_aux=Cy_aux+1;end
                    case 9
                        r_Cy=Cy_aux;r_sm=10;
```

```
                    case 10
                        r_ciclo=1;
                    otherwise
                        r_ciclo=r_ciclo;
                end

            end
%         case 'r'%Calculon del BOF
            led0=0;led0=0;
            r_XX  = zeros (1,16);
            r_YY  = zeros (1,16);
            r_DD  = zeros (1,16);
            r_BOF = zeros (1,16);

            XX=0;YY=0;MM=0;
            raiz=0;pot1=0;pot2=0;
            mayor=0;menor=31;
            indice=0;

            Cx_aux=0;Cy_aux=0;
            r_ciclo=0;r_sm=0;r_i=0;r_j=0;
            while r_ciclo==0
                switch r_sm
                    case  0 % inicializacion
                        Cx_aux=r_Cx;Cy_aux=r_Cy;
                        r_i=Cy_aux;r_j=Cx_aux
                        r_sm=r_sm+1;
                    case  1 %  90 grados
                        if r_imagen(r_i,r_j)==1 r_i=r_i-1;
                        else r_XX(1)=r_j;r_YY(1)=r_i+1;
                            r_i=Cy_aux;r_j=Cx_aux;r_sm=r_sm+1;
                        end
                    case  2 %  67.5 grados
                        if r_imagen(r_i,r_j)==1 r_j=r_j+1;r_i=r_i-2;
                        else if r_imagen(r_i+1,r_j)==0 r_j=r_j-1;r_i=r_i+2;end
                            r_XX(2)=r_j;r_YY(2)=r_i;
                            r_i=Cy_aux;r_j=Cx_aux;r_sm=r_sm+1;
                        end
                    case  3 %  45 grados
                        if r_imagen(r_i,r_j)==1 r_j=r_j+1;r_i=r_i-1;
                        else r_XX(3)=r_j-1;r_YY(3)=r_i+1;
                            r_i=Cy_aux;r_j=Cx_aux;r_sm=r_sm+1;
                        end
                    case  4 %  22.5 grados
                        if r_imagen(r_i,r_j)==1 r_j=r_j+2;r_i=r_i-1;
                        else if r_imagen(r_i,r_j-1)==0 r_j=r_j-2;r_i=r_i+1;end
                            r_XX(4)=r_j;r_YY(4)=r_i;
                            r_i=Cy_aux;r_j=Cx_aux;r_sm=r_sm+1;
                        end
                    case  5 %   0(360 grados)
                        if r_imagen(r_i,r_j)==1 r_j=r_j+1;
                        else r_XX(5)=r_j-1;r_YY(5)=r_i;
                            r_i=Cy_aux;r_j=Cx_aux;r_sm=r_sm+1;
                        end
                    case  6 % 337.5 grados
                        if r_imagen(r_i,r_j)==1 r_j=r_j+2;r_i=r_i+1;
                        else if r_imagen(r_i,r_j-1)==0 r_j=r_j-2;r_i=r_i-1;end
                            r_XX(6)=r_j;r_YY(6)=r_i;
                            r_i=Cy_aux;r_j=Cx_aux;r_sm=r_sm+1;
                        end
                    case  7 % 315 grados
                        if r_imagen(r_i,r_j)==1 r_j=r_j+1;r_i=r_i+1;
                        else r_XX(7)=r_j-1;r_YY(7)=r_i-1;
```

```
            r_i=Cy_aux;r_j=Cx_aux;r_sm=r_sm+1;
        end
    case  8 % 315 grados
        if r_imagen(r_i,r_j)==1 r_j=r_j+1;r_i=r_i+2;
        else if r_imagen(r_i-1,r_j)==0 r_j=r_j-1;r_i=r_i-2;end
            r_XX(8)=r_j;r_YY(8)=r_i;
            r_i=Cy_aux;r_j=Cx_aux;r_sm=r_sm+1;
        end
    case  9 % 270 grados
        if r_imagen(r_i,r_j)==1 r_i=r_i+1;
        else r_XX(9)=r_j;r_YY(9)=r_i-1;
            r_i=Cy_aux;r_j=Cx_aux;r_sm=r_sm+1;
        end
    case 10 % 247.5 grados
        if r_imagen(r_i,r_j)==1 r_j=r_j-1;r_i=r_i+2;
        else if r_imagen(r_i-1,r_j)==0 r_j=r_j+1;r_i=r_i-2;end
            r_XX(10)=r_j;r_YY(10)=r_i;
            r_i=Cy_aux;r_j=Cx_aux;r_sm=r_sm+1;
        end
    case 11 % 225 grados
        if r_imagen(r_i,r_j)==1 r_j=r_j-1;r_i=r_i+1;
        else r_XX(11)=r_j+1;r_YY(11)=r_i-1;
            r_i=Cy_aux;r_j=Cx_aux;r_sm=r_sm+1;
        end
    case 12 % 202.5 grados
        if r_imagen(r_i,r_j)==1 r_j=r_j-2;r_i=r_i+1;
        else if r_imagen(r_i,r_j+1)==0 r_j=r_j+2;r_i=r_i-1;end
            r_XX(12)=r_j;r_YY(12)=r_i;
            r_i=Cy_aux;r_j=Cx_aux;r_sm=r_sm+1;
        end
    case 13 % 180 grados
        if r_imagen(r_i,r_j)==1 r_j=r_j-1;
        else r_XX(13)=r_j+1;r_YY(13)=r_i;
            r_i=Cy_aux;r_j=Cx_aux;r_sm=r_sm+1;
        end
    case 14 % 157.5 grados
        if r_imagen(r_i,r_j)==1 r_j=r_j-2;r_i=r_i-1;
        else if r_imagen(r_i,r_j+1)==0 r_j=r_j+2;r_i=r_i+1;end
            r_XX(14)=r_j;r_YY(14)=r_i;
            r_i=Cy_aux;r_j=Cx_aux;r_sm=r_sm+1;
        end
    case 15 % 135 grados
        if r_imagen(r_i,r_j)==1 r_j=r_j-1;r_i=r_i-1;
        else r_XX(15)=r_j+1;r_YY(15)=r_i+1;
            r_i=Cy_aux;r_j=Cx_aux;r_sm=r_sm+1;
        end
    case 16 % 112.5 grados
        if r_imagen(r_i,r_j)==1 r_j=r_j-1;r_i=r_i-2;
        else if r_imagen(r_i+1,r_j)==0 r_j=r_j+1;r_i=r_i+2;end
            r_XX(16)=r_j;r_YY(16)=r_i;
            r_i=Cy_aux;r_j=Cx_aux;r_sm=r_sm+1;
        end

    case 17
        indice=1;r_sm=r_sm+1;
    case 18 % calculo de la magnitud
        if Cx_aux>r_XX(indice) XX=Cx_aux-r_XX(indice)+1;
        else XX=r_XX(indice)-Cx_aux+1;end
        if Cy_aux>r_YY(indice) YY=Cy_aux-r_YY(indice)+1;
        else YY=r_YY(indice)-Cy_aux+1;end
        MM=XX*XX+YY*YY;r_sm=r_sm+1;
    case 19
        raiz=0;pot1=0;Pot2=0;r_sm=r_sm+1;
```

```
                    case 20
                        raiz=raiz+1;pot1=raiz*raiz;if pot1>MM r_sm=r_sm+1;end
                    case 21
                        pot2=(raiz-1)*(raiz-1);
                        if (pot1-MM)>(MM-pot2) raiz=raiz-1;end
                        r_sm=r_sm+1;
                    case 22
                        r_BOF(indice)=raiz;
                        if raiz>mayor mayor=raiz;end
                        if raiz<menor menor=raiz;end
                        indice=indice+1;if indice>16 r_sm=23;else r_sm=18;end
                    case 23 %
                        if (mayor-menor)>4 led0=1;else led2=1;end
                        %r_XX = Cx_pos;
                        %r_YY = Cy_pos;
                        r_sm=r_sm+1;
                    case 24
                        r_ciclo=1;
                    otherwise
                        r_ciclo=r_ciclo;
                end
            end
            r_tiempo2=toc(r_tiempo1);
        case 't'%Ver resultados
            %Ver resultados
            %Contorno
            %r_contorno=r_imagen;
            r_contorno=zeros (r_filas,r_columnas);
            for i=1:r_filas
                for j=1:r_columnas
                    if     r_arriba   (j)==i r_contorno(i,j)=1;
                    elseif r_abajo    (j)==i r_contorno(i,j)=1;
                    elseif r_izquierda(i)==j r_contorno(i,j)=1;
                    elseif r_derecha  (i)==j r_contorno(i,j)=1;
                    elseif r_Cx==j & r_Cy==i r_contorno(i,j)=1;
                    else %r_contorno(i,j)=0;
                    end
                end
            end
            %Frontera
            %r_frontera=r_imagen;
            r_frontera= zeros (r_filas,r_columnas);
            for indice=1:16
                for i=1:r_filas
                    for j=1:r_columnas
                        if     r_XX(indice)==j & r_YY(indice)==i
r_frontera(i,j)=1;
                        elseif r_Cx==j & r_Cy==i r_frontera(i,j)=1;
                        else %r_frontera(i,j)=0;
                        end
                    end
                end
             end
            % figuras
            figure,imshow(r_imagen_original);
%             figure,imshow(r_contorno);
%             figure,imshow(r_frontera);
%             figure;
%             subplot(1,3,1);imshow(r_imagen);
%             subplot(1,3,2);imshow(r_contorno);
%             subplot(1,3,3);imshow(r_frontera);

            figure;
```

```
            subplot(2,4,1),
            imshow(r_imagen_original);grid;%mostrar imagen original
            title('Imagen Original');%xlabel('Columnas');ylabel('Filas');

            subplot(2,4,2), imshow(imagen_grises);grid;%mostrar a escala de
grises
            title('Escala de Grises');%xlabel('Columnas');ylabel('Filas');

            subplot(2,4,3), imshow(bw);grid;%mostrar imagen binarizada
            title('Binarización');%xlabel('Columnas');ylabel('Filas');

            subplot(2,4,4), imshow(bw2);grid;%mostrar imagen complementada
            title('Complemento');%xlabel('Columnas');ylabel('Filas');

            subplot(2,4,5), imshow(r_contorno);%mostrar el contorno
            title('Contorno');%xlabel('Columnas');ylabel('Filas');

            subplot(2,4,6), imshow(r_frontera);%mostrar las coordenadas graficas
            title('Coordenadas');%xlabel('Columnas');ylabel('Filas');

            subplot(2,4,7), plot(r_XX,r_YY,'x',r_Cx,r_Cy,'+');grid;%mostrar las
coordenadas
            title('Coordenadas');%xlabel('columnas');ylabel('filas');
            axis equal;axis ij;axis([1 columnas 1 filas]);

            subplot(2,4,8), plot(r_BOF);grid;%mostrar el BOf
            %axis([0 17 r_filas/5 r_filas/2]);title('BOF(x) para x=1..16');
            axis([0 17 5 30]);title('BOF(x) para x=1..16');

        otherwise
            disp(' ')
            disp('Incorrecto!!!')
            pause(2)
    end

end
%% Cerrar puerto serie
 %fclose(Puerto6);
 delete(Puerto6)
 clear Puerto6
 disp(' ')
 disp('Comunicación terminada.')
 pause(3)
 closepreview
```

ANEXO 1

COSTOS

Tabla AN1.1 COSTO DEL SISTEMA

Elemento	**Pesos MXN (Mexicanos)**	**Dólares USD (Estadounidenses)**
Tarjeta de Desarrollo Spartan-3E Starter Kit	3500.00	280.00
Tarjeta de Desarrollo BASYS2	1200.00	96.00
Visualizador de cristal líquido	80.00	6.40
Cable serial RS-232 a USB	120.00	9.60
Módulo de radio frecuencia (x3)	1050.00	84.00
Total	**5950.00**	**476.00**

Tabla AN1.2 COSTO DE LA ESTACIÓN DE TRABAJO

Elemento	**Pesos MXN (Mexicanos)**	**Dólares USD (Estadounidenses)**
Brazo robótico K-680	720.00	57.60
Microcontrolador PIC18F4550	90.00	7.20
Manejadores de potencia UL2003(x2)	20.00	1.60
Convertidor de voltaje MAX3232	20.00	1.60
Relevadores electromecánicos(x7)	35.00	2.80
Reguladores de voltaje (5v y 3.3v)	20.00	1.60
Convertidor de voltaje RS-232 a TTL	120.00	9.60
Adaptador de voltaje 110vAC a 12vDC	60.00	4.80
Fuentes de luz (x2)	80.00	6.40
Total	**1165.00**	**93.20**

ANEXO 2

ESTACIÓN DE TRABAJO

JUSTIFICACIÓN

Con la finalidad de poder observar que, realmente el sistema desarrollado en este trabajo podría darnos como resultado el cálculo de los valores que conforman la función de frontera BOF; comparamos estos resultados, tanto de una figura cúbica como de una cilíndrica, para luego recoger uno de ellos de su posición inicial. Por lo que se implementó una estación de trabajo pequeña, la cual describimos a continuación.

DIAGRAMA ESQUEMÁTICO

Tenemos un brazo robótico y dos bases giratorias que reciben órdenes de movimiento del circuito de control, este a su vez es comandado desde el exterior vía comunicación RS-232; alimentado por una fuente de poder que entrega los voltajes de potencia y de señal necesarios. Por encima de la base 1 donde se encuentran los objetos inicialmente, está ubicada una cámara de video USB. En la parte de iluminación, por la parte posterior la fuente de luz 1 y por la parte superior la fuente de luz 2 a 45 grados de dirección hacia el objeto, ambas direccionales. (Figura AN2.1)

DESARROLLO DE LA ESTACIÓN

Se adquirieron todos los componentes que integrarían la estación de trabajo, a continuación se estudió sobre las especificaciones de funcionamiento y se hicieron las pruebas con cada uno independientemente. Para adaptar el brazo desconectamos su batería y conectamos los motores de corriente continua a nuestro circuito.

El circuito de control fue desarrollado programando un PIC18F4550, en lenguaje PIC C, que es de alto nivel, tipo C pero orientado a estos microcontroladores. Esta parte lleva además manejadores de potencia para los motores de corriente directa del brazo y de los motores a pasos de las bases giratorias, y relevadores electromecánicos normalmente abiertos y cerrados para su encendido o apagado. En determinados puertos de entrada se conectaron: un sensor de fuerza ubicado en la pinza del efector final para coger el objeto, un final de carrera que detecta si el brazo está arriba o abajo, dos microinterruptores que detienen el giro de la base del brazo en uno u otro sentido, y un sensor óptico de ranura para poder parar el motor al abrir la pinza del efector final.

La fuente de poder toma los 110 voltios de la red eléctrica y nos entrega 12, 5 y 3.3 voltios a través de un adaptador y 2 reguladores de voltaje. También las dos fuentes de iluminación se conectan a

la red. En el caso de la cámara de video, es conectada a un puerto USB y toma del mismo la energía para su funcionamiento.

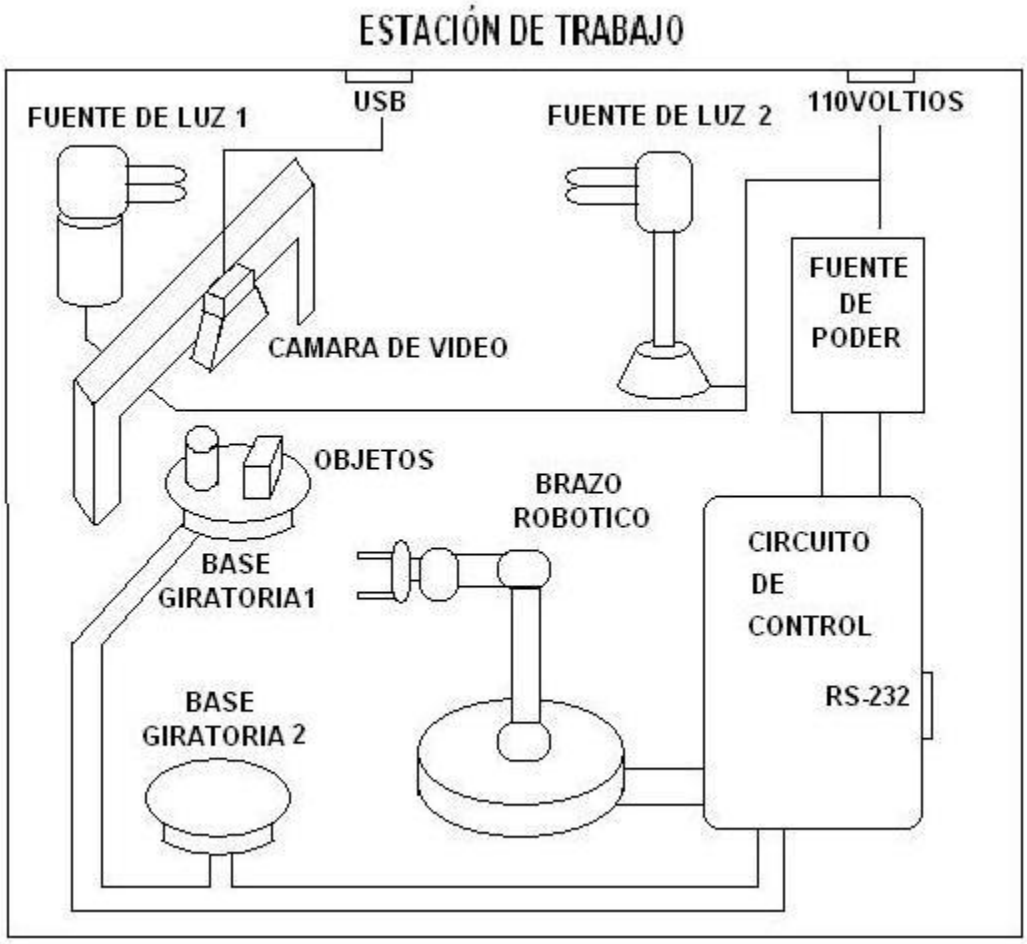

Figura AN2.1 Descripción gráfica de los elementos de la estación de trabajo. [Elaboración propia]

BRAZO ROBÓTICO

Kit de brazo robótico con control remoto alámbrico: K-680 (Imágenes AN2.1 y AN2.2)
Entrada: 6Voltios corriente directa (4 xD) (simétrica +3Voltios/-3Voltios)
Capacidad de carga de la tenaza: 100 gramos.
[Fuente: Manual de usuario V0608]

SENSOR DE FUERZA

Sensor de fuerza resistivo FSR(Force Sensing Resistor):
Variación aproximada de resistencia: 10 Kohms – 100Kohms (ohmios)
Fuerza sobre un área de 0.81cm2 (centímetros cuadrados): 9Kg (kilogamos)
[Fuente: Catálogo de Interlink]

El botón M1 abre y cierra la tenaza, Los botones M2, M3 y M4 mueven el brazo mecánico, El botón M5 gira la base del brazo. El interruptor ON/OFF sirve para encender o apagar el LED del brazo.

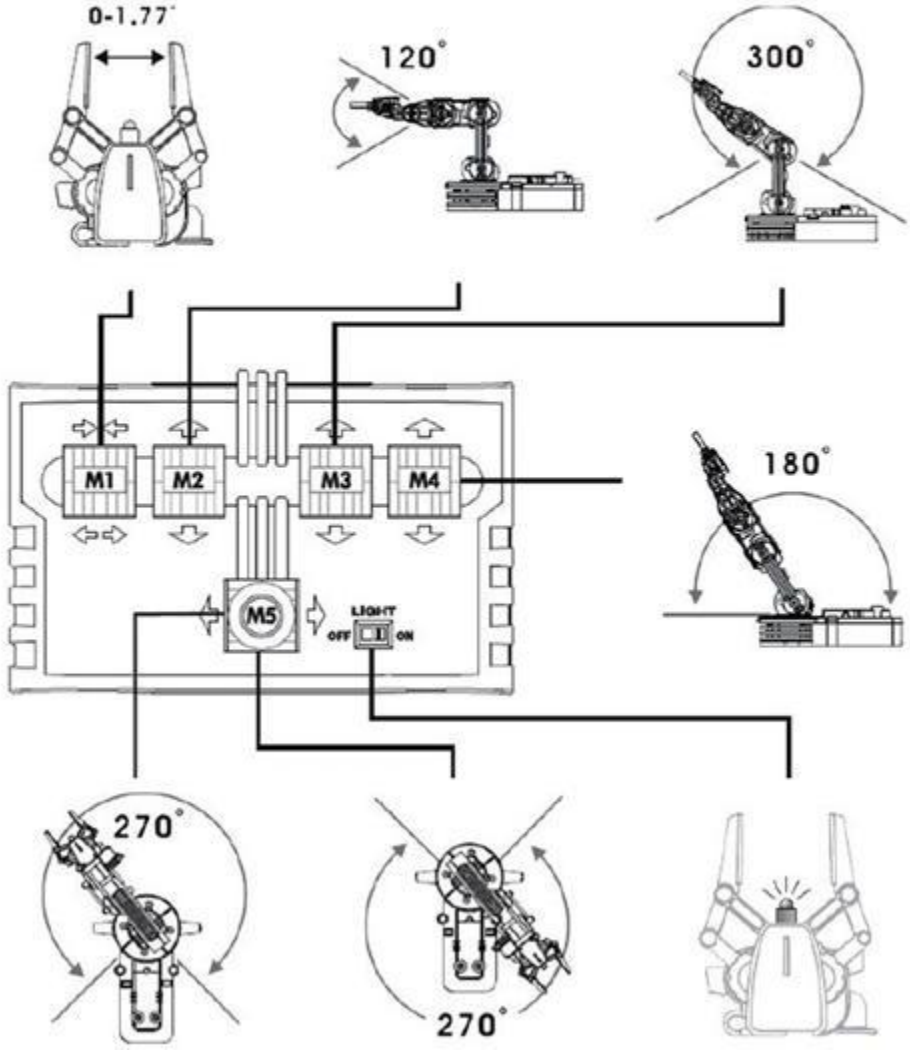

Imagen AN2.1 Operación del brazo robótico. [Manual V0608]

SENSOR ÓPTICO DE RANURA

Sensor óptico de ranura de 5 pines:
Ánodo y Cátodo del diodo emisor de luz interno: 3.3v y 0v (voltios).
Voltaje de alimentación y tierra: 5v y 0v (voltios)
Salida del sensor: Depende del voltaje de alimentación.(voltios)
Apertura de la ranura: 10mm (milímetros)
[Fuente: Hoja técnica del sensor)]

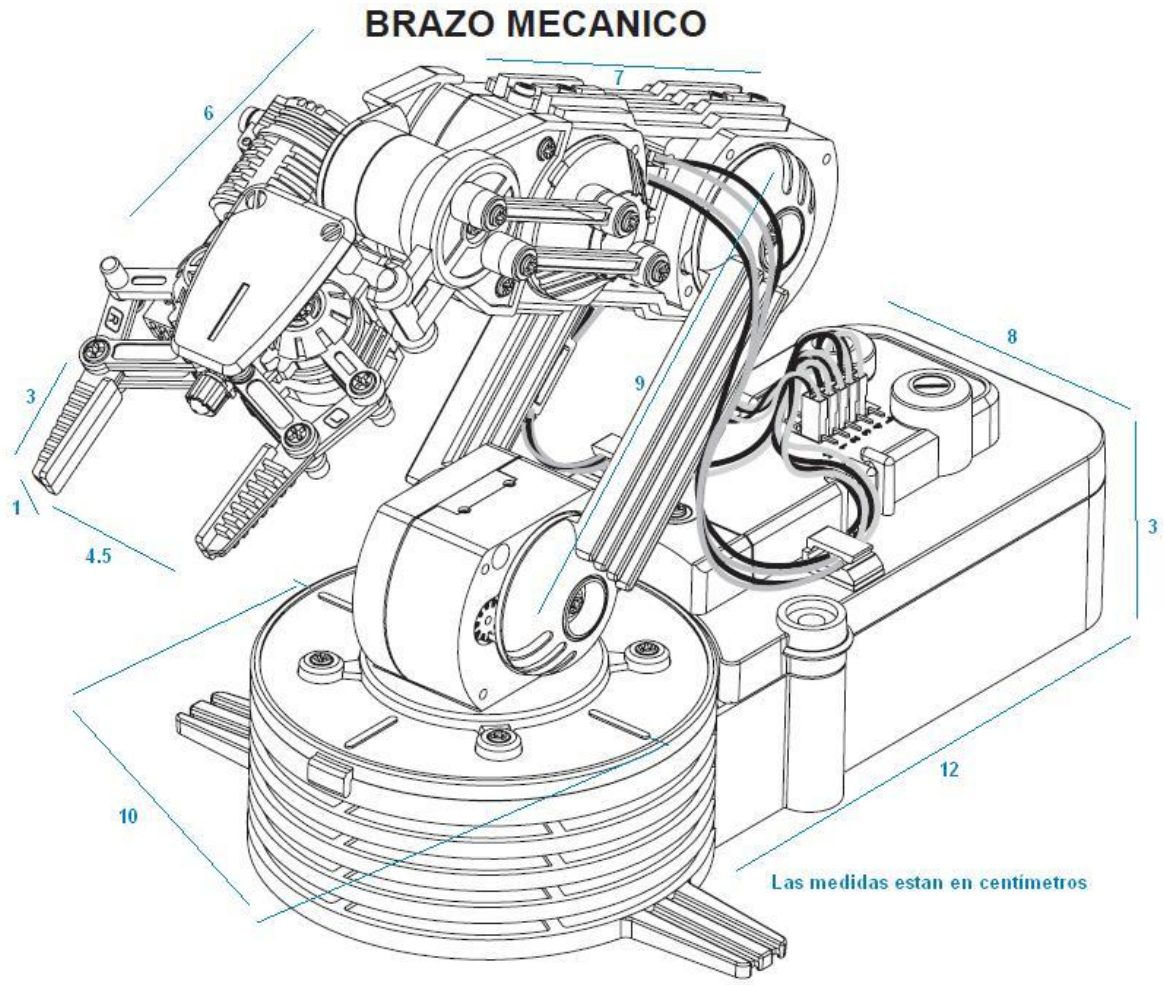

Imagen AN2.2 Dimensiones del brazo robótico. [Manual V0608]

FINAL DE CARRERA Y MICROINTERRUPTORES

Final de carrera normalmente cerrado.
Microinterruptores normalmente abiertos.
[Producto comercial]

MOTORES

Cinco motores de corriente continua DC(Direct Current): 3.3v (voltios)
[Incluidos con el brazo robótico]

Dos motor a pasos:
Tipo: unipolar clase B
Alimentación: 36v (voltios)
Pasos: 7.5G/paso (grados por paso)
[Fuente: Astrosyn Stepper]

OTROS ELEMENTOS

Manejador de potencia: ULN2803 (circuito integrado)
Microcontrolador: PIC18F4550 (circuito integrado)
Convertidor de voltaje: MAX3232 (ciruito integrado)
Relevadores electromecánicos: RAS05 (normalmente abierto)
Reguladores de voltaje: LM7805 (5 voltios) y LD1117V33 (3.3 voltios)
Cámara de video: Wb918la HP (Heweltt Packard)
Conector para comunicación serial RS-232: DB9 hembra (Data Bit)
Lámparas para iluminación: 110v (voltios)

IMPLEMENTACIÓN Y COMUNICACIÓN CON EL FPGA

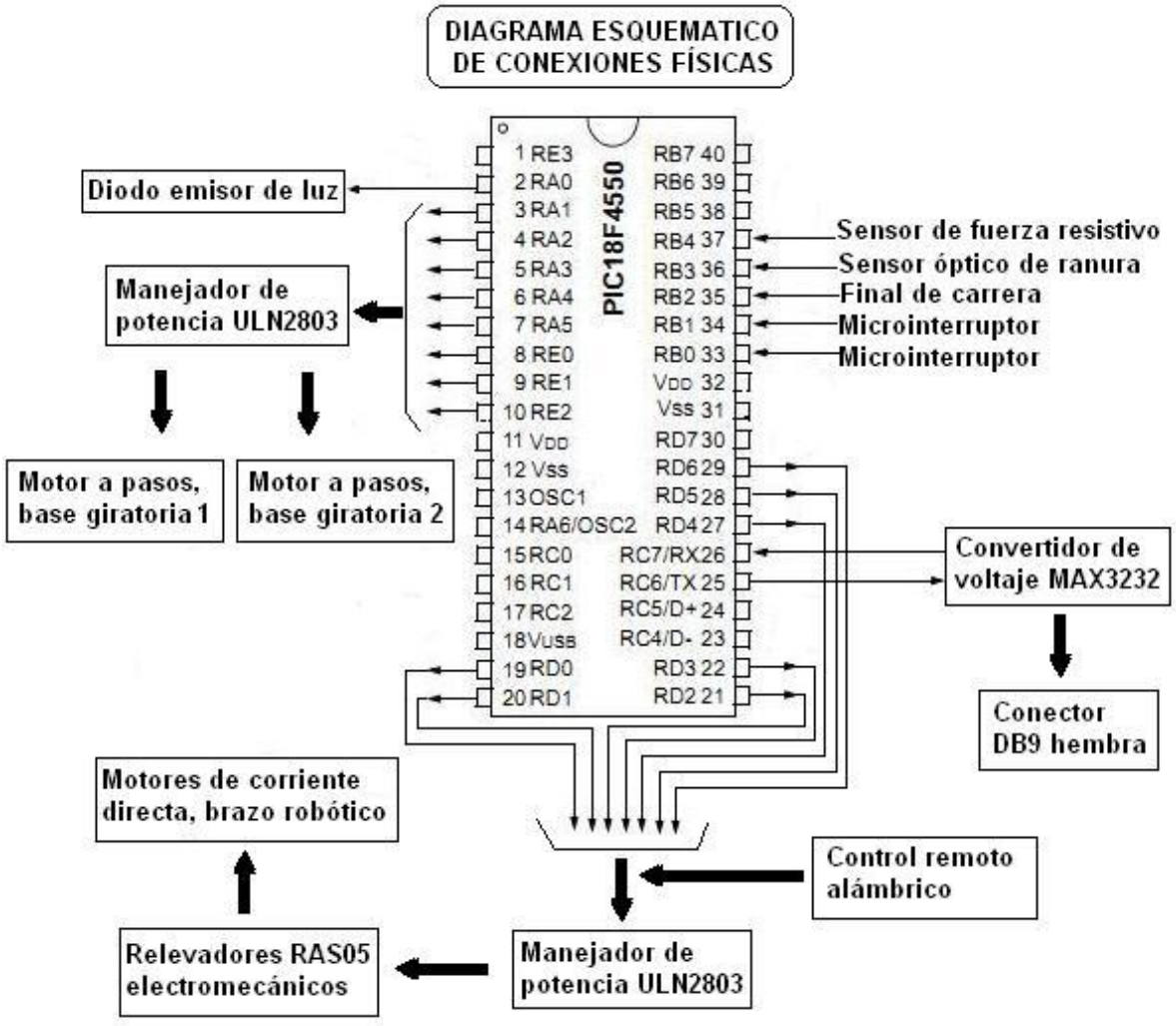

Figura AN2.2 Hardware del circuito de control de la estación de trabajo. [Elaboración propia]

La comunicación con el FPGA se realiza a través el puerto serial RS-232, conectando un cable al conector DB9; entre el microcontrolador y los motores del brazo y bases giratorias, usamos los manejadores de potencia en conjunto con los relevadores electromecánicos.(Figura AN2.2)La

estación de trabajo resultante tiene un área de 61cm por 53cm, la cámara está sobre los objetos a 15cm, la fuente de luz 1 ilumina los objetos por su parte posterior y la fuente de luz 2 elevada a 30cm de altura incide a unos 45 grados por el frente. El objeto cúbico tiene las siguientes dimensiones: 1.5cmx1.5cm.1.8cm y el cilíndrico: 2.7cm de alto y 1.5cm de diámetro.

PROGRAMA CODIFICADO EN PIC-C

```
//Raul Romero Ayala – UNAM – Mexico DF (raulromero2mil1@yahoo.com)
#include <18F4550.h>
#fuses xt,nowdt,noput,noprotect,brownout,nomclr,nolvp
#use delay(clock=1Mhz)//pruebas con cristal externo de 20MHz y 8 MHz
#use rs232 (baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)
#use fast_io(B)//para los sensores conectados al puerto B de entrada
void main() {
int i;//variable auxiliar para led de status en modo testigo al iniciar
int foco;//variable auxiliar que guarda estado del foco blanco del robot
char ch;//variable que guarda tecla de ingreso o salida del modulo de control
char op;//variable que guarda tecla de opcion de comando a ejecutar
char rtu[8];//Estado de los sensores conectados al puerto B de entrada
char mtu[13];//Estado de las opciones elegidas para realizar un comando
set_tris_b(0b11111111);//Sensores
set_tris_d(0b00000000);//Motores dc del brazo robotico
set_tris_a(0b000000);//4fases del motor2 y fase 1 del motor1 (motores a pasos)
set_tris_e(0b000);//las otras 3 fases del motor1 (motor a pasos)
output_low(pin_d0);output_low(pin_d1);output_low(pin_d2);output_low(pin_d3);
output_low(pin_d4);output_low(pin_d5);output_low(pin_d6);output_low(pin_d7);
output_low(pin_a0);output_low(pin_a1);output_low(pin_a2);output_low(pin_a3);
output_low(pin_a4);output_low(pin_a5);
output_low(pin_e0);output_low(pin_e1);output_low(pin_e2);
for (i=1;i<=4;i++) {
  output_low(pin_a0);delay_ms(400);
  output_high(pin_a0);delay_ms(400);}
printf("Comunicacion establecida\r");
mtu="0000000000000";//el byte esta invertido PUERTO D comandos
rtu="00000000";//el byte esta invertido PUERTO B sensores
//printf("\r mtu= ");for (i=1;i<=8;i++)printf("%c",mtu[i-1]);//mtu
printf("\r Sensores= ");for (i=8;i>=1;i--)printf("%c",rtu[i-1]);//rtu
printf("\r\r <Presione (W) para comenzar y (Q) para detener>");printf("\r\r    ");
rtu[0]=0;rtu[1]=0;rtu[2]=0;rtu[3]=0;rtu[4]=0;rtu[5]=0;rtu[6]=0;rtu[7]=0;
mtu[0]=0;mtu[1]=0;mtu[2]=0;mtu[3]=0;mtu[4]=0;mtu[5]=0;mtu[6]=0;mtu[7]=0;
mtu[8]=0;mtu[9]=0;mtu[10]=0;mtu[11]=0;mtu[12]=0;
foco=0;
while (true){//Programa principal
  while (ch!='w') ch=getch();
  printf ("\r ON ");
  while (ch!='q'){
```

```
if (input(pin_b0)) rtu[0]='1';else rtu[0]='0';//microswitch1
if (input(pin_b1)) rtu[1]='1';else rtu[1]='0';//microswitch2
if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';//final de carrera
if (input(pin_b3)) rtu[3]='1';else rtu[3]='0';//sensor de ranura
if (input(pin_b4)) rtu[4]='1';else rtu[4]='0';//sensore de fuerza
if (input(pin_b5)) rtu[5]='1';else rtu[5]='0';//no sensor
if (input(pin_b6)) rtu[6]='1';else rtu[6]='0';//no sensor
if (input(pin_b7)) rtu[7]='1';else rtu[7]='0';//no sensor
//-----------------------------------------------------------------
if (mtu[1]=='1') { //comando r -> Girar motor1 hacia la derecha
  //Girar Motor1 a pasos en sentido de las agujas del reloj
  //4 veces da una vuelta
  for (i=1;i<=4;i++) {
    output_high(pin_e0);delay_ms(400);output_low(pin_e0);
    output_high(pin_e1);delay_ms(400);output_low(pin_e1);
    output_high(pin_e2);delay_ms(400);output_low(pin_e2);
    output_high(pin_a5);delay_ms(400);output_low(pin_a5);
  };
  mtu[1]='0';
  //printf("\r Proceso terminado");//Mandar termino de proceso
  };
//-----------------------------------------------------------------
if (mtu[2]=='1') { //comando t -> Girar motor1 hacia la izquierda
  //Girar Motor1 a pasos en contra de las agujas del reloj
  output_high(pin_e2);delay_ms(400);output_low(pin_e2);
  output_high(pin_e1);delay_ms(400);output_low(pin_e1);
  output_high(pin_e0);delay_ms(400);output_low(pin_e0);
  output_high(pin_a5);delay_ms(400);output_low(pin_a5);
  mtu[2]='0';
  //printf("\r Proceso terminado");//Mandar termino de proceso
  };
//-----------------------------------------------------------------
if (mtu[3]=='1') { //comando y -> Girar motor2 hacia la derecha
  //Girar Motor2 a pasos en sentido de las agujas del reloj
  //4 veces de una vuelta
  for (i=1;i<=4;i++) {
    output_high(pin_a2);delay_ms(400);output_low(pin_a2);
    output_high(pin_a3);delay_ms(400);output_low(pin_a3);
    output_high(pin_a4);delay_ms(400);output_low(pin_a4);
    output_high(pin_a1);delay_ms(400);output_low(pin_a1);
  };
  mtu[3]='0';
  //printf("\r Proceso terminado");//Mandar termino de proceso
  };
//-----------------------------------------------------------------
if (mtu[4]=='1') { //comando u -> Girar motor2 hacia la izquierda
```

```
  //Girar Motor2 a pasos en contra de las agujas del reloj
  output_high(pin_a4);delay_ms(400);output_low(pin_a4);
  output_high(pin_a3);delay_ms(400);output_low(pin_a3);
  output_high(pin_a2);delay_ms(400);output_low(pin_a2);
  output_high(pin_a1);delay_ms(400);output_low(pin_a1);
  mtu[4]='0';
  //printf("\r Proceso terminado");//Mandar termino de proceso
  };
//---------------------------------------------------------------
if (mtu[7]=='1') { //comando z -> Girar base brazo robotico a izquierda
  //output_high(pin_d0);//giro normal del motor dc de la base
  if (input(pin_b1)) rtu[1]='1';else rtu[1]='0';
  if (rtu[1]=='0') output_high(pin_d5);
  while (rtu[1]=='0') {
    if (input(pin_b1)) rtu[1]='1';else rtu[1]='0';
  };
  output_low(pin_d5);
  //output_low(pin_d0);
  mtu[7]='0';
 };//Si el sensor en el puerto b1 se activa, ya se llego.
  //Sensor switch izquierdo conectado
//---------------------------------------------------------------
if (mtu[8]=='1') { //comando x -> Girar base brazo robotico a derecha
  output_high(pin_d0);//giro inverso del motor dc de la base
  if (input(pin_b0)) rtu[0]='1';else rtu[0]='0';
  if (rtu[0]=='0') output_high(pin_d5);
  while (rtu[0]=='0') {
    if (input(pin_b0)) rtu[0]='1';else rtu[0]='0';
  };
  output_low(pin_d5);
  output_low(pin_d0);
  mtu[8]='0';
 };//Si el sensor en el puerto b0 se activa, ya se llego.
  //Sensor switch derecho conectado
//---------------------------------------------------------------
if (mtu[9]=='1') {   //comando c -> Subir brazo robotico
  //output_high(pin_d0);//giro normal del motor dc del codo
  if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
  if (rtu[2]=='1') output_high(pin_d3);
  while (rtu[2]=='1') {
    if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
  };
  output_low(pin_d3);
  //output_low(pin_d0);
  mtu[9]='0';
 };//Si el sensor en el puerto b2 se DESACTIVA, ya se llego.
```

```
  //Sensor final de carrera DESactivado
//-----------------------------------------------------------------
if (mtu[10]=='1') {   //comando v -> Bajar brazo robotico
  output_high(pin_d0);//giro inverso del motor dc del codo
  if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
  if (rtu[2]=='0') output_high(pin_d3);
  while (rtu[2]=='0') {
    if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
  };
  output_low(pin_d3);
  output_low(pin_d0);
  mtu[10]='0';
 };//Si el sensor en el puerto b2 se activa, ya se llego.
  //Sensor final de carrera activado
//-----------------------------------------------------------------
if (mtu[11]=='1') {   //comando b -> Cerrar gripper de brazo robotico
  //output_high(pin_d0);//giro normal del motor dc del gripper
  if (input(pin_b4)) rtu[4]='1';else rtu[4]='0';
  if (rtu[4]=='0') output_high(pin_d1);
  while (rtu[4]=='0') {
    if (input(pin_b4)) rtu[4]='1';else rtu[4]='0';
  };
  output_low(pin_d1);
  //output_low(pin_d0);
  mtu[11]='0';
 };//Si el sensor en el puerto b4 se activa, ya se llego.
  //Sensor de fuerza presionado
//-----------------------------------------------------------------
if (mtu[12]=='1') {   //comando n -> Abrir gripper de brazo robotico
  output_high(pin_d0);//giro inverso del motor dc del gripper
  if (input(pin_b3)) rtu[3]='1';else rtu[3]='0';
  if (rtu[3]=='1') output_high(pin_d1);
  while (rtu[3]=='1') {
    if (input(pin_b3)) rtu[3]='1';else rtu[3]='0';
  };
  output_low(pin_d1);
  output_low(pin_d0);
  mtu[12]='0';
 };//Si el sensor en el puerto b3 se activa, ya se llego.
  //Sensor optico de ranura bloqueado //logica negada
//-----------------------------------------------------------------
if (mtu[0]=='1') {   //comando m -> encender/apagar foco del gripper
  if (foco==0) {   //invierte el ultimo estado del foco
    foco=1;
    output_high(pin_d6);
  }
```

```
    else {
      foco=0;
      output_low(pin_d6);
    };
    mtu[0]='0';
  };
 //---------------------------------------------------------------
 if (mtu[5]=='1') {  //comando p -> Inicializar brazo robotico
    //comando n -> Abrir gripper de brazo robotico
    output_high(pin_d0);//giro inverso del motor dc del gripper
    if (input(pin_b3)) rtu[3]='1';else rtu[3]='0';
    if (rtu[3]=='1') output_high(pin_d1);
    while (rtu[3]=='1') {
      if (input(pin_b3)) rtu[3]='1';else rtu[3]='0';
    };
    output_low(pin_d1);
    output_low(pin_d0);
    //Si el sensor en el puerto b3 se activa, ya se llego.
    //Sensor optico de ranura bloqueado //logica negada
    //comando c -> Subir brazo robotico
    //output_high(pin_d0);//giro normal del motor dc del codo
    if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
    if (rtu[2]=='1') output_high(pin_d3);
    while (rtu[2]=='1') {
      if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
    };
    output_low(pin_d3);
    //output_low(pin_d0);
    //Si el sensor en el puerto b2 se DESACTIVA, ya se llego.
    //Sensor final de carrera DESactivado
    //comando z -> Girar base brazo robotico a izquierda
    //output_high(pin_d0);//giro normal del motor dc de la base
    if (input(pin_b1)) rtu[1]='1';else rtu[1]='0';
    if (rtu[1]=='0') output_high(pin_d5);
    while (rtu[1]=='0') {
      if (input(pin_b1)) rtu[1]='1';else rtu[1]='0';
    };
    output_low(pin_d5);
    //output_low(pin_d0);
    //Si el sensor en el puerto b1 se activa, ya se llego.
    //Sensor switch izquierdo conectado
    mtu[5]='0';//termino incializacion de brazo robotico
 };
 //---------------------------------------------------------------
 if (mtu[6]=='1') {  //comando o -> Recoger objeto
       //comando x -> Girar base brazo robotico a derecha
```

```
output_high(pin_d0);//giro inverso del motor dc de la base
if (input(pin_b0)) rtu[0]='1';else rtu[0]='0';
if (rtu[0]=='0') output_high(pin_d5);
while (rtu[0]=='0') {
  if (input(pin_b0)) rtu[0]='1';else rtu[0]='0';
};
output_low(pin_d5);
output_low(pin_d0);
//Si el sensor en el puerto b0 se activa, ya se llego.
//Sensor switch derecho conectado
//comando v -> Bajar brazo robotico
output_high(pin_d0);//giro inverso del motor dc del codo
if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
if (rtu[2]=='0') output_high(pin_d3);
while (rtu[2]=='0') {
  if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
};
output_low(pin_d3);
output_low(pin_d0);
//Si el sensor en el puerto b2 se activa, ya se llego.
//Sensor final de carrera activado
//comando b -> Cerrar gripper de brazo robotico
//output_high(pin_d0);//giro normal del motor dc del gripper
if (input(pin_b4)) rtu[4]='1';else rtu[4]='0';
if (rtu[4]=='0') output_high(pin_d1);
while (rtu[4]=='0') {
  if (input(pin_b4)) rtu[4]='1';else rtu[4]='0';
};
output_low(pin_d1);
//output_low(pin_d0);
//Si el sensor en el puerto b4 se activa, ya se llego.
//Sensor de fuerza presionado
delay_ms(1500);//espera despues de coger objeto
//comando c -> Subir brazo robotico
//output_high(pin_d0);//giro normal del motor dc del codo
if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
if (rtu[2]=='1') output_high(pin_d3);
while (rtu[2]=='1') {
  if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
};
output_low(pin_d3);
//output_low(pin_d0);
//Si el sensor en el puerto b2 se DESACTIVA, ya se llego.
//Sensor final de carrera DESactivado
//comando z -> Girar base brazo robotico a izquierda
//output_high(pin_d0);//giro normal del motor dc de la base
```

```
if (input(pin_b1)) rtu[1]='1';else rtu[1]='0';
if (rtu[1]=='0') output_high(pin_d5);
while (rtu[1]=='0') {
  if (input(pin_b1)) rtu[1]='1';else rtu[1]='0';
};
output_low(pin_d5);
//output_low(pin_d0);
//Si el sensor en el puerto b1 se activa, ya se llego.
//Sensor switch izquierdo conectado
//comando v -> Bajar brazo robotico
output_high(pin_d0);//giro inverso del motor dc del codo
if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
if (rtu[2]=='0') output_high(pin_d3);
while (rtu[2]=='0') {
  if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
};
output_low(pin_d3);
output_low(pin_d0);
//Si el sensor en el puerto b2 se activa, ya se llego.
//Sensor final de carrera activado
delay_ms(1000);//espera debido al peso del objeto
//Sube y baja el brazo para estabilización
//comando c -> Subir brazo robotico
//output_high(pin_d0);//giro normal del motor dc del codo
if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
if (rtu[2]=='1') output_high(pin_d3);
while (rtu[2]=='1') {
  if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
};
output_low(pin_d3);
//output_low(pin_d0);
//Si el sensor en el puerto b2 se DESACTIVA, ya se llego.
//Sensor final de carrera DESactivado
//comando v -> Bajar brazo robotico
output_high(pin_d0);//giro inverso del motor dc del codo
if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
if (rtu[2]=='0') output_high(pin_d3);
while (rtu[2]=='0') {
  if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
};
output_low(pin_d3);
output_low(pin_d0);
//Si el sensor en el puerto b2 se activa, ya se llego.
//Sensor final de carrera activado
delay_ms(1500);//espera fin de estabilización
//comando n -> Abrir gripper de brazo robotico
```

```
      output_high(pin_d0);//giro inverso del motor dc del gripper
      if (input(pin_b3)) rtu[3]='1';else rtu[3]='0';
      if (rtu[3]=='1') output_high(pin_d1);
      while (rtu[3]=='1') {
        if (input(pin_b3)) rtu[3]='1';else rtu[3]='0';
      };
      output_low(pin_d1);
      output_low(pin_d0);
      //Si el sensor en el puerto b3 se activa, ya se llego.
      //Sensor optico de ranura bloqueado //logica negada
      //comando c -> Subir brazo robotico
      //output_high(pin_d0);//giro normal del motor dc del codo
      if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
      if (rtu[2]=='1') output_high(pin_d3);
      while (rtu[2]=='1') {
        if (input(pin_b2)) rtu[2]='1';else rtu[2]='0';
      };
      output_low(pin_d3);
      //output_low(pin_d0);
      //Si el sensor en el puerto b2 se DESACTIVA, ya se llego.
      //Sensor final de carrera DESactivado
      //girar motor2 4 vueltas
      for (i=1;i<=8;i++) {
        output_high(pin_a2);delay_ms(400);output_low(pin_a2);
        output_high(pin_a3);delay_ms(400);output_low(pin_a3);
        output_high(pin_a4);delay_ms(400);output_low(pin_a4);
        output_high(pin_a1);delay_ms(400);output_low(pin_a1);
      };
      mtu[6]='0';//se recogio objeto y se dejo en otra posición
    };
//---------------------------------------------------------------
    if(kbhit()) {
      op=getch();
      if (op=='m') mtu[0]='1';//foco en el gripper del robot
      if (op=='r') mtu[1]='1';//Girar Motor1 sentido reloj (Motor1 a pasos)
      if (op=='t') mtu[2]='1';//Girar Motor1 contra reloj
      if (op=='y') mtu[3]='1';//Girar Motor2 sentido reloj (Motor2 a pasos)
      if (op=='u') mtu[4]='1';//Girar Motor2 contra reloj
      if (op=='p') mtu[5]='1';//inicializar brazo (HOME)
      if (op=='o') mtu[6]='1';//recoger objeto
      if (op=='z') mtu[7]='1';//Girar base a izquierda
      if (op=='x') mtu[8]='1';//Girar base a derecha
      if (op=='c') mtu[9]='1';//Subir brazo
      if (op=='v') mtu[10]='1';//Bajar brazo
      if (op=='b') mtu[11]='1';//Cerrar gripper
      if (op=='n') mtu[12]='1';//Abrir gripper
```

```
      if (op=='e'){ch='w';//Mandar estado de los sensores
        //printf("\r Sensores= ");for (i=1;i<=8;i++)printf("%c",rtu[i-1]);
        printf("\r Sensores= ");for (i=8;i>=1;i--)printf("%c",rtu[i-1]);
      };
      if (op=='q') ch=op;
      //output_bit(pin_b2,mtu[0]);
      }//if-else
    else {
      output_low(pin_a0);delay_ms(400);
      output_high(pin_a0);delay_ms(400);
      };//end-if-else
  }//end-while
  //printf ("\b\b\bOFF");
  printf ("\r OFF");
  //output_low(pin_b1);
}//end-while
}//end.
```

REFERENCIAS BIBLIOGRÁFICAS

[1] I. Lopez-Juarez, R. Rios-Cabrera, M. Peña-Cabrera, and R. Osorio-Comparan. *Learning and Fast Object Recognition in Robot Skill Acquisition: A New Method*. J.A. Carrasco-Ochoa et al. (Eds.): MCPR 2010, LNCS 6256, pp. 40–49.

[2] M.Peña, I.López, R.Osorio . *Invariant Object Recognition Robot Vision System for Assembly*. Proceedings of the Electronics, Robotics and Automotive Mechanics Conference (CERMA'06). 2006 IEEE.

[3] Mario Peña-Cabrera, Ismael López-Juárez y Reyes Rios-cabrera. *Proceso de Aprendizaje con Algoritmo Robusto para la Obtención del POSE de Objetos en Líneas de Ensamble con Robots en Tiempo Real (RT)*. Informacíon Tecnológica. 17(2). La Serena 2006, pp 61-69.

[4] M. Peña, I López-Juárez, J. Corona, K.Ordaz. *MANUFACTURA INTELIGENTE UTILIZANDO VISION PARA ROBOTS*. 2004 IFAC.

[5] M. Peña-Cabrera, H. Gómez, R. Osorio, I. López-Juárez. *Object Location in Manufacturing Cells using Artificial Vision*. Electronics, Robotics and Automotive Mechanics Conference . DOI 10.1109/CERMA.2009.78. IEEE. pp 215-219.

[6] M. Peña-Cabrera, I. López-Juárez, R. Ríos-Cabrera. *A Learning Approach for On Line Object Recognition Tasks*. Proceedings of the Fifth Mexican International Conference in Computer Science (ENC'04). 2004 IEEE

[7] Chunhua Li, Kun He, Jiliu Zhou . *Edge Detection of Image on the Local Feature*. DOI 10.1109/IITA.2008.403. IEEE. pp 326-330.

[8] Hu tao, Guo baoping. *An Improved Run-Based Algorithm for Fast and Exact Boundary Extraction*. DOI 10.1109/IIS.2009.73. IEEE. pp 240-243.

[9] Tao Hu, Baoping Guo. *Fast and Exact Boundary Extraction of High Resolution PCB Images*. ICIEA 2009. IEEE. pp 2392-2395.

[10] Shimon Edelman, Heinrich H. Biilthoff. *Modeling human visual object recognition*. 1992 IEEE. 4. pp 37-42.

[11] Ah-Hwee Tan. *Adaptive Resonance Associative Map: A Hierarchical ART system for Fast Stable Associative Learning*. 1992 IEEE. 1. pp 860-865.

[12] Si Wei Lu, Jun Shen. *ARTIFICIAL NEURAL NETWORKS FOR BOUNDARY EXTRACTION*. 1996 IEEE. pp 2270-2275.

[13] P. Chalimbaud, F. Berry. *Design of an Imaging System based on FPGA Technology and CMOS Imager*. 2004 IEEE. pp 407-411.

[14] John Cole and Larry Garey, Kenneth B. Kent. *Rapid Prototyping Projection Algorithms with FPGA Technology*. 2009 IEEE International Symposium on Rapid System Prototyping. pp 95-101.

[15] Bogdan Belean, Monica Borda, Bertrand LeGal, Raul Malutan. *FPGA Technology And Parallel Computing Towards Automatic Microarray Image Processing*. 2011 IEEE. pp 607-610.

[16] Przemyslaw Brylski, Michal Strzelecki. *PARALLEL DIGITAL IMAGE PROCESSOR IMPLEMENTED IN FPGA TECHNOLOGY*.

[17] Eddie Angel Sobrado Malpartida. *SISTEMA DE VISIÓN ARTIFICIAL PARA EL RECONOCIMIENTO Y MANIPULACIÓN DE OBJETOS UTILIZANDO UN BRAZO ROBOT. Tesis.* Perú. 2003.

[18] Luis Alejandro Mora. *INTRODUCCIÓN AL LENGUAJE DE DESCRIPCIÓN DE HARDWARE (VHDL)*. Venezuela. Mayo 2009.

[19] Sin autor. *FPGA (CAMPO DE MATRIZ DE PUERTAS PROGRAMABLES)*. Acceso al documento a través de cualquier navegador por internet.

[20] Juan Mario Peña Cabrera. *APRENDIZAJE Y RECONOCIMIENTO INVARIANTE DE OBJETOS EN ENSAMBLE CON ROBOTS EMPLEANDO REDES NEURONALES ARTIFICIALES. Tesis.* México. Diciembre 2005.