UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Implementación de un sistema de reconocimiento de formas en FPGA

Proyecto de investigación

José Enrique Camacho Zambrano

Ingeniería Electrónica

Trabajo de titulación presentado como requisito
para la obtención del título de
Ingeniero Electrónico

Quito, 21 de mayo de 2018

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

COLEGIO DE CIENCIAS E INGENIERÍAS

HOJA DE CALIFICACIÓN
DE TRABAJO DE TITULACIÓN

Implementación de un sistema de reconocimiento de formas en FPGA

José Enrique Camacho Zambrano

Calificación:

Nombre del profesor, Título académico Luis Miguel Prócel, Ph.D.

Firma del profesor _____

Quito, 21 de mayo de 2018

DERECHOS DE AUTOR

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Firma del estudiante: _____

Nombres y apellidos: José Enrique Camacho Zambrano

Código: 00117803

Cédula de Identidad: 1717410854

Lugar y fecha: Quito, 21 de mayo de 2018

AGRADECIMIENTOS

Quiero expresar mi más profundo y sincero agradecimiento a mis padres, quienes me apoyaron incondicionalmente durante toda mi carrera universitaria y me dieron las fuerzas necesarias para superar todas las adversidades que se presentaron a lo largo del camino.

Agradecer a mi tutor de tesis Luis Miguel Prócel por toda la ayuda brindada y hacer que me involucre en el mundo de la visión por computadora. Además quiero agradecer la ayuda Juan Romero en el desarrollo del presente trabajo y a mis compañeros.

RESUMEN

El presente proyecto propone el desarrollo en MATLAB de un algoritmo de detección de figuras, así como su posterior implementación en hardware por medio de una tarjeta FPGA usando Verilog. El algoritmo de detección se basa en los conceptos sobre los momentos de Hu, los cuales permiten caracterizar una imagen sin importar su transformación de similitud. Además se presenta el algoritmo de reducción de ruido, Non-Local Means, el cual expone el concepto de la distancia euclidiana utilizada para poder reconocer una imagen logo o figura dentro de una imagen de prueba.

El trabajo presenta las etapas de desarrollo en MATLAB y la optimización del algoritmo con el fin de poder sintetizar el sistema en hardware. Los resultados de reconocimiento presentan un correcto funcionamiento del algoritmo de detección de figuras tanto en el ámbito de software como en FPGA. De igual forma, se describen las ventajas de la implementación en hardware sobre la implementación en software como es el caso de la rapidez y el paralelismo al momento de procesar una señal.

Palabras clave: Visión por computadora, Momentos de Hu, algoritmo Non-Local Means, transformaciones de similitud, Verilog.

ABSTRACT

The present Project proposes the development in MATLAB of a figure detection algorithm, as well its subsequent implementation in hardware in a FPGA using Verilog. The detection algorithm is based on the concepts of Hu moments, which allows to characterize an image regardless its similarity transformation. In addition, the noise reduction algorithm Non-Local Means is present, which show the concept of Euclidean distance used to recognize an imagen logo or figure in a test image.

The project present the stages of development in MATLAB and the optimization of the algorithm in order to be ablo to synthesize the system in hardware. The recognition results show a correct functioning of the figure detection algorithm in both, software and FPGA. Similarly, it describes the advantages of the implementation in hardware over the implementation in software, such as the case of speed and parallelism in processing a signal.

Key words: Computer vision, Hu moments, Non-Local Menas algorithm, similarity transformtions, Verilog.

CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Desde la aparición de la computación digital, se han desarrollado grandes avances en el área de visión por computadora la cual trata de describir el mundo que nos rodea a través de imágenes analizando sus propiedades tales como iluminación, distribución de color o forma (Szeliski, 2010). Actualmente, la visión por computadora presenta varias aplicaciones en nuestra vida diaria las cuales incluyen: reconocimiento de objetos, seguidores de movimiento, detección de rostros, captura de movimiento, entre otros. Sin embargo, desarrollar un algoritmo robusto que presente un margen de error mínimo es sumamente complicado debido a la gran cantidad de variables que se presentan a la hora de reconocer un objeto y a la gran cantidad de información que se debe procesar.

Hoy en día la implementación de reconocimiento de imágenes se lo realiza vía software, en donde no se considera los elementos que hay detrás como el manejo de datos, el uso de memoria o el manejo de operaciones aritméticas. La implementación en hardware toma en cuenta todos los mencionados anteriormente con el afán de reducir la complejidad matemática y optimizar el algoritmo para utilizar la menor cantidad de recursos posibles.

El uso de un FPGA (Field Programmable Gate Array) en el ámbito de visión computacional presenta grandes beneficios gracias al procesamiento en paralelo de una misma señal, la versatilidad de poder manejar los datos y la interpretación del procesamiento de imágenes en un alto nivel de abstracción creando algoritmos más sofisticados (Crockett et al., 2014). Por esta razón se puede crear un sistema modular integrado que realice el procesamiento de imágenes y reconocimiento de formas. El presente trabajo tiene como objetivo realizar la implementación en hardware de un sistema de reconocimiento de formas utilizando los conceptos teóricos sobre los momentos de Hu y el algoritmo Non Local Means los cuales serán detallados en la siguiente sección.

METODOLOGÍA

Descriptores de imágenes

Permiten caracterizar y obtener los patrones de una imagen analizando sus propiedades como iluminación, distribución de color o forma. Un ejemplo claro es el algoritmo Scale Invariant Feature Transform(SIFT) el cual propone un método de caracterización de imágenes el cual es invariante a cambios de iluminación, ruido y transformaciones de similitud tales como rotación, traslación y escalamiento (Lowe, 2004).

Momentos Espaciales

Los momentos espaciales son características intrínsecas de una imagen. En esta sección se repasaran los conceptos teóricos sobre los momentos espaciales geométricos, centroide y momentos de inercia los cuales en su conjunto dan origen a los momentos invariantes de Hu.

Momento espacial geométrico

El momento (m, n) de probabilidad de densidad conjunta $p(x, y)$ es definido como:

$$M(m,n) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^m y^n p(x,y)dx\,dy \tag{1.1}$$

El momento central está dado por:

$$U(m,n) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (x-\bar{x})^m (y-\bar{y})^n p(x,y)dx\,dy \tag{1.2}$$

Donde $\bar{x}, \bar{y}$ son los promedios marginales de $p(x, y)$. Estos conceptos fueron aplicados para el análisis de forma por Hu (Pratt, 2007). Los conceptos presentados fueron extendidos hacia una imagen discreta $F(j, k)$ reemplazando la sumatoria continua por una sumatoria espacial, por lo tanto se define el momento espacial geométrico de una imagen como:

$$M_U(m,n) = \sum_{j=1}^{J}\sum_{k=1}^{K} \left(x_j\right)^m (y_k)^n F(j,k) \tag{1.3}$$

Donde $F(j,k)$ da el valor de intensidad de un pixel dadas las coordenadas (j,k), j es el número de filas de la imagen mientras que k es el número de columnas. Por lo tanto el número total de pixeles es el resultado de $j \times k$. Los denominados momentos espaciales de bajo-orden se presentan a continuación:

$$M_U(0,0) = \sum_{j=1}^{J} \sum_{k=1}^{K} F(j,k) \tag{1.4}$$

$$M_U(0,1) = \sum_{j=1}^{J} \sum_{k=1}^{K} (y_k)^n F(j,k) \tag{1.5}$$

$$M_U(1,0) = \sum_{j=1}^{J} \sum_{k=1}^{K} (x_j)^m F(j,k) \tag{1.6}$$

La ecuación de orden cero (1.4) es denominada superficie de imagen, la ecuación (1.5) se le conoce como *momento fila de primer orden* mientras que la ecuación (1.6) es el *momento columna de primer orden*. En base a los momentos espaciales mencionados se determina el centroide o centro de gravedad de la imagen:

$$\bar{x} = \frac{M(1,0)}{M(0,0)} \tag{1.7}$$

$$\bar{y} = \frac{M(0,1)}{M(0,0)} \tag{1.8}$$

A partir de centro de gravedad se define el momento espacial central como:

$$U(m,n) = \sum_{j=1}^{J} \sum_{k=1}^{K} (x - \bar{x})^m (y - \bar{y})^n p(x,y) dx\, dy \tag{1.9}$$

Los momentos de inercia de segundo orden de la imagen están definidos por las siguientes expresiones:

$$U(0,2) = \sum_{j=1}^{J}\sum_{k=1}^{K}(x-\bar{x})^m(y-\bar{y})^n p(x,y)dx\,dy \tag{1.10}$$

$$U(2,0) = \sum_{j=1}^{J}\sum_{k=1}^{K}(x-\bar{x})^m(y-\bar{y})^n p(x,y)dx\,dy \tag{1.11}$$

$$U(1,1) = \sum_{j=1}^{J}\sum_{k=1}^{K}(x-\bar{x})^m(y-\bar{y})^n p(x,y)dx\,dy \tag{1.12}$$

Donde las ecuaciones (1.10), (1.11) y (1.12) son el momento de inercia de fila, momento de inercia de columna y el momento de inercia fila-columna respectivamente. A partir de estas ecuaciones se derivan los 7 momentos de Hu.

Momentos de Hu

Los 7 momentos de Hu son proyecciones invariantes a transformaciones de similitud tales como traslación, rotación y escalamiento (Pratt, 2007). Hu propuso la normalización de los momentos centrales a partir de la siguiente expresión:

$$V(m,n) = \frac{U(m,n)}{[M(0,0)]^{\propto}} \tag{1.13}$$

Donde:

$$\propto = \frac{\boldsymbol{m+n}}{\boldsymbol{2}} + \mathbf{1} \tag{1.14}$$

Para $m + n = 2,3, \ldots$

La normalización de los momentos centrales dio origen a siete momentos espaciales descritos por Hu a través de las siguientes expresiones:

$$h_1 = V(2,0) + V(0,2) \tag{1.15}$$

$$h_2 = [V(2,0) - V(0,2)]^2 + 4[V(1,1)]^2 \tag{1.16}$$

$$h_3 = [V(3,0) - 3V(1,2)]^2 + [V(0,3) - 3V(2,1)]^2 \tag{1.17}$$

$$h_4 = [V(3,0) + V(1,2)]^2 + [V(0,3) - V(2,1)]^2 \tag{1.18}$$

$$\begin{aligned} h_5 = {} & [V(3,0) - 3V(1,2)][V(3,0) + V(1,2)][[V(3,0) + V(1,2)]^2 \\ & -3[V(0,3) + V(2,1)]^2] + [3V(2,1) - V(0,3)][V(0,3) \\ & +V(2,1)][3[V(3,0) + V(1,2)]^2 - [V(0,3) + V(2,1)]^2] \end{aligned} \tag{1.19}$$

$$\begin{aligned} h_6 = {} & [V(2,0) - V(0,2)][[V(3,0) + V(1,2)]^2 - [V(3,0) + V(1,2)]^2] \\ & +4V(1,1)[V(3,0) + V(1,2)][V(0,3) + V(2,1)] \end{aligned} \tag{1.20}$$

$$\begin{aligned} h_7 = {} & [3V(2,1) - V(0,3)][V(3,0) + V(1,2)] \\ & [[V(3,0) + V(1,2)]^2 - 3[V(0,3) + V(2,1)]^2] \\ & +[3V(1,2) - V(3,0)][V(0,3) + V(2,1)][3[V(3,0) + V(1,2)]^2 \\ & -[V(0,3) + V(2,1)]^2] \end{aligned} \tag{1.21}$$

Algoritmo Non Local Means

Non Local Means es un algoritmo utilizado en la eliminación de ruido de una imagen el cual está definido por la fórmula:

$$NL[u](x) = \frac{1}{C(x)} \int_{\Omega} e^{-\frac{(G_a * |u(x+.) - u(y+.)|^2)(0)}{h^2}} u(y)\, dy \tag{1.22}$$

Donde G_a es el núcleo Gaussiano con una desviación estándar a, h es un parámetro de filtración y $C(x)$ es un factor de normalización dado por la expresión:

$$C(x) = \int_{\Omega} e^{-\frac{(G_a * |u(x+.) - u(z+.)|^2)(0)}{h^2}} dz \tag{1.23}$$

La ecuación (1.22) del algoritmo se basa en el uso de la distancia euclidiana entre pixeles representado por la expresión $|u(x+.) - u(y+.)|^2$, donde $u(x, y)$ es el valor del pixel de una imagen dada sus coordenadas x, y (Baudes, Coll y Morel, 2005).

Desarrollo del algoritmo de detección de formas en MATLAB

En base al algoritmo Non Local Means, se analizó el uso de la distancia euclidiana para poder desarrollar un algoritmo de detección de formas a partir de una imagen logo, $I_L(x)$, la cual contiene la figura a encontrar en una imagen de prueba, $I_P(y)$. Utilizando los descriptores propuestos por Hu, podemos caracterizar la imagen logo y de esta manera analizar la imagen de prueba a través de un proceso de ventaneo, donde se escanea la imagen $I_P(y)$ con la finalidad de encontrar la figura descrita por el logo sin importar la transformación afín que esta pudiera tener.

Este procedimiento conlleva en modificar la ecuación (1.22) de la siguiente forma:

$$NLM(x) = \frac{1}{C(x)} \sum_{y \, \epsilon \, J} e^{\left(\frac{D\left(d^I(x), d^J(y)\right)}{h^2} \right)} J(y) \tag{1.24}$$

Donde $d^I(x)$ es el descriptor calculado en la imagen logo, $d^J(y)$ es el descriptor calculado para la ventana de la imagen a analizar y $D\left(d^I(x), d^J(y)\right)$ es la distancia entre ambos descriptores. Sin embargo, con el objetivo de simplificar la ecuación para su implementación en hardware se mantuvo solo la distancia entre los descriptores de ambas imágenes. Por lo tanto la ecuación final queda de la siguiente forma:

$$NLM(x) = \sum_{y \, \epsilon \, J} D\left(d^I(x), d^J(y)\right) \tag{1.25}$$

En un caso perfecto la distancia euclidiana entre los momentos de Hu de la imagen de logo y la ventana que contiene la figura a encontrar seria cero, indicando el reconocimiento de la figura. Sin embargo, existen diversos factores como el tamaño de las ventanas o la transformación de similitud que afectan mínimamente la caracterización al momento de analizar las ventanas de la imagen de prueba. Por esta razón, se estableció un valor de threshold mínimo el cual permite el reconocimiento de la figura, es decir, si la distancia entre los

descriptores es menor o igual al valor de threshold, significa que en la ventana que cumple esta condición se encuentra la figura a reconocer.

Al realizar el desarrollo del algoritmo en MATLAB se optó por crear dos funciones denominadas *Momentos_Hu_Im* y *Logo_detection*. La función *Momentos_Hu_Im* calcula los 7 momentos de Hu para una imagen dada, mientras que la función *Logo_detection* se encarga de realizar el ventaneo de la imagen de prueba y detectar la figuras similares a la imagen de logo. La descripción detallada del funcionamiento de ambas funciones se muestra a continuación en el siguiente pseudocódigo:

Algoritmo de cómputo Momentos de Hu:

$$Hu[7] = Momentos\_Hu\_Im(imagen)$$

1. Ingresada la matriz Im de una imagen RGB se transforma a escala de grises en dos dimensiones.
2. La imagen a escala de grises Im se transforma a formato $double$ para realizar operaciones aritméticas.
3. Se obtiene el tamaño (M, N) de la imagen Im
4. Se calculan los momentos espaciales $M00, M10, M01$
5. Se calcula el centro de gravedad xjp, ykp
6. Se calculan los momentos de inercia $U11, U12, U21, U20, U02, U30, U03$
7. Se normalizan los momentos centrales para luego calcular los 7 momentos de Hu[7].

Algoritmo de detección de figuras:

$$Imagen\_final = Logo\_detection(wH, wW, saltoH, saltoW, F1, F2)$$

1. Ingresada la imagen $F1$, $imagen\ de\ logo$, se calculan los momentos de Hu utilizando la función $Hu1[7] = Momentos\_Hu\_Im(F1)$
2. Ingresada la imagen $F2$, $imagen\ de\ prueba$, se obtiene el tamaño (M, N) de la imagen $F2$.

3. Se crear una imagen $F3$, $imagen\,final$, en escala de grises de la imagen $F2$.
4. Se determina la dimensión de la ventana, $F2Window$, para la imagen $F2$ dada por las variables wH (alto) y wW (ancho).
5. Se calculan los momentos de Hu de la ventana mediante la función $Hu2[7] = Momentos\_Hu\_Im(F2Window)$
6. Se obtiene la distancia euclidiana, D, entre los descriptores Hu1 y Hu2.
7. Se comprara la distancia obtenida con un valor de $thershold$, al no cumplirse la condición se obtiene una nueva ventada desplazada por el valor de $saltoH$ y $saltoW$.
8. Al cumplirse la condición anterior se detiene el proceso de ventaneo y dadas las coordenadas de la ventana $(y1, y2)$ se sobrescribe en la imagen $F2$ el valor de los pixeles correspondientes a la ventana que contiene la figura con un valor de cero.
9. Se invierte el valor de los pixeles correspondientes a la ventana en la imagen $F3$. Se reanuda el proceso de ventaneo hasta escanear la totalidad de la imagen.
10. Se muestra el resultado final generado en la imagen $F3$.

En el algoritmo $Logo\_detection$, literal 8, se realiza una escritura en la imagen de prueba, cambiando los pixeles correspondientes a la ventana a un valor de cero con la finalidad de borrar la figura encontrada y evitar que en el próximo salto de la ventana se vuelva a reconocer. Por otra parte, en el literal 9 se invierten los valores de los pixeles de la imagen final con el objetivo de mostrar la posición donde se encuentran las figuras reconocidas.

Implementación del algoritmo de reconocimiento en hardware

El objetivo de la implementación es poder sintetizar el algoritmo de detección en hardware y mostrar el resultado de la imagen final en una pantalla a través de un puerto VGA.

Para la implementación del algoritmo de reconocimiento se utilizó una tarjeta FPGA ZYBO 7000. La estrategia a tomar para poder realizar la implementación del algoritmo es a través de la creación de un sistema modular, es decir, desarrollar módulos que realicen una tarea específica en base al código desarrollado en MATLAB. Para realizar la escritura, síntesis e implementación se utilizó la herramienta de diseño XILINX VIVADO. El lenguaje de descripción de hardware utilizado para realizar la programación de los módulos es Verilog (Harris, Harris, 2007).

En primer lugar, se realizó la implementación de la memoria de logo la cual almacena la figura que se va a reconocer. Debido a la capacidad de la unidad de procesamiento lógico en la tarjeta ZYBO fue necesario reducir el tamaño de las imágenes con el fin de poder sintetizar las diferentes memorias que almacenan la imagen de logo, la imagen de prueba y la imagen final. Para la memoria logo se realizó el diseño de una memoria solo de lectura (ROM) presentada a continuación:

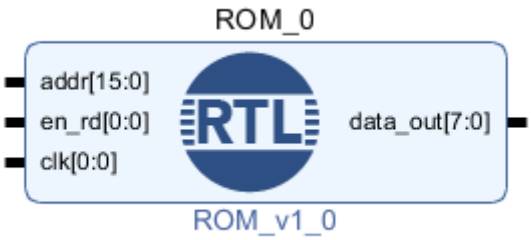

Figura 1. Diseño de bloque de la memoria ROM de logo

En la Figura 1 podemos apreciar las conexiones necesarias para el funcionamiento de la memoria donde $addr$ es la dirección de la memoria en 16 bits, en_rd es la señal de activación y clk es la señal de reloj. En este caso la imagen logo tiene un tamaño de 60×70 pixeles, por lo tanto la capacidad de almacenamiento de la memoria es de 4200 bytes. La salida del módulo, $data\_out$, es de 8 bits debido a que cada pixel de la imagen es representada en escala de grises por lo que el rango de valor de un pixel varía de 0 a 255.

Posteriormente, se realizó la implementación de la función *Momentos_Hu_Im.* Para su implementación se diseñó un módulo específico para realizar el cálculo de cada variable que se presenta en la función. Durante esta etapa se realizó modificaciones al código original de MATLAB con el fin de facilitar su implementación en la tarjeta FPGA, sin afectar el funcionamiento del código. Dichos cambios serán puntualizados en la descripción de cada módulo.

Los módulos a implementar para poder calcular los momentos de Hu son: M00, M01, M10, xjp_ykp, U11, U20, U02, V_Hu11, V_Hu02, V_Hu20, Hu1, Hu2_1, Hu2_2 y Hu2. El siguiente esquema muestra el diagrama de bloque de los módulos M00, M01, M10 y xjp_ykp:

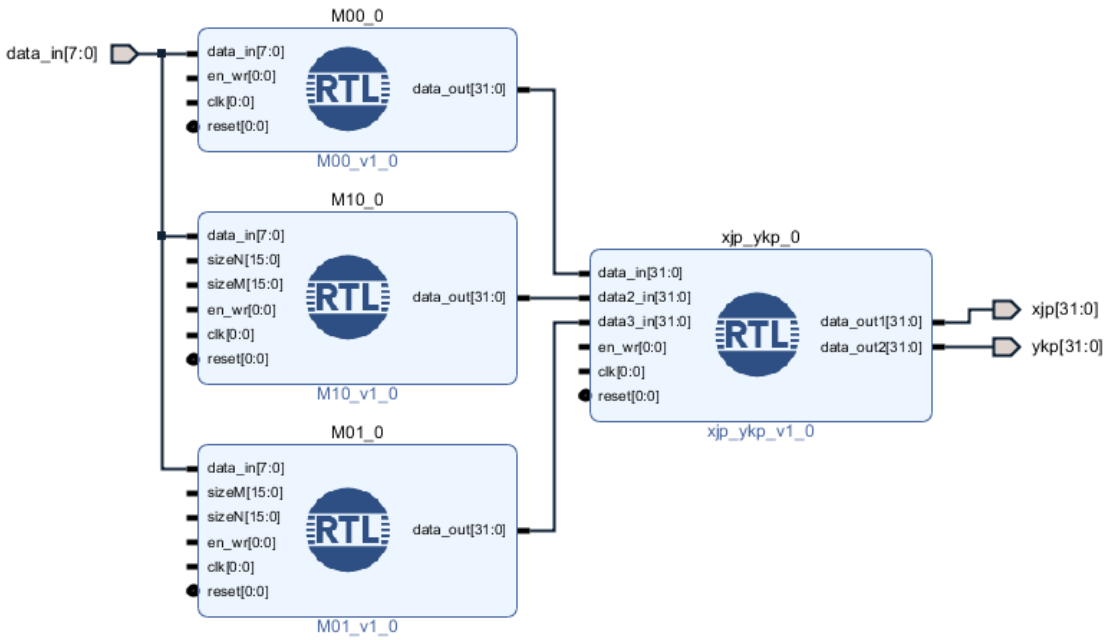

Figura 2. Diseño de bloque del módulos M00, M01, M10 y xjp_ykp

Donde $data\_in$ es la entrada de datos provenientes de la memoria, $sizeM$ - $sizeN$ son las dimensiones de la imagen y en_wr son las señales de activación de los módulos mostrados. En este caso el módulo M00_0 corresponde a la implementación de la ecuación (1.4), el módulo M01_0 a la ecuación (1.5), el módulo M01_0 a la ecuación (1.6) y el módulo xjp_ykp a las ecuaciones (1.7) y (1.8). El módulo xjp_ykp presenta la operación de división aritmética, sin embargo, en Verilog el resultado de la división entre dos valores es un entero mientras que en

el código original de MATLAB se presenta el resultado con decimales. La no incorporación de decimales para obtener los valores de xjp y ykp, representa una reducción en la precisión al momento de realizar el cálculo de los momentos de Hu. Sin embargo, no genera un cambio significativo en el funcionamiento del algoritmo.

El siguiente esquema muestra la implementación de los módulos U11, U20, U02, V_Hu11, V_Hu02 y V_Hu20:

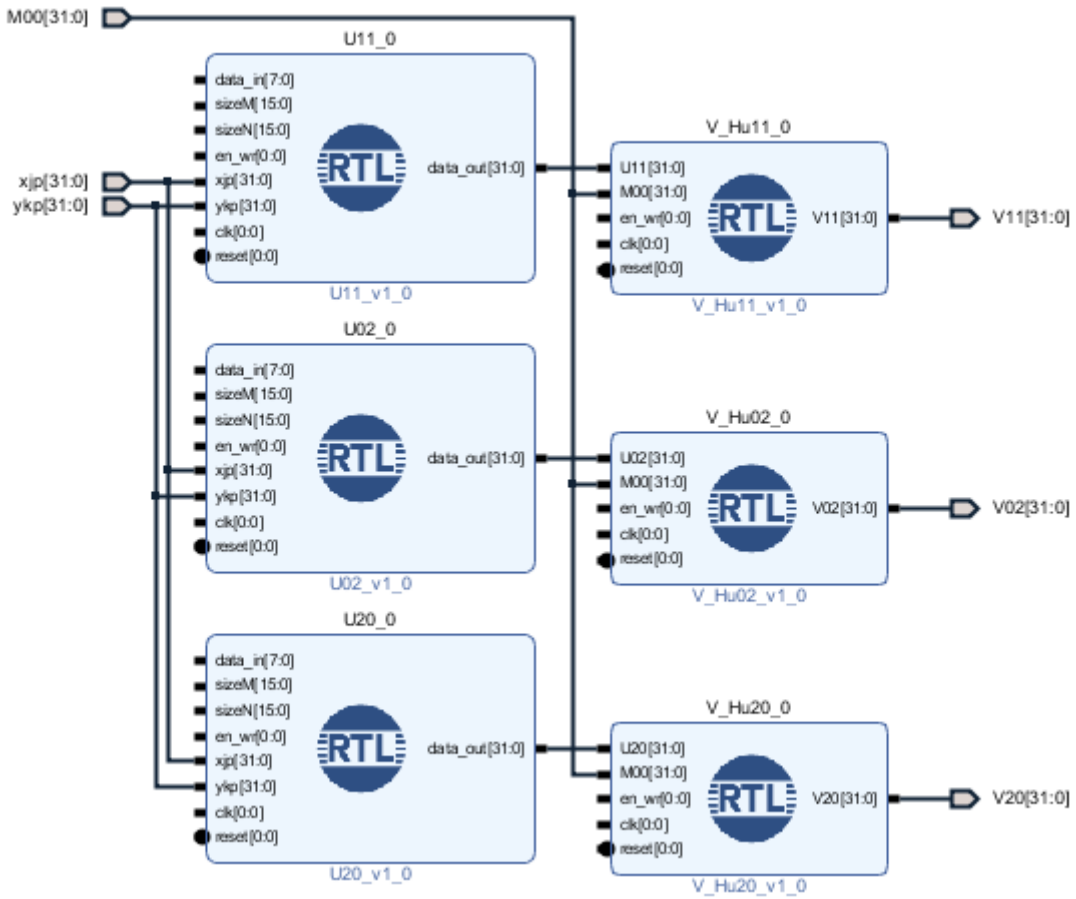

Figura 3. Diseño de bloque de los módulos U11, U20, U02, V_Hu11, V_Hu02 y V_Hu20

Donde xjp corresponde a la salida $data\_out1$ y ykp a la salida $data\_out2$ del módulo xjp_ykp presentado en la Figura 3. En este caso el módulo U11_0 corresponde a la implementación de la ecuación (1.10), el módulo U02_0 a la ecuación (1.11), el módulo U20_0 a la ecuación (1.12). A partir del tercer momento de Hu (1.17) se utilizan los momentos de

inercia U12, U21, U30 y U03. No obstante, se decidió no realizar la implementación de dichos momentos ya que como se puede apreciar en la Tabla 1, desde el tercer momento de Hu, los valores calculados no son significativos al instante de evaluar la distancia euclidiana.

Los módulos V_Hu11, V_Hu02 y V_Hu20 corresponden a la implementación de la ecuación (1.13) para cada caso. Dichos módulos presentaron dificultades en su implementación debido a la naturaleza de la ecuación (1.13) donde se debe realizar una división aritmética. En este caso es necesario el uso de decimales por lo que se implementó un módulo de división de punto fijo (Skalicky, 2011). El módulo de división presenta una salida de 32 bits donde los 15 bits se usan para representar los decimales de la división y los 17 bits restantes representan el resultado entero.

Otra dificultad encontrada al momento de realizar la implementación de la ecuación (1.13) fue el exponencial de M00. Debido a que solo se está trabajando en 32 bits, el resultado de la operación exponencial sobrepasa el valor máximo que se puede representar con este número de bits. En consecuencia, se cambió la operación exponencial por una multiplicación con un factor de 1000. Este cambio modifica el valor final de los momentos de Hu y por consiguiente el valor de threshold establecido, a pesar de ello, no altera el funcionamiento del algoritmo de reconocimiento. Todos los cambios propuestos fueron probados y comprobados en MATLAB. Por último, el siguiente esquema muestra los bloques finales correspondientes a los módulos Hu1, Hu2_1, Hu2_2 y Hu2:

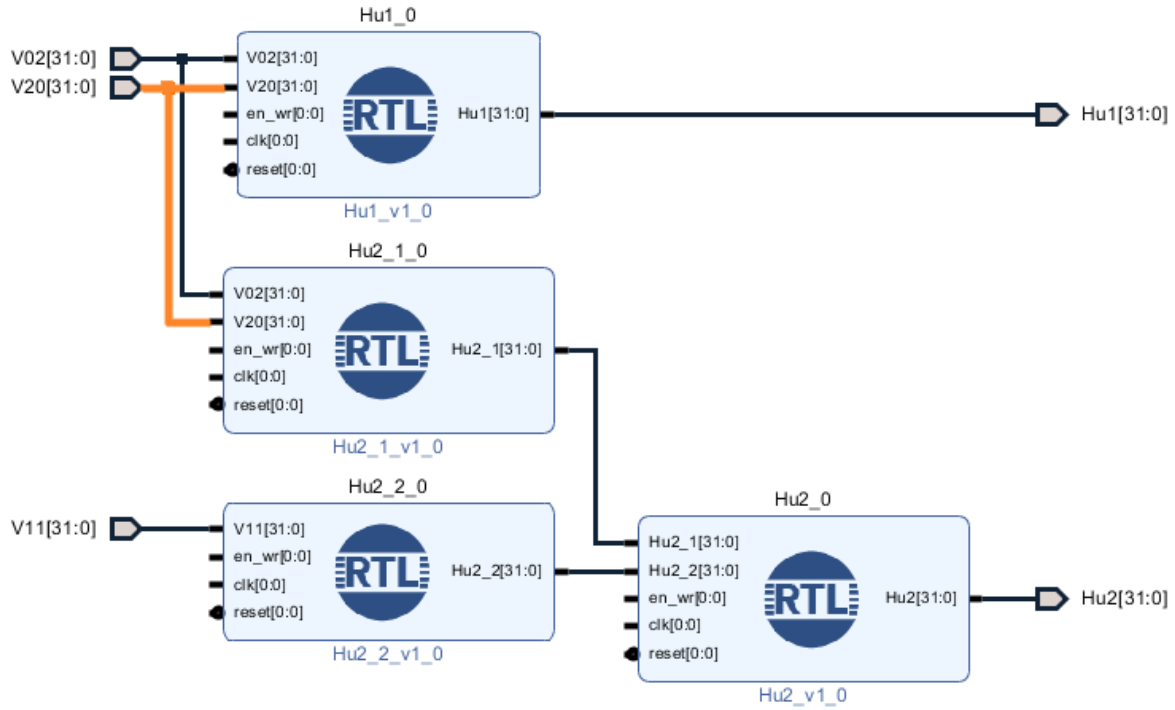

Figura 4. Diseño de bloque de los módulos Hu1, Hu2_1, Hu2_2 y Hu2.

En la Figura 4 se observa el módulo Hu1_0 que calcula el valor del primer momento de Hu correspondiente a la ecuación (1.15) mientras que el módulo Hu2_0 corresponde a la implementación de la ecuación (1.16) que es el valor del segundo momento de Hu. El módulo Hu2_0 necesita de dos módulos anteriores, Hu2_1_0 y Hu2_2_0, que ejecuten las operaciones $[V(2,0) - V(0,2)]^2$ y $4[V(1,1)]^2$ respectivamente. A causa de que en esta etapa se está trabajando con números en punto fijo, fue necesario la implementación de un módulo de multiplicación en punto fijo (Skalicky, 2011), para realizar las operaciones exponenciales antes descritas. El siguiente paso es desarrollar los módulos de control los cuales consisten en una máquina de estados que se encargan de pedir los datos a las memorias para ser enviados los módulos antes expuestos para realizar el cálculo de los momentos de Hu. El primer módulo de control creado se denominó *control_1* cuyo propósito es calcular los momentos de Hu de la imagen de logo almacenada en la memoria ROM. Para esto se diseñó una máquina de estados con un total de 7 estados los cuales se representan en el siguiente diagrama:

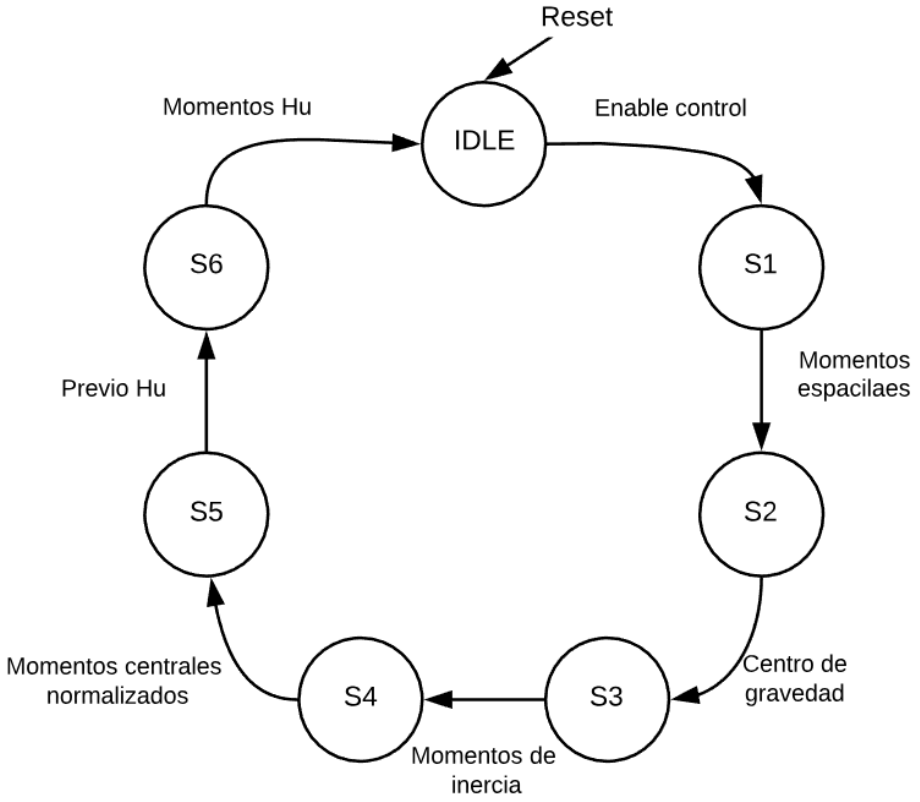

Figura 5. Diagrama de estados del módulo *control_1*

El funcionamiento de cada estado se describe en la siguiente tabla:

Tabla 1. Descripción de los estados del módulo *control_1*

Estado	**Proceso**
IDLE	Estado de espera, no se activa ningún módulo
1	Activación y envió de datos de la memoria ROM y activación los módulos M00, M01, M10
2	Activación del módulo xjp_ykp
3	Envío de datos de la memoria ROM y activación de los módulos U11, U20, U02
4	Activación de los módulos V_Hu11, V_Hu02, V_Hu20
5	Activación de los módulos Hu2_1, Hu2_2
6	Activación de los módulos Hu1, Hu2
RESET	Reinicia los módulos a su estado inicial

La activación secuencial de los módulos permite que los mismos entren en funcionamiento una vez que los datos necesarios para ejecutar las operaciones estén listos con el fin de optimizar los procesos y evitar errores al momento del caculo. De la misma forma se realizó el diseño de un módulo de control denominado *control_2*, el cual realiza el cálculo de los momentos de Hu de las ventanas para la imagen de prueba. En este caso se utilizaron dos memorias tipo RAM (Altera Corporation, 2010) las cuales almacenan la imagen de prueba, donde la primera memoria es aquella que va a ser analizada mientras que la segunda memoria contiene la imagen que va a ser mostrada donde el valor de los pixeles es invertido al momento de reconocer una figura. Diseño de bloque de la memoria RAM se presenta a continuación el cual presentada a continuación:

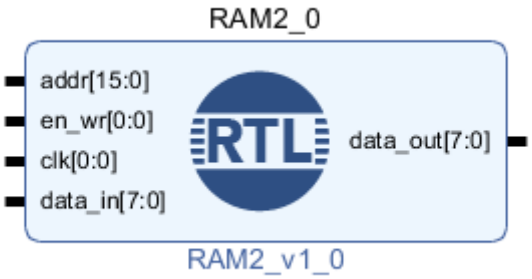

Figura 6. Diseño de bloque memoria RAM

A diferencia de la memoria ROM, la memoria RAM presenta la entrada $data\_in$ la cual envía el dato que se va a escribir. En este caso la imagen de prueba a utilizar tiene un tamaño de 155×155, por lo tanto la capacidad de la memoria es 24025 bytes. El módulo *control_2* tiene un total de 9 estados representados en el siguiente diagrama:

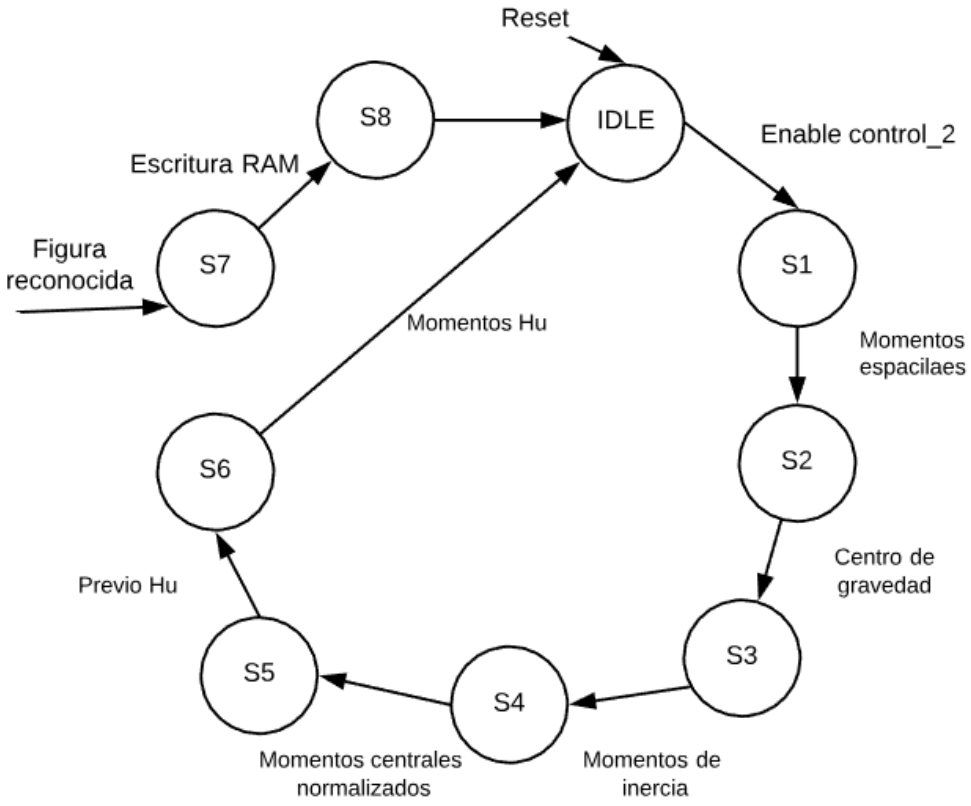

Figura 7. Diagrama de estados del módulo *control_2*

El funcionamiento de cada estado se describe en la siguiente tabla:

Tabla 2. Descripción de los estados del módulo *control_2*

Estado	**Proceso**
IDLE	Estado de espera, no se activa ningún módulo
1	Activación y envió de datos de la memoria RAM1 y activación los módulos M00, M01, M10
2	Activación del módulo xjp_ykp
3	Envío de datos de la memoria RAM1 y activación de los módulos U11, U20, U02
4	Activación de los módulos V_Hu11, V_Hu02, V_Hu20
5	Activación de los módulos Hu2_1, Hu2_2
6	Activación de los módulos Hu1, Hu2
7	Activación de la señal de escritura para la memoria RAM1 y RAM2
8	Escritura de las memorias RAM1 y RAM2
RESET	Reinicio de los módulos a su estado inicial

El estado 7 y 8 solo es alcanzado cuando existe el reconocimiento de la figura de logo. Estos estados son controlados por un módulo general denominado *global* el cual contiene los módulos de control antes descritos y realiza la función de ventaneo para el módulo control_2. Además realiza el reconocimiento de las figuras al analizar los momentos de Hu provenientes de ambos módulos de control. El módulo global presenta 7 estados los cuales están representados en el siguiente diagrama:

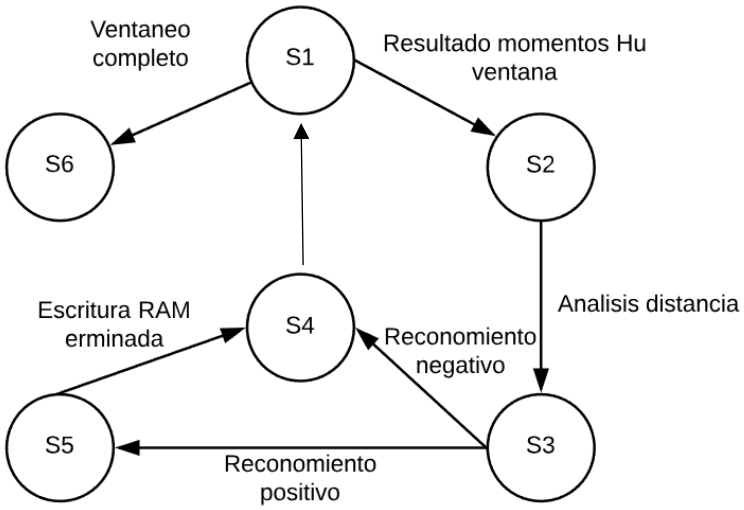

Figura 8. Diagrama de estados del módulo *global*

El funcionamiento de cada estado se describe a continuación en la siguiente tabla:

Tabla 3. Descripción de los estados del módulo *global*

Estado	**Proceso**
1	Activación de los módulos *control_1* y *control_2*. Si el proceso de ventaneo se ha completado da paso al estado 6.
2	Análisis de la distancia euclidiana entre los momentos de Hu de la imagen de prueba y la imagen de la ventana.
3	Si el reconocimiento es negativo se realiza el salto de columna para la siguiente ventana dando paso al estado 4. Si el reconocimiento es positivo se activa la señal de escritura en la memoria RAM y paso al estado 5.
4	Activa la señal de RESET para el módulo *control_2* e ingresa las nuevas coordenadas de la ventana.
5	Espera hasta que los datos sean escritos en la memoria RAM y da paso al estado 4.
6	Activación de los módulos VGA.

Implementación de los módulos VGA

Con la finalidad de poder proyectar el resultado final de la imagen, se realizó la implementación de un módulo VGA (GitHub, 2017) donde se transforma la información de 8 bits de cada pixel de la imagen en un vector de 6 bits. En consecuencia, se escogió los 6 bits más significativos de cada pixel. La resolución del módulo VGA es de 640×480 y es controlado por un módulo *Controller*, el cual se encarga de enviar los datos de la memoria RAM2 al módulo VGA para que la imagen sea mostrada en pantalla. El diseño final de la implementación se representa en el siguiente diagrama de bloques:

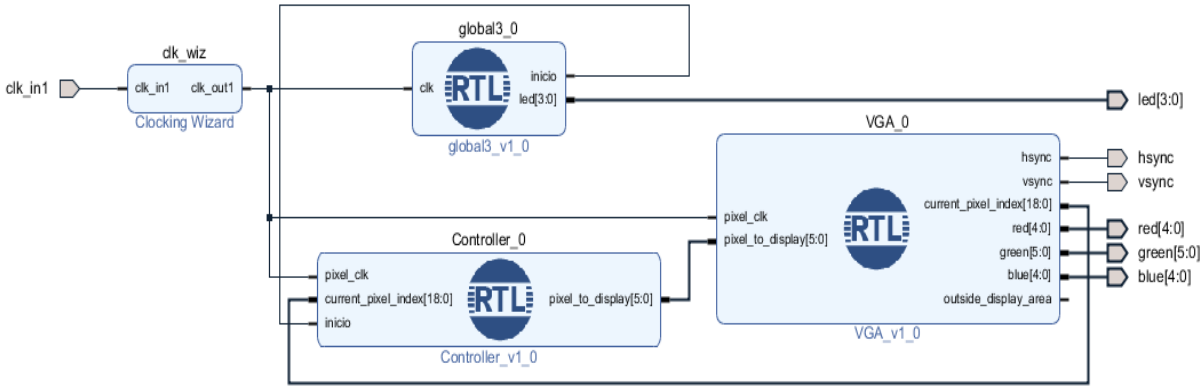

Figura 9. Diagrama de bloque implementación final

La frecuencia de reloj utilizada en la implementación es de 25 MHz. Este valor fue escogido debido al funcionamiento del módulo VGA el cual opera a dicha frecuencia (Digilent, 2014). Como se puede apreciar el modulo final presenta los puertos, hsync, vsync, red, green y blue que envían las señales al VGA y además se añadió un puerto para las luces led de la tarjeta. El uso de los leds es para poder mostrar el binario el número de figuras que detecto el sistema.

RESULTADOS

Resultados en MATLAB

La figura a reconocer al momento de realizar las pruebas en MATLAB fue la figura de un corazón, teniendo como referencia el análisis hecho por Pratt (Pratt, 2007), donde la figura que se caracteriza con los momentos de Hu es la misma. Por lo tanto, la imagen de logo para utilizar en las pruebas de reconocimiento es la siguiente:

Figura 10. Imagen Logo con figura de corazón de tamaño 146×142 pixeles

Los momentos de Hu de la imagen logo calculados en MATLAB se muestran a continuación en la siguiente tabla:

Tabla 4. Momentos de Hu Imagen Logo

Momento de Hu	**Valor**
h_1	$6.6320\ e^{-4}$
h_2	$8.2234\ e^{-10}$
h_3	$4.1995\ e^{-11}$
h_4	$3.7752\ e^{-13}$
h_5	$-1.4948\ e^{-24}$
h_6	$-1.0782\ e^{-17}$
h_7	$-4.9346\ e^{-26}$

La primera imagen de prueba en la cual se realizó la detección del logo es la siguiente:

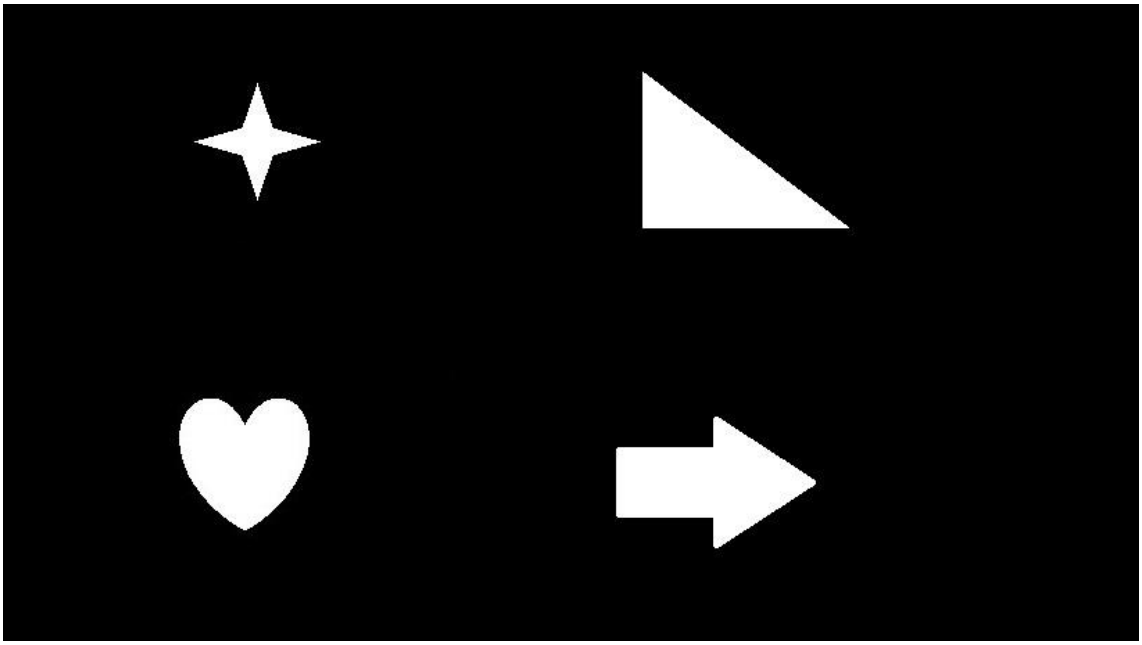

Figura 11. Imagen de prueba con varias figuras de tamaño 460×819 pixeles

Debido a la naturaleza de la función *Logo_detection* es necesario establecer los parámetros de ventaneo. Para esta prueba se escogió una ventana que tenga las dimensiones necesarias para analizar los momentos de Hu de todas las figuras por igual, por lo que el alto y ancho de la ventana establecido en esta ocasión es $wH = 250$ y $wW = 350$ respectivamente. Los valores para el salto de la ventana con respecto al alto y ancho son: $saltoH = 15$ y $saltoW = 13$. Estos valores de salto fueron escogidos con el fin de escanear la imagen en su totalidad y acelerar el proceso de ventaneo. Por último, el valor de threshold establecido es $5e^{-7}$, este valor se propuso tras realizar varias iteraciones de prueba y error con el objetivo de encontrar un valor que permita reconocer la figura con un mínimo margen de error. El resultado final de la imagen se presenta a continuación:

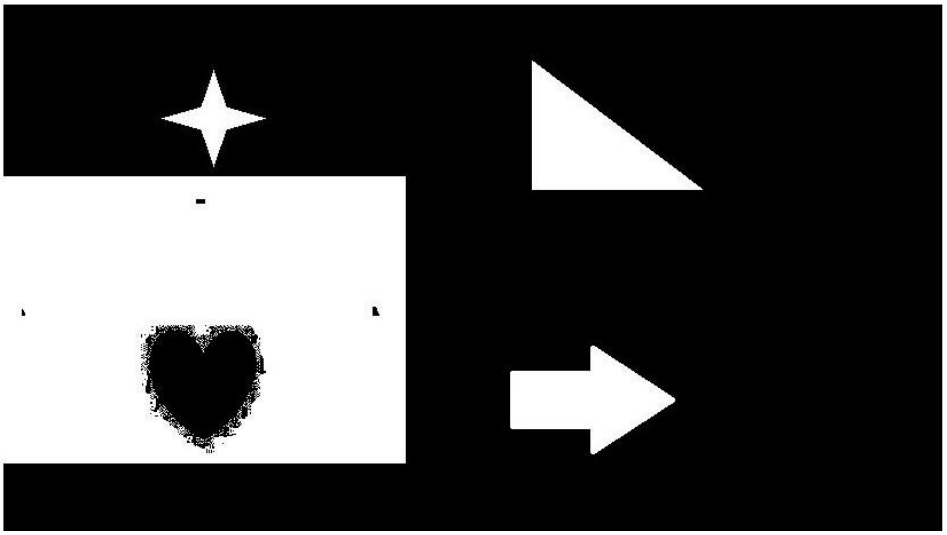

Figura 12. Imagen Final - Resultado de reconocimiento de la figura

La Figura 12 muestra un correcto funcionamiento del algoritmo de detección logrando identificar el corazón en la parte inferior izquierda de la imagen. A continuación se realizó una segunda prueba donde la imagen de prueba presenta 5 figuras de corazón distribuidos en toda la imagen, donde dos de estas figuras presentan rotación. En este caso se evaluó la capacidad del algoritmo de reconocer figuras que presenten transformaciones de similitud. La imagen de prueba es la siguiente:

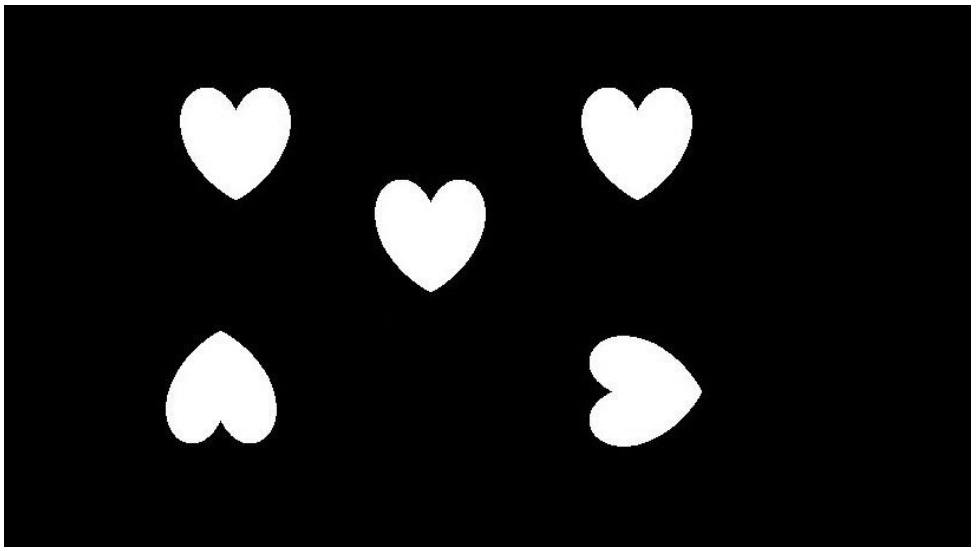

Figura 13. Imagen de prueba con varios corazones de tamaño 460×819 pixeles

Para esta prueba se escogió una ventana que tenga las mismas dimensiones de la imagen de logo la cual presenta un tamaño de 146×142 pixeles, por lo tanto lo $wH = 146$ y $wW = 142$. Los valores para el salto de la ventana con respecto al alto y ancho son: $saltoH = 15$ y $saltoW = 13$. El valor de threshold se mantiene con respecto a la prueba anterior. El resultado de la imagen final es la siguiente:

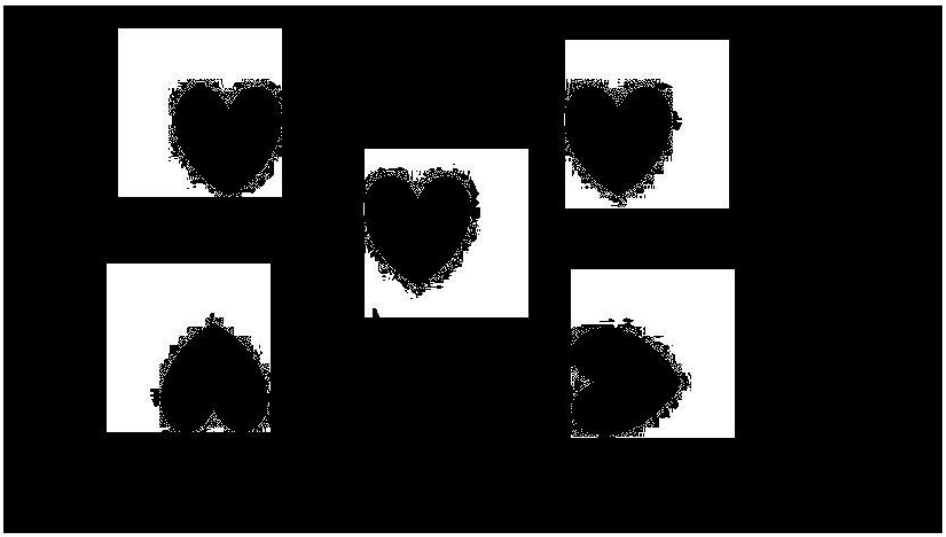

Figura 14. Imagen Final - Resultado de reconocimiento de las figuras

La Figura 14 muestra un exitoso reconocimiento de todas las figuras de corazón que se encuentran en la imagen, incluso de aquellas que presentan rotación. A partir del correcto funcionamiento del algoritmo de detección de figuras se procede a realizar la implementación en hardware.

Resultados en Hardware

La imagen de prueba que se utilizó para la implementación es la siguiente:

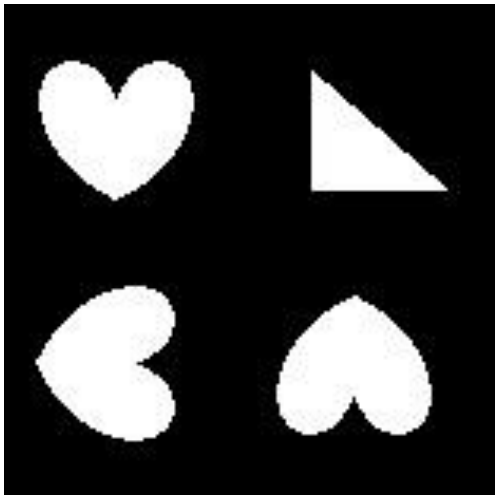

Figura 15. Imagen de prueba utilizada en la implementación de tamaño de 155×155

La imagen de logo utilizada en la implementación es similar a la Figura 10, con la diferencia que el tamaño se redujo a una imagen de 60×70 pixeles. El resultado de la detección mostrado en pantalla a través del VGA se presenta a continuación:

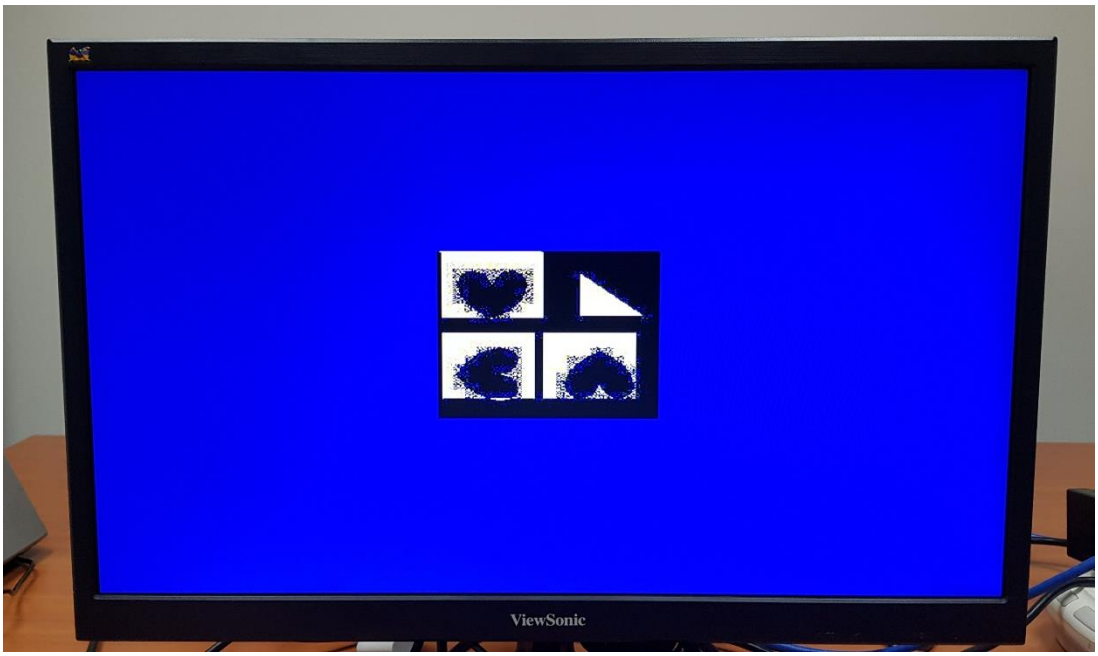

Figura 16. Implementación en funcionamiento

Para la detección de las figuras se usó un valor de threshold de 0.002136. En la siguiente tabla comparamos la velocidad de operación del algoritmo en MATLAB con los tiempos de la tarjeta FPGA obtenidos de la simulación para cada módulo:

Tabla 5. Tiempo de comparación promedio entre MATLAB y FPGA

Variable	**Tiempo en MATLAB** $[ms]$	**Tiempo en Simulación** $[ms]$
M00	0.076	0.031
M01	0.204	0.031
M10	0.242	0.031
xjp, ykp	0.308	$1e^{-5}$ (1 ciclo de reloj)
U11	0.344	0.031
U20	0.186	0.031
U02	0.159	0.031
V11	0.119	0.001
V02	0.082	0.001
V20	0.060	0.001
Hu1	0.070	$1e^{-5}$ (1 ciclo de reloj)
Hu2	0.072	0.00122

Como se puede apreciar el tiempo de ejecución del algoritmo implementado en hardware es menor al tiempo de ejecución en MATLAB. Podemos observar el efecto de paralelismo donde una misma señal es procesada por varios módulos a la vez como es el caso de los módulos de momentos espaciales (M00, M10, M01) los cuales presentan el mismo tiempo de ejecución ya que operan a la vez.

DISCUCIONES Y CONCLUSIONES

En el presente trabajo se logró con éxito el desarrollo en software de un algoritmo de detección de figuras basadas en la teoría de los Momentos de Hu y el algoritmo Non Local Means, así como su implementación en hardware usando HDL (Verilog) en la tarjeta FPGA ZYBO 7000, demostrando la capacidad de poder implementar en hardware algoritmos de alto nivel como son los algoritmos de visión.

La principal dificultad en el presente trabajo radica en la complejidad del leguaje Verilog. Se encontró un desafío al momento de cargar la imagen debido a que no se pudo utilizar la RAM propia de la tarjeta, sin embargo se crearon diferentes módulos de memorias para cumplir con el objetivo de implementación. Durante el proceso de implementación se realizó modificaciones al código original del algoritmo de detección para poder generar su síntesis. Los cambios realizados provocaron una mínima reducción en la precisión del algoritmo, pero ayudo a aumentar la velocidad de procesamiento del mismo al hacer que los módulos reduzcan su complejidad matemática buscando alternativas que no afecten al funcionamiento general del mismo.

Tal como se puede apreciar en la Tabla 5 existe una clara ventaja en cuanto al tiempo de procesamiento del FPGA con relación al código ejecutado de MATLAB lo que demuestra la eficiencia que presenta el algoritmo detección implementado en hardware. Esto permite que en el futuro se pueda implementar un sistema de reconocimientos de figuras en tiempo real haciendo uso de los recursos de procesamiento de la tarjeta FPGA, así como de una cámara digital. Para futuros proyectos se recomienda en trabajar con un formato de manejo de bits en punto flotante con el fin de obtener una mayor capacidad de almacenamiento en los resultados de las operaciones y poder trabajar con imágenes que tengan un tamaño más grande.

REFERENCIAS

Altera Corporation. (2010). *Verilog HDL: Single-Port RAM*. Recuperado de: https://www.altera.com/support/support-resources/design-examples/design-software/verilog/ver-single-port-ram.html

Baudes, A., Coll, B., & Morel, J. (2005). *A review of Image Denoising Algorithms*. Society for Industrial and Applied Mathematics, vol. 4, no. 2, pp. 490-530.

Crockett, L. H., Elliot, R. A., Enderwitz, M. A., & Stewart, R. W. (2014). *The Zynq book: Embedded processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 all programmable SoC*. Glasgow: Strathclyde Academic Media.

Digilent. (2014). *ZYBO Reference Manual.* Recuperado de: https://www.xilinx.com

GitHub (2017). Zybo-VGA-Pong. Recuperado de: https://github.com/Kagehiko/Zybo-VGA-Pong

Harris, D. M., & Harris, S. L. (2007). *Digital design and computer architecture*. San Francisco, CA: Morgan Kaufmann.

Lowe, D. (2004). *Distinctive image features from scale-invariant keypoints.* International Journal on Computer Vision, vol. 60, no. 2, pp. 91-110.

Mikolajczyk, K., Tuytelaars, T., Schmid, C., Zisserman, A., Matas, J., Schaalitzky, F., Kadir, T. & Van Gool. L. (2005). *A Comparison of Affine Region Detectors.* International Journal of Computer Vision, vol. 65, no. 1-2, pp. 43-72.

Pratt, W. K. (2007). *Digital image processing* (4th ed.). New York: Wiley.

Skalicky, S. (2011). *Fixed_point_arithmetic_parameterized.* Recuperado de: https://opencores.org/project/fixed_point_arithmetic_parameterized

Szeliski, R. (2010). *Computer vision: Algorithms and applications*. New York: Springer.