



**CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS
AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL**

UNIDAD ZACATENCO

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

SECCIÓN DE ELECTRÓNICA DEL ESTADO SÓLIDO

**“SISTEMA DE DESARROLLO DE REDES NEURONALES
CELULARES”**

T E S I S

Que presenta

ING. JOSE DE JESUS MORALES ROMERO

Para obtener el grado de

MAESTRO EN CIENCIAS

En la especialidad de

INGENIERÍA ELÉCTRICA

Directores de la Tesis:

Dr. Felipe Gómez Castañeda

Dr. José Antonio Moreno Cadenas

México, D.F.

Noviembre del 2015

Agradecimientos

A CONACYT por el apoyo económico que me otorgo para realizar mi programa de maestría.

A mi familia por todo el apoyo incondicional que me ha brindado durante mis estudios.

A mis asesores Felipe Gómez, José Antonio Moreno por toda su ayuda y apoyo para realizar esta tesis, además de M.C. Luis Martín por su colaboración y consejos, a mis maestros y personal de apoyo.

A mis amigos y compañeros que me han apoyado y dando consejos durante mi programa de maestría.

Dedicatoria

A mi familia, por todo el apoyo que me han otorgado y principalmente a mi madre por su apoyo incondicional.

Contenido

Agradecimientos.....	i
Dedicatoria.....	ii
Resumen	viii
Introducción.....	ix
Objetivo	x
Objetivos particulares.....	x
Capítulo 1. Teoría de Redes Neuronales Celulares	1
1.1 Introducción.....	1
1.2 Arquitectura de la CNN.....	1
1.2.1 Vecindario de una CNN	2
1.2.2 Ecuaciones de la CNN.....	4
1.2.3 Función de salida.....	5
1.2.4 Modelo eléctrico de una neurona.....	6
1.2.5 Condiciones de frontera.....	7
1.3 Estabilidad de una CNN	8
1.4 Plantillas de retroalimentación y control de una CNN	8
1.5 Métodos para búsqueda de plantillas	10
1.6 Aplicaciones de CNN en procesamiento de imágenes	12
1.6.1 Removedor de Ruido.....	12
1.6.2 Extractor de bordes para imágenes binarias	13
1.6.3 Extractor de bordes para imágenes en grises.....	14
1.6.4 Extractor de sombras	15
1.6.5 Detector de conectividad global	16
1.7 Conclusiones.....	17

1.8	Referencias.....	19
Capítulo 2. Multiplexado en tiempo de una CNN		21
2.1	Introducción	21
2.2	Pasos para multiplexar en tiempo una CNN	21
2.2.1	Obtención de bloques	22
2.2.2	Orden lexicográfico	23
2.3	Errores en el multiplexado de la CNN.....	24
2.3.1	Reducción del error	25
2.4	Repetición del tipo de bloques.....	27
2.5	Algoritmo general para el multiplexado en tiempo de una CNN	27
2.6	CNN para el multiplexado en tiempo	30
2.7	Conclusiones	30
2.8	Referencias.....	32
Capítulo 3. Descripción de una CNN multiplexada en tiempo con MATLAB y SIMULINK		
	33	
3.1	Introducción	33
3.2	Algoritmo de una CNN multiplexada en tiempo	33
3.3	Creación de una CNN de 4×4 Neuronas en SIMULINK	34
3.3.1	Integrador Euler.....	35
3.3.2	Neurona en SIMULINK	39
3.3.3	Comprobación del estado estable	41
3.3.4	Condición de frontera	42
3.3.5	Prueba de la neurona en SIMULINK	43
3.3.6	Creación de la CNN de 4×4 Neuronas	44
3.4	Control de la CNN con MATLAB	49

3.4.1	Proceso “Carga los parámetros”	49
3.4.2	Proceso “Calcula la cantidad de bloques”	51
3.4.3	Proceso “Obtiene el bloque”	52
3.4.4	Proceso “Tipo de bloque”	53
3.4.5	Proceso “Actualiza salida y estados”	54
3.5	Funcionamiento de la CNN con MATLAB y SIMULINK	55
3.5.1	Removedor de ruido	55
3.5.2	Extractor de Bordes	55
3.5.3	Extractor de sombras	56
3.5.4	Detector de conectividad global	58
3.6	Conclusiones	59
3.7	Referencias	60
Capítulo 4. Descripción de una CNN multiplexada en tiempo en VHDL		61
4.1	Introducción	61
4.2	Diagrama a bloques de la CNN	62
4.3	Descripción de una Neurona en VHDL	65
4.4	Descripción de la CNN 4x4 en VHDL	67
4.5	Multiplexado de la CNN en VHDL	68
4.6	Prueba de la CNN	70
4.6.1	Removedor de ruido	71
4.6.2	Extractor de bordes	72
4.6.3	Extractor de sombras	72
4.6.4	Detector de conectividad global	74
4.7	Conclusiones	75
4.8	Referencias	76

Resultados.....	77
Conclusiones.....	89
Trabajo futuro	91
Apéndice.....	92
A. Código en MATLAB para el multiplexado de una CNN.	92
B. Código en VHDL para el multiplexado de una CNN.	98
C. Código en MATLAB para la comunicación con el FPGA.	125

Preface

Since Cellular Neural Networks were introduced, there have been several studies about these, such as search templates, numerical integration methods to solve the state equation of each of the neurons in the network, and these networks have been used to solve problems such as the image processing. However, it notes that these solutions lead to a large consumption of resources, which is why this paper has proposed a method of reducing the resources needed to perform well CNN and process very large images. The method introduced in this paper is multiplexing CNN.

In Chapter 1, we talk mainly about the theory of Cellular Neural Networks. In this chapter we talk about architecture and stability of CNN also we will explain what are the templates and existing methods for searching the templates, as these define what CNN will do. Finally, some examples of use of Cellular Neural Network are given as image processor. In Chapter 2, we talk mainly about the theory of multiplexing of CNN and its characteristics and general algorithm, along with their advantages and disadvantages. In addition, we will talk about the conditions that must be met for CNN multiplexing is performed correctly.

In chapter 3, we talk about the simulator that has been done on MATLAB and Simulink, which performs multiplexing of our CNN, to verify the correct operation that has this so that it can be physically implemented. The proper functioning of the simulator performing basic image processing will be verified. In Chapter 4, the most important part of our work will be done, which is the physical realization of our multiplex CNN for image processing. We will talk about how it was developed using VHDL to describe the circuit to be implemented within the FPGA. In addition, proper operation performing basic image processing is checked.

Finally, in the section "Results" of this paper some examples of processing real images using our implementation with FPGA, along with some recommendations and limitations of our implementation are given.

Resumen

Desde que las Redes Neuronales Celulares fueron introducidas, se han realizado diversos estudios a cerca de estas, como por ejemplo la búsqueda de plantillas, métodos de integración numérica que resuelvan la ecuación de estado de cada una de las neuronas dentro de la red, así como se han utilizado estas redes para la solución de problemas como lo es el procesamiento de imágenes. Sin embargo, cabe destacar que estas soluciones conllevan a un consumo muy grande de recursos, es por esto que este trabajo se ha propuesto un método de reducir los recursos necesarios para realizar la CNN y así procesar imágenes muy grandes. El método introducido en este trabajo consiste en la multiplexación de la CNN.

En el capítulo 1 se habla principalmente sobre la teoría de las Redes Neuronales Celulares. Dentro de este capítulo se hablara sobre la arquitectura y la estabilidad de una CNN, además se explicará lo que son las plantillas y los métodos que existen para la búsqueda de las plantillas, ya que estas definen lo que realizarla la CNN. Finalmente, se dan algunos ejemplos del uso de la Red Neuronal Celular como procesador de imágenes. En el capítulo 2, se habla principalmente sobre la teoría del multiplexado de una CNN, así como sus características y su algoritmo general, además de sus ventajas y desventajas. Además, se hablará sobre las condiciones que se deben de cumplir para que el multiplexado de la CNN se realice de forma correcta.

En el capítulo 3, se habla sobre el simulador que se ha realizado sobre MATLAB y SIMULINK, que realiza el multiplexado de nuestra CNN, para verificar el correcto funcionamiento que tiene esta y poderlo así implementar físicamente. Se verificará el correcto funcionamiento del simulador realizando procesamiento de imágenes básicas. En el capítulo 4 se realizará la parte más importante de nuestro trabajo, lo cual es la realización física de nuestra CNN multiplexada para el procesamiento de imágenes. Se hablará de cómo fue desarrollada utilizando VHDL para describir el circuito que será implementado dentro del FPGA. Además, se comprobará su correcto funcionamiento realizando el procesamiento de imágenes básicas.

Finalmente, en la parte de "Resultados" de este trabajo se dan algunos ejemplos del procesamiento de imágenes reales utilizando nuestra implementación con el FPGA, además de algunas recomendaciones y limitaciones que tiene nuestra implementación.

Introducción

En la actualidad, el campo de la electrónica ha estado creciendo en forma exponencial y un ejemplo de esto es el desarrollo de los procesadores digitales; estos procesadores tienen grandes capacidades de procesamiento y velocidades muy altas, sin embargo, éstos aún no pueden resolver varios problemas de forma óptima y rápida, como lo es el procesamiento de imágenes y video, reconocimiento de patrones, resolución de ecuaciones diferenciales, entre otras, debido a su naturaleza de procesamiento serial que poseen este tipo de procesadores. Es por esto que se han propuesto varias alternativas para solucionar esta problemática como lo son las Redes Neuronales Celulares que poseen una gran capacidad de procesamiento paralelo.

Desde que L. O. Chua y L. Yang presentaron las Redes Neuronales Celulares en el año de 1988, se han realizado diversos estudios acerca de éstas. Algunos de estos estudios incluyen como por ejemplo, la optimización y búsqueda de plantillas robustas para el procesamiento de diferentes tareas, otros estudios realizan una comparativa sobre los diversos métodos de integración numérica que resuelvan la ecuación de estado dentro de las Redes Neuronales Celulares, también se han realizado estudios sobre la resolución de problemas en el campo de la visión y el procesamiento de imágenes.

Dentro de estos estudios se presenta el procesamiento de imágenes, sin embargo, debido a que como veremos más adelante, la CNN utilizada debe de ser del mismo tamaño que la imagen a ser procesada. Debido a lo anterior, se han realizado propuestas como lo es el multiplexado de la CNN, esto quiere decir, que se van procesando fracciones de la imagen original dentro de la CNN, esto hasta completar la imagen. Esta propuesta resulta atractiva debido a que con esta técnica se reducen los recursos necesarios que se requieren para el procesamiento de imágenes.

Objetivo

El objetivo de este trabajo es presentar el desarrollo de una CNN digital de 4×4 neuronas la cual será implementada físicamente en un FPGA, sin embargo esta CNN, a diferencia de otras CNN's presentadas en diversos trabajos, tendrá la capacidad de ser multiplexada. Además se desarrollará un simulador de CNN igualmente de 4×4 neuronas, el cual será implementado dentro del programa MATLAB utilizando SIMULINK para el procesamiento paralelo de las neuronas. El objetivo de realizar esta implementación y simulador es debido a que actualmente las CNN's se han ido desarrollando e investigado sus características y formas de optimización, y dentro de estas investigaciones se encuentra el procesamiento de imágenes, con lo cual nuestra CNN desarrollada se utilizará principalmente para diferentes tipos de procesamiento de imágenes de cualquier tamaño, además de que ésta será capaz de realizar procesamiento de imágenes en escala de grises y no únicamente de imágenes en escala de blancos y negros, como ocurre en otros trabajos realizados previamente.

Objetivos particulares

Los objetivos particulares de este trabajo son primero el aprendizaje sobre la teoría de las CNN's, así como sus ventajas y desventajas del uso de las CNN's para la resolución de problemas, segundo el desarrollo de un simulador CNN multiplexado para comprobar el comportamiento de esta, y finalmente el desarrollo de una CNN dentro de un FPGA para realizar procesamiento de imágenes.

Capítulo 1. Teoría de Redes Neuronales Celulares

1.1 Introducción

Hoy en día diversas aplicaciones de la electrónica como lo son en el campo de la computación, almacenamiento de datos, comunicaciones, control, procesamiento de imágenes, reconocimiento de patrones, video, etc., podrían ser resueltas por procesadores digitales que trabajan en forma serial, sin embargo, hay aplicaciones donde este tipo de procesadores están limitados, como son el reconocimiento de patrones, el procesamiento de imágenes y video, procesamiento no lineal en general y solución de ecuaciones diferenciales, entre otras. Para resolver este tipo de problemas, se han propuesto varias alternativas de solución como son los circuitos integrados CNN, autómatas celulares, retinas de silicio, circuitos integrados de aplicación específica (ASICs), entre otros.

Las Redes Neuronales Celulares (CNN por sus siglas en inglés “Cellular Neural Networks”) fueron introducidas por L. O. Chua y L. Yang en el año de 1988. Las Redes Neuronales Celulares poseen características de procesamiento paralelo de las Redes Neuronales y la interconexión de los Autómatas Celulares; en las primeras, éstas se caracterizan porque pueden ser integradas tecnológicamente como un circuito analógico no lineal que son capaces de procesar señales en tiempo real y en las segundas porque pueden ser identificadas como un arreglo de elementos regulares analógicos [1].

La CNN está compuesta de unidades básicas que están interconectadas entre sí en forma de arreglo teniendo así características como el procesamiento paralelo, dinámica en tiempo continuo e interacción global entre los elementos de la red.

1.2 Arquitectura de la CNN

Como se mencionó anteriormente, las CNN son arreglos de elementos básicos, estos elementos son llamados *celdas* o *neuronas*, estas *neuronas* están compuestas de elementos eléctricos como resistencias, capacitores y fuentes de corriente controladas por voltaje y fuentes independientes de voltaje [1] [2]. La interconexión de las *neuronas* es similar a las encontradas en los autómatas celulares. Esto es, que una neurona dentro de una CNN está conectada a las neuronas vecinas. Las neuronas que se encuentran adyacentes entre sí pueden

interactuar directamente entre ellas y las neuronas que no están conectadas directamente pueden afectarse indirectamente entre ellas debido a los efectos de propagación de la dinámica en tiempo continuo de la CNN [1] y [3].

Teóricamente una CNN puede ser de diversas dimensiones como es mostrado en los trabajos [1], [2] y [4], sin embargo, es más frecuente encontrar CNN's de dos dimensiones. Una arquitectura estándar de una CNN, como la mostrada en la Fig. 1.1, está compuesta de un arreglo rectangular de MN neuronas ($C(i,j)$) donde M es la cantidad de filas, N la cantidad de columnas, e (i,j) son las coordenadas cartesianas correspondientes a la posición de la neurona [2] y [3].

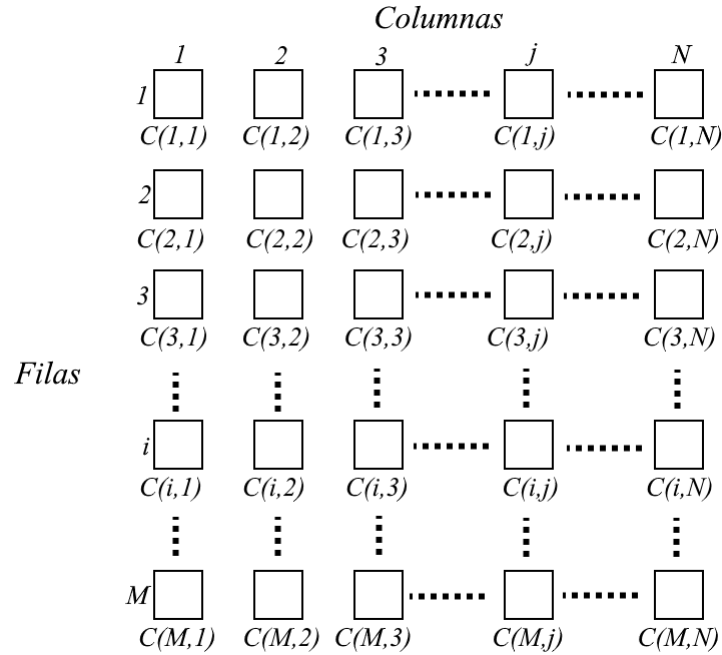


Fig. 1.1 Estructura básica de una CNN

1.2.1 Vecindario de una CNN

El vecindario $N_r(i,j)$ o esfera de influencia de radio r en una neurona $C(i,j)$ dentro de una CNN está definido como el conjunto de todas las neuronas del vecindario que satisfacen la siguiente propiedad [1] y [2]:

$$N_r(i,j) = \{C(k,l) | \max\{|k-i|, |l-j|\} \leq r, 1 \leq k \leq M; 1 \leq l \leq N\} \quad (1.1)$$

Donde r es un entero positivo.

Algunas veces nos referiremos al vecindario $N_r(i, j)$ como un vecindario de $((2r + 1) \times (2r + 1))$ neuronas para el caso de una CNN de dos dimensiones [1] y [2]. En la Fig. 1.2 observamos un vecindario de $r = 1$ o también llamado un vecindario de (3×3) neuronas.

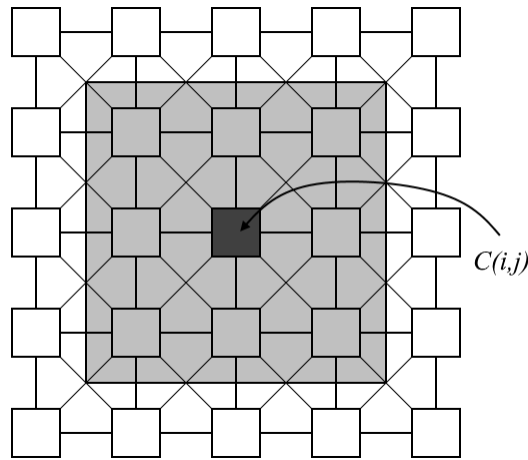


Fig. 1.2 CNN con $r = 1$, (vecindario de (3×3) neuronas)

Se llama *neurona regular* si y solo si todas las neuronas dentro del vecindario N_r existen, de otra forma se les llama *neuronas de frontera*; esto es mostrado en la Fig. 1.3.

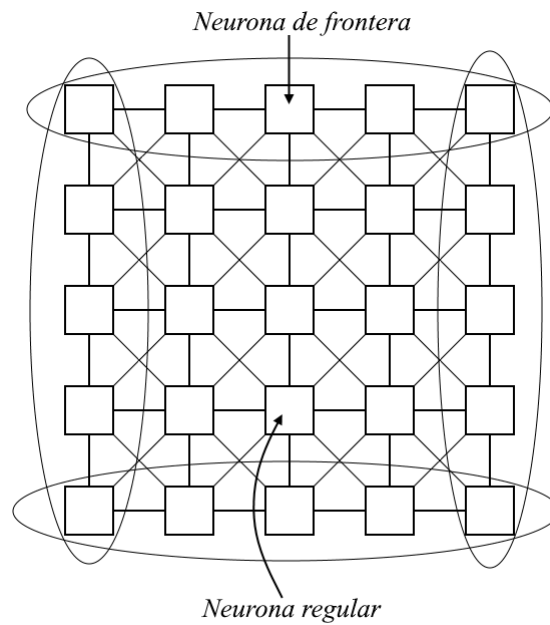


Fig. 1.3. Tipos de neurona.

Cuando $r > N/2$ y $M = N$, tenemos una CNN totalmente conectada, donde cada neurona está conectada a cada una de las demás neuronas y $N_r(i, j)$ es toda la matriz. Este caso extremo corresponde a la clásica red de Hopfield [2].

1.2.2 Ecuaciones de la CNN

La dinámica de una neurona $C(i, j)$ en la CNN está definida por la siguiente *ecuación de estado* [1] y [2]:

$$C \frac{dv_{xij}(t)}{dt} = -\frac{1}{R_x} v_{xij}(t) + \sum_{C(k,l) \in N_r(i,j)} A(i, j; k, l) v_{ykl}(t) + \sum_{C(k,l) \in N_r(i,j)} B(i, j; k, l) v_{ukl} + I_{ij} \quad (1.2)$$

Donde $1 \leq i \leq M$ y $1 \leq j \leq N$.

Haciendo $C = 1$ y $R_x = 1$ en la ecuación (1.2), obtenemos la ecuación normalizada de estado:

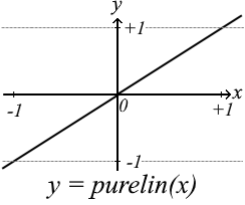
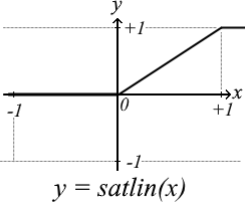
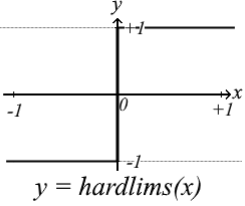
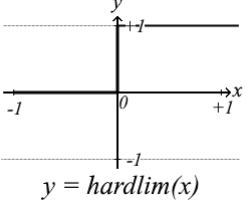
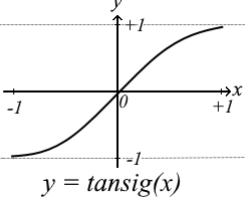
$$\frac{dv_{xij}(t)}{dt} = -v_{xij}(t) + \sum_{C(k,l) \in N_r(i,j)} A(i, j; k, l) v_{ykl}(t) + \sum_{C(k,l) \in N_r(i,j)} B(i, j; k, l) v_{ukl} + I_{ij} \quad (1.3)$$

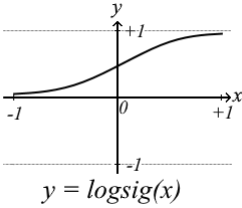
Donde v_{xij} es el estado de la neurona $C(i, j)$ y su condición inicial es asumida para tener una magnitud menor o igual a 1, v_{uij} es la entrada de la neurona $C(i, j)$, y es asumida para ser una constante con magnitud menor o igual a 1, v_{ukl} son las estradas de las neuronas vecinas, v_{yij} es la salida de la neurona $C(i, j)$, v_{ykl} son las salidas de las neuronas vecinas e I es el umbral, $A(i, j; k, l)$ es llamada plantilla de retroalimentación y $B(i, j; k, l)$ es llamada plantilla de control, (i, j) indican el índice de la neurona actual y (k, l) indican los índices vecinales de la red neuronal.

1.2.3 Función de salida

Además, se establece la función de salida, que es la única función no lineal dentro de la neurona. Existen varias funciones de salida utilizadas, algunas de estas funciones se muestran en la siguiente tabla:

Tabla 1.1. Funciones de transferencia en redes neuronales

Función	Símbolo	Definición
Lineal		$purelin(x) = x$
Saturación lineal positiva		$satlin(x) = \begin{cases} 0, & \text{si } x \leq 0 \\ x, & \text{si } 0 < x < 1 \\ 1, & \text{si } x \geq 1 \end{cases}$
Signo		$hardlims(x) = \begin{cases} -1, & \text{si } x < 0 \\ 1, & \text{si } x \geq 0 \end{cases}$
Signo positiva		$harlim(x) = \begin{cases} 0, & \text{si } x < 0 \\ 1, & \text{si } x \geq 0 \end{cases}$
Sigmoidal tangente hiperbólica		$tansig(x) = \frac{2}{1 + e^{-2x}} - 1$

<p>Sigmoidal logarítmica</p>		$\text{logsig}(x) = \frac{1}{1 + e^{-x}}$
----------------------------------	---	---

Sin embargo, la función de salida más utilizada para el procesamiento de imágenes es la función de transferencia “*Saturación Lineal Simétrica*”, que es representada por la siguiente ecuación:

$$v_{yij}(t) = \frac{1}{2} (|v_{xij}(t) + 1| - |v_{xij}(t) - 1|) \quad (1.4)$$

En la Fig. 1.4 se muestra la gráfica de la función de salida, que es la función de transferencia de “*Saturación Lineal Simétrica*”, donde para el procesamiento de imágenes se tiene que si $v_{yij} = 1$ correspondiente a un pixel negro y para $v_{yij} = -1$ corresponde a un pixel blanco.

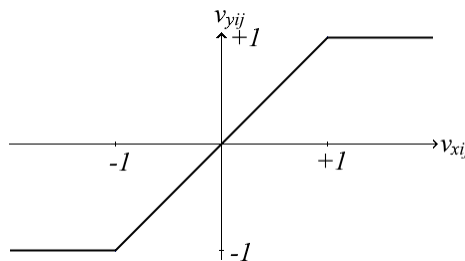


Fig. 1.4 Función de transferencia “*Saturación Lineal Simétrica*”

1.2.4 Modelo eléctrico de una neurona

El modelo eléctrico típico de una neurona es mostrado en la siguiente figura, donde u representa la entrada, x el estado e y representa la salida, donde este modelo de neurona sigue la ecuación (1.2).

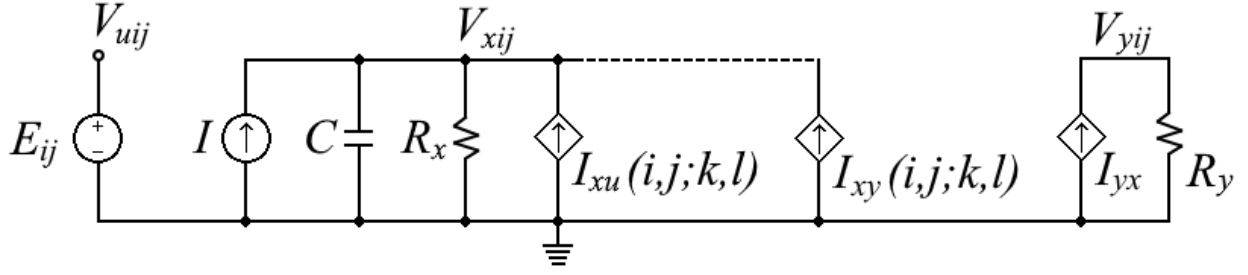


Fig. 1.5. Modelo eléctrico de una neurona en la CNN

El voltaje en el nodo v_{xij} representa el estado de la neurona, el voltaje en el nodo v_{uij} representa la entrada de la neurona, y el voltaje en el nodo v_{yij} representa la salida. Las fuentes $I_{xy}(i, j; k, l)$ e $I_{xu}(i, j; k, l)$ son fuentes de corriente controladas por voltaje que tienen las características de la ecuación (1.2) donde representan:

$$I_{xy}(i, j; k, l) = A(i, j; k, l)v_{ykl} \quad (1.5)$$

$$I_{xu}(i, j; k, l) = B(i, j; k, l)v_{ukl} \quad (1.6)$$

Como se mencionó anteriormente, el único elemento no lineal es la función de transferencia “*Saturación Lineal Simétrica*” que es representada por la fuente de corriente controlada por voltaje $I_{yx} = \left(\frac{1}{R_y}\right)v_{xij}$ donde observamos que depende del estado de la neurona v_{xij} .

1.2.5 Condiciones de frontera

Como se mencionó en 1.2.1 existen la neurona regular y la neurona de frontera. Para este último caso las neuronas deben de cumplir las condiciones de frontera para su correcto funcionamiento, para esto existen tres de condiciones de frontera que son las más utilizadas [2] dentro de las CNN, las cuales se mencionan a continuación:

1. *Condición de frontera fija (Dirichlet)*: En este tipo de frontera los valores de las variables de estado v_{xkj} y salida v_{ykl} de las neuronas de frontera están asignados a un valor constante, generalmente 0.

2. *Condición de frontera de flujo cero (Neumann)*: En este tipo de frontera los valores de las variables de estado y la salida son iguales a los valores de las neuronas adyacentes.
3. *Condición de frontera periódica (Toroidal)*: En este tipo de frontera los valores de las variables de estado y la salida son iguales a los valores de las neuronas que se encuentran en el otro extremo de la CNN.

1.3 Estabilidad de una CNN

La función básica de una CNN para el procesamiento de imágenes es transformar una imagen de entrada en su respectiva imagen de salida, para ello cada pixel de una imagen será procesada por una neurona. Con lo cual aquí se restringen los valores de salida a $[-1,1]$. Sin embargo, las entradas de los pixeles pueden contener diversos niveles de grises con lo cual se restringen estas entradas a:

$$|v_{uij}| \leq 1, \quad 1 \leq i \leq M, 1 \leq j \leq N \quad (1.7)$$

Como se demuestra en el trabajo realizado por L. O. Chua y L. Yang [1], se tiene que para que no oscile la red se debe de cumplir que:

$$A(i, j; i, j) > 1 \quad (1.8)$$

Con esto también se garantiza que la salida de la neurona será binaria.

1.4 Plantillas de retroalimentación y control de una CNN

En el caso de una CNN con vecindario $r = 1$ y considerando parámetros invariantes, las plantillas de retroalimentación (plantilla \mathbf{A}), control (plantilla \mathbf{B}) y el umbral (I) se pueden representar de la siguiente manera:

$$\mathbf{A} = \begin{bmatrix} a_{i-1,j-1} & a_{i-1,j} & a_{i-1,j+1} \\ a_{i,j-1} & a_{i,j} & a_{i,j+1} \\ a_{i+1,j-1} & a_{i+1,j} & a_{i+1,j+1} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} b_{i-1,j-1} & b_{i-1,j} & b_{i-1,j+1} \\ b_{i,j-1} & b_{i,j} & b_{i,j+1} \\ b_{i+1,j-1} & b_{i+1,j} & b_{i+1,j+1} \end{bmatrix}, I_{i,j} \quad (1.9)$$

Ahora considere una neurona típica $C(i, j)$ dentro del vecindario $N_r(i, j)$ como sigue:

$C(i-1, j-1)$	$C(i-1, j)$	$C(i-1, j+1)$
$C(i, j-1)$	$C(i, j)$	$C(i, j+1)$
$C(i+1, j-1)$	$C(i+1, j)$	$C(i+1, j+1)$

Podemos así escribir para el caso de la plantilla **A**:

$$\begin{aligned}
\sum_{C(k,l) \in N_r(i,j)} A(i,j;k,l)y_{kl} &= \sum_{|k-i| \leq 1} \sum_{|l-j| \leq 1} A(k-i, l-j)y_{kl} \\
&= a_{-1,-1}y_{i-1,j-1} + a_{-1,0}y_{i-1,j} + a_{-1,1}y_{i-1,j+1} + a_{0,-1}y_{i,j-1} + a_{0,0}y_{i,j} \\
&\quad + a_{0,1}y_{i,j+1} + a_{1,-1}y_{i+1,j-1} + a_{1,0}y_{i+1,j} + a_{1,1}y_{i+1,j+1} \\
&= \sum_{k=-1}^1 \sum_{l=-1}^1 a_{k,l}y_{i+k,j+l}
\end{aligned} \tag{1.10}$$

De esta forma podemos reescribir la ecuación anterior como:

$$\begin{aligned}
&\stackrel{\triangle}{=} \begin{array}{|c|c|c|} \hline a_{-1,-1} & a_{-1,0} & a_{-1,1} \\ \hline a_{0,-1} & a_{0,0} & a_{0,1} \\ \hline a_{1,-1} & a_{1,0} & a_{1,1} \\ \hline \end{array} \otimes \begin{array}{|c|c|c|} \hline y_{i-1,j-1} & y_{i-1,j} & y_{i-1,j+1} \\ \hline y_{i,j-1} & y_{i,j} & y_{i,j+1} \\ \hline y_{i+1,j-1} & y_{i+1,j} & y_{i+1,j+1} \\ \hline \end{array} = A \otimes Y_{i,j} \tag{1.11}
\end{aligned}$$

Donde el símbolo \otimes representa la suma de los productos punto. En matemáticas discretas esta operación es llamada “convolución espacial” [2].

De la misma forma podemos hacer el análisis para la plantilla **B**:

$$\begin{aligned}
\sum_{C(k,l) \in N_r(i,j)} B(i,j;k,l)u_{kl} &= \sum_{|k-i| \leq 1} \sum_{|l-j| \leq 1} B(k-i, l-j)u_{kl} \\
&= b_{-1,-1}u_{i-1,j-1} + b_{-1,0}u_{i-1,j} + b_{-1,1}u_{i-1,j+1} + b_{0,-1}u_{i,j-1} + b_{0,0}u_{i,j} \\
&\quad + b_{0,1}u_{i,j+1} + b_{1,-1}u_{i+1,j-1} + b_{1,0}u_{i+1,j} + b_{1,1}u_{i+1,j+1} \\
&= \sum_{k=-1}^1 \sum_{l=-1}^1 b_{k,l}u_{i+k,j+l}
\end{aligned} \tag{1.12}$$

Escribiendo la ecuación anterior como:

$$\cong \begin{array}{|c|c|c|} \hline b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ \hline b_{0,-1} & b_{0,0} & b_{0,1} \\ \hline b_{1,-1} & b_{1,0} & b_{1,1} \\ \hline \end{array} \otimes \begin{array}{|c|c|c|} \hline u_{i-1,j-1} & u_{i-1,j} & u_{i-1,j+1} \\ \hline u_{i,j-1} & u_{i,j} & u_{i,j+1} \\ \hline u_{i+1,j-1} & u_{i+1,j} & u_{i+1,j+1} \\ \hline \end{array} = B \otimes U_{i,j} \quad (1.13)$$

1.5 Métodos para búsqueda de plantillas

Las plantillas A y B , junto con el valor de umbral I y el estado inicial $v_{xij}(0)$, determinan el tipo de procesamiento que realizará la CNN; a diferencia de otras estructuras neuronales, no existe un método único para encontrar dichos parámetros [5] y [6].

Los métodos para encontrar los valores de los parámetros de la CNN se pueden agrupar como: **métodos analíticos** [7] - [10], estos métodos se basan en una serie de reglas locales que caracterizan la dinámica de la neurona dentro de la CNN. Para realizar esto, se forma un conjunto de desigualdades que deben ser resueltas mediante un método de optimización; **algoritmos de aprendizaje local** [11]-[15], estos algoritmos se basan en los algoritmos de entrenamiento de otras Redes Neuronales, estos algoritmos pueden ser “*recurrent backpropagation*”, “*backpropagation through time*” y el aprendizaje de “*perceptron*”; **algoritmos de aprendizaje global** [16] y [17], similar a los algoritmos de aprendizaje local, los cuales presentan pares de datos de entrada-salida, donde estos datos son presentados de manera global a toda la CNN y no solamente a la neurona de forma local. Como vemos, en estos algoritmos su nivel de dificultad se va incrementando debido al incremento de número de variable, además esto produce que se presenten dificultades para encontrar la solución de las ecuaciones desarrolladas.

Los **métodos heurísticos** han sido desarrollados para cubrir los problemas de los métodos anteriormente mencionados. La búsqueda de plantillas ha sido tratado como un importante estudio de optimización y por lo tanto, muchos métodos basados en *Inteligencia Artificial* han sido propuestos como lo son: Optimización de Nube de Partículas también llamado *PSO* (por sus siglas en inglés de *Particle Swarm Optimization*), Algoritmos Genéticos también llamado *GA* (por sus siglas en inglés de *Genetic Algorithm*), Evolución Diferencia también llamado *DE* (por sus siglas en inglés de *Differential Evolution*), etc.

Algunos de los métodos para la búsqueda numérica de plantillas son presentados a continuación:

Método simplex: Este método fue propuesto por Koji Nakai y Akio Ushida en el año de 1995 [18]. Este método de búsqueda de plantillas se basa en forzar la salida dependiendo de un determinado patrón de entrada, con esto se crean diversas desigualdades y finalmente con éstas, crear una función objetivo.

Establecida la función objetivo se procede a buscar la solución óptima utilizando el método simplex, el cual tiene la ventaja de no utilizar derivadas en su algoritmo, sin embargo, este método no es tan eficiente como los algoritmos que usan derivadas, además de que conforme va incrementado el número de variables de la función objetivo, no se llega a un resultado óptimo.

El algoritmo simplex crea una serie de puntos alrededor de un punto inicial (estos puntos se crean haciendo una reflexión, una expansión y una contracción) de una función objetivo con dimensión n , donde el peor punto es sustituido iterativamente hasta encontrar la convergencia de la función objetivo.

En el trabajo realizado en [5] se muestra el uso de este método para encontrar las plantillas.

Método ABC: El algoritmo “Colonia de Abejas Artificial” también llamado ABC (por sus siglas en inglés de “*Artificial Bee Colony*”) es un algoritmo basado en el comportamiento de búsqueda de alimento de las abejas, este tipo de algoritmo pertenece a los llamados algoritmos bio-inspirados.

En el trabajo realizado en [6] se utiliza este tipo de algoritmo para búsqueda de plantillas. Este algoritmo fue en un principio desarrollado por Dervis Karaboga en 2005. Como se mencionó anteriormente, este algoritmo se basa en la búsqueda de alimento por parte de las abejas, donde existen tres tipos de abejas: las abejas trabajadoras, las abejas observadoras y las abejas exploradoras. En un principio, las abejas exploradoras son enviadas a la búsqueda de fuentes de alimento, después las abejas observadoras son enviadas a las fuentes de alimento que encontraron las abejas exploradoras y calculan la cantidad de alimento encontrado, en una tercera fase se asegura que las abejas exploradoras busquen

nuevo alimento aleatoriamente cercano al alimento encontrado anteriormente; aquí la cantidad de alimento representa la solución buscada.

1.6 Aplicaciones de CNN en procesamiento de imágenes

Una de las aplicaciones de las CNN es en el procesamiento de imágenes debido a su característica de procesamiento paralelo. A continuación se mostrarán algunos ejemplos en el procesamiento de imágenes con CNN.

Para comprobar la respuesta de la CNN se utilizará un simulador creado por Simón Moser and Eric Pfaffhauser del Instituto de Procesamiento de Información y Señales de Zúrich, cuya versión es 2.1 y cuya última revisión se realizó en Diciembre de 1998, con dirección electrónica: http://www.isiweb.ee.ethz.ch/haenggi/CNN_web/CNNsim_adv.html

1.6.1 Removedor de Ruido

Este proceso radica en recuperar una imagen cuando existe ruido en ella; se considerará ruido cuando un pixel tenga un color diferente al de sus vecinos verticales u horizontales, estos se muestran en la siguiente figura:

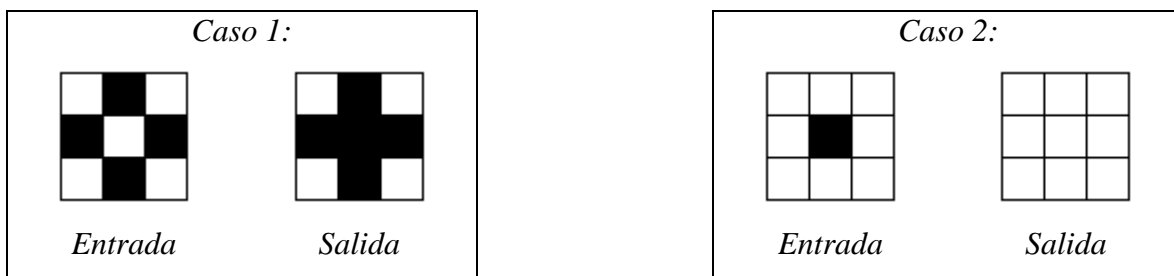


Fig. 1.6. Píxeles que son considerados como ruido, en ambos casos el píxel del centro es el considerado ruido.

Para ejemplificar el removedor de ruido, se utilizarán las siguientes plantillas y valor de umbral, los cuales son tomados de [18]:

$$A = \begin{bmatrix} 0 & 3 & 0 \\ 3 & 2 & 3 \\ 0 & 3 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = 1.0$$

Con lo cual obtenemos la siguiente respuesta en el simulador antes mencionado:

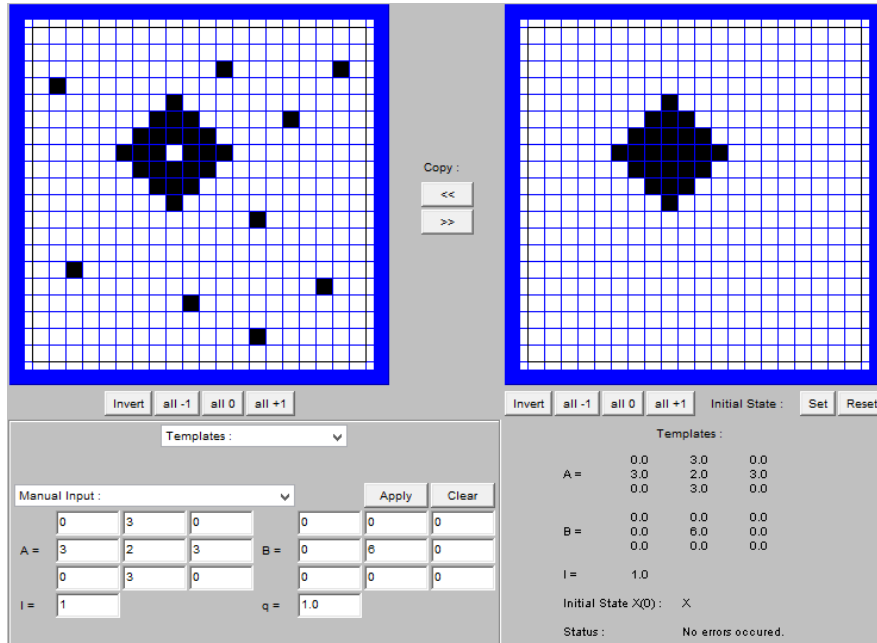


Fig. 1.7. Ejemplo Removedor de Ruido

1.6.2 Extractor de bordes para imágenes binarias

La obtención de bordes es importante ya que permite reducir la cantidad de información de la imagen y solo quedarse con la información más esencial de la imagen; las consideraciones para este ejemplo son las siguientes: Si un pixel es de color blanco su salida debe de ser blanco, si un pixel es de color negro y todos sus vecinos son de color blanco su salida debe de ser blanco de otro modo la salida es de color negro; éstos se ejemplifican en la siguiente figura.

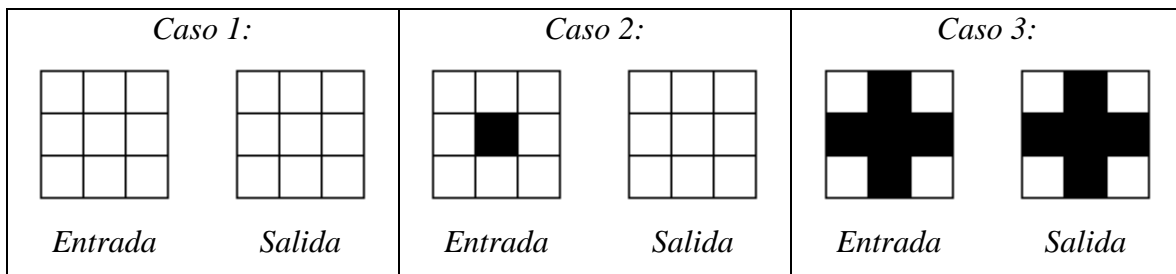


Fig. 1.8. Pixeles a considerar para extractor de bordes, en estos casos el pixel central será el procesado.

Para ejemplificar el uso de la CNN en el extractor de bordes, se utilizan las siguientes plantillas y valor de umbral, estos valores son tomados de [5]:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & -2 & 0 \\ -2 & 5 & -2 \\ 0 & -2 & 0 \end{bmatrix}, I = -1.0$$

Con lo cual obtenemos la siguiente respuesta:

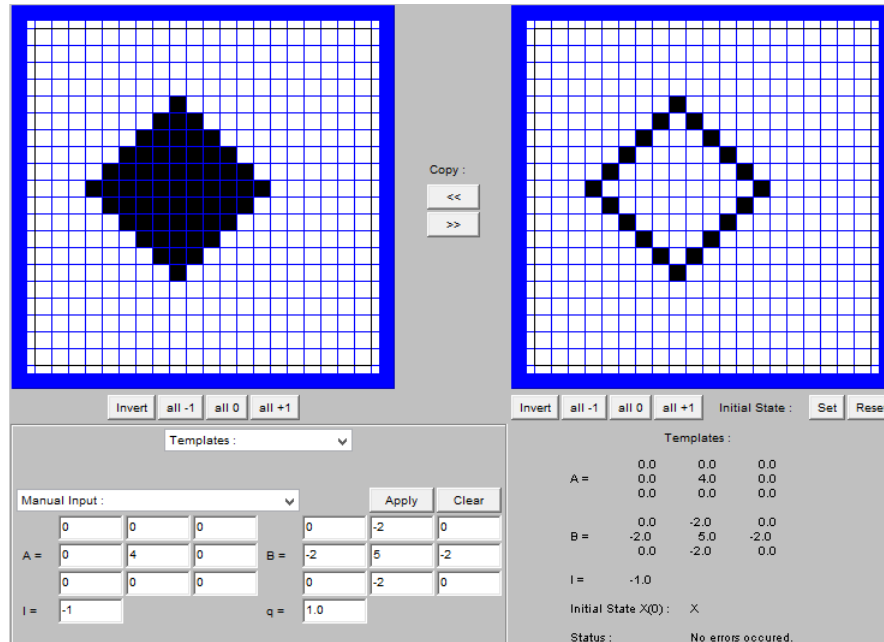


Fig. 1.9. Ejemplo Extractor de bordes

1.6.3 Extractor de bordes para imágenes en grises

Parecido al anterior, sin embargo, en esta ocasión se procesan imágenes en escala de grises, lo cual significa que los valores de entrada ya no solo serán 1's o -1's sino tendrán valores intermedios, para esto se utilizarán las siguientes plantillas, estos valores son tomados de [2].

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}, I = -0.5$$

Cuya respuesta es la siguiente:

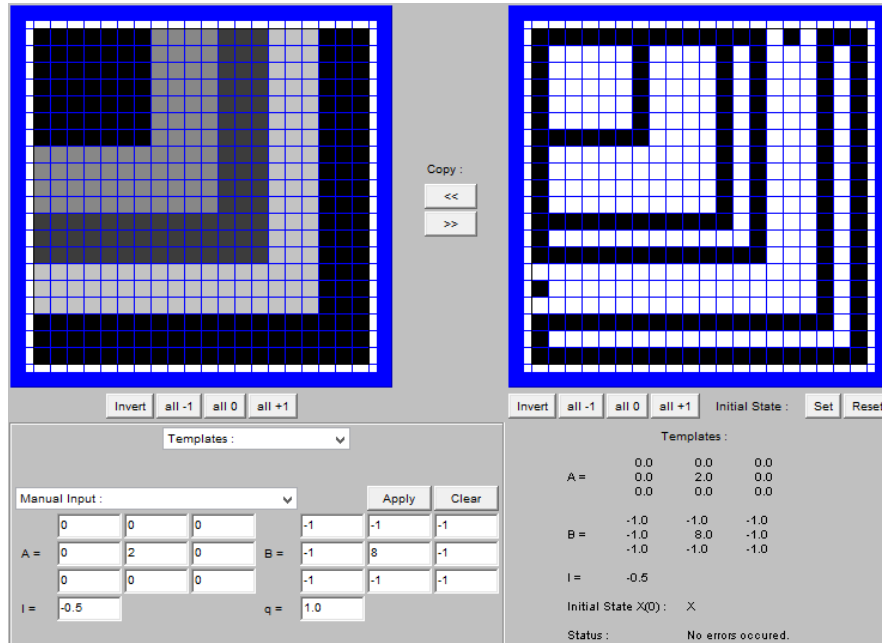


Fig. 1.10. Ejemplo Extractor de Bordes para imágenes en escala de Grises

1.6.4 Extractor de sombras

En este procesamiento se hace una proyección de la imagen, en este caso hacia la izquierda; para este ejemplo se hacen las siguientes consideraciones: si un pixel es negro su salida siempre será negro, si un pixel es blanco y el pixel a la derecha es blanco su salida será en blanco y finalmente si un pixel es blanco y el pixel a la derecha es color negro su salida será negro, esto se ejemplifica en la siguiente figura.

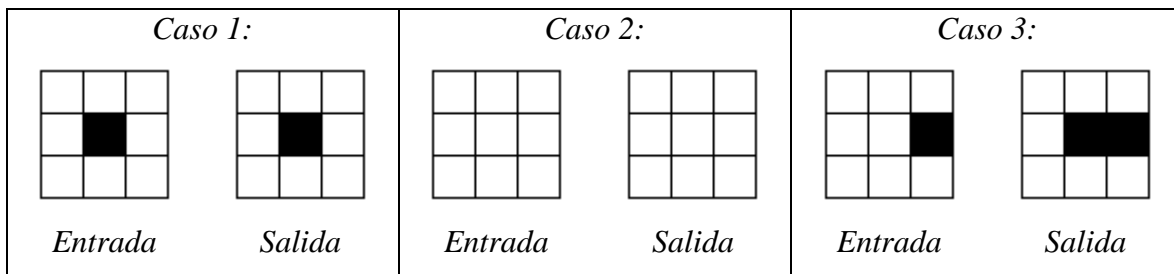


Fig. 1.11. Pixeles a considerar para extractor de sombra, en estos casos el pixel central será el procesado.

Para realizar esto se utilizarán las siguientes plantillas y valor de umbral, los cuales son obtenidos de [5]:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2.1 & 2 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.9 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = 1.0$$

Con lo cual obtenemos la siguiente respuesta:

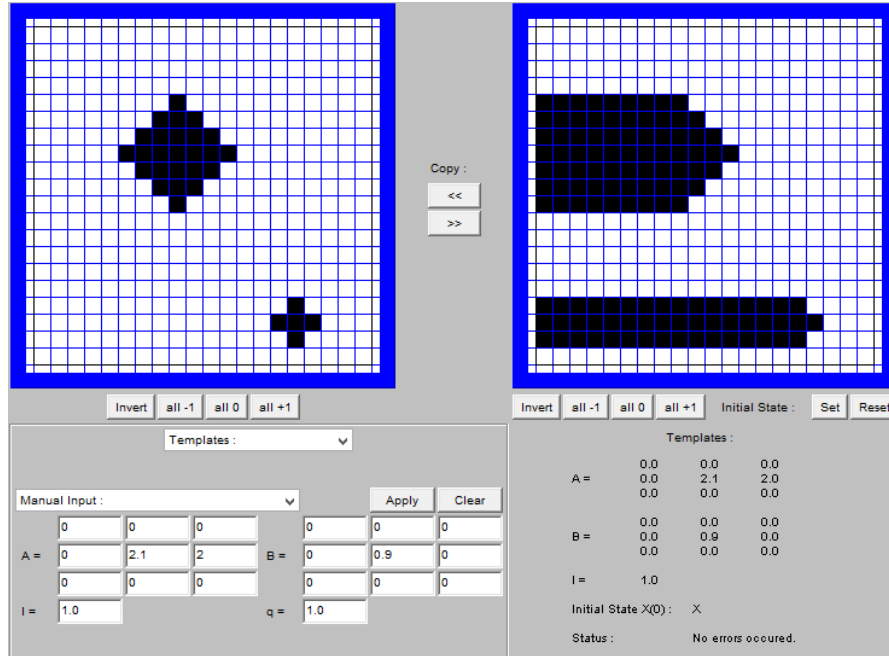


Fig. 1.12. Ejemplo Extractor de Sombras

En este ejemplo hay que hacer una aclaración: la proyección de sombra dependerá del tiempo de procesamiento de la CNN, así conforme avance el tiempo se irá expandiendo la sombra en la CNN, sin embargo, para este trabajo se ha determinado que se detenga la simulación hasta llegar a un punto de saturación.

1.6.5 Detector de conectividad global

Este ejemplo puede ser visto como un sistema que encuentra la salida en un laberinto; para este ejemplo se toman en consideración los siguientes puntos: La salida del laberinto tendrá un ancho de un pixel además de que éste tendrá que ser blanco, si el pixel es negro su salida tendrá que ser de color negro, si la entrada es un pixel blanco y tres de sus vecinos son de color negro su salida tendrá que ser de color negro de otra forma será de color blanco. En la siguiente figura se muestra los puntos anteriormente dichos.

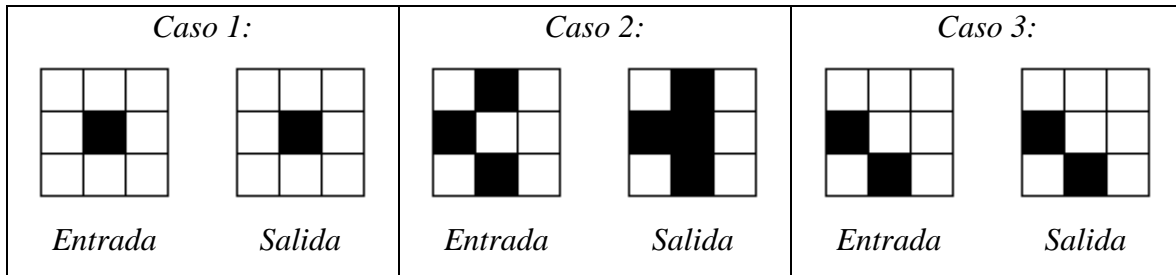


Fig. 1.13. Píxeles a considerar para el detector de conectividad global, en estos casos el píxel central será el procesado.

Para mostrar esto se utilizarán las siguientes plantillas y valor de umbral.

$$A = \begin{bmatrix} 0 & 4.4 & 0 \\ 4.4 & 3.6 & 4.4 \\ 0 & 4.4 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 10.7 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = 7.0$$

Con lo cual obtenemos la siguiente respuesta:

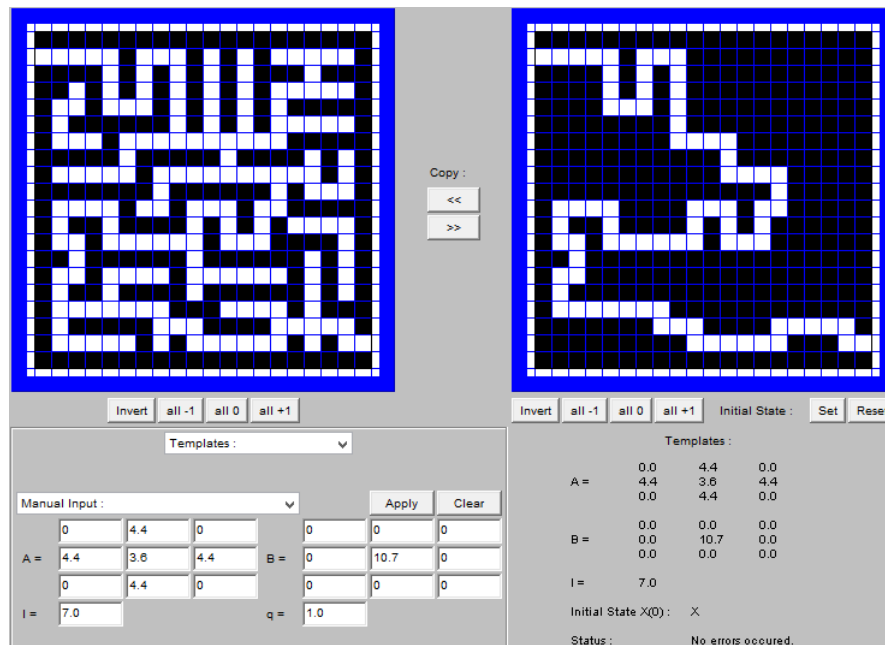


Fig. 1.14. Ejemplo de Detector de conectividad global

1.7 Conclusiones

En este capítulo se han introducido los conceptos básicos de la CNN, estos conceptos incluyen el funcionamiento de la CNN y la arquitectura de una CNN. También se describió la unidad básica de las CNN llamadas neuronas o celdas, además se mostraron las ecuaciones que describen el funcionamiento de una neurona, se realizó una mención especial en las

plantillas debido a que éstas son muy importantes dentro de las CNN ya que indican a ésta qué tipo de procesamiento realizarán, ya sea para el procesamiento de imágenes o para realizar otro tipo de procesamientos que pueda realizar la CNN, como el resolver ecuaciones diferenciales. También fueron descritos los métodos para realizar la búsqueda de las plantillas.

Todos los conceptos anteriormente mencionados son importantes ya que con base a ellos se realizará tanto un simulador como una implementación física en un FPGA de una CNN.

Dentro de este capítulo se demostró el uso de la CNN como procesador de imágenes utilizando diferentes tipos de plantillas, algunos de estos procesamientos fueron el removedor de ruido, extractor de bordes, extractor de sombras y detector de conectividad global.

1.8 Referencias

- [1] Leon O. Chua and L. Yang, "Cellular Neural Networks: Theory & Applications", IEEE Trans. Circuits and Systems, vol. CA-35, 1988, pp. 1257-1290.
- [2] Leon O. Chua and Tamás Roska, "Cellular neural networks and visual computing: Foundations and applications", Cambridge U.K., Cambridge University Press 2004, Ch. 2, pp. 7-34.
- [3] Valerio Cimagalli and Marco Balsi, "Cellular Neural Networks: A Review", E. Caianiello (ed): "Neural Nets WIRN Vietri-93", World Scientific, Singapore, 1994, pp. 55-84.
- [4] H. Harrer, "Multiple Layer Discrete-Time Cellular Neural Networks Using Time-Variant templates", IEEE Trans. On Circuits and Systems II vol. 40, no 3, March 1993, pp. 191-199.
- [5] Jesus Ezequiel Molinar Solís, "Circuito Integrado Analógico CMOS con Arquitectura de Red Neuronal Celular", M.C. I.E. CINVESTAV, México, D.F. 2002.
- [6] S. Parmaksizoğlu and M. Alçı, "A Novel Cloning Template Designing Method by Using an Artificial Bee Colony Algorithm for Edge Detection of CNN Based Imaging Sensor", Sensors, vol. 11, no. 12, pp. 5337-5359, May 2011.
- [7] Leon O. Chua and P. Thiran, "An Analytic Method for Designing Simple Cellular Neural Networks", IEEE Trans. On Circuits and Systems, vol. 38, no 11, November 1991, pp. 1332-1341.
- [8] I. Fijfar and F. Bratkovic, "Design of Monotonic Binary-Valued Cellular Neural Networks", CNNA '96, Fourth IEEE International Workshop on Cellular Neural Networks and their Applications, Seville, 1996.
- [9] M. Hanggi and G.]S. Moschitz, "An exact and direct analytical method for the design of optimally robust CNN templates", IEEE Trans. On Circuits and Systems, vol. 46, no. 2, 1999, pp. 304-311.
- [10] Zarandy, A. "The Art of Template Design", International Journal of Circuit Theory and Applications, 27, pp. 5-23, 1999.

- [11] J. A. Nossek, "Design and Learning with Cellular Neural Networks", Proc. IEEE International Workshop on Cellular Neural Networks and their Applications, Rome, Dec. 1994, pp. 137-146.
- [12] C. Guzelis and S. Karamahut, "Recurrent Perceptron Learning Algorithm for Completely Stable Cellular Neural Networks", Proc. IEEE International Workshop on Cellular Neural Networks and their Applications. Rome, 1994, pp. 177-182.
- [13] B. Mirzai, Z. Cheng and G. S. Moschytz, "Learning Algorithms for Cellular Neural Networks", Proc. IEEE Int. Symp. Circuits Systems, Moterey CA. June 1998, vol. 3, pp. 159-162.
- [14] Zou, F.; Schwarz, S.; Nossek, J. A., "Cellular Neural Network Design Using a Learning Algorithm". In Proc, IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '90), Budapest, Hungary, 16-19 December 1990, pp. 73-81.
- [15] Tetlaff, R., Wolf, D. A. "Learning Algorithm for the Dynamics of CNN with Nonlinear Templates Part I: Discrete-Time Case". In Proc. IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '96), Seville, Spain, 24-26 June, 1996, pp. 461-466.
- [16] T. Kozek, T. Roska and Leon O. Chua, "Genetic Algorithm for CNN Template Learning", IEEE Trans. On Circuits and Systems I, vol. 40, pp. 392-402, June 1993.
- [17] Shuo Y. W. and Lin C. T. "Image Descreening by GA-CNN Based Texture Classifications", Trans. Circuits and Systems, 2004, vol. 51, pp. 2287-2299.
- [18] Koji Nakai and Akio Ushida, "Design Technique of Cellular Neural Network", Electronics and Communications in Japan, Part 3, vol. 78, No. 3, 1995.

Capítulo 2. Multiplexado en tiempo de una CNN

2.1 Introducción

Una forma de procesar una imagen es haciéndolo como en [1], donde cada pixel es mapeado dentro de una CNN, esto es que cada pixel será procesado por una neurona que conforma la CNN.

Dentro de un punto de vista de implementación en hardware y software, éste es un enfoque muy exhaustivo debido a la cantidad de neuronas que se requerían. Por ejemplo, para imágenes de 10×10 pixeles se tendría un total de 100 pixeles, con lo cual con el enfoque antes mencionado deberíamos de tener una CNN de 10×10 neuronas, donde cada neurona procesará un pixel que compone la imagen y tener así un total de 100 neuronas. Sin embargo, actualmente las imágenes pueden llegar a ser mucho mayores de 512×512 pixeles, con lo cual tendríamos un total de 262,144 pixeles con lo cual deberíamos de tener una CNN de 512×512 neuronas, con un total de 262,144 neuronas. Como vemos en este enfoque, la CNN requerirá la misma cantidad de neuronas, lo cual lo hace inviable debido al consumo de recursos que para ésta se requieren. Una alternativa a este enfoque es multiplexar la CNN en la imagen como se muestra en [2-4].

La principal ventaja de esta técnica de multiplexar la CNN, radica en el hecho de que no se requiere una CNN del mismo tamaño que la imagen a ser procesada, sino que basta una CNN de tamaño reducido para poder procesar imágenes de diversos tamaños. Sin embargo, esta metodología conlleva ciertos errores, los cuales veremos posteriormente, además de que hay que tener ciertas consideraciones en el multiplexado para poder procesar las imágenes de cualquier tamaño.

2.2 Pasos para multiplexar en tiempo una CNN

Para multiplexar la CNN, primero que nada se debe de tomar en cuenta el tamaño de la CNN con la cual se va a trabajar. Por ejemplo, una CNN con tamaño de 4×4 neuronas, una de 8×8 neuronas, etc., y así tener bloques de la imagen con la cual se va a trabajar del mismo tamaño que la CNN, después se procederá a procesar cada uno de estos bloques en algún orden lexicográfico.

2.2.1 Obtención de bloques

Para ejemplificar de una mejor manera la obtención de los bloques de la imagen, lo explicaremos con un ejemplo. De esta forma, tengamos en cuenta una CNN de 8×8 neuronas y una imagen como la que se muestra en la Fig. 2.1, la cual es de 24×24 píxeles (donde cada cuadrado representa un píxel). En este caso iremos tomando bloques de 8×8 píxeles, con lo cual cada uno de estos bloques será procesado por el bloque de la CNN.

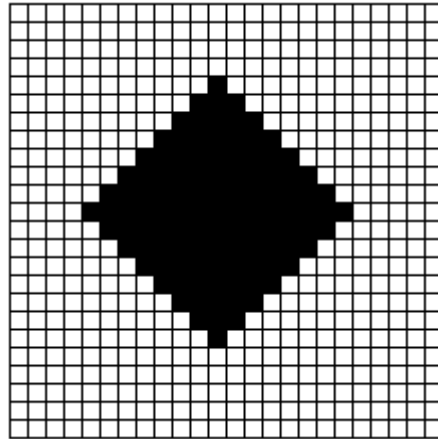


Fig. 2.1. Imagen de muestra.

Por lo tanto, tendríamos una imagen conformada por 3×3 bloques con un total de 9 bloques, donde cada bloque está compuesto de 8×8 píxeles. Cada uno de los bloques, como mencionamos anteriormente, será procesado por la CNN de 8×8 neuronas. En la Fig. 2.2 se muestra cómo se dividiría esta imagen para formar dichos bloques.

Hay que considerar que no siempre la imagen será un múltiplo de la CNN que se tenga, es por esto que si la imagen no es múltiplo de la CNN se puede proceder a procesar la imagen de diferentes formas. Generalmente se puede aplicar alguna de las formas de las condiciones de frontera vistas en 1.2.5.

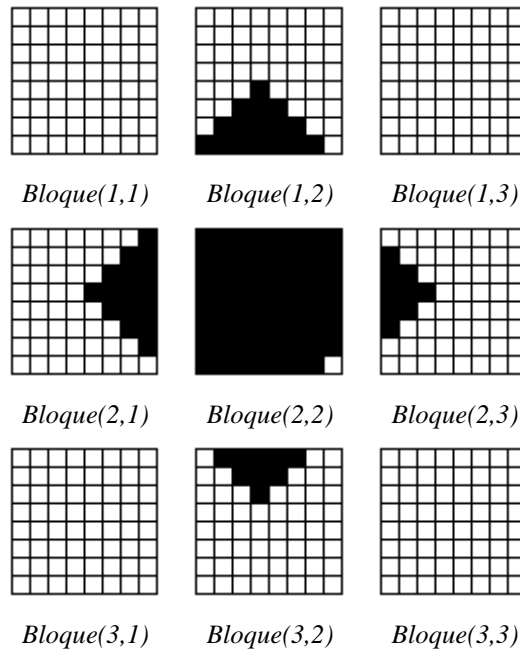


Fig. 2.2. Conformación de los bloques en una imagen dada.

2.2.2 Orden lexicográfico

Los bloques escaneados por la CNN deben de seguir un orden lexicográfico, esto es un orden en el cual los bloques serán procesados, por ejemplo, en la Fig. 2.3 se muestra cómo se realiza un escaneo de izquierda a derecha y de arriba hacia abajo.

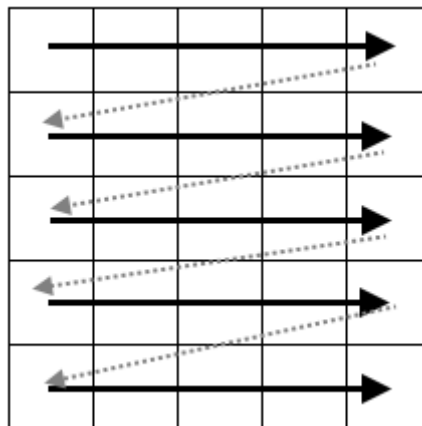


Fig. 2.3. Orden lexicográfico de Izquierda a derecha de arriba hacia abajo

Para ejemplificar este orden lexicográfico tomemos la imagen en bloques vista en la Fig. 2.2, donde se irán procesando los bloques en el siguiente orden: *Bloque(1,1)*, *Bloque(1,2)*, *Bloque(1,3)*, *Bloque(2,1)*, ..., *Bloque(3,3)*.

Otro ejemplo de orden lexicográfico sería como el mostrado en la Fig. 2.4, donde éste comienza a realizar el procesamiento de los bloques de arriba hacia abajo y de izquierda a derecha. Para ejemplificar este orden lexicográfico tomemos la imagen en bloques vista en la Fig. 2.2, donde se irán procesando los bloques en el siguiente orden: *Bloque(1,1)*, *Bloque(2,1)*, *Bloque(3,1)*, *Bloque(1,2)*, ..., *Bloque(3,3)*. En nuestro trabajo hemos utilizado un orden lexicográfico como el mostrado en la Fig. 2.3.

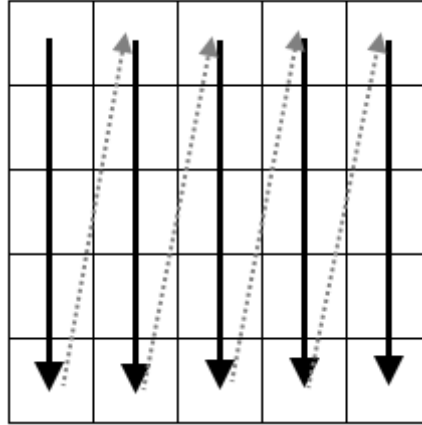


Fig. 2.4. Orden lexicográfico de arriba hacia abajo de izquierda a derecha

2.3 Errores en el multiplexado de la CNN

Otro punto a tomar en cuenta, es que en este enfoque de multiplexado de la CNN conlleva a dos tipos de errores en el cálculo de los píxeles en el borde para cualquier bloque de CNN, debido a que son calculados sin el efecto de sus píxeles vecinos.

Estos dos tipos de errores son mostrados en la ecuación (2.1) y la ecuación (2.2) respectivamente, las cuales corresponden a una CNN con vecindario de $r = 1$.

$$\varepsilon_{ij}^A = \sum_{i=1}^{i=3} a_{i,j+1} y_{i,j+1}(t) \quad (2.1)$$

$$\varepsilon_{ij}^B = \sum_{i=1}^{i=3} b_{i,j+1} \text{sign}(u_{i,j+1}) \quad (2.2)$$

Donde la ecuación (2.1) representa el error debido a la ausencia de las salidas de las neuronas vecinas y la ecuación (2.2) representa el error debido a la ausencia de las entradas de las neuronas vecinas y donde $sign(\cdot)$ representa la función *signo*, esta función es representada en la ecuación (2.3).

$$sign(x) = \begin{cases} +1, & \text{si } x > 0 \\ 0, & \text{si } x = 0 \\ -1, & \text{si } x < 0 \end{cases} \quad (2.3)$$

2.3.1 Reducción del error

Para reducir los errores en el procesamiento de los bloques vistos anteriormente, se realizan las siguientes consideraciones.

Para reducir el error de la ecuación (2.1) que es debida a la ausencia de las salidas $y_{i,j}$, se toma en cuenta lo siguiente.

- Se hace un traslape entre dos bloques como es mostrado en la Fig. 2.5, el ancho mínimo de este traslape debe de ser dos veces el radio r del vecindario de la CNN. Se recomienda que este traslape sea mayor, sin embargo, para el procesamiento de una CNN donde importa más el resultado final que la evolución de las neuronas que conforman la CNN, con un vecindario con 2 veces r es suficiente.

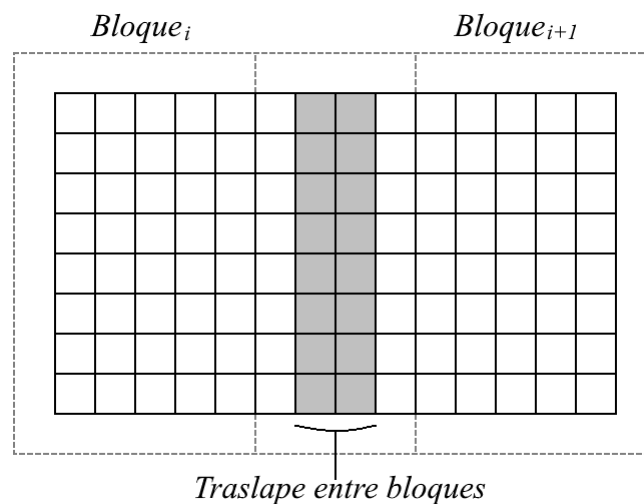


Fig. 2.5. Traslape entre bloques de CNN.

En este punto hay que hacer la consideración de que debido al traslape de los bloques, existen pixeles que son procesados dos veces, sin embargo, el valor guardado y utilizado como resultado final depende del bloque. Por ejemplo, en la Fig. 2.5, los valores guardados para el primer bloque “*Bloque_i*” se guardan los valores de salida y estado desde la primera hasta la penúltima columna que conforman el bloque; para el bloque “*Bloque_{i+1}*” los valores guardados de salida y estado corresponde desde la segunda columna hasta la última. En otras palabras, la salida y estado del traslape (parte sombreada en la figura) corresponden para el lado izquierdo del bloque “*Bloque_i*” y la parte derecha para el bloque “*Bloque_{i+1}*”.

Otra consideración al traslape entre bloques, es que debido a esto, se incrementará la cantidad de bloques que hay dentro de la imagen.

El error mostrado en la ecuación (2.2), es debido a la ausencia de las señales de entrada y la función “Saturación Lineal Simétrica” de los pixeles vecinos. Para minimizar este error se toma en cuenta la siguiente consideración:

- Se coloca un cinturón de pixeles alrededor de la sub-imagen de la imagen original igual al radio de la CNN, por ejemplo si $r = 1$, se coloca un cinturón con ancho de un pixel, o si $r = 2$, se coloca un cinturón con ancho de 2 pixeles. Este punto se muestra en la Fig. 2.6 el cual muestra un cinturón con ancho de 1 pixel que corresponde a un vecindario con $r = 1$.

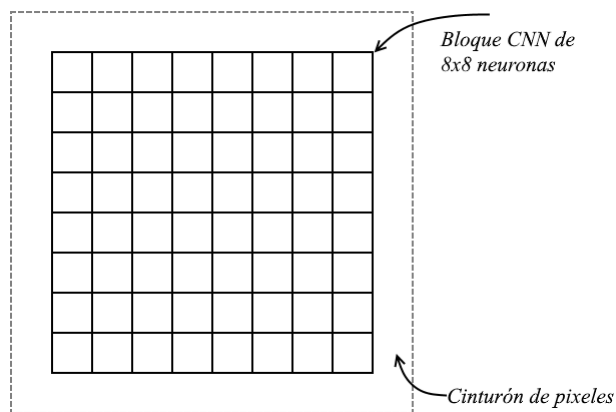


Fig. 2.6. Cinturón de pixeles en un bloque.

2.4 Repetición del tipo de bloques

Para un simulador, hay que tomar en cuenta que hay que ahorrar tiempo en el procesamiento de los datos de la CNN. Como vemos, al momento de multiplexar la CNN en la imagen, se pueden tener muchos casos donde los bloques puedan tener todos los píxeles en blanco o en negro. Tomando en cuenta lo anterior, basta con solo simular la primera vez que aparezca alguno de estos casos y guardar los valores obtenidos en la simulación. Después, cada vez que aparezca alguno de los casos anteriores, solo se necesitará rescatar los valores obtenidos en la primera simulación de estos.

Para ejemplificar esto tomemos el caso visto en la Fig. 2.2. En esta imagen observamos que tenemos 4 bloques que son totalmente blancos (sin considerar el traslape anteriormente mencionado) los cuales son *Bloque(1,1)*, *Bloque(1,3)*, *Bloque(3,1)* y *Bloque(3,3)*, así solo sería necesario procesar un solo bloque blanco. En este caso basta con solo procesar el *Bloque(1,1)* y guardar sus resultados de salida y estado, para así cuando aparezcan el resto de los bloques blancos, en este caso el *Bloque(1,3)*, *Bloque(3,1)* y *Bloque(3,3)*, bastara con recuperar los datos guardados de estado y salida del *Bloque(1,1)* y colocarlo en el resto de los bloques. Con esta forma solo se procesarían un total de 6 bloques y no los 9 bloques obtenidos en la imagen, ahorrando de esta forma tiempo y cantidad de bloques procesados.

2.5 Algoritmo general para el multiplexado en tiempo de una CNN

Antes de realizar el multiplexado de una CNN para el procesamiento de una imagen, hay que crear la CNN. Esta CNN puede ser definida de diversas formas como es mostrado en [3] y [4], para poder así tener un mejor control sobre el algoritmo a implementar en el multiplexado. Además, hay que definir el tamaño de la CNN con la cual vamos a trabajar, y tenerlo en consideración para algunos de los pasos que conforman el algoritmo.

Un algoritmo general para el multiplexado de la CNN es mostrado en la Fig. 2.7. A continuación se explicarán cada uno de los bloques en forma breve.

Carga los parámetros: Este bloque se encarga de leer los diferentes parámetros que se pasarán a la CNN para realizar el multiplexado, como lo son los parámetros de plantillas y umbral, así como el valor de la frontera que se utilizará.

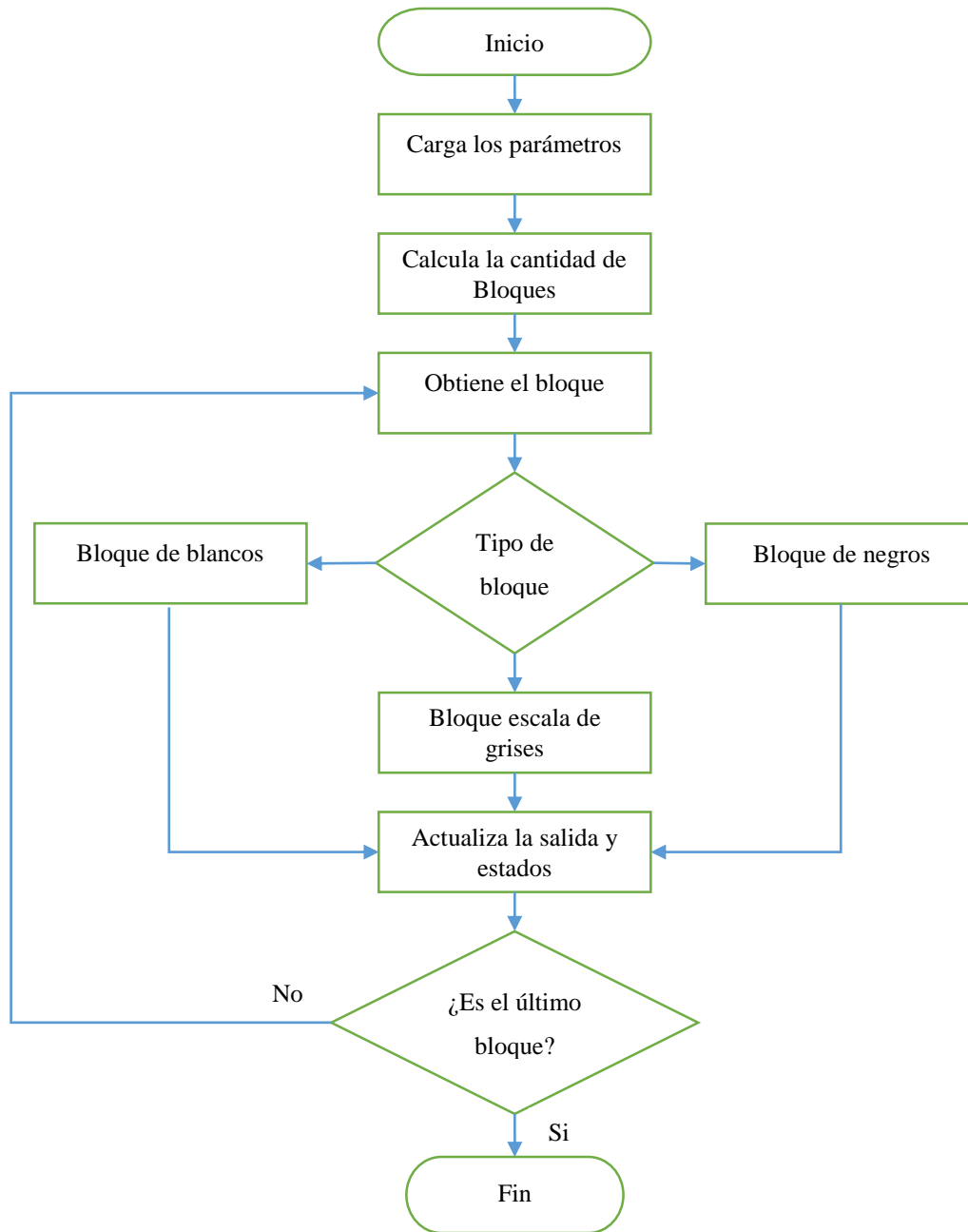


Fig. 2.7. Algoritmo general para el multiplexado en tiempo de una CNN.

Calcula la cantidad de bloques: Este bloque se encarga de calcular la cantidad de bloques que conforman la imagen, sin tomar en cuenta si hay bloques repetidos de blancos y negros, solo se realiza un cálculo general del total de bloques, esto basado en el tamaño de la CNN utilizada, además de tomar en cuenta el traslape y su tamaño.

Obtiene el bloque: Este bloque del algoritmo se encarga de llevar el control de los bloques que ya han sido procesados, además de indicar cuál será el próximo bloque a ser procesado, esto tomando en cuenta el orden lexicográfico utilizado en el procesamiento de la imagen.

Tipo de bloque: Este bloque del algoritmo se encarga de determinar el tipo de bloque de pixeles que ha obtenido el bloque del algoritmo “*Obtiene el bloque*”, este indica si es un bloque blanco, negro, o uno de escala de grises (si se determina que es de escala de grises puede ser que también sea un bloque que tiene solo blancos y negros), y así lo envía a “*Bloques de blancos*”, “*Bloques de negros*” o “*Bloque escala de grises*” dependiendo de cuál se determinó.

Bloque de blancos: Este bloque se encarga de realizar el procesamiento del primer bloque de pixeles que está totalmente en blanco. Después de esto, guarda los resultados obtenidos de la salida de la CNN y su estado, para poder ser utilizadas por los siguientes bloques totalmente en blancos y que no sean procesados de nuevo, de esta forma se ahorra tiempo en el procesamiento del multiplexado en tiempo de la CNN, ya que no se requiere realizar nuevamente los cálculos para estos bloques.

Bloque de negros: Este bloque se encarga de realizar el procesamiento del primer bloque de pixeles que está totalmente en negro, después de esto guarda los resultados obtenidos de la salida de la CNN y su estado, para poder ser utilizadas por los siguientes bloques totalmente en negros y que no sean procesados de nuevo, de esta forma se ahorra tiempo en el procesamiento del multiplexado en tiempo de la CNN, ya que no se requiere realizar nuevamente los cálculos en estos bloques.

Bloque escala de grises: Este bloque del algoritmo se encarga de realizar el procesamiento del bloque de pixeles que contiene tanto pixeles blancos como negros, o ya sea del procesamiento de un bloque que contenga pixeles en escala de grises.

Actualiza la salida y estados: Este bloque se encarga de guardar los resultados tanto de la salida como el estado de las neuronas, entregados por el procesamiento de la CNN. Esto lo realiza siguiendo lo mencionado anteriormente sobre el hecho de que dependiendo del tipo de bloque, qué resultados de las neuronas va a guardar, ya sea si guarda hasta la penúltima

columna o hasta la última o si guarda empezando desde la primera columna o desde la segunda columna de la CNN.

¿Es el último bloque?: Este bloque del algoritmo se encarga de determinar si el bloque procesado anteriormente es el último de los bloques o si no lo es, con esto determina si pasa a procesar el siguiente bloque o finalmente muestra los resultados con el bloque “Fin”.

Fin: Este bloque se encarga de mostrar los resultados obtenidos en el procesamiento de la imagen utilizando esta técnica de multiplexado en tiempo de la CNN.

2.6 CNN para el multiplexado en tiempo

Una parte importante en el multiplexado de la CNN es realizar la CNN. Existen diversas formas de realizar esta CNN, como por ejemplo como se realiza en [1] y [5]. En las referencias [3] y [4] se dan dos ejemplos de cómo se pueden realizar estas CNN de diferente forma.

Como vemos en la referencia [3] se utiliza SIMULINK para crear la CNN y después multiplexarla, y en la referencia [4] se crea la CNN utilizando un lenguaje de programación. Para nuestro trabajo, utilizaremos SIMULINK, sin embargo, veremos más adelante que nuestra CNN creada en SIMULINK difiere a la creada en [3].

Esta CNN creada a través de un modelo por computadora debe de comportarse lo más parecido a una CNN diseñada con tecnología digital como lo es el FPGA, es por esto que deben de tomarse las consideraciones necesarias para poder realizar esto.

2.7 Conclusiones

En este capítulo se trató el tema del multiplexado de una CNN y su importancia de realizarlo, además de las desventajas y los errores que se generan al realizar esta técnica para el procesamiento de imágenes, sin embargo, se mostraron algunas formas de reducir estos errores creados.

También se mostró un algoritmo general para el multiplexado de la CNN, además de dar una pequeña explicación de cada uno de los pasos que conforman dicho algoritmo; este

algoritmo es importante, ya que con base a este se realizará este trabajo de tesis. Como veremos más adelante, la CNN creada en este trabajo para el procesamiento de imágenes, utiliza gran parte del algoritmo mostrado en este capítulo.

2.8 Referencias

- [1] Leon O. Chua and L. Yang, "Cellular Neural Networks: Theory & Applications", IEEE Trans. Circuits and Systems, vol. CA-35, pp. 1257-1290, 1988.
- [2] Chi-Chien Lee and José Pineda de Gyves, "Time-Multiplexing CNN Simulator", IEEE Inter. Symp. On Circuits and Systems, 1994, vol. 6, pp. 407-410, 1994.
- [3] A. A. H. EL-Shafei and M.I. Sobhy, "A Time-Multiplexing simulator for Cellular Neural Network (CNN) using SIMULINK", IEEE Inter. Symp. On Circuits and Systems, 1998, vol. 3, pp. 167-170.
- [4] A. A. H. EL-Shafei and M.I. Sobhy, "A Time-Multiplexing simulator for Cellular Neural Network (CNN)", Cellular Neural Networks and Their Applications Proceedings, 1998 Fifth IEEE International Workshop on, pp. 14-18, 1998.
- [5] Leon O. Chua and Tamás Roska, "Cellular neural networks and visual computing: Foundations and applications", Cambridge U. K., Cambridge University Press, 2004, ch. 2, pp. 7-34.

Capítulo 3. Descripción de una CNN multiplexada en tiempo con MATLAB y SIMULINK

3.1 Introducción

Como mencionamos en el capítulo anterior, realizar una CNN como la mencionada en [1] tomaría demasiados recursos tanto en software como en hardware, es por eso que se han propuestos trabajos como los mostrados en [2-4], donde se ha multiplexado la CNN dando buenos resultados. Estos trabajos se han enfocado principalmente en el procesamiento de imágenes.

El multiplexado de la CNN para imágenes representa una buena opción para su realización tanto en software y hardware debido a la disminución de cantidad de neuronas que se requieren para realizar el procesamiento de imágenes y con esto la reducción de recursos que se requieren para realizarla, ya sea en software o hardware, es por ello que en este trabajo realizaremos una CNN para el procesamiento de imágenes, esta CNN será de 4×4 neuronas con vecindario de $r = 1$.

En este capítulo, nos concentraremos en realizar un simulador de la CNN para posteriormente realizarla en hardware utilizando un FPGA y describiéndola en VHDL. Para realizar el simulador utilizaremos SIMULINK y MATLAB. Sin embargo, para realizar este simulador y poderlo hacer que coincida lo más posible a la que se realizará en VHDL se tomarán criterios en los cálculos, debido a que en VHDL la descripción del circuito lo haremos digital por lo tanto solo podremos utilizar cierta resolución numérica para estos.

La importancia de tener un simulador para nuestra CNN que realizaremos en hardware radica en el hecho de que necesitamos saber cómo se comportará nuestra CNN a la resolución de problemas, este simulador será utilizado para el procesamiento de imágenes.

3.2 Algoritmo de una CNN multiplexada en tiempo

El algoritmo utilizado para crear el simulador con MATLAB y SIMULINK fue basado en el mostrado en la Fig. 2.7, sin embargo, este algoritmo lo adecuaremos para poder ser implementado utilizando tanto MATLAB como SIMULINK.

Este algoritmo es mostrado en la siguiente figura. Como vemos, parece diferir de aquel mostrado en la Fig. 2.7, sin embargo, como mencionamos anteriormente, este algoritmo toma en cuenta la parte del lenguaje en el que lo vamos a implementar. Además de esto, en el algoritmo se toma en cuenta que la creación de la CNN fue a través de SIMULINK.

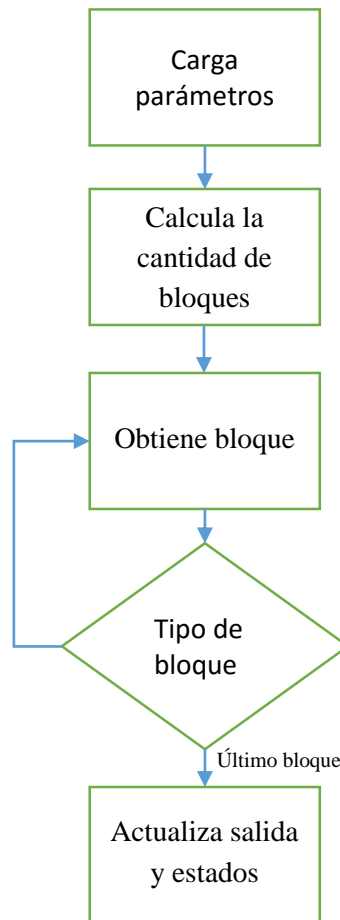


Fig. 3.1. Algoritmo utilizado para el multiplexado de la CNN en tiempo.

3.3 Creación de una CNN de 4×4 Neuronas en SIMULINK

El simulador creado en este capítulo fue derivado de la ecuación de estado normalizada de la CNN, ecuación (1.3) vista en el primer capítulo, y para poder realizarlo se utilizó MATLAB y SIMULINK. El uso de SIMULINK es debido a que dicha ecuación de estado puede ser descrita en forma de bloques, además de que podemos crear la neurona y replicarla para crear la matriz de 4×4 neuronas y éstas procesen su información en forma paralela, de forma similar a como lo haría una CNN real.

Antes de crear la CNN de 4×4 neuronas y de explicar el algoritmo general para el multiplexado de CNN, explicaremos cómo se creó la neurona para después reproducirla y crear la CNN; dicha neurona puede ser descrita en forma de bloques como se muestra en la siguiente figura.

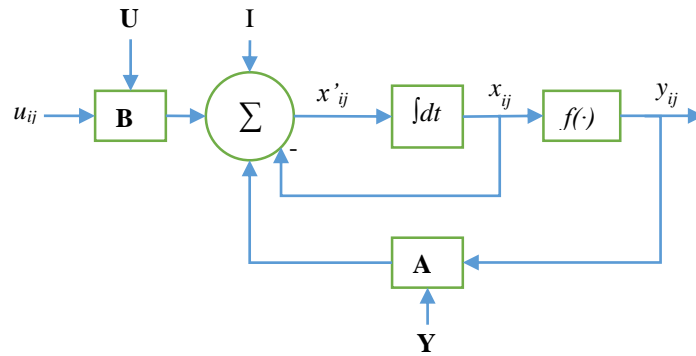


Fig. 3.2. Diagrama a bloques de una neurona $C(i, j)$.

Donde \mathbf{Y} representa la matriz de las salidas de las neuronas dentro del vecindario, \mathbf{U} representa la matriz de las entradas de las neuronas dentro del vecindario, \mathbf{A} representa la matriz de retroalimentación, \mathbf{B} representa la matriz de control, I representa el umbral, el símbolo Σ representa la sumatoria de los valores de las sumatorias de las plantillas y el valor negativo del estado $x_{i,j}$, $f(\cdot)$ representa la función de transferencia y finalmente $y_{i,j}$ representa la salida de la neurona $C(i,j)$.

3.3.1 Integrador Euler

Para poder resolver la ecuación de estado de la CNN, se requiere resolver la ecuación diferencial y para poder realizar esto se requiere de hacer una integración numérica, como se observa en el diagrama a bloques, lo que quiere decir que cada neurona tendrá un integrador. Además debemos tener en cuenta que las neuronas trabajan en forma paralela lo que quiere decir que todas van resolviendo su respectiva ecuación diferencial al mismo tiempo que las demás. Es por esto que utilizaremos SIMULINK ya que permite describir la neurona y reproducirla para que resuelvan sus respectivas ecuaciones de estado en forma paralela.

Aunque SIMULINK incluye bloques para realizar integración numérica, se optó por crear otro bloque debido a que se implementará en hardware posteriormente, y así tener una simulación funcional con la cual se pueda verificar la implementación.

Aunque existen diversos métodos de integración numérica como Euler, Euler hacia atrás, Euler mejorado, y los métodos de Runge-Kuta, se optó por realizar el método de Euler hacia atrás debido a su sencillez y facilidad de realizarlo en hardware. En la siguiente tabla se muestran las diferentes fórmulas de los métodos de integración numérica.

Tabla 3.1. Fórmulas de integración numérica.

Método	Formula
Euler	$y_{n+1} = y_n + f(t_n, y_n)h$
Euler hacia atrás	$y_{n+1} = y_n + f(t_{n+1}, y_{n+1})h$
Euler mejorado	$y_{n+1} = y_n + \frac{f_n + f(t_n + h, y_n + hf_n)}{2}h$
Runge-Kuta	$y_{n+1} = y_n + h \left(\frac{k_{n1} + 2k_{n2} + 2k_{n3} + k_{n4}}{6} \right)$ <p>Donde:</p> $k_{n1} = f(t_n, y_n),$ $k_{n2} = f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_{n1}\right),$ $k_{n3} = f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_{n2}\right),$ $k_{n4} = f(t_n + h, y_n + hk_{n3}).$

Donde $n = 0,1,2 \dots$

Como mencionamos, la CNN se describirá en MATLAB y SIMULINK lo cual incluye el desarrollo del integrador en SIMULINK; en la figura 3.3, se muestra el integrador a bloques en SIMULINK, el cual sigue la siguiente fórmula del integrador Euler hacia atrás:

$$y_{n+1} = y_n + f(t_{n+1}, y_{n+1})h, \quad n = 0,1,2 \dots \quad (3.1)$$

Donde h es el paso fijo, al cual colocamos el valor de 0.001, y $f(t_n, y_n)$ es la función a ser evaluada.

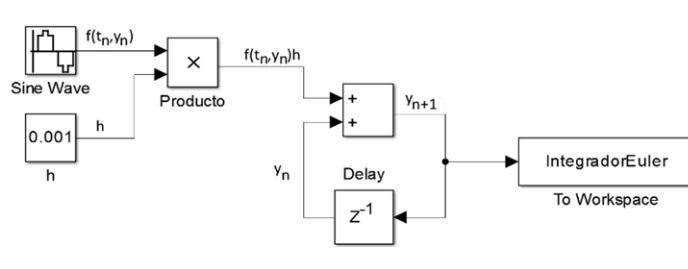


Fig. 3.3. Integrador Euler realizado en SIMULINK

Si recordamos, es importante tener las condiciones iniciales de la ecuación de estado de la CNN, como fue mencionado en el punto 1.2.2. Es por ello, que la condición inicial es cargada dentro del bloque “Delay”, la cual contiene un parámetro llamado condición inicial y es ahí donde colocamos la condición inicial del integrador, que a su vez será la condición inicial de la neurona. En la Fig. 3.4 mostramos el parámetro de la condición inicial; recordemos que esta condición inicial deberá encontrarse en el rango de $[-1,1]$. En la siguiente figura mostramos dónde se coloca la condición inicial mencionada anteriormente dentro del bloque “Delay”, marcada con un óvalo.

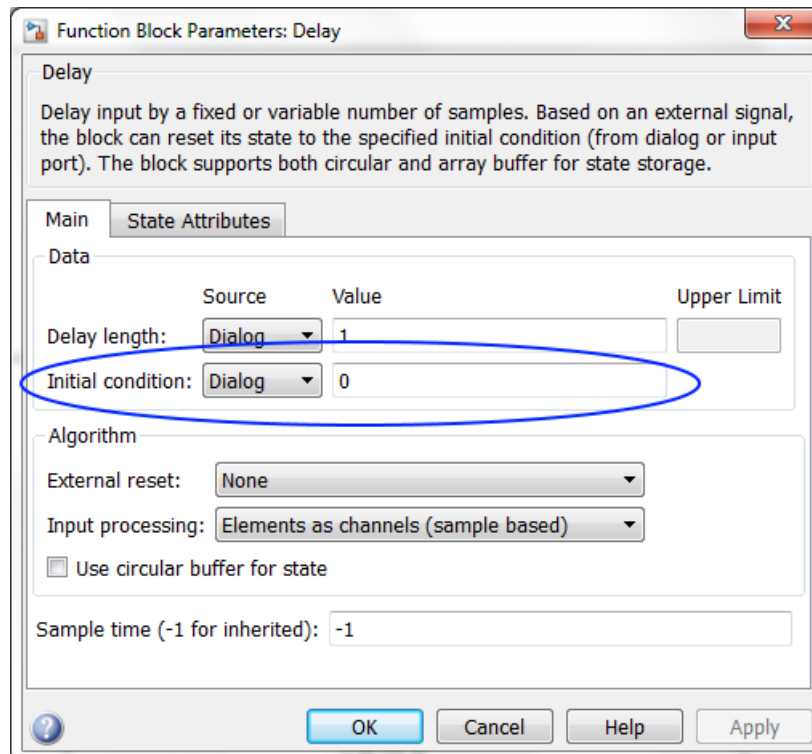


Fig. 3.4. Parámetro donde se colocará la condición inicial de la neurona.

Para comprobar el funcionamiento del integrador Euler hacia atrás realizado con bloques en SIMULINK, lo compararemos con el integrador discreto ya establecido en el mismo SIMULINK, el cual contiene diversos métodos de integración, sin embargo, sólo se hará una comparativa con el método de Euler hacia atrás. La respuesta del integrador a una señal sinusoidal discreta con frecuencia de 10Hz y una amplitud de 1V, es mostrada en la Fig. 3.5.

Como observamos, ambas respuestas se parecen mucho, sin embargo, para hacer una mejor comparativa entre ambas señales de respuesta a la señal sinusoidal de entrada realizaremos una figura donde contenga ambas respuestas, la Fig. 3.6 muestra dichas respuesta. Como observamos, la señal del integrador a bloques sigue la respuesta obtenida con el integrador de SIMULINK. Esto muestra que el integrador realizado a bloques dentro de SIMULINK se comporta de forma adecuada, lo cual nos indica que es factible realizar el integrador dentro de VHDL, dándonos resultados aceptables.

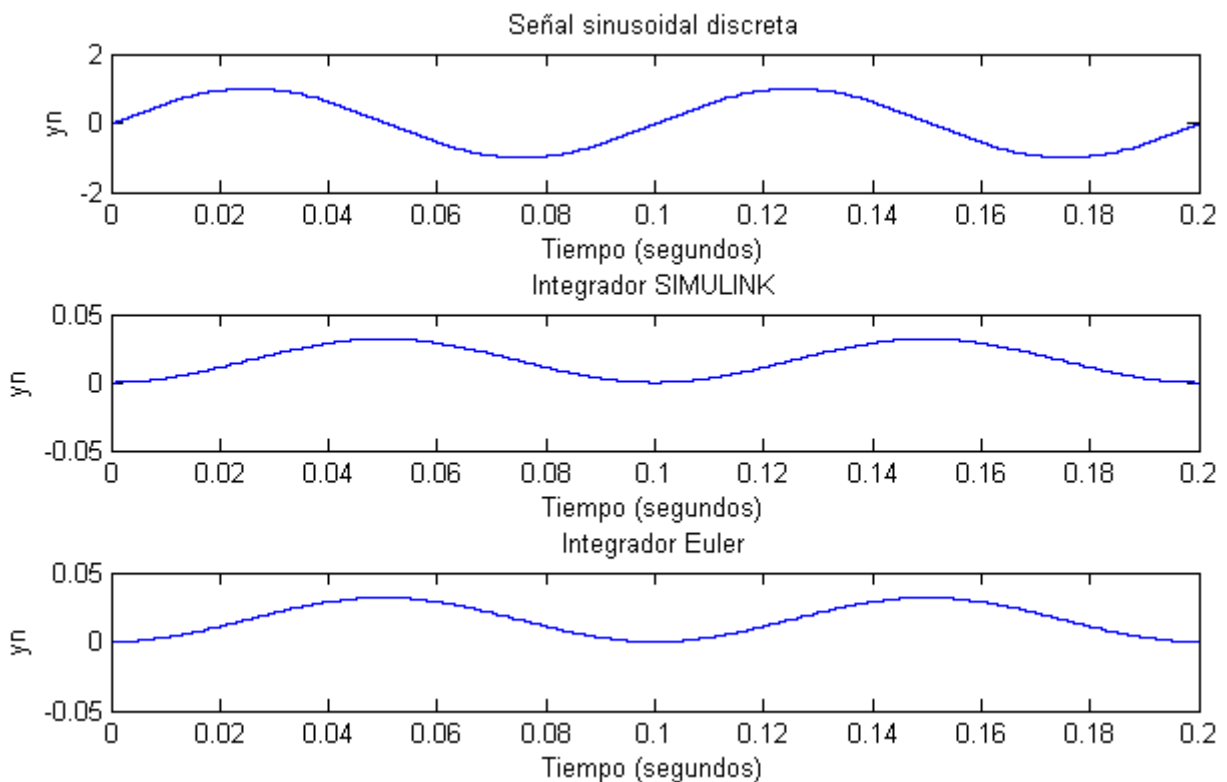


Fig. 3.5. Respuesta del integrador de SIMULINK y el Euler a bloques a una señal sinusoidal discreta.

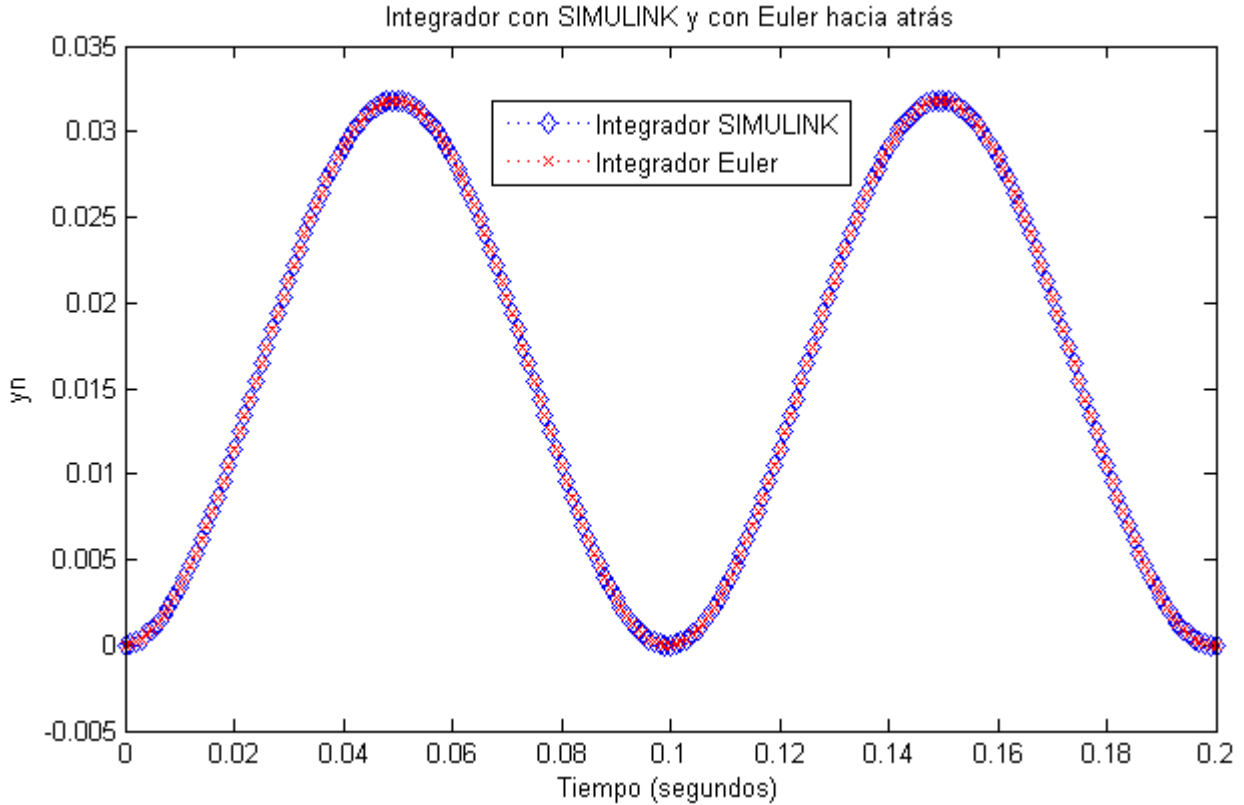


Fig. 3.6. Respuesta del integrador Euler con bloques y el integrador de SIMULINK a una entrada sinusoidal.

3.3.2 Neurona en SIMULINK

Para crear la CNN utilizando MATLAB Y SIMULINK, debemos primero describir la unidad básica de la CNN, esto es, la neurona. La descripción de la neurona la realizaremos en SIMULINK siguiendo la descripción a bloques de la neurona como vimos en el punto 3.3. El primer paso para realizar esto es resolver la ecuación diferencial, es por eso que en el punto anterior se desarrolló el integrador Euler con SIMULINK.

La descripción de la ecuación (1.3), nos indica que debemos hacer un producto entre cada uno de los componentes de las plantillas con su respectiva salida o entrada de las neuronas vecinas:

$$\sum_{C(k,l) \in N_r(i,j)} A(i,j;k,l)v_{ykl}(t) + \sum_{C(k,l) \in N_r(i,j)} B(i,j;k,l)v_{ukl} \quad (3.2)$$

Esto es hacer la sumatoria de cada uno de los productos de los componentes de la plantilla de retroalimentación $A(i, j; k, l)$ por la respectiva salida de las neuronas del vecindario $v_{ykl}(t)$, como fue mostrado en la ecuación (1.10) y de la misma forma hacer la sumatoria de cada uno de los productos de los componentes de la plantilla de control $B(i, j; k, l)$ por las respectivas entradas de las neuronas del vecindario, como fue mostrado en la ecuación (1.12).

De la ecuación diferencial del estado de la neurona, observamos que hay que sumar el valor de umbral y restarle el valor del estado de la misma neurona a las sumatorias y así poder integrar el resultado para encontrar el estado de la neurona. Con esto vemos que esta ecuación diferencial depende de su mismo estado, además a la salida del estado de la neurona hay que hacerla el argumento de la función de transferencia de Saturación Lineal Simétrica para poder así obtener la salida de la neurona.

La descripción de la neurona a bloques la cual sigue la ecuación del estado de la neurona normalizada es mostrada en la Fig. 3.7.

Ahí comprobamos que se han colocado las sumatorias de las plantillas de retroalimentación remarcada como “Sumatoria de retroalimentación”, la de control remarcada como “Sumatoria de control”, el valor de umbral marcado como I; se muestra el integrador Euler hacia atrás remarcado como “Integrador Euler” y la función de saturación lineal simétrica remarcada como “función de salida”.

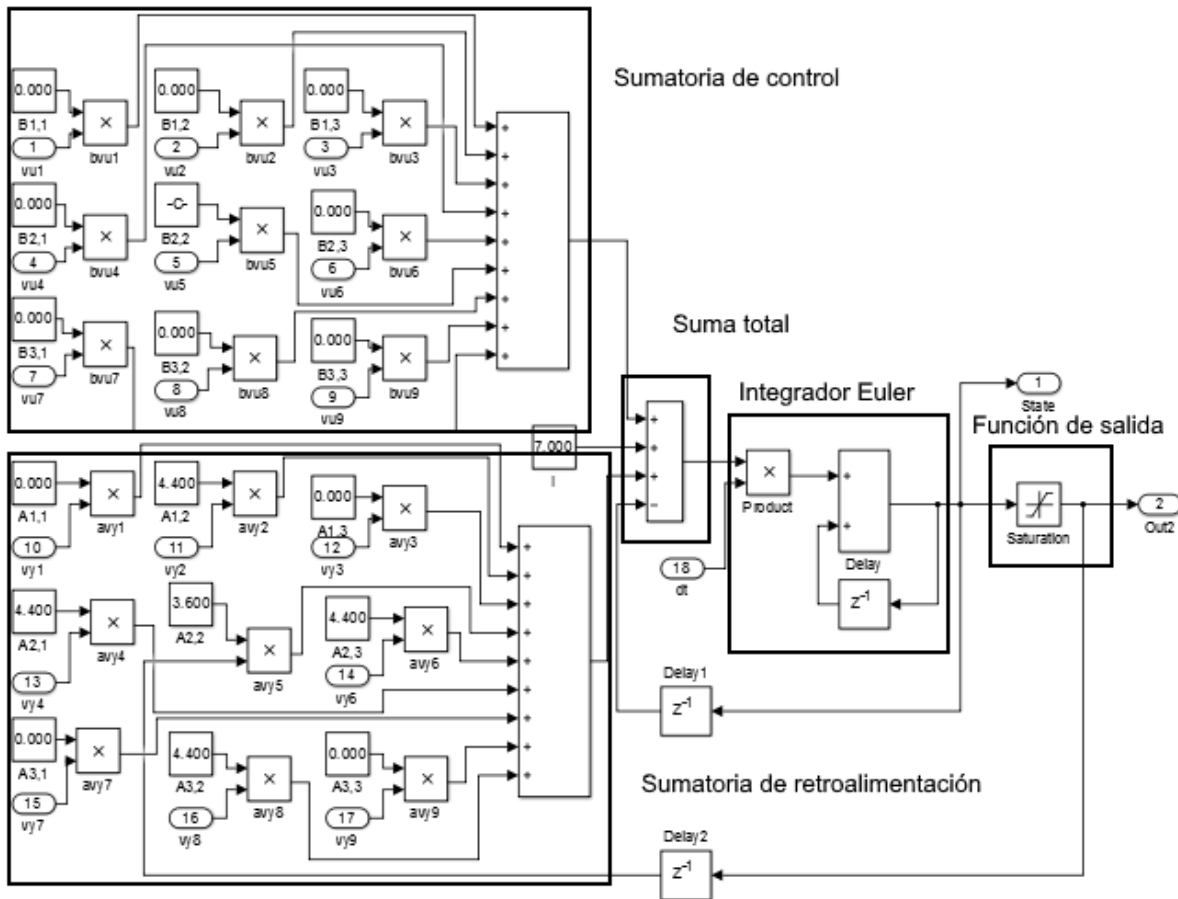


Fig. 3.7. Descripción de la neurona en SIMULINK.

3.3.3 Comprobación del estado estable

Una vez creada la neurona, se tendrá que tomar en cuenta que debido a que se va a multiplexar la CNN es necesario detener la simulación cuando todas las neuronas lleguen a un estado estable, para poder así ahorrar tiempo en el simulado. En otras palabras, cuando todas las neuronas cumplan con el valor, ya sea ≤ -1 o $\geq +1$, se detenga la simulación y así poder continuar con la simulación del siguiente bloque de imagen, de lo contrario la CNN siempre encontraría un solo bloque de la imagen y no pasaría al siguiente bloque. Hacer esto también nos ahorra tiempo debido que aunque puede ser definido el tiempo máximo de simulación de la CNN en SIMULINK, muchas veces la CNN llega al estado estable antes de que se cumpla ese tiempo.

Este bloque de comprobación del estado estable es colocado a la salida de la neurona; en la siguiente figura, se muestra cómo se realizó dicho bloque, el cual entrega a su salida un valor lógico de “1” cuando cumple alguna de las condiciones anteriormente mencionadas.

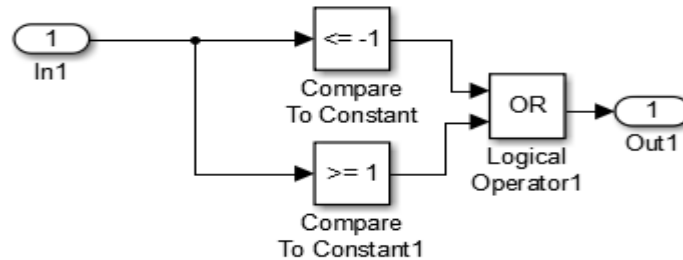


Fig. 3.8. Comprobador de estado estable, colocado a la salida de la neurona.

Además, éste se colocó dentro de un subsistema para hacerlo replicable y poderlo colocar en cada una de las salidas de la neurona y después colocar un bloque que esté observando el valor lógico (se colocó un bloque AND, para comprobar los valores lógicos de las salidas) entregado por este subsistema creado para comprobar la salida de las neuronas, y así cuando todas cumplan con alguna de las condiciones detener la simulación, para esto se colocó un bloque que detiene la simulación.

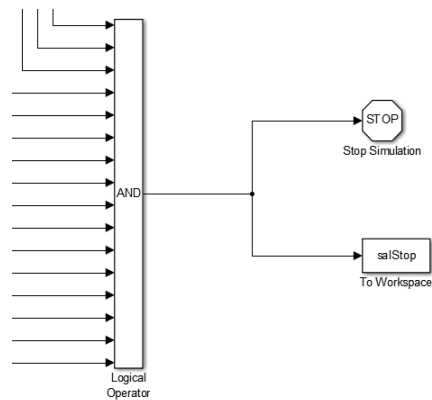


Fig. 3.9. Condición de paro de la CNN.

3.3.4 Condición de frontera

Para la CNN que se está desarrollando, utilizaremos la condición de frontera *Condición de frontera fija (Dirichlet)* mostrada en el punto 1.2.5. El valor que tomaremos para la condición de frontera será de 0. Debido a que todas las neuronas de frontera tendrán

el mismo valor de 0, solo será necesario colocar en las neuronas de frontera dicha condición con un bloque constante de 0.

3.3.5 Prueba de la neurona en SIMULINK

Para comprobar el funcionamiento de la neurona creada en SIMULINK, se realizará una simulación con las plantillas para quitar ruido. Esto lo haremos en una imagen de 3×3 pixeles, de las cuales sólo el pixel que se encuentra en el centro tendrá un valor diferente al resto. Además de que los pixeles que rodean a éste, se tomará el valor contrario a la neurona central como las condiciones de frontera de la neurona.

Los valores de prueba serán los siguientes:

- Los valores de frontera serán pixeles negros; esto es, valores de uno.
- El valor de entrada de la neurona será un pixel blanco; esto es, un valor de menos uno.
- La condición inicial de la neurona será la misma que la entrada a la neurona en el tiempo 0; esto es, un valor de menos uno.
- El paso de tiempo para el integrador Euler de la neurona será de 0.001s.
- Las plantillas para quitar ruido serán las siguientes:

$$A = \begin{bmatrix} 0 & 3 & 0 \\ 3 & 2 & 3 \\ 0 & 3 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = 1.0$$

Establecidos los valores de prueba de la neurona, se procede a comprobar el funcionamiento de la neurona.

En la siguiente figura tenemos los resultados obtenidos durante la simulación del funcionamiento de la neurona. Como observamos, el valor obtenido en la salida evoluciona de un valor de -1 a $+1$, además de esto también observamos la evolución del estado de la neurona y como vemos, al igual que la salida, también evoluciona teniendo valores que cambian de un valor de -1 a un valor aproximado de $+2.78$.

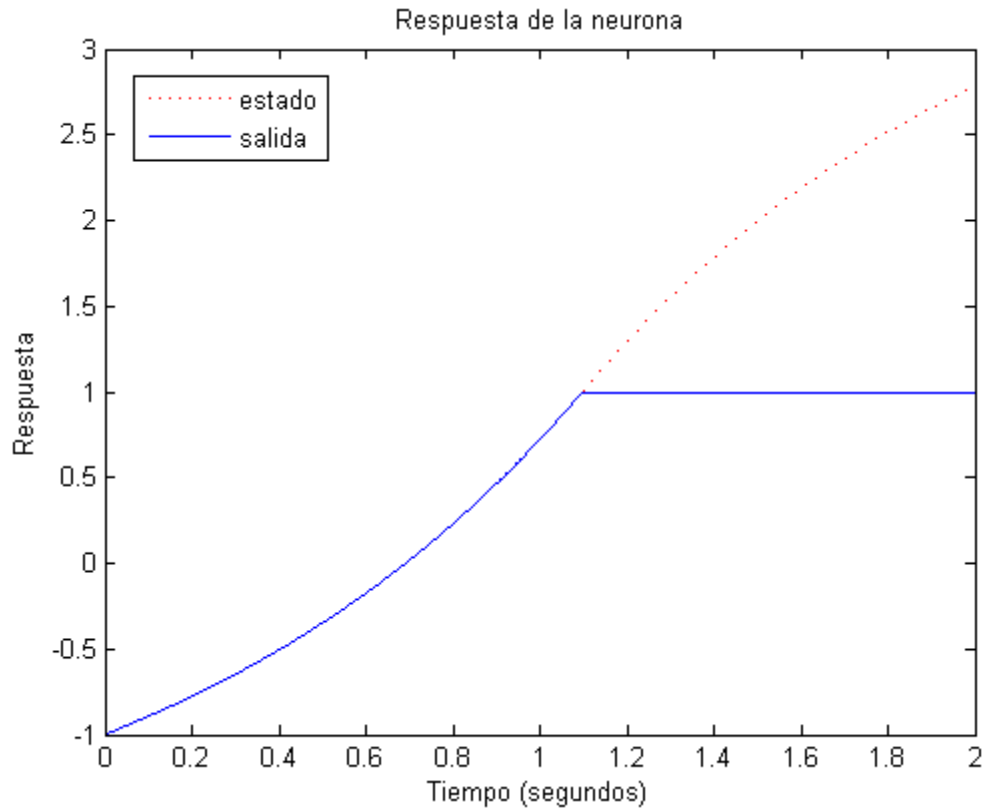


Fig. 3.10. Evolución de la neurona en el tiempo a una entrada de -1.

Con estos valores obtenidos, comprobamos el correcto funcionamiento de la neurona ya que el pixel procesado cambia de un valor blanco al negro.

3.3.6 Creación de la CNN de 4 × 4 Neuronas

Comprobado el funcionamiento de la neurona, es momento de crear la CNN, pero para esto se colocó la neurona como un bloque dentro de SIMULINK para poder facilitar la interconexión entre neuronas; esto es mostrado en la siguiente figura.



Fig. 3.11. Bloque de Neurona.

Donde vu1 a vu9 representan las entradas del vecindario de la neurona propia, vy1 a vy9 representan la salida de las neuronas del vecindario, dt es el paso de tiempo del integrador Euler hacia atrás (h dentro de la fórmula visto en 3.3.1), State representa el estado de la neurona y OutN representa la salida de la neurona.

La Fig. 3.12 muestra la CNN creada en SIMULINK.

En la Fig. 3.13 se muestra un acercamiento a la CNN de 4×4 neuronas creada en SIMULINK, donde se muestra la interconexión entre las diferentes neuronas. Como observamos, la interconexión se realizó utilizando “Buses” para hacerlo más entendible.

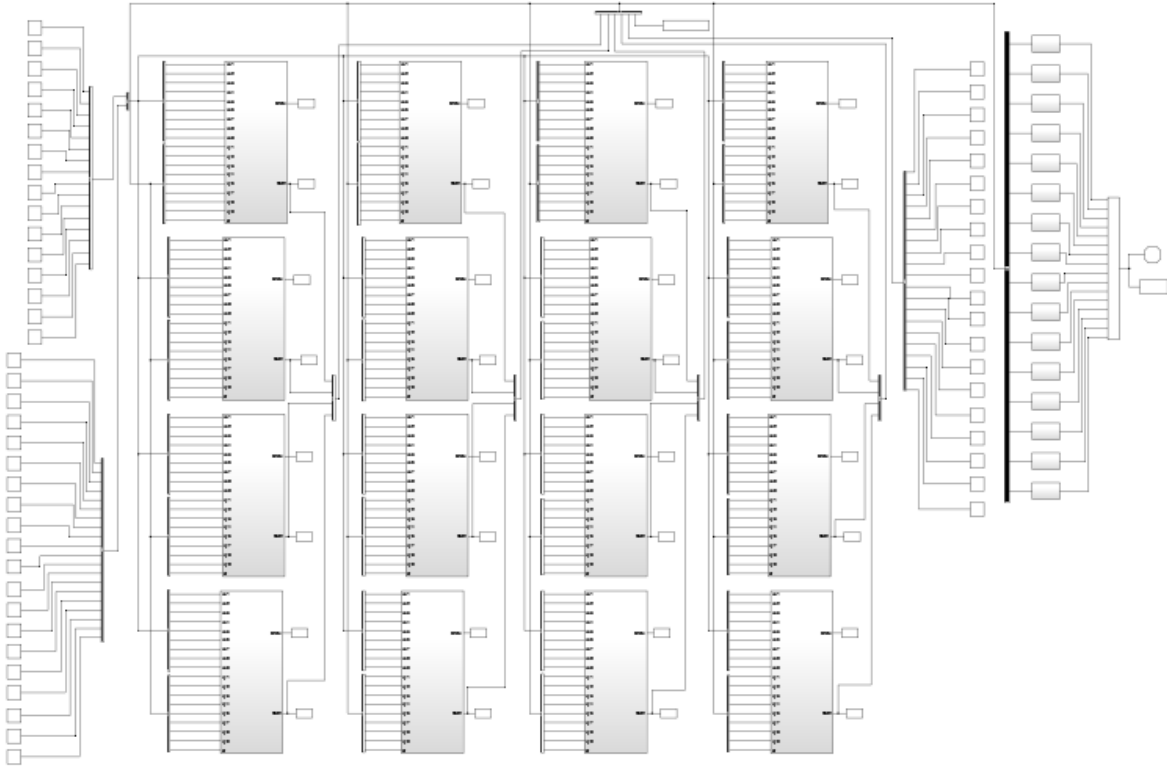


Fig. 3.12. CNN de 4x4 Neuronas creada en SIMULINK.

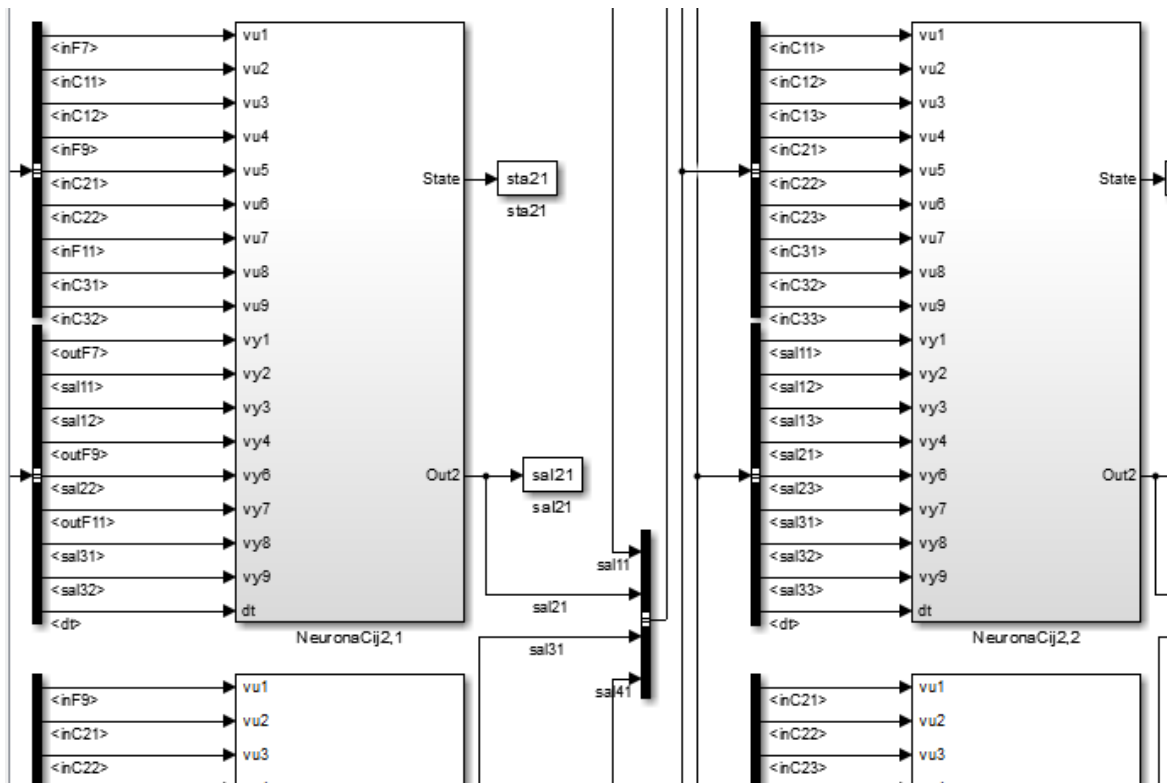
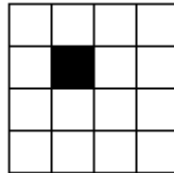


Fig. 3.13. Interconexión entre Neuronas dentro de la CNN.

Para comprobar el correcto funcionamiento de la CNN creada, se realizará un ejercicio similar al realizado con la neurona. Se realizará una simulación para quitar ruido, con las siguientes consideraciones:

- La imagen de prueba será la siguiente:



- La condición inicial de las neuronas será la misma que la imagen de entrada.
- El valor de frontera será de 0, como se mencionó anteriormente.
- El paso de tiempo del integrador Euler hacia atrás será de 0.001s.
- Las plantillas para quitar ruido serán las siguientes:

$$A = \begin{bmatrix} 0 & 3 & 0 \\ 3 & 2 & 3 \\ 0 & 3 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = 1.0$$

Establecidas las condiciones con las cuales se comprobará la CNN, se procede a realizar la simulación. En la Fig. 3.14 se muestra la evolución de las salidas de las neuronas pertenecientes a la CNN, donde “N11” representa la salida de la Neurona 11, “N12” la salida de la Neurona 12, “N21” la salida de la neurona 21 y así sucesivamente. Esta representación de las neuronas sigue como N_{ij} , donde N se refiere a una neurona, i y j, representan las coordenadas de la neurona dentro de la CNN.

En la Fig. 3.15 se muestran los estados obtenidos durante la simulación de la CNN, donde de la misma manera que la figura anterior, “ N_{ij} ” representa el estado de la Neurona.

Como observamos en las figuras anteriores, la Neurona N22 (en color rojo) cambia de un valor 1 a -1, en otras palabras cambia de un pixel negro a un pixel en blanco. Con esto comprobamos el funcionamiento correcto de la CNN.

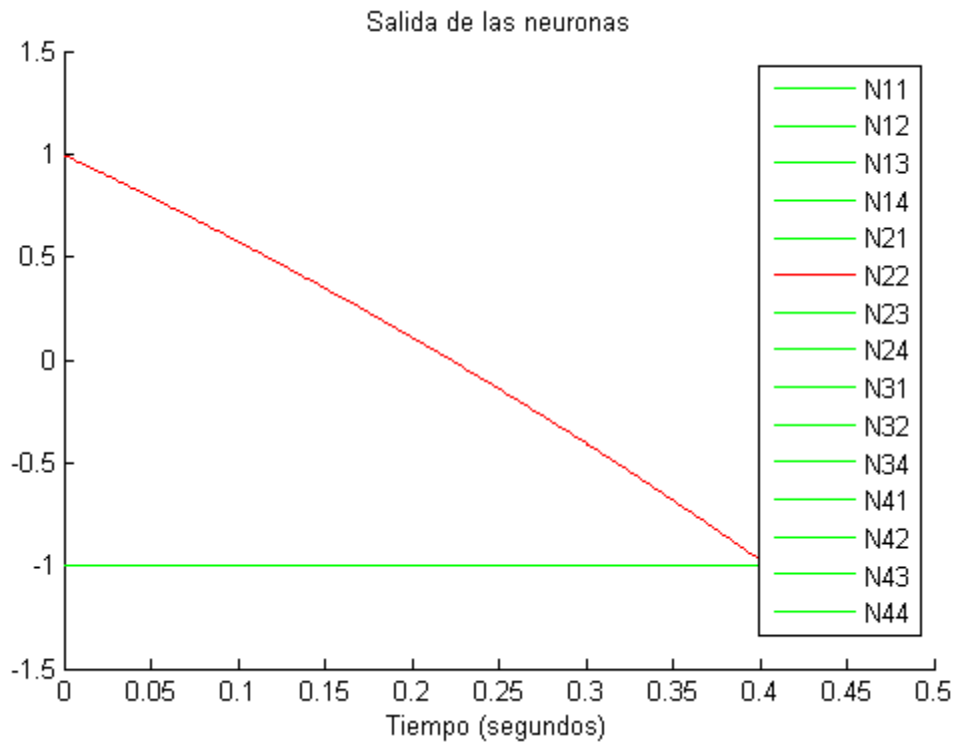


Fig. 3.14. Salida de las neuronas en la prueba del funcionamiento de la CNN.

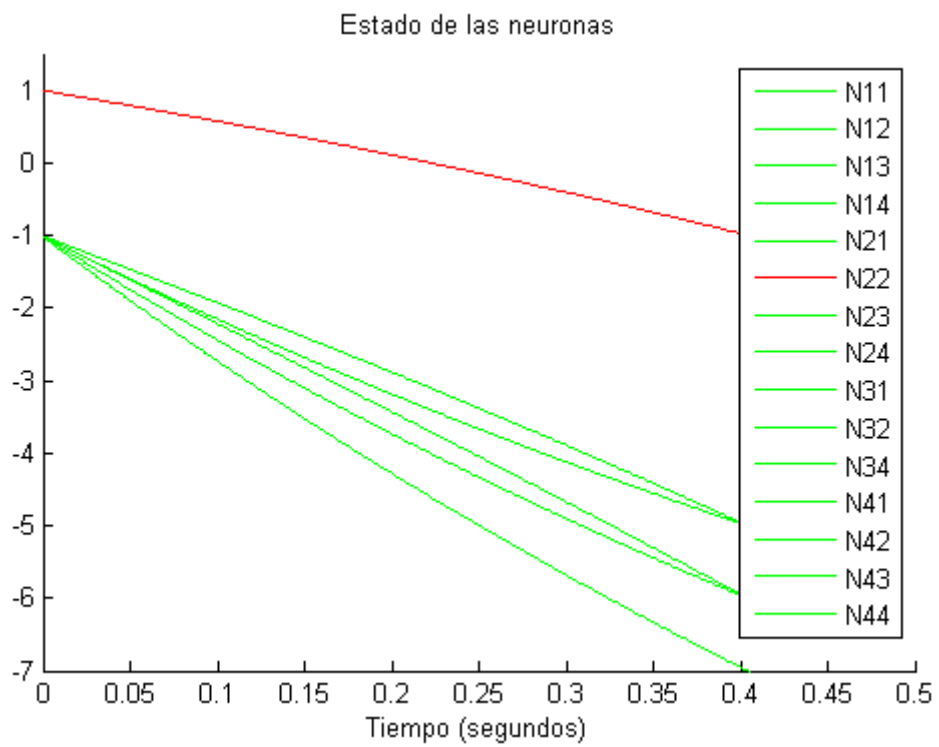


Fig. 3.15. Estado de las neuronas en la prueba del funcionamiento de la CNN.

3.4 Control de la CNN con MATLAB

Comprobado el funcionamiento de la CNN realizado en SIMULINK, se creó el código necesario para multiplexar la CNN, además de poder cargar la imagen a ser procesada y sus respectivos parámetros. También se despliega la imagen antes de procesar y después de procesarla, todo esto basándonos en el algoritmo visto en el punto 2.5 y utilizando el algoritmo creado para usarlo con MATLAB visto en el punto 3.2.

Los pasos del algoritmo se detallan a continuación, además de que se mencionarán las condiciones necesarias para reducir los errores que se mostraron en 2. En cada paso del algoritmo general se indicará a qué pasos corresponde del algoritmo visto en 3.2.

3.4.1 Proceso “Carga los parámetros”

En este primer paso se procede a realizar los siguientes pasos:

- Lee la imagen.
- Convierte la imagen a escala de grises o escala de blancos y negros.
- Convierte la imagen a una imagen tipo CNN.
- Lee los parámetros **A**, **B** e **I**.
- Carga los parámetros en la CNN de SIMULINK.

Cuando se lee la imagen en MATLAB, lo hace en valores RGB (por sus siglas en inglés de Red, Green and Blue). Estos valores van de un valor de 0 a 255 para cada uno de sus componentes, debido a esto y a que se procesarán imágenes en tonos de grises, es necesario convertir la imagen a color en tonos de grises.

Esta conversión de valores RGB a valores en tonos de grises se realizó con la función en MATLAB “*rgb2gray*”, la cual utiliza la siguiente fórmula:

$$Gray = 0.2989 * R + 0.5870 * G + 0.1140 * B$$

Donde *R* representa la componente en Rojo, *G* la componente en Verde y *B* la componente Azul en la imagen cargada.

Convertida la imagen a tonos de grises, se procede a comprobar que la imagen tiene valores de 0 a 255, donde 0 representa un pixel negro, 255 representa un pixel blanco y los

valores intermedios a éstos, son la representación de las tonalidades de grises. Sin embargo, como mencionamos anteriormente, la CNN procesa valores entre -1 a +1, es por esto que es necesario convertir los valores de 0 a 255 a valores de -1 a +1.

Lo anterior se puede observar en la siguiente tabla, donde se observa cómo es la representación de un pixel en MATLAB y cómo es ese mismo pixel dentro de la CNN:

Tabla 3.2. Representación de pixeles en MATLAB y CNN.

Pixel	MATLAB	CNN
Blanco	255	-1
Negro	0	+1

Para cambiar los valores numéricos entregados por MATLAB a los necesarios para que pueda ser procesado por la CNN, se utilizó la siguiente la fórmula:

$$x = 1 - \frac{2 * y}{255} \tag{3.3}$$

Donde x representa un pixel de CNN e y representa un pixel entregado por MATLAB. Para comprobar la ecuación, realizaremos el cálculo para los dos casos extremos donde se tiene un pixel en blanco y un pixel en negro, además de mostrar algunos ejemplos de pixeles intermedios, esto es mostrado en la siguiente tabla:

Tabla 3.3. Comprobación de la fórmula de conversión numérica.

Pixel CNN	Formula	Valor obtenido (pixel CNN)
Blanco (255)	$1 - \frac{2 * 255}{255}$	-1
191	$1 - \frac{2 * 191}{255}$	-0.498
127	$1 - \frac{2 * 127}{255}$	0.0039

64	$1 - \frac{2 * 64}{255}$	+0.498
Negro (0)	$1 - \frac{2 * 0}{255}$	+1

Como observamos en la tabla, la fórmula funciona de manera correcta. Sin embargo, si el pixel tiene un valor fuera de rango, la fórmula anterior nos entregará un valor erróneo, es por eso que es necesario asegurarse que los valores entregados al momento de convertir la imagen a tonos de grises contengan valores entre [0, 255].

Obtenidos los valores necesarios de los pixeles, es necesario pasar los valores de las plantillas y umbral a la CNN dentro de SIMULINK. Esto es realizado mediante una función, la cual se encarga de colocar dichos valores a la CNN. En el apéndice A se muestra el listado completo del programa realizado en MATLAB.

Después de esto, se procede a leer los parámetros de las plantillas y el umbral. Seguido a esto, los valores de las plantillas y el umbral son cargados dentro de la CNN creada en SIMULINK.

3.4.2 Proceso “Calcula la cantidad de bloques”

En este paso del algoritmo se procede a realizar lo siguiente:

- Comprueba el tamaño de la imagen.
- Calcula la cantidad de bloques en X e Y.

Antes de explicar la necesidad de comprobar el tamaño de la imagen, primero se explicará cómo se calcula la cantidad de bloques que serán procesadas por la CNN. El primer punto a tomar en cuenta para calcular la cantidad de bloques es el tamaño de nuestra CNN y después de esto tomar en cuenta el traslape que se estará realizando entre los bloques.

Para calcular la cantidad de bloques tanto en X como en Y, se utilizarán las siguientes ecuaciones, donde estas ecuaciones tomaron en cuenta el tamaño de la CNN y el traslape que existirá entre bloques:

$$\text{Bloque}Y = \frac{1}{2}(m - 2) \quad (3.4)$$

$$\text{Bloque}X = \frac{1}{2}(n - 2) \quad (3.5)$$

Donde *BloqueY* y *BloqueX* representan la cantidad total de bloques que habrá en “Y” y en “X” respectivamente, *m* y *n* representan la cantidad de filas y columnas respectivamente.

Como vemos de las ecuaciones, si *m* o *n* son valores impares, el resultado tendrá valores decimales y no solo enteros. Esto quiere decir que la imagen podría tener pixeles sin procesar y esto es un resultado indeseable. Es por esto que para pasar de un valor decimal en el resultado a un entero, se agrega ya sea una fila donde dé como resultado este tipo de valor. Así si por ejemplo, si la imagen tiene filas impares se agregará una fila con lo cual se convertirá a par y de esta forma ya podrán ser procesados todos los pixeles en su totalidad. Sin embargo, se debe de considerar que la fila o columna agregada no deben de tener valores aleatorios ya que debido a la dinámica de la CNN podría llegar a tener valores erróneos en los pixeles de la última fila o columna de la imagen original. Es por eso que se ha propuesto que esta fila o columna agregada tenga los mismos valores que sus vecinos, de esta forma se reduce el error provocado al momento de agregarlos. Finalmente, esta fila o columna agregada será removida de la imagen procesada, ya que ésta no es parte de la imagen original.

3.4.3 Proceso “Obtiene el bloque”

En este paso del algoritmo se realiza lo siguiente:

- Se apunta al bloque correspondiente.
- Se obtienen los valores de dicho bloque.

- Pregunta si fue el último bloque, si fue así termina, de lo contrario regresa y apunta al siguiente.

En este paso se obtiene el bloque que será procesado por la CNN, y para determinar qué bloque será procesado, se basa en la cantidad de bloques que hay en la imagen. Es por eso que se calcularon en el paso anterior, de esta forma sólo basta con empezar desde el primer bloque hasta el último, tanto en las filas como en las columnas. Y para llevar un mejor control de esto, se crearon dos bucles, uno para llevar el control de los bloques en X y otro para llevar el control de los bloques en Y.

Para ejemplificar esto, tengamos una imagen con 10×10 pixeles, y aplicando las ecuaciones (3.4) y (3.5) tendremos una cantidad de $BloqueX = 4$ y $BloqueY = 4$, en forma matricial tendríamos $Bloque(1,1), Bloque(1,2) , \dots , Bloque(1,4), \dots , Bloque(2,1), \dots$, hasta el $Bloque(4,4)$, los cuales podemos representar como $Bloque(i,j)$ donde (i,j) representan las coordenadas del *Bloque*. Ahora basta con colocar dos bucles que vayan recorriendo la cantidad de bloques en i y en j desde $(1,1)$ hasta $(BloqueX, BloqueY)$, donde $BloqueX$ representa la cantidad máxima de bloques en X y $BloqueY$ representa la cantidad máxima de boques en Y, de esta forma se irán tomando primero el $Bloque(1,1)$, después el $Bloque(1,2)$, y así sucesivamente hasta el $Bloque(4,4)$.

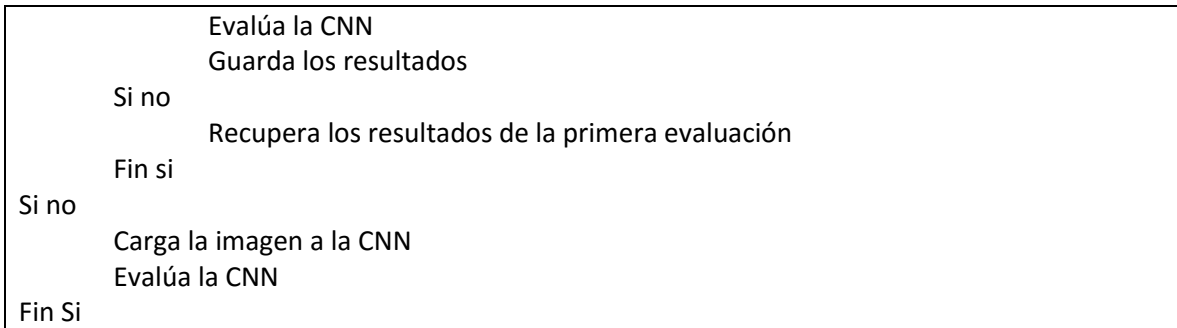
3.4.4 Proceso “Tipo de bloque”

Obtenido el bloque a ser procesado, se procede a determinar el tipo de bloque que es y a ser evaluado éste. Esta parte la mostraremos con el siguiente pseudoalgoritmo:

```

Checa el tipo de bloque obtenido
Si el bloque es blanco
    Si es el primer blanco
        Carga la imagen a la CNN
        Evalúa la CNN
        Guarda los resultados
    Si no
        Recupera los resultados de la primera evaluación
    Fin si
Si el bloque es negro
    Si es el primer negro
        Carga la imagen a la CNN

```

Como observamos en el pseudoalgoritmo, primero se checa qué tipo de bloque es. Para hacer esto comparamos cada uno de los pixeles que componen este bloque con los valores numéricos a los cuales correspondería un bloque en blanco o uno negro. Recordemos que un blanco corresponde a -1 y un negro corresponde a +1, de esta forma si el bloque contiene solamente valores de -1 en todos sus pixeles, querrá decir que es un bloque de blancos, si solo contiene valores numéricos de 1 querrá decir que es un bloque de negros; caso contrario a éstos querrá decir que es un bloque que contiene blancos y negros o en determinado caso será un bloque en escala de grises.

Determinado el tipo de bloque procederemos dependiendo el tipo de bloque. Por ejemplo, si el bloque es blanco se preguntará si es el primer blanco y si lo es, este bloque será procesado y se guardarán sus resultados en memoria, y si no lo es, entonces se recuperarán los resultados guardados del primer bloque procesado. De esta forma se ahorrará tiempo en el procesamiento de la imagen, ya que con esto evitamos estar evaluando el mismo bloque repetidamente. Para el caso de que el bloque sea negro, se actúa de forma similar al bloque blanco. Por último, si el bloque es en blanco y negro o en determinado caso en escala de grises, éste será procesado de forma normal.

3.4.5 Proceso “Actualiza salida y estados”

En este paso se realiza lo siguiente:

- Actualiza los valores de salida final de la CNN.
- Actualiza los valores de estado final de la CNN.

Como se mencionó en el punto 2.3.1, los valores guardados dependerán de la ubicación del bloque. En dicho punto se mencionó la forma de guardar los valores en un caso

donde sólo hay dos bloques, sin embargo, la misma lógica es aplicada para el caso donde existan un mayor número de bloques, esto es, que se van guardando los resultados finales hasta la penúltima fila, columna o ambas, dependiendo del traslape.

3.5 Funcionamiento de la CNN con MATLAB y SIMULINK

Mostrado el funcionamiento del multiplexado de la CNN, se mostrará una serie de ejemplos que muestran el funcionamiento de la CNN.

3.5.1 Removedor de ruido

Para mostrar este ejemplo utilizaremos las mismas plantillas que fueron utilizadas en el Capítulo 1 para el removedor de ruido, las cuales son las siguientes:

$$A = \begin{bmatrix} 0 & 3 & 0 \\ 3 & 2 & 3 \\ 0 & 3 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = 1.0$$

A continuación se muestra la imagen de 20×20 pixeles procesada y el tiempo que tardó el simulador en realizar el procesamiento.

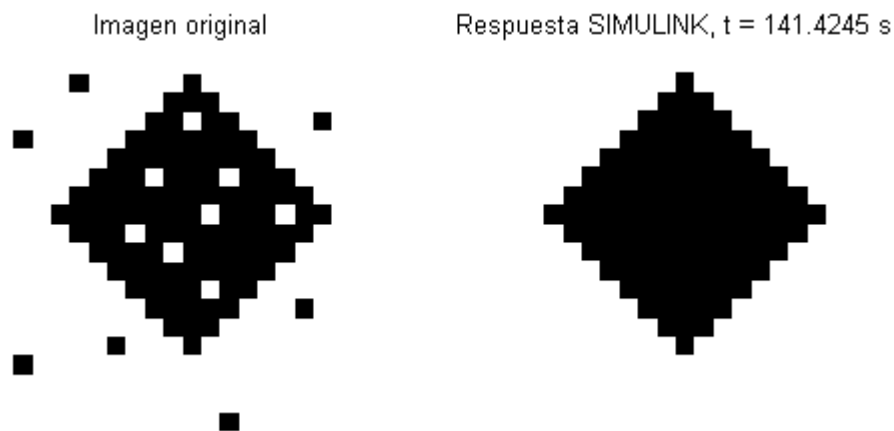


Fig. 3.16. Removedor de ruido.

3.5.2 Extractor de Bordes

La imagen procesada para mostrar este ejemplo es de 20×20 pixeles y se utilizan las siguientes plantillas:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & -2 & 0 \\ -2 & 5 & -2 \\ 0 & -2 & 0 \end{bmatrix}, I = -1.0$$

El resultado del procesamiento se muestra a continuación:

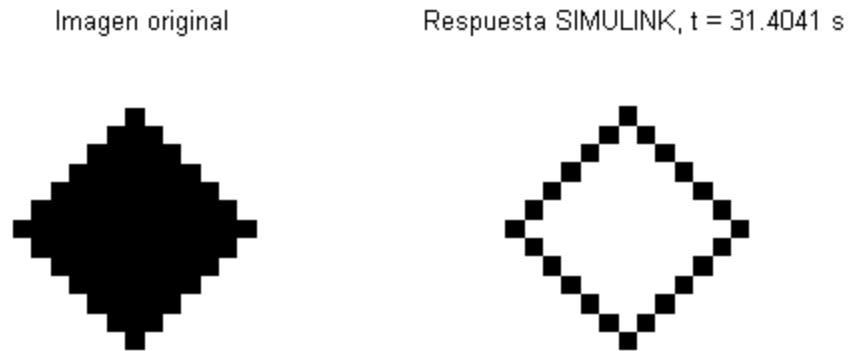


Fig. 3.17. Extractor de bordes.

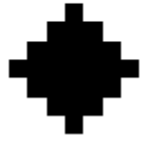
3.5.3 Extractor de sombras

En este ejemplo se hace una proyección de la imagen hacia la izquierda, donde para este simulador, como comentamos anteriormente, se detiene la simulación cuando en la imagen todas las neuronas que componen a la CNN llegan a saturación. Esto causa que no se llegue al mismo valor visto en el ejemplo del Capítulo 1, sin embargo, podemos procesar la imagen tantas veces como queramos donde la imagen de entrada para la siguiente vez que se procese, será la imagen de salida en el procesamiento anterior. Por ejemplo, para la imagen procesada en Fig. 3.18 su salida será la entrada para la imagen que se procesará por segunda vez, dando el resultado de la Fig. 3.19, finalmente en la imagen mostrada en la Fig. 3.20 se ha procesado 5 veces; utilizaremos las mismas plantillas vistas en el Capítulo 1:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2.1 & 2 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.9 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = 1.0$$

Como observamos en los resultados, según la cantidad de veces que se procesa la imagen, va aumentando la sombra.

Imagen original



Respuesta SIMULINK, t = 58.6212 s



Fig. 3.18. Extractor de sombras procesada una vez.

Imagen original

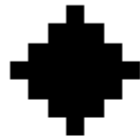


Respuesta SIMULINK, t = 106.1463 s



Fig. 3.19. Extractor de sombras procesada dos veces.

Imagen original



Respuesta SIMULINK, t = 285.5129 s



Fig. 3.20. Extractor de sombras procesada cinco veces.

3.5.4 Detector de conectividad global

Recordando lo que se mencionó en el Capítulo 1, podemos ver al detector de conectividad global como un proceso que busca la salida en un laberinto. En este ejemplo, tomamos las mismas consideraciones que en el ejemplo pasado, donde se tiene que procesar la imagen varias veces para llegar al valor corrector. Tomamos los mismos valores que el Capítulo 1, además procesamos la misma imagen. Las plantillas son las siguientes:

$$A = \begin{bmatrix} 0 & 4.4 & 0 \\ 4.4 & 3.6 & 4.4 \\ 0 & 4.4 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 10.7 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = 7.0$$

Las imágenes obtenidas son las siguientes:

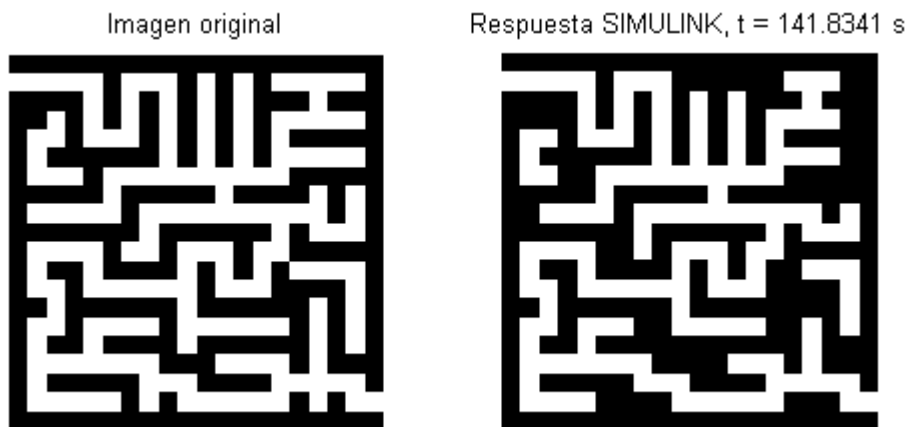


Fig. 3.21. Detector de conectividad global procesado una sola vez.

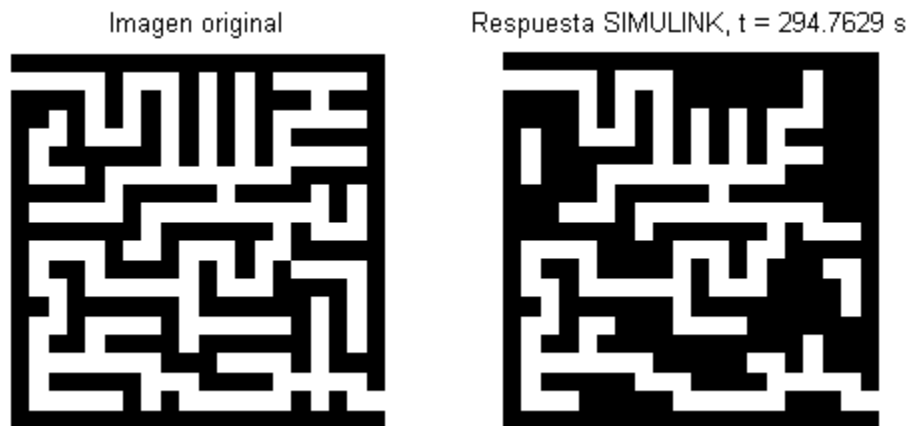


Fig. 3.22. Detector de conectividad global procesado dos veces.

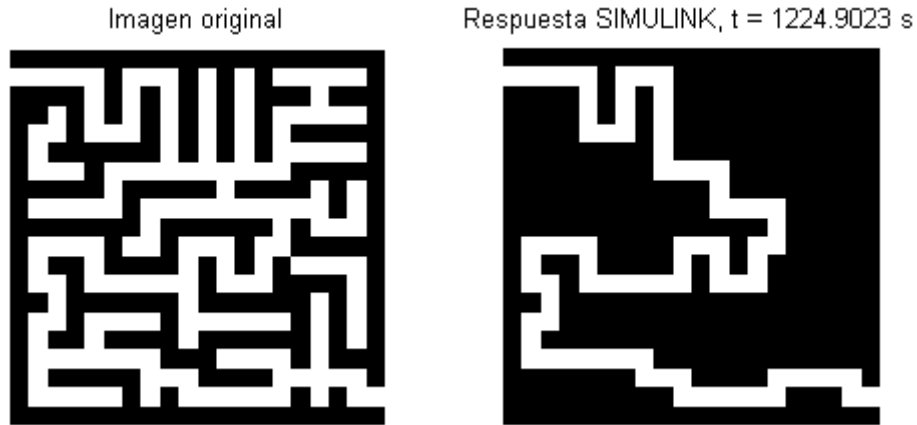


Fig. 3.23. Detector de conectividad global procesado doce veces.

Como vemos, en este caso se necesitó procesar la imagen más veces, sin embargo, se llegó al mismo resultado visto en el Capítulo 1, de este mismo ejemplo.

3.6 Conclusiones

Como vimos, en este capítulo se creó un simulador para nuestra CNN multiplexada. Con éste se obtuvieron buenos resultados, además nos permite tomarlo como base para crear la CNN que realizaremos dentro del FPGA, y debido a que observamos buenos resultados observamos que es factible realizarla y obtener buenos resultados de igual forma.

Como observamos a lo largo de este capítulo, se deben de tomar ciertas consideraciones al momento de procesar la imagen, como lo fue el hecho de convertir la imagen a valores que reconozca la CNN, ya que como mencionamos, MATLAB no trata las imágenes de la misma forma que lo realiza una CNN. Otro aspecto importante respecto a este punto son las consideraciones que se deben de tener en el multiplexado para reducir los errores inherentes a éste.

En este capítulo observamos las mismas aplicaciones que fueron realizadas en el capítulo 1, observando que obtenemos resultados similares, sin embargo, como mencionamos, debemos de tener ciertas consideraciones en ciertas aplicaciones para obtener el resultado correcto.

3.7 Referencias

- [1] Leon O. Chua and L. Yang, "Cellular Neural Networks: Theory & Applications", IEEE Trans. Circuits and Systems, vol. CA-35, pp. 1257-1290, 1988.
- [2] Chi-Chien Lee and José Pineda de Gyves, "Time-Multiplexing CNN Simulator", IEEE Inter. Symp. On Circuits and Systems, 1994, vol. 6, pp. 407-410, 1994.
- [3] A. A. H. EL-Shafei and M.I. Sobhy, "A Time-Multiplexing simulator for Cellular Neural Network (CNN) using SIMULINK", IEEE Inter. Symp. On Circuits and Systems, 1998, vol. 3, pp. 167-170.
- [4] A. A. H. EL-Shafei and M.I. Sobhy, "A Time-Multiplexing simulator for Cellular Neural Network (CNN)", Cellular Neural Networks and Their Applications Proceedings, 1998 Fifth IEEE International Workshop on, pp. 14-18, 1998.
- [5] Boyce and DiPrima, "*Ecuaciones diferenciales y problemas con valores en la frontera*", 5a edición, Limusa, pp. 441-467.

Capítulo 4. Descripción de una CNN multiplexada en tiempo en VHDL

4.1 Introducción

Como se mencionó en el capítulo 1, existen diversas formas de resolver varios problemas que requieren una gran demanda de cómputo matemático y un alto nivel de comunicación entre procesadores como lo son “la difusión”, “las ecuaciones de Laplace y Onda”. Una forma de resolverlos es utilizando circuitos integrados de aplicación específica, sin embargo, estos son muy costosos y el tiempo de diseño es grande [1].

En este capítulo mostraremos la CNN digital creada en FPGA. Esta CNN está multiplexada como la creada en el capítulo anterior y en [2] – [4], sin embargo, la CNN digital creada en FPGA difiere a la creada en el capítulo anterior y a los vistos en [1–4].

La principal diferencia entre la CNN digital vista en [1] y la propuesta en este trabajo, es que la primera solo puede procesar imágenes binarias, esto es, solo imágenes en blanco y negro, y la propuesta que se presenta aquí, puede procesar imágenes en escala de grises. Otra diferencia es la forma de resolver la ecuación de estado de la CNN, en [1] se ha aplicado el método de integración de Euler directamente a la ecuación de estado para digitalizarla y poder resolverla. Como veremos más adelante, en nuestro caso lo hemos realizado más a la forma vista en el capítulo anterior.

Algunas de las diferencias entre estas CNN son, como mencionamos anteriormente, el hecho de que la creada en FPGA no realiza la comprobación de si el bloque es blanco, negro o está en escala de grises, así como que la resolución numérica está limitada debido al hecho de que fue creada digitalmente, el resto de los bloques son parecidos. Sin embargo, debido a que en esta ocasión son creados digitalmente, tienen diferencias.

La tarjeta de desarrollo utilizada para crear la CNN es Atlys™ Spartan-6 FPGA Development Board, la cual tiene las siguientes características para el desarrollo de la CNN:

- FPGA Xilinx Spartan-6 LX45
- Un puerto USB-UART
- Oscilador CMOS a 100MHz
- 8 LEDs

Como observamos, la tarjeta de desarrollo contiene el FPGA Xilinx Spartan-6 LX45, la cual contiene las siguientes características para desarrollar la CNN:

- 6822 capas (slices) la cual cada una contiene cuatro tablas de búsqueda (LUTs) de 6 entradas y 8 flip-flops
- Bloque rápido de memoria de acceso aleatorio (RAM) de 2.1Mbits
- 58 Procesadores de señal digital (DSPs)

El puerto USB-UART lo utilizaremos para poder comunicar al FPGA con MATLAB, ya que a través de éste se recibirán, por un lado, la imagen a ser procesada, además de los parámetros necesarios para poder procesar la imagen como lo son las plantillas **A** y **B**, el umbral, y la cantidad de pixeles en x y en y .

El oscilador lo utilizaremos a 100MHz, lo cual nos garantiza que trabajaremos a una alta velocidad. Los LEDs los utilizaremos para estar monitoreando el estado de la CNN.

4.2 Diagrama a bloques de la CNN

En la Fig. 4.1 se muestra el diagrama a bloques de la CNN multiplexada. A continuación daremos una pequeña explicación de cada uno de los módulos (en el apéndice B se encuentra el código completo).

Receptor: Este módulo se encarga de recibir los datos seriales provenientes de MATLAB y colocarlos en un registro de 8 bits, para así poder ser guardados por el módulo Pila cuando estos datos corresponden a los parámetros de la CNN o en la RAM cuando estos datos corresponden a la imagen a ser procesada.

Pila: Éste se encarga de guardar los primeros datos recibidos por el receptor dentro de una serie de registros que los almacenarán, ya que éstos pertenecen a los parámetros de la CNN como lo son: $xMax$ (este dato es la cantidad de pixeles de la imagen en x), $yMax$ (este dato es la cantidad de pixeles de la imagen en y), plantilla **A**, plantilla **B**, umbral I . Cuando haya terminado de almacenar estos parámetros, la pila pasará el dato recibido como un pixel para que sea almacenado como parte de la imagen dentro de la memoria RAM.

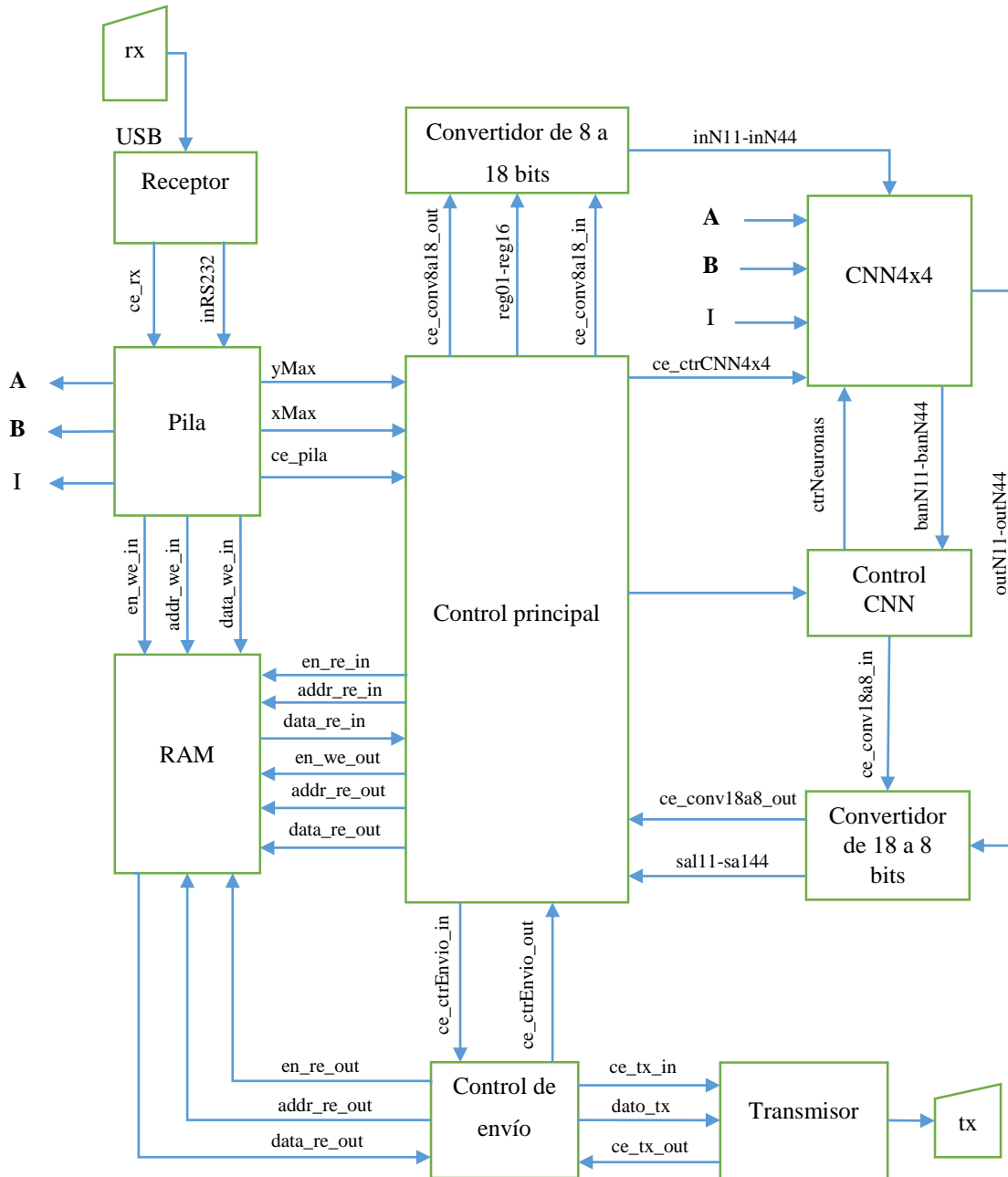


Fig. 4.1. Diagrama a bloques de la CNN multiplexada.

RAM: Este módulo contiene dos bloques de RAM, cada bloque es de 65536 localidades, cada localidad con un tamaño de 8 bits; un bloque es utilizado para guardar la imagen de entrada recibida por el “Receptor” y guardada por la “Pila”; el segundo bloque de RAM es utilizada para guardar la imagen de salida por el “Control principal” además de ser leída por el “Control de envío”.

Control principal: Este módulo se encarga de controlar todos los pasos que se requieren para realizar el multiplexado de la CNN, como lo son el cálculo de bloques, la lectura de las sub-imágenes a ser procesadas, la carga de los datos dentro de la CNN_{4x4} , además de guardar los resultados procesados por la CNN_{4x4} y finalmente de indicar cuándo se requiere regresar la imagen procesada hacia MATLAB serialmente.

Convertidor de 8 a 18 bits: Como mencionamos en el capítulo anterior, MATLAB al leer la imagen, la lee con valores de 0 a 255, estos datos son pasados directamente al FPGA. Es por esta razón que se necesita hacer una conversión como la mostrada en 3.4.1. Sin embargo, como observamos en la fórmula (3.3), se requiere hacer una división, es por esto, que se ha cambiado ligeramente la fórmula, donde esta división se sustituyó por una multiplicación, quedando la fórmula como sigue:

$$x = 1 - (0.007843018 * y) \quad (4.1)$$

Donde $\frac{2}{255} \approx 0.007843018$, esto debido a la resolución de 18 bits.

CNN_{4x4} : Este módulo contiene las 16 neuronas creadas individualmente y después interconectadas. Este módulo recibe los parámetros, además de la sub-imagen a ser procesada, además está controlado por un registro de control proveniente del módulo de “Control de la CNN”, a la salida se entregan los valores procesados por las neuronas. Este módulo se explicará más adelante.

Control de la CNN: Este módulo se encarga de llevar el control de la CNN_{4x4} , esto es, le dice a la CNN_{4x4} qué realizar, además de comprobar si la CNN_{4x4} ha llegado a un estado estable y detener el proceso; este módulo será explicado con mayor detalle más adelante.

Convertidor de 18 a 8 bits: Después de realizar el procesado de la CNN y haber llegado a un estado estable, se convierten los datos entregados por la CNN_{4x4} a valores ASCII, esto debido a que puedan ser reconocidos de mejor manera por MATLAB. Si el valor entregado por la CNN_{4x4} es un valor de 1, se convierte a un valor de “0” en ASCII y si el valor es -1 es convertido a un valor de “1” en ASCII.

Control de envío: Después de procesada la imagen y haberse guardado en memoria, es momento de regresar la imagen de salida vía RS232. Es por eso que este módulo va leyendo la imagen procesada guardada en RAM pixel por pixel, y va pasando los pixeles al módulo del “*Transmisor*” para que sean enviados.

Transmisor: Este módulo se encarga de recibir los datos pasados por el módulo “*Control de envío*” y enviarlos vía RS232 hacia MATLAB.

4.3 Descripción de una Neurona en VHDL

El diseño de la neurona en VHDL se realiza bajo las siguientes consideraciones:

Se han tomado como entradas para los multiplicadores una resolución de 18 bits, esto debido a que para estos multiplicadores su resolución máxima es de 36 bits, tomando en cuenta esto, los operandos no deben de exceder de 18 bits. De estos 18 bits se ha tomado 1 bit como el bit de signo, 6 bits para la parte entera y finalmente 11 bits para la parte fraccionaria, esto nos da una resolución cercana entre $(-64, 64)$.

El tamaño de los registros de entrada para las plantillas, y valores de entrada y salida de los pixeles, además del valor del paso de tiempo para el integrador, será de 18 bits. El valor de umbral tendrá un tamaño de 36 bits, este último debido a que se sumará directamente y no requiere de ser multiplicado.

Las plantillas **A** y **B** e igualmente el umbral **I** se tomarán como variables, esto debido a que son recibidas desde MATLAB, con lo cual no sabremos qué valores tendrá. Con esto tendremos la ventaja de que desde MATLAB podemos pasarle las plantillas y así realizar diferentes tipos de procesamiento sobre la imagen.

La constante de tiempo para el integrador será fija, y ésta tendrá un valor de 0.000976563 que es el valor más cercano a 0.001 que se puede alcanzar con una resolución de 18 bits.

La salida de la neurona entregará un valor contenido en 18 bits. A continuación se muestra el diagrama a bloques que será implementado en VHDL para la neurona. En el apéndice B se muestra el código completo en VHDL.

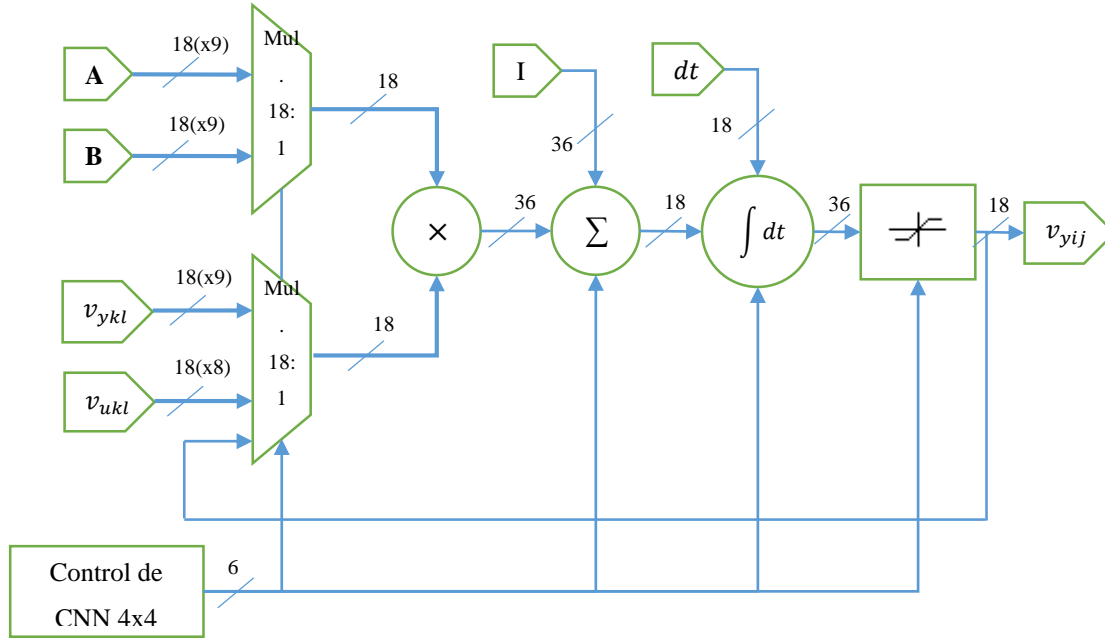


Fig. 4.2. Diagrama a bloques de una neurona en VHDL.

Para comprobar el funcionamiento de esta neurona se realizará el mismo ejemplo visto para la neurona descrita en MATLAB, la cual es como sigue:

- Los valores de frontera serán pixeles negros, esto es, valores de uno.
- El valor de entrada de la neurona será un pixel blanco, esto es, un valor de menos uno.
- La condición inicial de la neurona será la misma que la entrada a la neurona en el tiempo 0, esto es, un valor de menos uno.
- El paso de tiempo para el integrador Euler de la neurona será de 0.000976563s.
- Las plantillas de ruido serán las siguientes:

$$A = \begin{bmatrix} 0 & 3 & 0 \\ 3 & 2 & 3 \\ 0 & 3 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = 1.0$$

En la Fig. 4.3 se muestra la evolución de la neurona. Como vemos, la neurona evoluciona de un valor de -1 a un valor de 1 (la salida de esta neurona aparece como sal dentro de la figura), con esto comprobamos que la neurona se comporta de manera correcta.

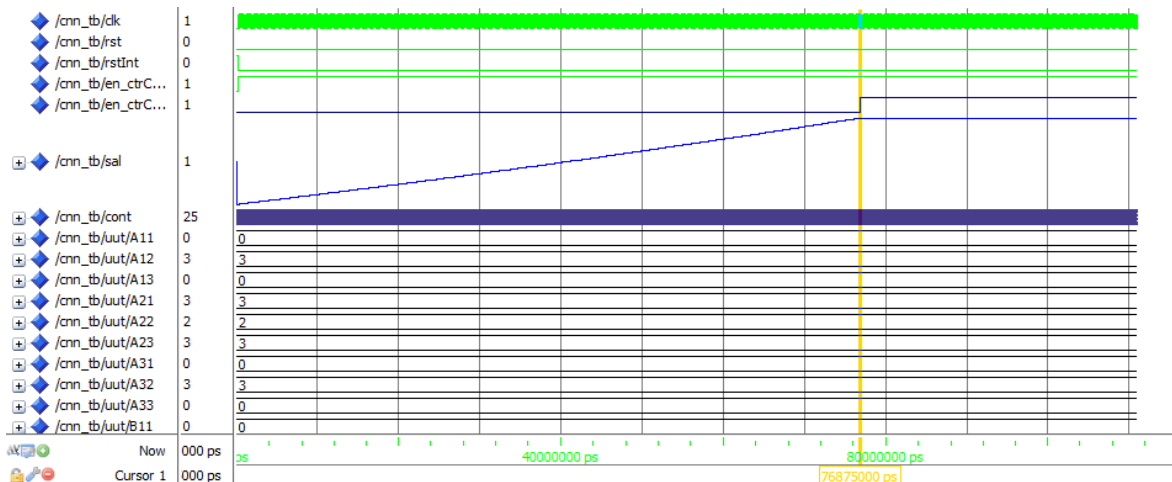


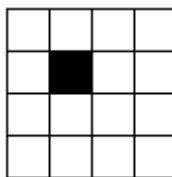
Fig. 4.3. Evolución de una neurona en VHDL para un ejemplo de quitar ruido en un pixel.

4.4 Descripción de la CNN 4x4 en VHDL

Descrita la neurona, se procede a describir la CNN 4x4 dentro de VHDL. Para poder lograr esto se integró la neurona como una entidad y de esta forma sólo se repite esta neurona y se realiza la interconexión. En el apéndice B se observa el código para esto.

Para comprobar el funcionamiento de la CNN 4×4 descrita en VHDL se hace un ejercicio similar al mostrado en el capítulo anterior. Se realizará una simulación para quitar ruido, con las siguientes consideraciones:

- La imagen de prueba será la siguiente:



- La condición inicial de las neuronas será la misma que la imagen de entrada.
- El valor de frontera será de 0.
- El paso de tiempo del integrador Euler hacia atrás será de 0.000976563s.
- Las plantillas para quitar ruido serán las siguientes:

$$A = \begin{bmatrix} 0 & 3 & 0 \\ 3 & 2 & 3 \\ 0 & 3 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = 1.0$$

En la siguiente figura se muestra la evolución de las neuronas.

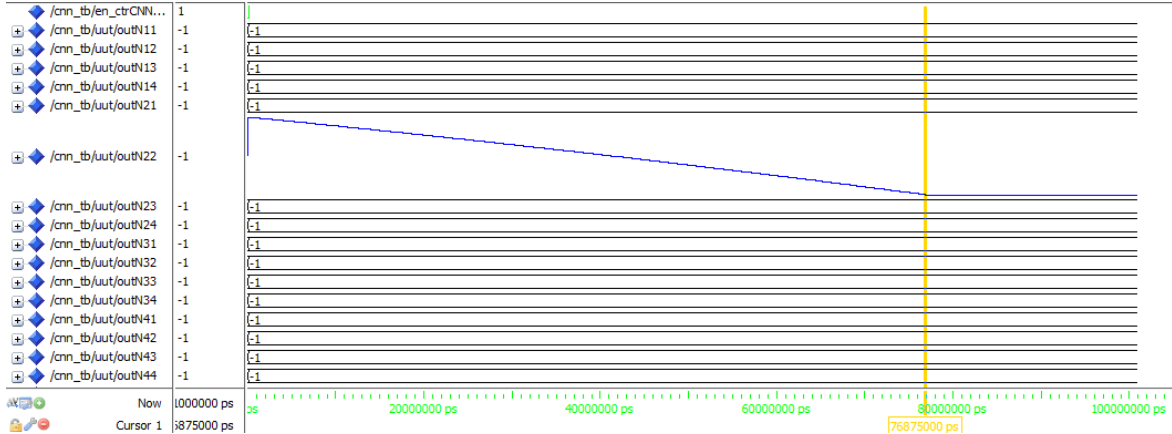


Fig. 4.4. Evolución de una CNN de 4x4 neuronas, ejemplo de quitar ruido en un pixel, en este caso el pixel de la neurona C(2,2).

Como observamos, la salida de la neurona que contiene el pixel que es considerado ruido evoluciona de un valor +1 (negro) a un valor de -1 (blanco), el resto de los pixeles se mantienen en el color blanco, estos resultados nos muestran el correcto funcionamiento de la CNN.

4.5 Multiplexado de la CNN en VHDL

Para realizar el multiplexado de la CNN en VHDL se desarrolló un circuito secuencia mixto que controlará cada uno de los procesos vistos en la Fig. 4.1. Este módulo se muestra como “Control principal” dentro de dicha figura, en la Fig. 4.5 se muestra el circuito secuencial mixto.

A continuación se dará una explicación breve de la función de cada uno de los pasos en el diagrama de estados de la Fig. 4.5:

Lee los parámetros: Recibe la señal “rx” que es el receptor RS232, se van leyendo primero los parámetros de la CNN como lo son las plantillas **A** y **B**, el valor de umbral I, además del valor máximo en x y el valor máximo en y de la imagen a ser procesada, al finalizar la lectura pasa al estado de “Lee imagen de entrada”.

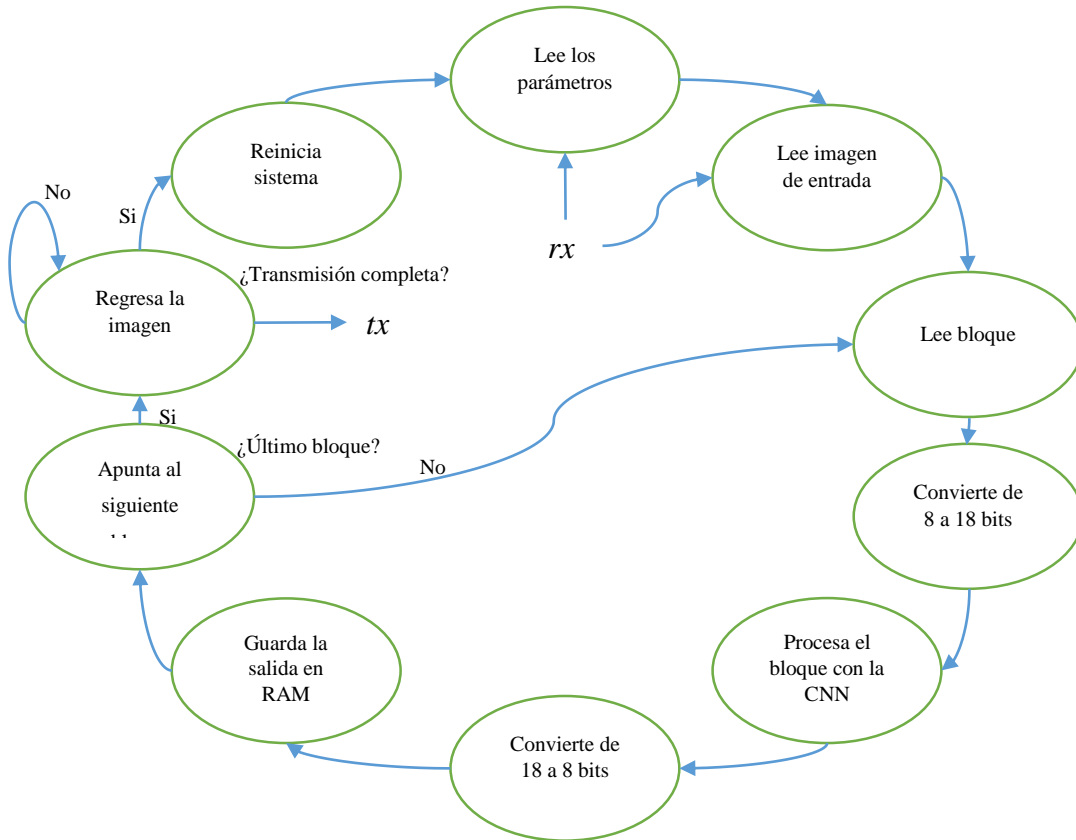


Fig. 4.5. Diagrama de flujo del circuito secuencial mixto para el control de la CNN.

Lee imagen de entrada: Al igual que el estado de “Lee los parámetros” recibe la señal “rx”. En este caso, después de completar la lectura de los parámetros se va leyendo la imagen a ser procesada la cual se va almacenando en un bloque de memoria RAM, al finalizar la lectura de la imagen pasa al estado “Lee bloque”.

Lee bloque: En este estado lee la sub-imagen a ser procesada por la CNN desde la memoria RAM; para realizar esto toma el apuntador que entrega el estado de “Apunta al siguiente bloque”, al finalizar pasa la imagen al estado “Convierte de 8 a 18 bits”, esto debido a que la imagen recibida está en formato de 8 bits que entrega MATLAB.

Convierte de 8 a 18 bits: Este estado convierte la sub-imagen en formato de 8 bits al formato de 18 bits, como fue mencionado anteriormente. Al finalizar, pasa los resultados al módulo que contiene la CNN de 4×4 neuronas.

Procesa el bloque con la CNN: En este estado recibe los datos del estado “Convierte de 8 a 18 bits” y los datos de “Lee los parámetros” y así procesar el bloque con la CNN de

4×4 neuronas, éste estará procesando dicho bloque hasta que obtenga en la salida de las 16 neuronas un estado estable; al finalizar pasa los resultados al estado “*Convierte de 18 a 8 bits*”.

Convierte de 18 a 8 bits: Después de procesado, el bloque se pasa a convertir los resultados de un formato de 18 bits a 8 bits, para que puedan ser transmitidos vía RS232, esta conversión es pasada al estado de “*Guarda la salida en RAM*”.

Guarda la salida en RAM: Al finalizar la conversión a 8 bits, se guardan los resultados obtenidos, tomando en cuenta las consideraciones que se mencionaron en los puntos 2.3.1, dentro de un bloque de memoria RAM; este bloque es diferente al bloque que contiene la imagen de entrada.

Apunta al siguiente bloque: En este estado se realiza todo el cálculo necesario para hacer que se apunte al siguiente bloque a ser procesado, esto tomando en cuenta los puntos mostrados en 2.3.1, además de comprobar si ya fue procesado el último bloque. Si es así, pasa al estado “*Regresa la imagen*”, de lo contrario, pasa al estado “*Lee bloque*”. Otro punto a considerar es que ya sea si fue reiniciado el sistema manualmente o si fue activado el estado “*Reinicia la CNN*” o si se inicia por primera vez, los valores aquí obtenidos son inicializados para que apunten al primer bloque.

Regresa la imagen: En este estado se realiza la transmisión de la imagen obtenida. Esto lo hace leyendo el bloque de memoria que contiene dicha imagen; al finalizar de enviar la imagen pasa al estado que reinicia el sistema.

Reinicia sistema: Este estado se encarga crear un reinicio interno que inicializa las variables utilizadas por el sistema, sin embargo, este estado no borra los bloques de memoria RAM; al finalizar regresa al estado inicial que es “*Lee los parámetros*”.

4.6 Prueba de la CNN

Para comprobar el funcionamiento de la CNN creada en VHDL, se mostrarán algunos ejemplos procesados con el FPGA. Aquí cabe mencionar que se carga la imagen dentro de MATLAB y si la imagen es en color la imagen es convertida a escala de grises, y esta imagen en escala de grises es enviada tal cual al sistema, para ser procesada vía RS232 recordando

que se utiliza a una velocidad de 115200 Bauds y que los datos de la imagen están en valores enteros de 8 bits sin signo.

Los recursos aproximados tomados por el FPGA para realizar el sistema completo del multiplexado de la CNN se muestran en la siguiente figura.

Device Utilization Summary (estimated values)				[1]
Logic Utilization	Used	Available	Utilization	
Number of Slice Registers	3270	54576	5%	
Number of Slice LUTs	8539	27288	31%	
Number of fully used LUT-FF pairs	2500	9309	26%	
Number of bonded IOBs	12	218	5%	
Number of Block RAM/FIFO	52	116	44%	
Number of BUFG/BUFGCTRLs	3	16	18%	
Number of DSP48A1s	35	58	60%	

Fig. 4.6. Recursos utilizados del FPGA para realizar la CNN multiplexada.

Como observamos, toma bastantes recursos; sobre todo al uso de los DSPs debido a que estos contienen los multiplicadores utilizados.

4.6.1 Removedor de ruido

Para mostrar este ejemplo, utilizaremos las mismas plantillas de removedor de ruido de los capítulos 1 y 3.

$$A = \begin{bmatrix} 0 & 3 & 0 \\ 3 & 2 & 3 \\ 0 & 3 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = 1.0$$

A continuación se muestra la imagen de 20×20 pixeles procesada y el tiempo que tardó el simulador en realizar el procesamiento.

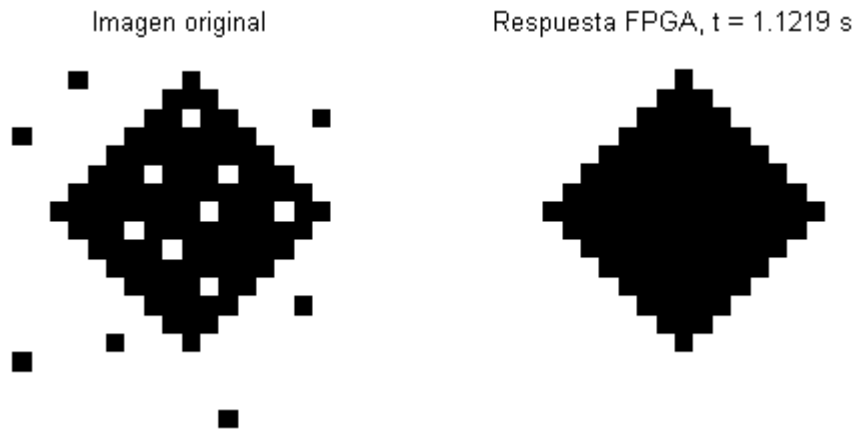


Fig. 4.7. Removedor de ruido procesado con la CNN multiplexada en el FPGA.

4.6.2 Extractor de bordes

Para mostrar este ejemplo utilizaremos las mismas plantillas de extractor de bordes de los capítulos 1 y 3.

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & -2 & 0 \\ -2 & 5 & -2 \\ 0 & -2 & 0 \end{bmatrix}, I = -1.0$$

El resultado del procesamiento con el FPGA es mostrado a continuación:

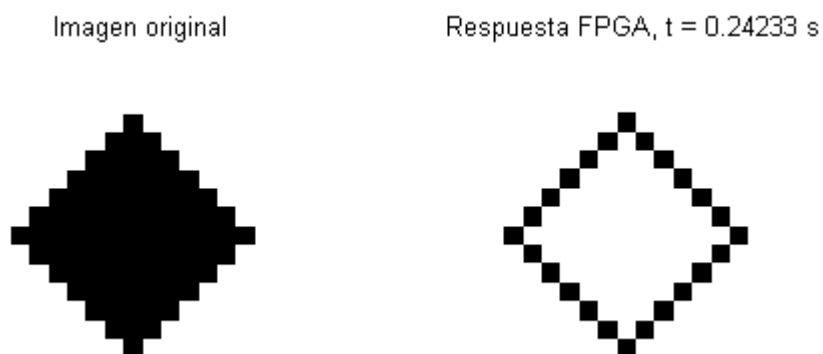


Fig. 4.8. Extractor de bordes procesado con la CNN multiplexada en el FPGA

4.6.3 Extractor de sombras

En este ejemplo se toman las consideraciones mostradas en el punto 3.5.3, además se toman las mismas plantillas vistas en los capítulos 1 y 3.

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2.1 & 2 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.9 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = 1.0$$

A continuación se muestran los resultados obtenidos:

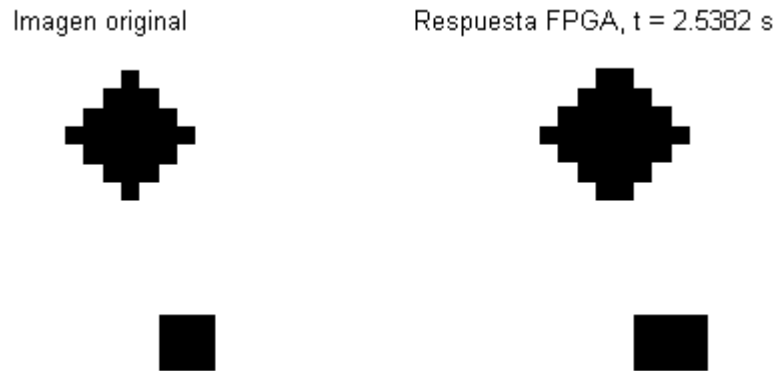


Fig. 4.9. Extractor de sombras procesada una vez con el FPGA.

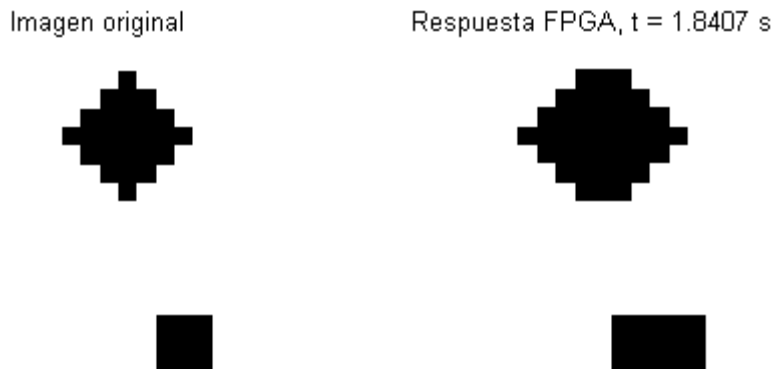


Fig. 4.10. Extractor de sombras procesada dos veces con el FPGA.

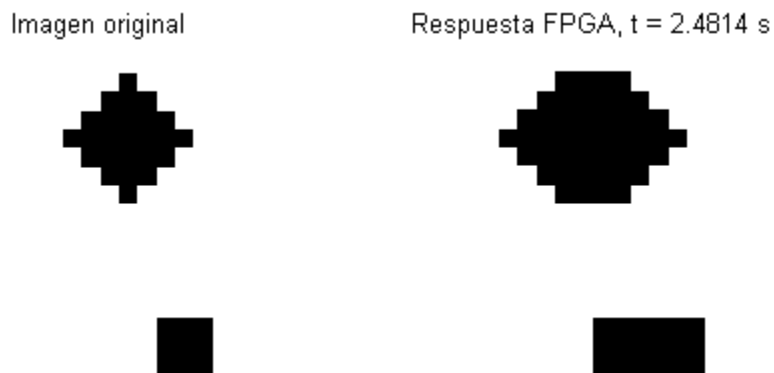


Fig. 4.11. Extractor de sombras procesada cinco veces con el FPGA.

4.6.4 Detector de conectividad global

En este ejemplo se toma en cuenta lo señalado en el punto 3.3.5, además se utilizan las mismas plantillas de los capítulos 1 y 3.

$$A = \begin{bmatrix} 0 & 4.4 & 0 \\ 4.4 & 3.6 & 4.4 \\ 0 & 4.4 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 10.7 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = 7.0$$

A continuación se muestran los resultados tomando el mismo número de iteraciones que se realizaron en el ejemplo del capítulo 3.

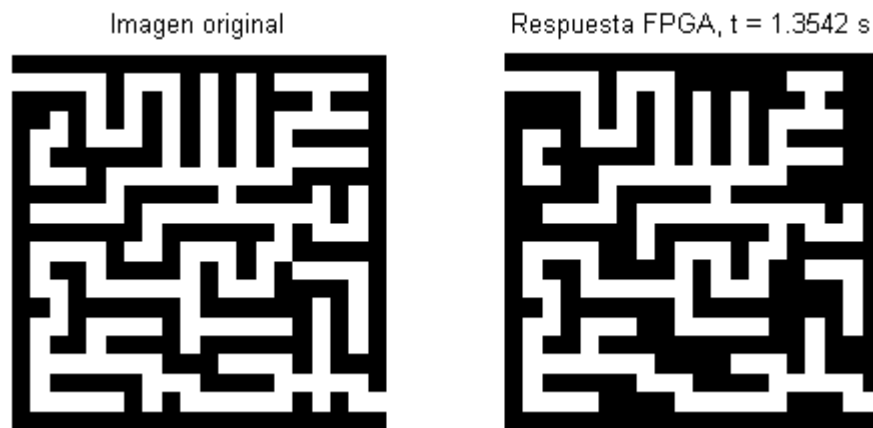


Fig. 4.12. Detector de conectividad global procesada una vez con el FPGA.

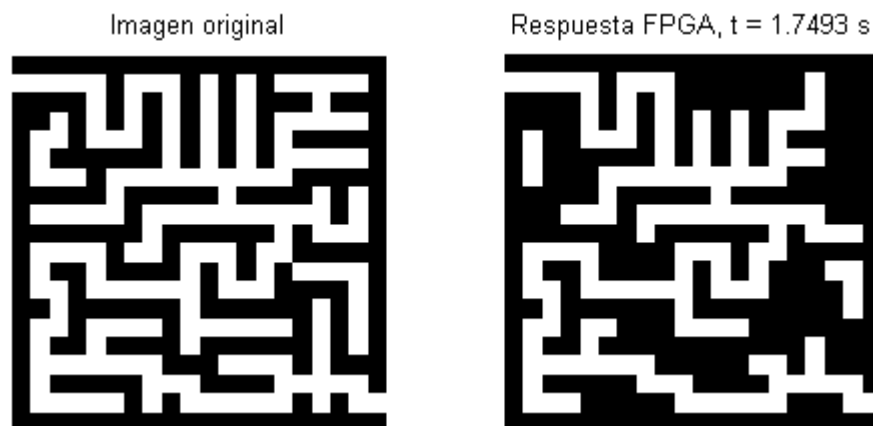


Fig. 4.13. Detector de conectividad global procesada dos veces con el FPGA.

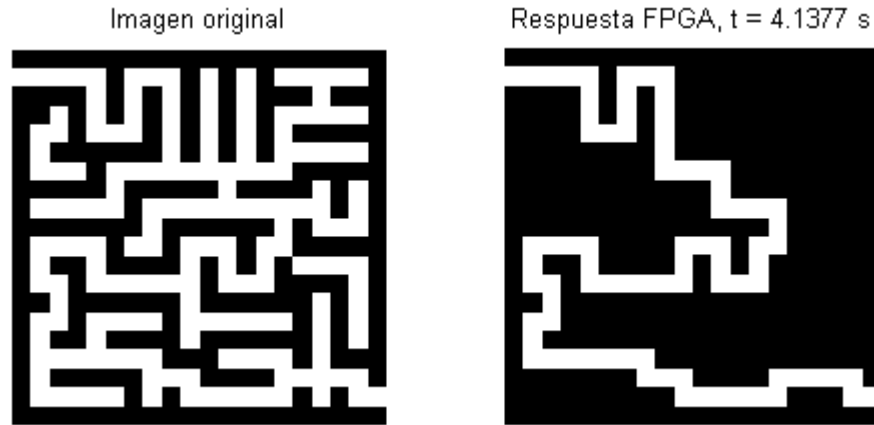


Fig. 4.14. Detector de conectividad global procesada doce veces con el FPGA.

4.7 Conclusiones

Como se observó a lo largo de este capítulo, se realizó la CNN digital implementada dentro de un FPGA de forma correcta. Además, como observamos, ésta requirió de varias consideraciones que fueron vistas a lo largo del capítulo, como es el hecho de considerar la resolución numérica para su correcto funcionamiento.

Además, observamos los recursos que se requirieron para implementar todos los módulos para el correcto funcionamiento de la CNN y se observó que fueron bastantes los recursos tomados del FPGA.

Como se observó en los resultados, la CNN implementada digitalmente funciona de forma correcta, lo cual indica que ésta puede realizar diversos tipos de procesamiento con esta técnica de multiplexado. Sin embargo, como se observará más adelante, hay consideraciones que hay que tomar en cuenta, como lo es el tipo de plantillas ingresadas a esta implementación.

4.8 Referencias

- [1] R. Grech, E. Gratt, I. Grech, J. Micallef, “Digital Implementation of Cellular Neural Networks”, Electronics, Circuits and Systems, 2008. ICES 2008. 15th IEEE International Conferences on, pp. 710-713, 2008.
- [2] Chi-Chien Lee and José Pineda de Gyves, “Time-Multiplexing CNN Simulator”, IEEE Inter. Symp. On Circuits and Systems, 1994, vol. 6, pp. 407-410, 1994.
- [3] A. A. H. EL-Shafei and M.I. Sobhy, “A Time-Multiplexing simulator for Cellular Neural Network (CNN) using SIMULINK”, IEEE Inter. Symp. On Circuits and Systems, 1998, vol. 3, pp. 167-170.
- [4] A. A. H. EL-Shafei and M.I. Sobhy, “A Time-Multiplexing simulator for Cellular Neural Network (CNN)”, Cellular Neural Networks and Their Applications Proceedings, 1998 Fifth IEEE International Workshop on, pp. 14-18, 1998.

Resultados

En este capítulo mostraremos algunos ejemplos de imágenes de mayor tamaño a las mostradas en los ejemplos de los capítulos anteriores, esto para mostrar el funcionamiento de la CNN propuesta en nuestro trabajo.

En esta primera parte, mostraremos imágenes procesadas tanto en el simulador desarrollado en MATLAB como el realizado dentro del FPGA. Además se calculará la correlación en MATLAB entre las imágenes del simulador y la entregada por SIMULINK con el objetivo de validar la comparación, esto se realizará con la función “*corr2 (imagen1, imagen2)*” de MATLAB.

Ejemplo 1: En la siguiente imagen, se encuentra el contorno utilizando las siguientes plantillas:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & -2 & 0 \\ -2 & 5 & -2 \\ 0 & -2 & 0 \end{bmatrix}, I = -1.0$$

La imagen procesada se muestra a continuación:

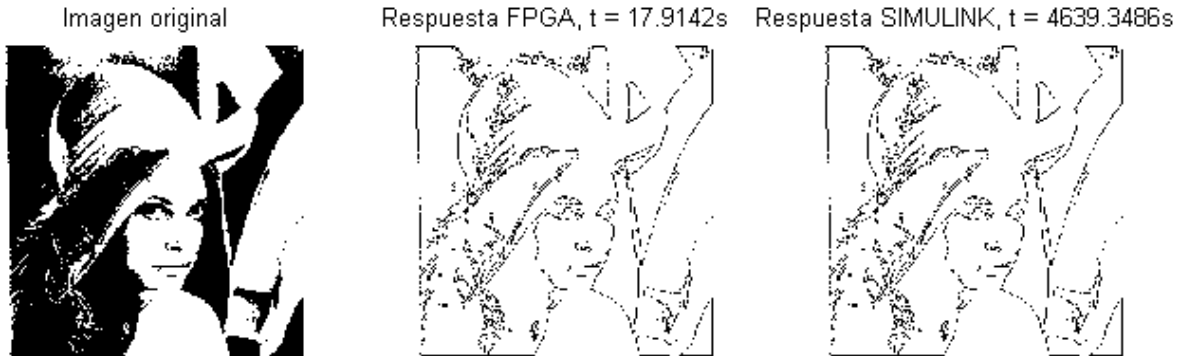


Esta imagen procesada tiene un tamaño de 263x267 píxeles en escala de grises, y como observamos, fue más rápido el procesamiento con el FPGA que la simulación. Mientras el procesado dentro del FPGA tomó 12.3807 segundos, la realizada con SIMULINK tomó 4982.2969 segundos. Además, se encontró una correlación entre ambas imágenes de 0.9998, lo que quiere decir que son muy parecidas las imágenes.

Ejemplo 2: De igual forma, en la siguiente imagen se encontrará el contorno utilizando las mismas plantillas:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & -2 & 0 \\ -2 & 5 & -2 \\ 0 & -2 & 0 \end{bmatrix}, I = -1.0$$

La imagen procesada se muestra a continuación:



Esta imagen procesada tiene un tamaño de 288x288 píxeles en escala de blancos y negros, y nuevamente observamos los tiempos, el cual es mucho más rápido para el procesamiento dentro del FPGA que el obtenido con el simulador en SIMULINK. El primero se realizó en un tiempo de 17.9142 s y el segundo en un tiempo de 4639.3486 s lo que representa 248.97 veces mayor que el realizado con el FPGA. La correlación encontrada en este caso fue de 1.

Ejemplo 3. En la siguiente figura se le ha agregado ruido, esto para posteriormente quitarlo utilizando la CNN; se utilizan las siguientes plantillas:

$$A = \begin{bmatrix} 0 & 3 & 0 \\ 3 & 2 & 3 \\ 0 & 3 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = 1.0$$

Recordemos que la salida procesada estará en blanco y negro. El ruido en la imagen es agregado con la función $J = imnoise(I, type)$ que está incluida dentro de MATLAB, donde I representa la imagen de entrada, J la imagen con ruido y $type$ es el tipo de ruido que es agregado, en nuestro caso hemos agregado un ruido tipo '*salt & pepper*'.

La imagen procesada es mostrada a continuación:

Imagen original



Respuesta FPGA, t = 21.0879 s



Ejemplo 4: En el siguiente ejemplo se muestra un removedor de ruido para la misma imagen que el anterior, sin embargo, en esta ocasión se muestra con plantillas diferentes. Las plantillas utilizadas se muestran a continuación:

$$A = \begin{bmatrix} 0 & 2.03 & 0 \\ 2.03 & 2.10 & 2.03 \\ 0 & 2.03 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3.99 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = -0.01$$

La imagen procesada es mostrada a continuación:

Imagen original



Respuesta FPGA, t = 21.856 s



Como observamos en la imagen, obtenemos una imagen que igualmente al anterior ha quitado el ruido.

Ejemplo 5. En este ejemplo se realiza el procesamiento de imagen en escala de grises el cual convierte dicha imagen en escala de blancos y negros, para realizar esto se utilizan las siguientes plantillas:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I = -z$$

En este ejemplo, z indica el umbral al cual el pixel será convertido a un pixel negro. En esta imagen se ha utilizado $z = 0.25$:

Imagen original



Respuesta FPGA, t = 82.2624 s



Ejemplo 6: En el siguiente ejemplo se muestra un extractor de bordes, el cual es utilizado para huellas digitales. Las plantillas utilizadas son las siguientes:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}, I = -0.5$$

El resultado de la imagen procesada con la CNN es mostrada a continuación.

Imagen original



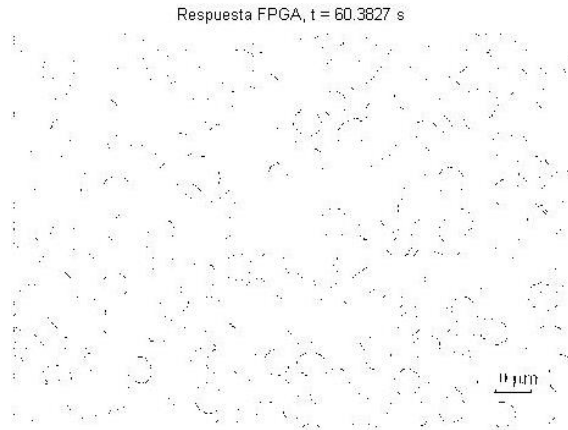
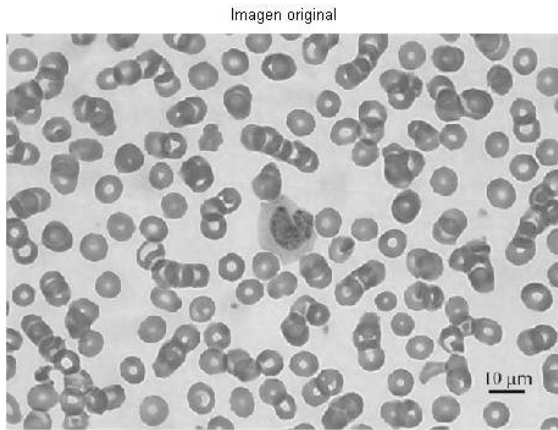
Respuesta FPGA, t = 13.1481 s



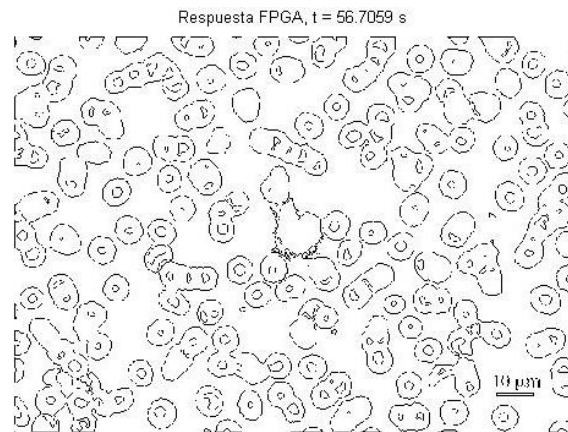
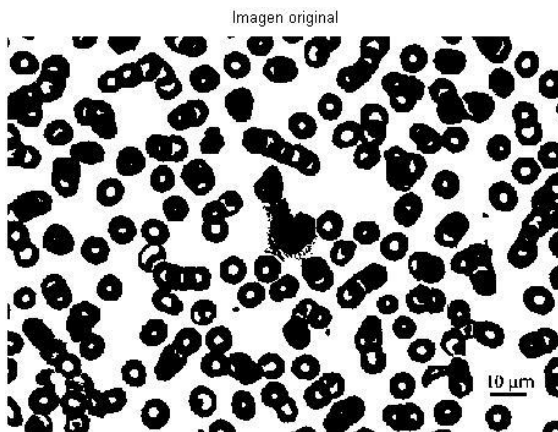
Ejemplo 7: En las siguientes imágenes se muestra el procesamiento de unas células, en éstas se encuentra los bordes. Este tipo de procesamiento puede ser de gran utilidad en el área médica y biológica, ya que como vemos, puede ser utilizada para contar la cantidad de células que existen en cierta área. Las plantillas utilizadas son las siguientes:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & -2 & 0 \\ -2 & 5 & -2 \\ 0 & -2 & 0 \end{bmatrix}, I = -1.0$$

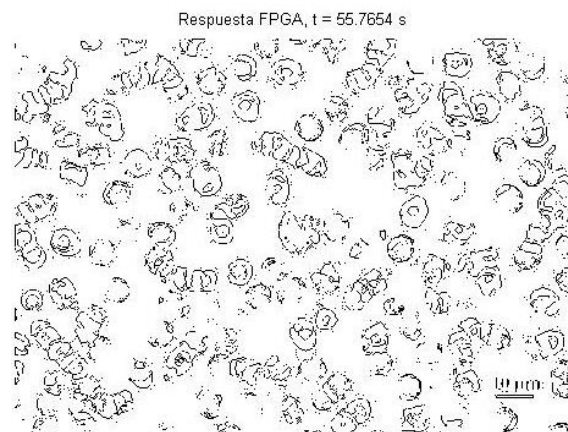
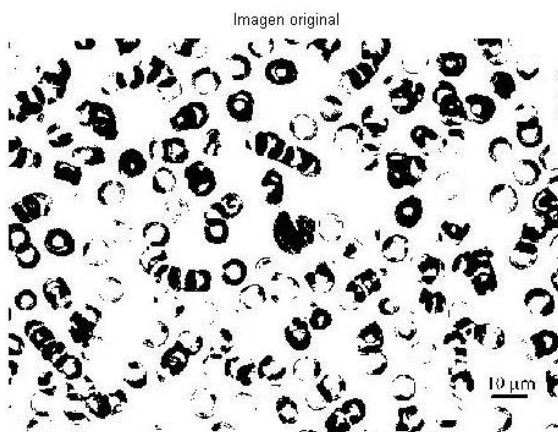
En la siguiente imagen se muestran células de la sangre; en (a) se muestra el procesamiento de la imagen en escala de grises, en (b) se muestra el procesamiento de la imagen en blanco y negro, y finalmente en (c) se realizó primeramente una segmentación global, después se procesó con la CNN.



(a) Células de la sangre en escala de grises en la parte izquierda, en la parte derecha el resultado de la búsqueda de contornos con la CNN.

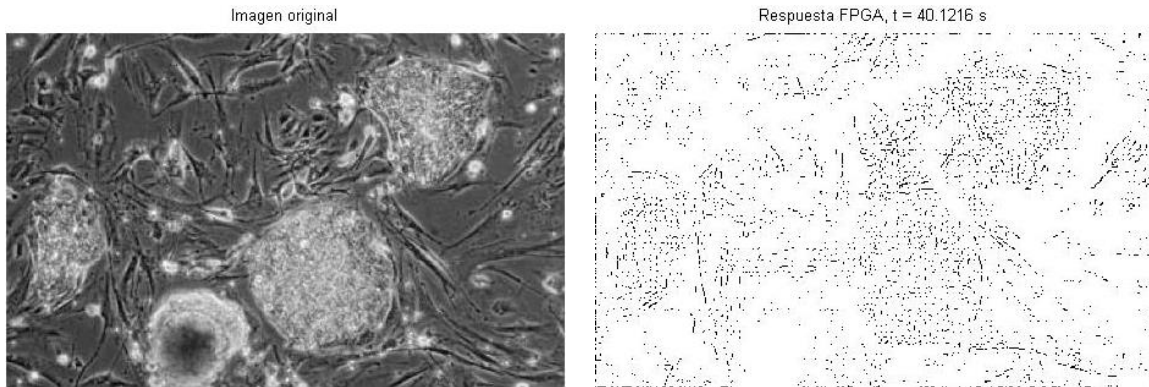


(b) Células de la sangre en escala de blancos y negros en la parte izquierda, en la parte derecha el resultado de la búsqueda de contornos con la CNN.

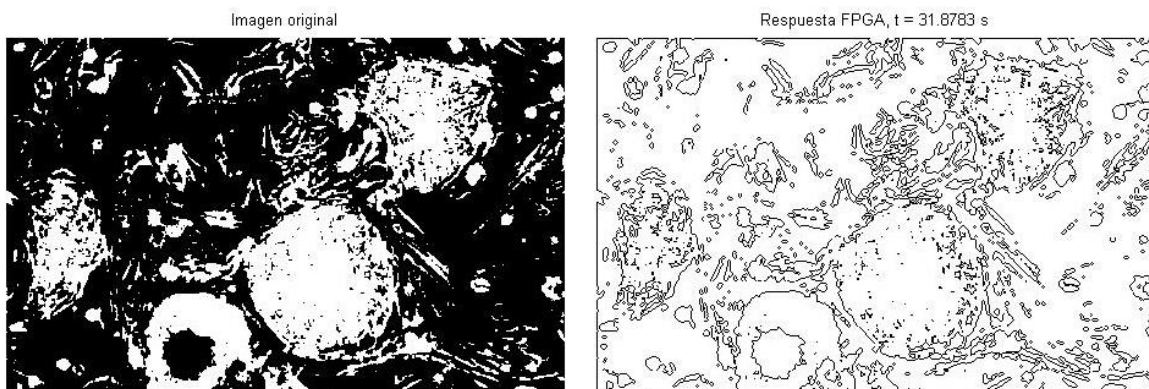


(c) Células de la sangre después de realizarles segmentación global en la parte izquierda, en la parte derecha el resultado de la búsqueda de contornos con la CNN.

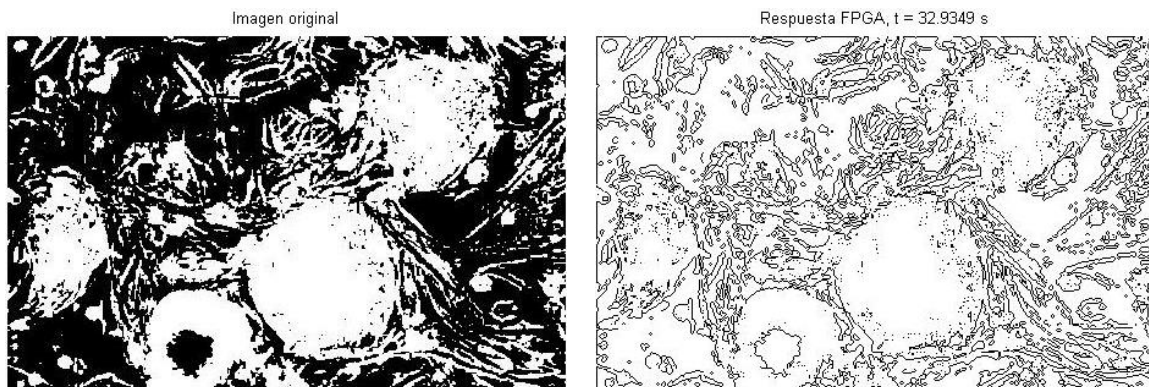
En las siguientes imágenes se muestran células regenerativas, en (a) se muestra el procesamiento de la imagen en escala de grises, en (b) se muestra el procesamiento de la imagen en blanco y negro, y finalmente en (c) se realizó primeramente una segmentación global, después se procesó con la CNN.



(a) *Células regenerativas en escala de grises en la parte izquierda, en la parte derecha el resultado de la búsqueda de contornos con la CNN.*

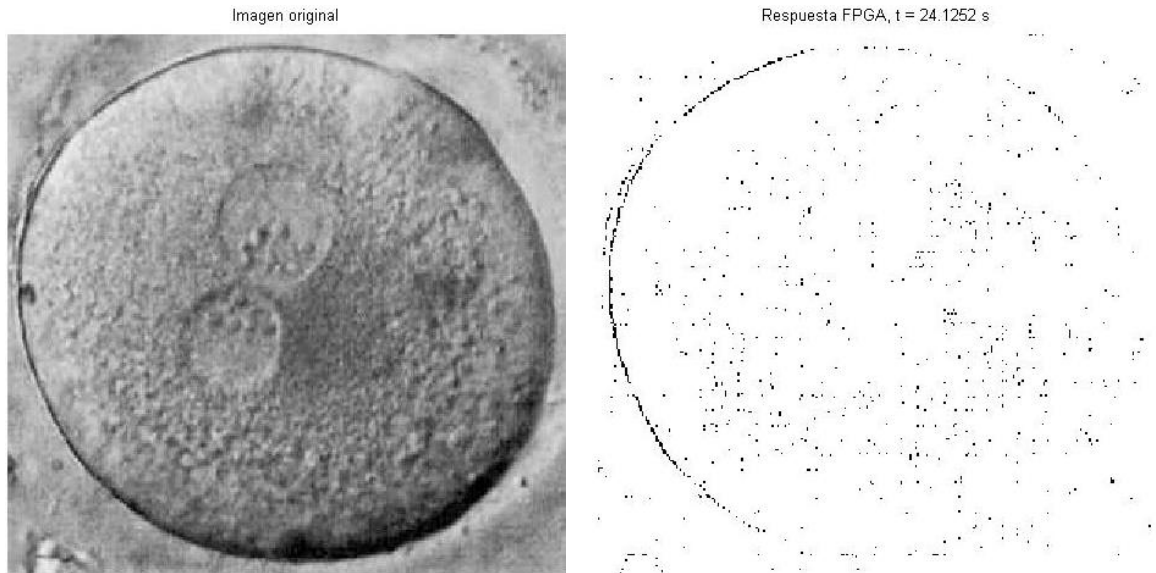


(b) *Células regenerativas en escala de blancos y negros en la parte izquierda, en la parte derecha el resultado de la búsqueda de contornos con la CNN.*

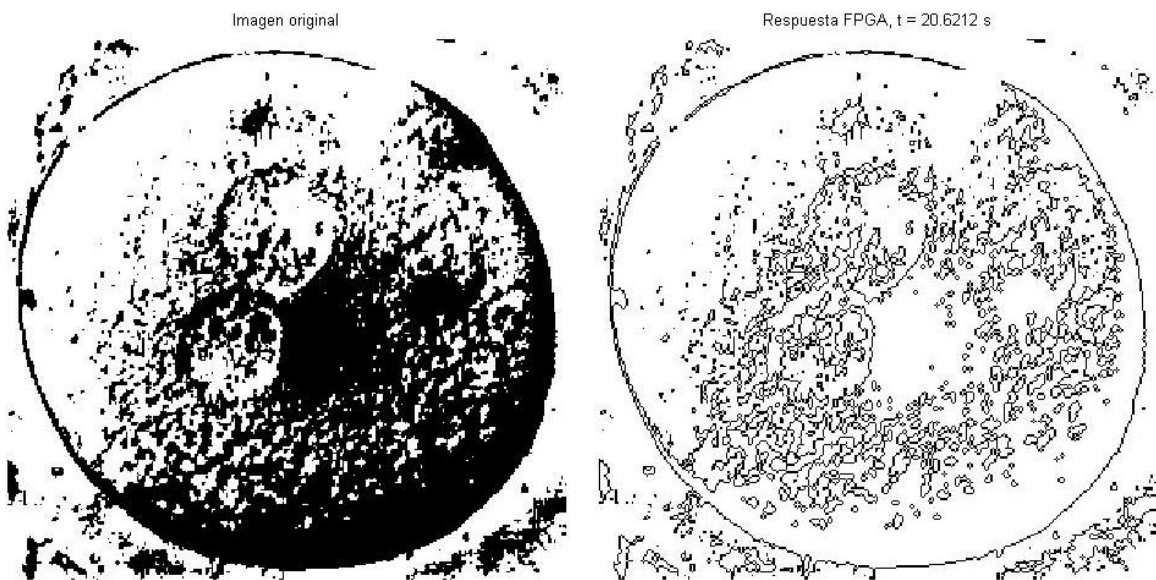


(c) *Células regenerativas después de realizarles segmentación global en la parte izquierda, en la parte derecha el resultado de la búsqueda de contornos con la CNN.*

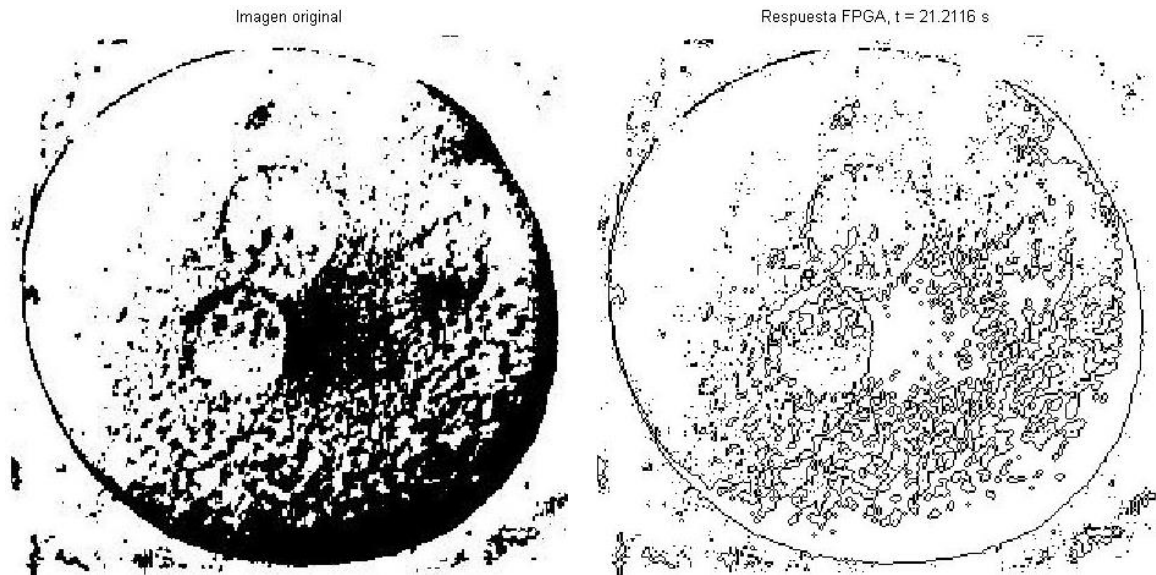
En las siguientes imágenes se muestra una célula madre, en (a) se muestra el procesamiento de la imagen en escala de grises, en (b) se muestra el procesamiento de la imagen en blanco y negro, y finalmente en (c) se realizó primeramente una segmentación global, después se procesó con la CNN.



(a) Células madre en escala de grises en la parte izquierda, en la parte derecha el resultado de la búsqueda de contornos con la CNN.



(b) Células madre en escala de blancos y negros en la parte izquierda, en la parte derecha el resultado de la búsqueda de contornos con la CNN.

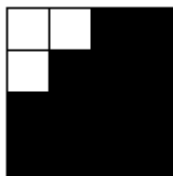


(c) Células regenerativas después de realizarles segmentación global en la parte izquierda, en la parte derecha el resultado de la búsqueda de contornos con la CNN.

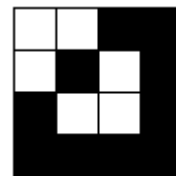
Un aspecto importante de este trabajo son las plantillas. Como vimos en el capítulo 4, la resolución numérica para los cálculos toma valores con resolución de 18 bits de punto fijo, donde se toma 1 bit para el signo, 6 bits para la parte entera y los 11 bits restantes para la parte fraccionaria. Esta resolución numérica puede ser insuficiente para realizar las sumas necesarias dentro de la CNN o el sumador de estado del integrador Euler de las neuronas. Para mostrar este punto tomemos las siguientes plantillas:

$$A = \begin{bmatrix} -0.24 & -5.43 & -1.29 \\ -7.31 & 62.71 & -7.31 \\ -1.29 & -5.43 & -0.24 \end{bmatrix}, B = \begin{bmatrix} -0.0058 & 0.06 & 0.13 \\ -0.07 & -34.27 & -0.07 \\ 0.13 & 0.06 & -0.0058 \end{bmatrix}, I = -1.6937$$

Estas plantillas son utilizadas para encontrar bordes en imágenes en escala de grises. Ahora vamos a procesar la imagen que se muestra en (a), con lo cual esperaríamos un resultado como es mostrado en (b).

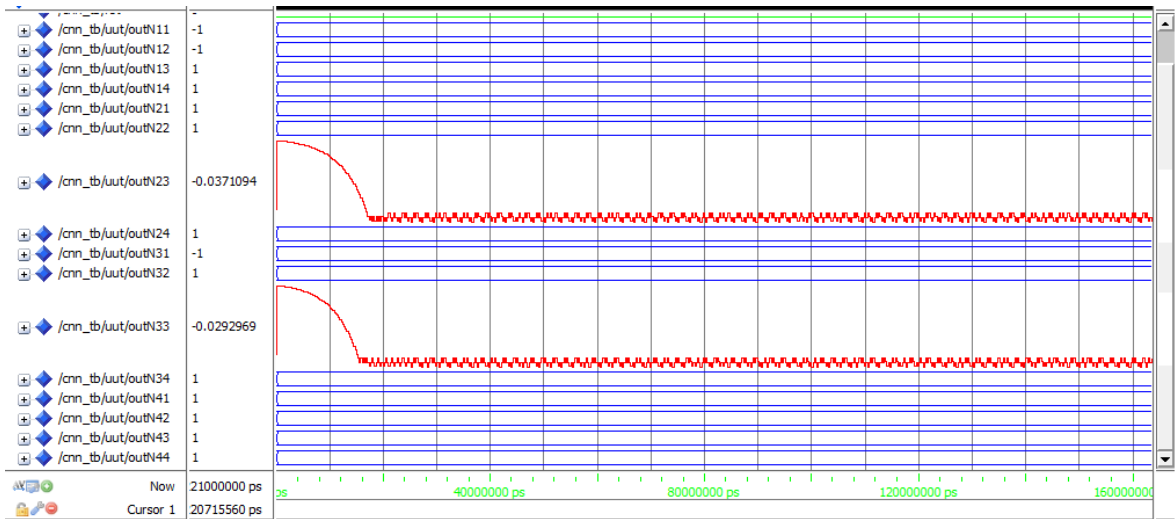


(a) Imagen de entrada.



(b) Imagen de salida deseada.

Al realizar el procesamiento dentro de la CNN realizada con el FPGA, se observa que comienzan a oscilar los valores de los pixeles que deberían de cambiar; esto es mostrado a continuación.

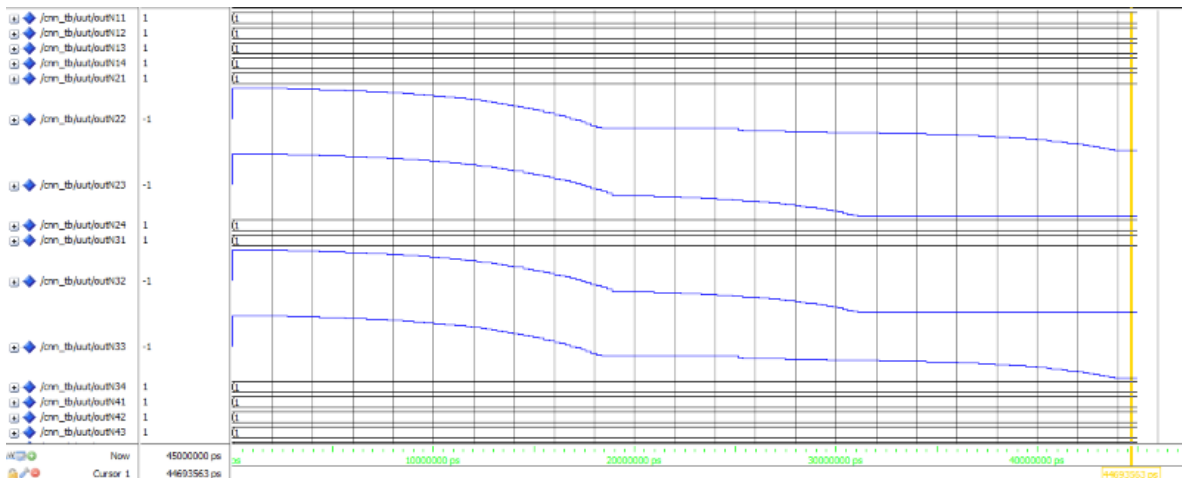


Esto es debido a que se desbordan numéricamente los acumuladores integrados dentro de cada neurona, pasando de un valor positivo a un valor negativo constantemente.

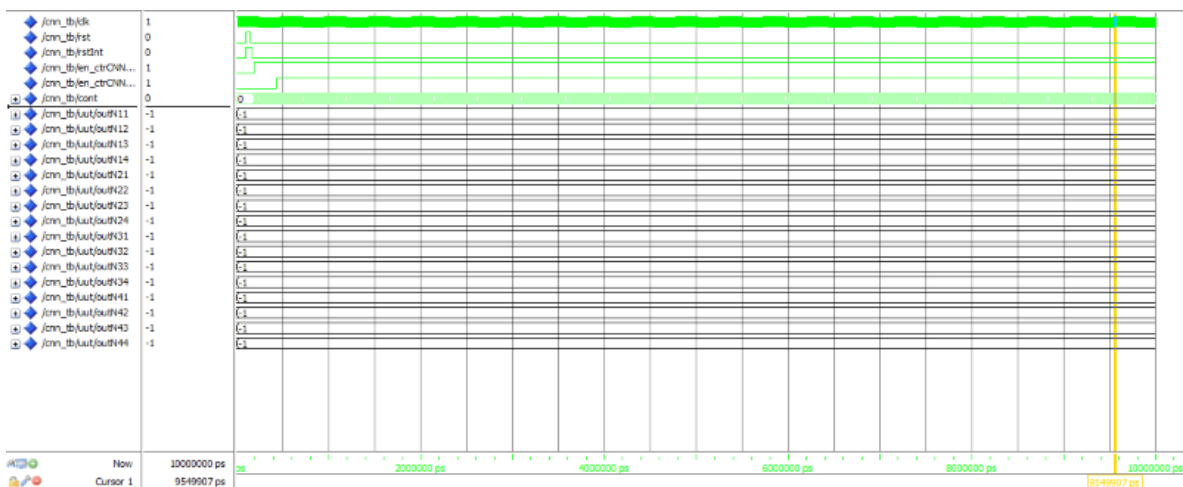
Otro punto es el hecho de que en este trabajo la condición de frontera es fija. Como mencionamos en capítulos anteriores, esta condición la dejamos fija con valor de 0; esta condición podría generar errores o resultados incorrectos. Para mostrar estos puntos anteriores, tomemos como base las siguientes plantillas, las cuales son para encontrar bordes en imágenes en escala de grises:

$$A = \begin{bmatrix} -0.24 & -5.43 & -1.29 \\ -7.31 & 62.71 & -7.31 \\ -1.29 & -5.43 & -0.24 \end{bmatrix}, B = \begin{bmatrix} -0.0058 & 0.06 & 0.13 \\ -0.07 & -34.27 & -0.07 \\ 0.13 & 0.06 & -0.0058 \end{bmatrix}, I = -1.6937$$

Primero procesaremos un bloque que contenga sólo valores de 1's; el resultado del procesamiento del bloque se muestra a continuación.



Como observamos en la figura, los valores obtenidos corresponden a los deseados, ya que cambian los cuatros valores del centro que compone a la CNN de 4×4 . Ahora haremos la comparación procesando en esta ocasión un bloque que contenga únicamente valores de -1's. El resultado del procesamiento se muestra en la siguiente figura.



Como observamos, en esta ocasión ninguno de los valores que rodean a la CNN ha cambiado de valor como se debería de esperar para un procesamiento de búsqueda de contornos. Esto es debido a que, como se mencionó anteriormente, las neuronas de frontera contienen un valor fijo de 0, esto es, un valor de gris y para que cambien las neuronas externas de la CNN esta frontera tendría que ser de valor +1.

Conclusiones

El principal objetivo de este trabajo ha sido el desarrollar un sistema CNN que sea multiplexado para el procesamiento de imágenes, y como fue mostrado durante el desarrollo de esta tesis, se ha logrado cumplir con este objetivo. Para comprobar el funcionamiento de nuestra CNN multiplexada digital, se realizó un simulador que nos indicará la forma en la que ésta iba a operar, la cual entregó resultados aceptables.

Como fue mostrado en el último capítulo, durante la implementación de la CNN multiplexada digital en FPGA, los resultados que entregó para el procesado de imágenes tanto en blanco y negro como en escala de grises, fueron buenos, sin embargo, como se mostró en el apartado de resultados de este trabajo, se tienen que considerar aspectos muy importantes para poder ser utilizada, en este caso se detalló el tipo de plantillas que pueden ser utilizadas.

En la siguiente tabla mostraremos algunas conclusiones importantes que se obtuvieron durante el desarrollo de la CNN multiplexada para el procesamiento de imágenes utilizando un FPGA.

Ventajas	Desventajas
<ul style="list-style-type: none">• Reducción de neuronas para el procesado de imágenes.• Reducción de recursos necesarios para el procesado de imágenes utilizando CNN.• Capacidad de procesar imágenes de prácticamente cualquier tamaño utilizando multiplexado de la CNN.	<ul style="list-style-type: none">• Aumento en el tiempo de procesado de imágenes comparado con otras CNN's implementadas dentro de ASIC's o a otras CNN's de mayor tamaño realizada en este trabajo.• Limitado debido a la resolución numérica con la cual fue implementada la CNN.• Posibles errores en los bloques de CNN.

Como vemos en las ventajas, esta propuesta de CNN multiplexada requiere de pocas neuronas para resolver problemas como lo es el procesamiento de imágenes, y no se requiere

de la misma cantidad de neuronas que se requerirían si no fuera así. Otro aspecto es que debido a la reducción de cantidad de neuronas de la CNN, se disminuye la cantidad de recursos que se requieren para poder realizarla. Otro aspecto fue el hecho de que gracias al multiplexado de la CNN se pueden procesar imágenes de prácticamente cualquier tamaño. Sin embargo, en este punto hay que aclarar, como se hizo en su momento, sobre las imágenes que tienen cantidades impares en las filas o columnas, donde se requiere agregar filas o columnas.

Dentro de las desventajas, observamos que el tiempo de procesado aumenta comparado con otras CNN's de mayor tamaño u otras CNN's implementadas analógicamente dentro de ASIC's. Esto debido al multiplexado que se realiza por el traslape que se existe para disminuir los errores inherentes al multiplexado. Esto provoca que existan pixeles, para el caso de procesamiento de imágenes, que se procesen dos veces. Otro aspecto es que debido a que la CNN fue implementada digitalmente, ésta tiene cierta resolución numérica. En este trabajo se utilizaron 18 bits como fue mencionado anteriormente, y esto provoca que sólo se puedan utilizar plantillas con resolución numérica que no sobrepasen cierto límite. Este límite en la resolución numérica de las plantillas puede provocar errores durante la evolución del estado de la neurona debido a que puede provocar oscilaciones, como fue mostrado en la parte de resultados de este trabajo.

Debido a estas desventajas, se propondrán trabajos futuros para poder resolver estas desventajas, además, de otros trabajos en los cuales podría ser utilizado éste como base.

Trabajo futuro

Dentro de los trabajos que se pueden realizar, está el realizar un programa para la búsqueda de plantillas utilizando la CNN que ha sido desarrollada en este trabajo. Esto con el fin de realizar otro tipo de procesamiento, o en dado caso, de buscar plantillas que realicen un mejor procesamiento de las imágenes. Lo anterior se podría realizar con algún procedimiento como la búsqueda de plantillas utilizando el algoritmo ABC visto en los capítulos anteriores.

Otro trabajo sería incrementar el tamaño de la CNN creada, ya que ésta es solo de 4×4 neuronas, lo cual es relativamente pequeña, o en dado caso, mejorar la CNN creada en este trabajo. Una de estas mejoras sería agregar la capacidad de indicarle a la CNN los valores iniciales, ya que en esta CNN toma por default los valores iniciales iguales a la imagen de entrada y hay casos donde éste no es necesario que sea de esa manera. Otra mejora sería la capacidad de cambiar el valor del paso del tiempo del integrador Euler, o en dado caso, realizar una mejora al integrador utilizado, esto para mejorar la respuesta de la CNN. Y finalmente, otra mejora propuesta a esta CNN sería mejorar la resolución numérica ya que, como se vio anteriormente, ésta realiza los cálculos utilizando una resolución de 18 bits, la cual puede ser incrementada.

Apéndice

A. Código en MATLAB para el multiplexado de una CNN.

A1. Programa principal (main.m)

```
% Programa principal para el multiplexado de la CNN 4x4
% Autor: Ing. José de Jesús Morales Romero
% SEES-CINVESTAV
clear
clc
imagen = imread('./imagenes/laberinto2.png'); % Carga la imagen
imagenCNNTemporal = im2imCNN(imagen,1); % Convierte a escala de grises
imagenA = imagenCNNTemporal;
imagenCNN = uint8(compruebaTamano(imagenCNNTemporal)); %Comprueba el tamaño de la imagen

%Plantillas
templateA = [0,4.4,0; 4.4,3.6,4.4; 0,4.4,0];
templateB = [0,0,0;0,10.7,0;0,0,0];
templateI = 7;

tic % Comienza el conteo de tiempo
FilasRed = 4; %Número de filas de la red neuronal
ColumnasRed = 4; %Número de columnas de la red neuronal
imagenCNNEntrada = im2CNN(imagenCNN); %Convierte a un tipo de imagen CNN
%Calcula la cantidad de bloques
[BloqueX, BloqueY, imagenCNNEntrada] = calculaBloques(imagenCNNEntrada);
%Coloca los templates A y B, además del Bias.
actualizaTemplates(templateA, templateB,templateI, FilasRed, ColumnasRed);
valFrontera = 0; % Valor de la frontera
actualizaFrontera(valFrontera); %Actualiza el valor de la frontera
%bAndW = 0, para bloque diferente a B o N, -1 Blancos y +1 Negros
bAndW = 0;
% 0 es el primero, 1 no es el primero
negros = 0;
blancos = 0;
nVeces = input('Ingrese la cantidad de iteraciones: ');
if(nVeces < 1)
    disp('No es posible procesar la imagen');
    toc;
else
    for mVeces=1:nVeces
        imagenCNNSalida = imagenCNNEntrada;
        imagenCNNestado = imagenCNNEntrada;
        for yBloque=1:BloqueY
            for xBloque=1:BloqueX
                [imagenCNNBloque, imagenCNNBloqueEdo] = obtenerImagenCNNBloque(yBloque, xBloque, ...
                    imagenCNNEntrada, imagenCNNestado);
                bAndW = checaBloque(imagenCNNBloque);
                if (bAndW == 0)
                    [salidaRed, estadoRed, estadoDeRed] = ...
                        evalCNN(FilasRed, ColumnasRed, imagenCNNBloque);
                elseif (bAndW == -1)
                    if (blancos == 0)
                        blancos = 1;
                        [salidaRed, estadoRed, estadoDeRed] = ...
                            evalCNN(FilasRed, ColumnasRed, imagenCNNBloque);
                        blancosSalida = salidaRed;
                        blancosEdo = estadoRed;
                    else
                        salidaRed = blancosSalida;
                        estadoRed = blancosEdo;
                    end
                elseif (bAndW == 1)
                    if (negros == 0)
                        negros = 1;
                        [salidaRed, estadoRed, estadoDeRed] = ...
                            evalCNN(FilasRed, ColumnasRed, imagenCNNBloque);
                        negrosSalida = salidaRed;
```

```

        negrosEdo = estadoRed;
    else
        salidaRed = negrosSalida;
        estadoRed = negrosEdo;
    end
end
end
[imagenCNNestado, imagenCNNsalida] = actualizaCNN(estadoRed,imagenCNNestado, ...
salidaRed, imagenCNNsalida, yBloque, xBloque, BloqueY, BloqueX);
end
end
imagenCNNEntrada = imagenCNNsalida;
end
imagenCNNSalidaCNN = conv2Img(imagenCNNsalida);
elapsedTime2 = toc; % Finaliza el medir tiempo
mensaje = ['El tiempo de ejecución con Simulink fue de: ', num2str(elapsedTime2), ' s.'];
disp(mensaje);
mensaje1 = ['Respuesta SIMULINK, t = ', num2str(elapsedTime2), ' s'];
figure('Name','CNN multiplexada','NumberTitle','off')
subplot(1,2,1);
imshow(imagenA);
title('Imagen original');
subplot(1,2,2);
imshow(imagenCNNSalidaCNN);
title(mensaje1);
end

```

A2. Función convertidor a escala de grises (im2imCNN)

```

function [imagenSalida] = im2imCNN(imagenEntrada, bandera)
%bandera = 1: Escala de grises
%bandera = 2: Escala de blanco y negros
imagenSalida = rgb2gray(imagenEntrada);
if(bandera == 2)
    %Convierte a escala de blanco y negro
    level = graythresh(imagenSalida);
    imagenSalidaTemporal = im2bw(imagenSalida, level);
    [m,n] = size(imagenSalidaTemporal);
    for i=1:m
        for j=1:n
            if(imagenSalidaTemporal(i,j) == 1)
                imagenSalida(i,j) = 255; %Es blanco el pixel
            else
                imagenSalida(i,j) = 0; %Es negro el pixel
            end
        end
    end
end
imagenSalida = uint8(imagenSalida);
end

```

A3. Función para comprobar el tamaño de la función (compruebaTamanio.m)

```

function [imagenCNN] = compruebaTamanio(imagenCNN)
[m,n] = size(imagenCNN);
if (rem(m,2)==1)
    m = m + 1;
    imagenCNNtemp = zeros(m,n);
    for i=1:(m-1)
        for j=1:n
            imagenCNNtemp(i,j) = imagenCNN(i,j);
        end
    end
    for i=1:n
        imagenCNNtemp(m,i) = imagenCNNtemp(m-1,i);
    end
    imagenCNN = imagenCNNtemp;
end
if(rem(n,2)==1)
    n = n + 1;
end

```



```

imagenCNNtemp = zeros(m,n);
for i=1:m
    for j=1:(n-1)
        imagenCNNtemp(i,j) = imagenCNN(i,j);
    end
end
for i=1:m
    imagenCNNtemp(i,n) = imagenCNNtemp(i,n-1);
end
imagenCNN = imagenCNNtemp;
end
end

```

A4. Función para convertir la imagen a valores de CNN (im2CNN.m)

```

function [imagenCNNentrada]=im2CNN(imagenGris)
imagenGris = uint8(imagenGris);
[m,n] = size(imagenGris);
imagenCNNentrada = zeros(m,n);
for i=1:m %Convierte la imagen a CNN
    for j=1:n
        imagenCNNentrada(i,j) = 1-((2.*double(imagenGris(i,j)))/255);
    end
end
end

```

A5. Función para calcular la cantidad de bloques (calculaBloques.m)

```

function [BloqueX, BloqueY, imagenCNN] = calculaBloques(imagenCNN)
[m,n] = size(imagenCNN);
BloqueY = fix(m/4) + fix((m-2)/4);
BloqueX = fix(n/4) + fix((n-2)/4);
end

```

A.6. Función para ingresar los valores de las plantillas a la CNN (actualizaTemplates.m)

```

function actualizaTemplates(A, B, Iij, FilasRed, ColumnasRed)
for i=1:FilasRed
    for j=1:ColumnasRed
        %Coloca los templates para la CNN
        for m=1:3
            for n=1:3
                templateA = sprintf(...
'set_param(''neuronav3/NeuronaCij%d,%d/A%d,%d'', 'value', '%.3f''), i,j,n,m,A(n,m));
                eval(templateA);
                templateB = sprintf(...
'set_param(''neuronav3/NeuronaCij%d,%d/B%d,%d'', 'value', '%.3f''), i,j,n,m,B(n,m));
                eval(templateB);
            end
        end
        % Coloca Bias
        Bias = sprintf(...
'set_param(''neuronav3/NeuronaCij%d,%d/I'', 'value', '%.3f''), i,j,Iij);
        eval(Bias);
    end
end
end

```

A7. Función para ingresar el valor de la frontera (actualizaFrontera)

```

function actualizaFrontera(valFrontera)
for i=1:20
    FronteraEntrada = sprintf(...
'set_param(''neuronav3/inF%d'', 'value', '%.3f''),i,valFrontera);
    eval(FronteraEntrada);
    FronteraSalida = sprintf(...
'set_param(''neuronav3/outF%d'', 'value', '%.3f''),i,valFrontera);

```

```

    eval(FronteraSalida);
end

```

A8. Función para obtener el bloque correspondiente (obtenerImagenCNNBloque.m)

```

function [imagenCNNBloque, imagenCNNBloqueEdo] = ...
    obtenerImagenCNNBloque(yBloque, xBloque, imagenCNNEntrada, imagenCNNEstado)
    imagenCNNBloque = ...
        imagenCNNEntrada((yBloque*2-1):(yBloque*2+2), (xBloque*2-1):(xBloque*2+2));
    imagenCNNBloqueEdo = ...
        imagenCNNEstado((yBloque*2-1):(yBloque*2+2), (xBloque*2-1):(xBloque*2+2));
end

```

A9. Función para determinar el tipo de bloque (chechaBloque.m)

```

function [bAndW] = checaBloque(imagenCNNBloque)
    tempBlancos = -ones(4,4); tempNegros = ones(4,4);
    if (isequal(imagenCNNBloque, tempBlancos) == 1)
        bAndW = -1;
    elseif (isequal(imagenCNNBloque, tempNegros) == 1)
        bAndW = 1;
    else
        bAndW = 0;
    end
end

```

A10. Función que hace la evaluación de la CNN a través de SIMULINK (evalCNN.m)

```

function [salidaRed, estadoRed, estadoDeRed] = evalCNN(FilasRed, ColumnasRed, imagenCNNBloque)
    for i=1:FilasRed
        for j=1:ColumnasRed
            imagenEntrada = sprintf('set_param(''neuronav3/inC%d_d'', 'value', '%.3f')', ...
                i, j, imagenCNNBloque(i, j)); %Imagen de entrada
            eval(imagenEntrada);
            InitialIntegral = sprintf('set_param(''neuronav3/NeuronaCij%d,%d/Delay'', ...
                'InitialCondition', '%.3f')', i, j, imagenCNNBloque(i, j));
            eval(InitialIntegral);
            InitialIntegral = sprintf('set_param(''neuronav3/NeuronaCij%d,%d/Delay1'', ...
                'InitialCondition', '%.3f')', i, j, imagenCNNBloque(i, j));
            eval(InitialIntegral);
            InitialIntegral = sprintf('set_param(''neuronav3/NeuronaCij%d,%d/Delay2'', ...
                'InitialCondition', '%.3f')', i, j, imagenCNNBloque(i, j));
            eval(InitialIntegral);
        end
    end
    sim('neuronav3'); %Se evalua la red neuronal
    [lastpoint, y] = size(stall1.Data); %Obtiene los últimos resultados
    salidaRed = [ sal11.Data(lastpoint, y), sal12.Data(lastpoint, y), ...
        sal13.Data(lastpoint, y), sal14.Data(lastpoint, y);
        sal21.Data(lastpoint, y), sal22.Data(lastpoint, y), ...
        sal23.Data(lastpoint, y), sal24.Data(lastpoint, y);
        sal31.Data(lastpoint, y), sal32.Data(lastpoint, y), ...
        sal33.Data(lastpoint, y), sal34.Data(lastpoint, y);
        sal41.Data(lastpoint, y), sal42.Data(lastpoint, y), ...
        sal43.Data(lastpoint, y), sal44.Data(lastpoint, y)];

    [lastpoint, y] = size(stall1.Data);
    estadoRed = [ stall1.Data(lastpoint, y), stal2.Data(lastpoint, y), ...
        stal3.Data(lastpoint, y), stal4.Data(lastpoint, y);
        sta21.Data(lastpoint, y), sta22.Data(lastpoint, y), ...
        sta23.Data(lastpoint, y), sta24.Data(lastpoint, y);
        sta31.Data(lastpoint, y), sta32.Data(lastpoint, y), ...
        sta33.Data(lastpoint, y), sta34.Data(lastpoint, y);
        sta41.Data(lastpoint, y), sta42.Data(lastpoint, y), ...
        sta43.Data(lastpoint, y), sta44.Data(lastpoint, y)];

    estadoDeRed = [sal11.Data, sal12.Data, sal13.Data, sal14.Data, ...
        sal21.Data, sal22.Data, sal23.Data, sal24.Data, ...

```

```

        sal31.Data, sal32.Data, sal33.Data, sal34.Data,...
        sal41.Data, sal42.Data, sal43.Data, sal44.Data];
end

```

A11. Función que guarda los resultados en memoria (actualizaCNN.m)

```

function [imagenCNNestado, imagenCNNsalida] = actualizaCNN(estadoRed,imagenCNNestado,...
    salidaRed,imagenCNNsalida, yBloque, xBloque, BloqueY, BloqueX)
ii = 1;
jj = 1;
if ((BloqueX == 1) && (BloqueY == 1))
    imagenCNNestado = estadoRed;
    imagenCNNsalida = salidaRed;
else
    if ((BloqueX == 1) && (BloqueY > 1))
        xini = 1;
        xfin = 4;
        if (yBloque == 1)
            yini = 1;
            yfin = 3;
            ii = 1;
        elseif (yBloque == BloqueY)
            yini = BloqueY*2;
            yfin = BloqueY*2 + 2;
            ii = 2;
        else
            yini = yBloque*2;
            yfin = yBloque*2 + 1;
            ii = 2;
        end
    elseif((BloqueX > 1) && (BloqueY == 1))
        yini = 1;
        yfin = 4;
        if (xBloque == 1)
            xini = 1;
            xfin = 3;
        elseif (xBloque == BloqueX)
            xini = BloqueX*2;
            xfin = BloqueX*2 + 2;
            jj = 2;
        else
            xini = BloqueX*2;
            xfin = BloqueX*2 + 1;
            jj = 2;
        end
    else
        if (yBloque == 1)
            yini = 1;
            yfin = 3;
            if (xBloque == 1)
                xini = 1;
                xfin = 3;
            elseif (xBloque == BloqueX)
                xini = BloqueX*2;
                xfin = BloqueX*2 + 2;
                jj = 2;
            else
                xini = xBloque*2;
                xfin = xBloque*2 + 1;
                jj = 2;
            end
        elseif (yBloque == BloqueY)
            ii = 2;
            yini = yBloque*2;
            yfin = yBloque*2 + 2;
            if (xBloque == 1)
                xini = 1;
                xfin = 3;
                jj = 1;
            elseif (xBloque == BloqueX)

```

```

        xini = xBloque*2;
        xfin = xBloque*2 + 2;
        jj = 2;
    else
        xini = xBloque*2;
        xfin = xBloque*2 + 1;
        jj = 2;
    end
elseif (xBloque == 1)
    xini = 1;
    yini = yBloque*2;
    xfin = 3;
    yfin = yBloque*2 + 1;
    ii = 2;
    jj = 1;
elseif (xBloque == BloqueX)
    xini = xBloque*2;
    yini = yBloque*2;
    xfin = xBloque*2 + 2;
    yfin = yBloque*2 + 1;
    ii = 2; jj = 2;
else
    xini = xBloque*2;
    yini = yBloque*2;
    xfin = xBloque*2 + 1;
    yfin = yBloque*2 + 1;
    ii = 2;
    jj = 2;
end
end
jtemp = jj;
for i=yini:yfin
    for j=xini:xfin
        imagenCNNestado(i,j) = estadoRed(ii,jj);
        imagenCNNsalida(i,j) = salidaRed(ii,jj);
        jj = jj + 1;
    end
    jj = jtemp;
    ii = ii + 1;
end
end
end
end

```

A12. Función que convierte la imagen entregada por CNN a imagen de MATLAB (conv2Img.m)

```

function [imagenSalida] = conv2Img(imagenEntrada)
[m,n] = size(imagenEntrada);
imagenSalida = imagenEntrada;
for j=1:m
    for i=1:n
        imagenSalida(j,i) = (255*(1-imagenEntrada(j,i)))/2;
    end
end
end
imagenSalida = uint8(imagenSalida);
end

```

B. Código en VHDL para el multiplexado de una CNN.

B1. Modulo principal (CNN.vhd)

```
-- CNN Multiplexada
-- Autor: Ing. José de Jesús Morales Romero
-- SEES-CINVESTAV
-- Puertos:
--   clk: Reloj a 100MHz.
--   rst: Reset general
--   rx: Receptor del RS232
--   tx: Transmisor del RS232
--   LEDs: LEDs que muestran el estado del sistema
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use work.componentes.all;

entity CNN is
    port(clk, rst, rx: in std_logic;
         LEDs: out std_logic_vector(7 downto 0);
         tx: out std_logic);
end CNN;

architecture arq_CNN of CNN is

    constant widthWord: integer := 8;
    constant widthAddr: integer := 16;
    constant sizeRAM: integer := 65536;
    signal we_in, we_out, en_in, en_out: std_logic;
    signal addr_re_in, addr_re_out: std_logic_vector(widthAddr-1 downto 0);
    signal addr_we_in, addr_we_out: std_logic_vector(widthAddr-1 downto 0);
    signal DI_in, DI_out, DO_in, DO_out: std_logic_vector(widthWord-1 downto 0);

    signal ce_rx, en_parametros, en_imagenGuardada, en_imagenIn, en_leeSubImg_in: std_logic;
    signal en_leeSubImg_out, en_conv8a18_in, en_conv8a18_out, en_ctrCNN4x4_in: std_logic;
    signal en_ctrCNN4x4_out, en_conv18a8_in, en_conv18a8_out, en_guardaSalida_in: std_logic;
    signal en_guardaSalida_out, en_Calcula_in, en_Calcula_out, en_Next, en_ctrEnvio_in:
    std_logic;
    signal en_ctrEnvio_out, ce_tx_in, ce_tx_out, rstInt: std_logic;

    signal inRS232, xMax, yMax: std_logic_vector(7 downto 0);
    signal Xini: std_logic_vector(widthAddr-1 downto 0) := (others => '0');
    signal fila, columna: std_logic_vector(widthWord-2 downto 0);
    signal maxFila, maxColumna: std_logic_vector(widthWord-2 downto 0);
    signal A11, A12, A13, A21, A22, A23, A31, A32, A33: std_logic_vector(17 downto 0);
    signal B11, B12, B13, B21, B22, B23, B31, B32, B33: std_logic_vector(17 downto 0);

    signal I: std_logic_vector(35 downto 0);

    signal reg01,reg02,reg03,reg04, reg05,reg06,reg07: std_logic_vector(widthWord-1 downto 0);
    signal reg08, reg09,reg10,reg11,reg12: std_logic_vector(widthWord-1 downto 0);
    signal reg13,reg14,reg15,reg16: std_logic_vector(widthWord-1 downto 0);

    signal inN11,inN12,inN13,inN14,inN21,inN22,inN23,inN24: std_logic_vector(17 downto 0);
    signal inN31,inN32,inN33,inN34,inN41,inN42,inN43,inN44: std_logic_vector(17 downto 0);
    signal outN11,outN12,outN13,outN14,outN21,outN22,outN23: std_logic_vector(17 downto 0);
    signal outN24,outN31,outN32,outN33,outN34,outN41,outN42: std_logic_vector(17 downto 0);
    signal outN43,outN44: std_logic_vector(17 downto 0);

    signal sal11,sal12,sal13,sal14,sal21,sal22,sal23: std_logic_vector(widthWord-1 downto 0);
    signal sal24,sal31,sal32,sal33,sal34,sal41,sal42: std_logic_vector(widthWord-1 downto 0);
    signal sal43,sal44: std_logic_vector(widthWord-1 downto 0);

    signal banN11, banN12, banN13, banN14, banN21, banN22, banN23, banN24: std_logic;
    signal banN31, banN32, banN33, banN34, banN41, banN42, banN43, banN44: std_logic;

    signal pixel, totalPixeles: std_logic_vector(widthWord-1 downto 0);
    signal salLEDs, dato_tx: std_logic_vector(7 downto 0);
```

```

signal clk_RAM, t_clk_RAM: std_logic;
signal contCLK: std_logic_vector(1 downto 0) := (others => '0');
constant inF: std_logic_vector(17 downto 0) := "00000000000000000000";
constant dt: std_logic_vector(17 downto 0) := "0000000000000000010";

signal ctrNeurona: std_logic_vector(5 downto 0);

begin

proceso_clkRAM: process(clk) begin
    if(rising_edge(clk))then contCLK <= contCLK + 1; t_clk_RAM <= contCLK(1); end if;
end process;

process(clk) begin
    if(rising_edge(clk))then clk_RAM <= t_clk_RAM; end if;
end process;

Receptor: rxRS232 port map(clk => clk, rst => rst, rx => rx, ce_rx => ce_rx,
    inRS232 => inRS232);

Parametros: pila port map(clk => clk, rst => rst, rstInt => rstInt, inRS232 => inRS232,
    ce_rx => ce_rx, en_parametros => en_parametros, xMax => xMax, yMax => yMax,
    A11 => A11, A12 => A12, A13 => A13, A21 => A21, A22 => A22, A23 => A23,
    A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12, B13 => B13,
    B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33, I => I,
    pixel => pixel);

RAM: moduloRAM generic map(widthWord => widthWord, widthAddr => widthAddr, sizeRAM =>
    sizeRAM)
    port map(clk => clk, we_in => we_in, en_in => en_in, addr_re_in => addr_re_in,
    addr_we_in => addr_we_in, DI_in => DI_in, DO_in => DO_in, we_out => we_out,
    en_out => en_out, addr_re_out => addr_re_out, addr_we_out => addr_we_out,
    DI_out => DI_out, DO_out => DO_out);

LeeImgIn: ctrWeRe generic map(widthWord => widthWord, widthAddr => widthAddr)
    port map(clk => clk, rst => rst, rstInt => rstInt, ce_rx => ce_rx,
    en_imagenIn => en_imagenIn, en_imagenGuardada => en_imagenGuardada, pixel =>
    pixel,
    we_in => we_in, addr_we_in => addr_we_in, DI_in => DI_in,
    totalPixeles => totalPixeles);

ControlCNN: ctrCNN generic map(widthWord => widthWord, widthAddr => widthAddr)
    port map(clk => clk, rst => rst, rstInt => rstInt, en_imagenIn => en_imagenIn,
    en_parametros => en_parametros, en_imagenGuardada => en_imagenGuardada,
    en_leeSubImg_in => en_leeSubImg_in, en_leeSubImg_out => en_leeSubImg_out,
    en_conv8a18_in => en_conv8a18_in, en_conv8a18_out => en_conv8a18_out,
    en_ctrCNN4x4_in => en_ctrCNN4x4_in, en_ctrCNN4x4_out => en_ctrCNN4x4_out,
    en_conv18a8_in => en_conv18a8_in, en_conv18a8_out => en_conv18a8_out,
    en_guardaSalida_in => en_guardaSalida_in, en_Next => en_Next, en_in => en_in,
    en_guardaSalida_out => en_guardaSalida_out, en_Calcula_in => en_Calcula_in,
    en_Calcula_out => en_Calcula_out, en_ctrEnvio_in => en_ctrEnvio_in,
    en_ctrEnvio_out => en_ctrEnvio_out, salLEDs => salLEDs, en_out => en_out);

LeeSubImagen: leeSubImg generic map(widthWord => widthWord, widthAddr => widthAddr)
    port map(clk_RAM => clk_RAM, en_leeSubImg_in => en_leeSubImg_in, Xini =>
    Xini,
    en_leeSubImg_out => en_leeSubImg_out, xMax => xMax, addr_re_in => addr_re_in,
    reg01 => reg01, reg02 => reg02, reg03 => reg03, reg04 => reg04, reg05 =>
    reg05,
    reg06 => reg06, reg07 => reg07, reg08 => reg08, reg09 => reg09, reg10 =>
    reg10,
    reg11 => reg11, reg12 => reg12, reg13 => reg13, reg14 => reg14, reg15 =>
    reg15,
    reg16 => reg16, DO_in => DO_in);

Convertidor8a18bits: conv8a18bits port map(clk => clk, rst => rst, rstInt => rstInt,
    en_conv8a18_in => en_conv8a18_in, en_conv8a18_out => en_conv8a18_out,
    reg01 => reg01, reg02 => reg02, reg03 => reg03, reg04 => reg04,
    reg05 => reg05, reg06 => reg06, reg07 => reg07, reg08 => reg08,
    reg09 => reg09, reg10 => reg10, reg11 => reg11, reg12 => reg12,
    reg13 => reg13, reg14 => reg14, reg15 => reg15, reg16 => reg16,

```

```

inN11 => inN11, inN12 => inN12, inN13 => inN13, inN14 => inN14,
inN21 => inN21, inN22 => inN22, inN23 => inN23, inN24 => inN24,
inN31 => inN31, inN32 => inN32, inN33 => inN33, inN34 => inN34,
inN41 => inN41, inN42 => inN42, inN43 => inN43, inN44 => inN44);

ControlCNN4x4: ctrCNN4x4 port map(clk => clk, rst => rst, rstInt => rstInt,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, banN11 => banN11,
banN12 => banN12, banN13 => banN13, banN14 => banN14, banN21 => banN21,
banN22 => banN22, banN23 => banN23, banN24 => banN24, banN31 => banN31,
banN32 => banN32, banN33 => banN33, banN34 => banN34, banN41 => banN41,
banN42 => banN42, banN43 => banN43, banN44 => banN44,
en_ctrCNN4x4_out => en_ctrCNN4x4_out);

N11: Neurona port map(clk => clk, banN => banN11, ctrNeurona => ctrNeurona,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, A11 => A11, A12 => A12, A13 => A13, A21 => A21,
A22 => A22, A23 => A23, A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12,
B13 => B13, B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33,
vu11 => inF, vu12 => inF, vu13 => inF, vu21 => inF, vu22 => inN11, vu23 => inN12,
vu31 => inF, vu32 => inN21, vu33 => inN22, vy11 => inF, vy12 => inF, vy13 => inF,
vy21 => inF, vy22 => outN11, vy23 => outN12, vy31 => inF, vy32 => outN21,
vy33 => outN22, I => I, dt => dt);

N12: Neurona port map(clk => clk, banN => banN12, ctrNeurona => ctrNeurona,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, A11 => A11, A12 => A12, A13 => A13, A21 => A21,
A22 => A22, A23 => A23, A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12,
B13 => B13, B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33,
vu11 => inF, vu12 => inF, vu13 => inF, vu21 => inN11, vu22 => inN12, vu23 => inN13,
vu31 => inN21, vu32 => inN22, vu33 => inN23, vy11 => inF, vy12 => inF, vy13 => inF,
vy21 => outN11, vy22 => outN12, vy23 => outN13, vy31 => outN21, vy32 => outN22,
vy33 => outN23, I => I, dt => dt);

N13: Neurona port map(clk => clk, banN => banN13, ctrNeurona => ctrNeurona,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, A11 => A11, A12 => A12, A13 => A13, A21 => A21,
A22 => A22, A23 => A23, A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12,
B13 => B13, B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33,
vu11 => inF, vu12 => inF, vu13 => inF, vu21 => inN12, vu22 => inN13, vu23 => inN14,
vu31 => inN22, vu32 => inN23, vu33 => inN24, vy11 => inF, vy12 => inF, vy13 => inF,
vy21 => outN12, vy22 => outN13, vy23 => outN14, vy31 => outN22, vy32 => outN23,
vy33 => outN24, I => I, dt => dt);

N14: Neurona port map(clk => clk, banN => banN14, ctrNeurona => ctrNeurona,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, A11 => A11, A12 => A12, A13 => A13, A21 => A21,
A22 => A22, A23 => A23, A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12,
B13 => B13, B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33,
vu11 => inF, vu12 => inF, vu13 => inF, vu21 => inN13, vu22 => inN14, vu23 => inF,
vu31 => inN23, vu32 => inN24, vu33 => inF, vy11 => inF, vy12 => inF, vy13 => inF,
vy21 => outN13, vy22 => outN14, vy23 => inF, vy31 => outN23, vy32 => outN24,
vy33 => inF, I => I, dt => dt);

N21: Neurona port map(clk => clk, banN => banN21, ctrNeurona => ctrNeurona,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, A11 => A11, A12 => A12, A13 => A13, A21 => A21,
A22 => A22, A23 => A23, A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12,
B13 => B13, B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33,
vu11 => inF, vu12 => inN11, vu13 => inN12, vu21 => inF, vu22 => inN21, vu23 => inN22,
vu31 => inF, vu32 => inN31, vu33 => inN32, vy11 => inF, vy12 => outN11, vy13 => outN12,
vy21 => inF, vy22 => outN21, vy23 => outN22, vy31 => inF, vy32 => outN31, vy33 =>
outN32,
I => I, dt => dt);

N22: Neurona port map(clk => clk, banN => banN22, ctrNeurona => ctrNeurona,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, A11 => A11, A12 => A12, A13 => A13, A21 => A21,
A22 => A22, A23 => A23, A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12,
B13 => B13, B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33,
vu11 => inN11, vu12 => inN12, vu13 => inN13, vu21 => inN21, vu22 => inN22, vu23 =>
inN23,
vu31 => inN31, vu32 => inN32, vu33 => inN33, vy11 => outN11, vy12 => outN12,
vy13 => outN13, vy21 => outN21, vy22 => outN22, vy23 => outN23, vy31 => outN31,
vy32 => outN32, vy33 => outN33, I => I, dt => dt);

N23: Neurona port map(clk => clk, banN => banN23, ctrNeurona => ctrNeurona,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, A11 => A11, A12 => A12, A13 => A13, A21 => A21,

```

```

A22 => A22, A23 => A23, A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12,
B13 => B13, B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33,
vu11 => inN12, vu12 => inN13, vu13 => inN14, vu21 => inN22, vu22 => inN23, vu23 =>
inN24,
vu31 => inN32, vu32 => inN33, vu33 => inN34, vy11 => outN12, vy12 => outN13,
vy13 => outN14, vy21 => outN22, vy22 => outN23, vy23 => outN24, vy31 => outN32,
vy32 => outN33, vy33 => outN34, I => I, dt => dt);

N24: Neurona port map(clk => clk, banN => banN24, ctrNeurona => ctrNeurona,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, A11 => A11, A12 => A12, A13 => A13, A21 => A21,
A22 => A22, A23 => A23, A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12,
B13 => B13, B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33,
vu11 => inN13, vu12 => inN14, vu13 => inF, vu21 => inN23, vu22 => inN24, vu23 => inF,
vu31 => inN33, vu32 => inN34, vu33 => inF, vy11 => outN13, vy12 => outN14, vy13 => inF,
vy21 => outN23, vy22 => outN24, vy23 => inF, vy31 => outN33, vy32 => outN34, vy33 =>
inF,
I => I, dt => dt);

N31: Neurona port map(clk => clk, banN => banN31, ctrNeurona => ctrNeurona,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, A11 => A11, A12 => A12, A13 => A13, A21 => A21,
A22 => A22, A23 => A23, A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12,
B13 => B13, B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33,
vu11 => inF, vu12 => inN21, vu13 => inN22, vu21 => inF, vu22 => inN31, vu23 => inN32,
vu31 => inF, vu32 => inN41, vu33 => inN42, vy11 => inF, vy12 => outN21, vy13 => outN22,
vy21 => inF, vy22 => outN31, vy23 => outN32, vy31 => inF, vy32 => outN41, vy33 =>
outN42,
I => I, dt => dt);

N32: Neurona port map(clk => clk, banN => banN32, ctrNeurona => ctrNeurona,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, A11 => A11, A12 => A12, A13 => A13, A21 => A21,
A22 => A22, A23 => A23, A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12,
B13 => B13, B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33,
vu11 => inN21, vu12 => inN22, vu13 => inN23, vu21 => inN31, vu22 => inN32, vu23 =>
inN33,
vu31 => inN41, vu32 => inN42, vu33 => inN43, vy11 => outN21, vy12 => outN22,
vy13 => outN23, vy21 => outN31, vy22 => outN32, vy23 => outN33, vy31 => outN41,
vy32 => outN42, vy33 => outN43, I => I, dt => dt);

N33: Neurona port map(clk => clk, banN => banN33, ctrNeurona => ctrNeurona,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, A11 => A11, A12 => A12, A13 => A13, A21 => A21,
A22 => A22, A23 => A23, A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12,
B13 => B13, B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33,
vu11 => inN22, vu12 => inN23, vu13 => inN24, vu21 => inN32, vu22 => inN33, vu23 =>
inN34,
vu31 => inN42, vu32 => inN43, vu33 => inN44, vy11 => outN22, vy12 => outN23,
vy13 => outN24, vy21 => outN32, vy22 => outN33, vy23 => outN34, vy31 => outN42,
vy32 => outN43, vy33 => outN44, I => I, dt => dt);

N34: Neurona port map(clk => clk, banN => banN34, ctrNeurona => ctrNeurona,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, A11 => A11, A12 => A12, A13 => A13, A21 => A21,
A22 => A22, A23 => A23, A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12,
B13 => B13, B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33,
vu11 => inN23, vu12 => inN24, vu13 => inF, vu21 => inN33, vu22 => inN34, vu23 => inF,
vu31 => inN43, vu32 => inN44, vu33 => inF, vy11 => outN23, vy12 => outN24, vy13 => inF,
vy21 => outN33, vy22 => outN34, vy23 => inF, vy31 => outN43, vy32 => outN44, vy33 =>
inF,
I => I, dt => dt);

N41: Neurona port map(clk => clk, banN => banN41, ctrNeurona => ctrNeurona,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, A11 => A11, A12 => A12, A13 => A13, A21 => A21,
A22 => A22, A23 => A23, A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12,
B13 => B13, B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33,
vu11 => inF, vu12 => inN31, vu13 => inN32, vu21 => inF, vu22 => inN41, vu23 => inN42,
vu31 => inF, vu32 => inF, vu33 => inF, vy11 => inF, vy12 => outN31, vy13 => outN32,
vy21 => inF, vy22 => outN41, vy23 => outN42, vy31 => inF, vy32 => inF, vy33 => inF,
I => I, dt => dt);

N42: Neurona port map(clk => clk, banN => banN42, ctrNeurona => ctrNeurona,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, A11 => A11, A12 => A12, A13 => A13, A21 => A21,
A22 => A22, A23 => A23, A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12,
B13 => B13, B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33,

```



```

    vu11 => inN31, vu12 => inN32, vu13 => inN33, vu21 => inN41, vu22 => inN42, vu23 =>
inN43,
vu31 => inF, vu32 => inF, vu33 => inF, vy11 => outN31, vy12 => outN32, vy13 => outN33,
vy21 => outN41, vy22 => outN42, vy23 => outN43, vy31 => inF, vy32 => inF, vy33 => inF,
I => I, dt => dt);

N43: Neurona port map(clk => clk, banN => banN43, ctrNeurona => ctrNeurona,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, A11 => A11, A12 => A12, A13 => A13, A21 => A21,
A22 => A22, A23 => A23, A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12,
B13 => B13, B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33,
vu11 => inN32, vu12 => inN33, vu13 => inN34, vu21 => inN42, vu22 => inN43, vu23 =>
inN44,
vu31 => inF, vu32 => inF, vu33 => inF, vy11 => outN32, vy12 => outN33, vy13 => outN34,
vy21 => outN42, vy22 => outN43, vy23 => outN44, vy31 => inF, vy32 => inF, vy33 => inF,
I => I, dt => dt);

N44: Neurona port map(clk => clk, banN => banN44, ctrNeurona => ctrNeurona,
en_ctrCNN4x4_in => en_ctrCNN4x4_in, A11 => A11, A12 => A12, A13 => A13, A21 => A21,
A22 => A22, A23 => A23, A31 => A31, A32 => A32, A33 => A33, B11 => B11, B12 => B12,
B13 => B13, B21 => B21, B22 => B22, B23 => B23, B31 => B31, B32 => B32, B33 => B33,
vu11 => inN33, vu12 => inN34, vu13 => inF, vu21 => inN43, vu22 => inN44, vu23 => inF,
vu31 => inF, vu32 => inF, vu33 => inF, vy11 => outN33, vy12 => outN34, vy13 => inF,
vy21 => outN43, vy22 => outN44, vy23 => inF, vy31 => inF, vy32 => inF, vy33 => inF,
I => I, dt => dt);

Convertidor18a8bits: conv18a8bits port map(clk => clk, rst => rst, rstInt => rstInt,
en_conv18a8_in => en_conv18a8_in, en_conv18a8_out => en_conv18a8_out,
outN11 => outN11, outN12 => outN12, outN13 => outN13, outN14 => outN14,
outN21 => outN21, outN22 => outN22, outN23 => outN23, outN24 => outN24,
outN31 => outN31, outN32 => outN32, outN33 => outN33, outN34 => outN34,
outN41 => outN41, outN42 => outN42, outN43 => outN43, outN44 => outN44,
sal11 => sal11, sal12 => sal12, sal13 => sal13, sal14 => sal14,
sal22 => sal22, sal23 => sal23, sal24 => sal24, sal31 => sal31,
sal32 => sal32, sal33 => sal33, sal34 => sal34, sal41 => sal41,
sal42 => sal42, sal43 => sal43, sal44 => sal44);

Guardar_Salida: guardaSalida generic map(widthWord => widthWord, widthAddr => widthAddr)
port map(clk_RAM => clk_RAM, en_guardaSalida_in => en_guardaSalida_in,
en_guardaSalida_out => en_guardaSalida_out, Xini => Xini, xMax => xMax,
fila => fila, columna => columna, maxFila => maxFila, DI_out => DI_out,
maxColumna => maxColumna, addr_we_out => addr_we_out, we_out => we_out,
sal11 => sal11, sal12 => sal12, sal13 => sal13, sal14 => sal14,
sal21 => sal21, sal22 => sal22, sal23 => sal23, sal24 => sal24,
sal31 => sal31, sal32 => sal32, sal33 => sal33, sal34 => sal34,
sal41 => sal41, sal42 => sal42, sal43 => sal43, sal44 => sal44);

Calcular_Valores: calculaValores generic map(widthWord => widthWord, widthAddr => widthAddr)
port map(clk => clk, clk_RAM => clk_RAM, rst => rst, rstInt => rstInt,
en_Calcula_in => en_Calcula_in, en_Calcula_out => en_Calcula_out,
en_Next => en_Next, totalPixeles => totalPixeles, fila => fila,
columna => columna, maxFila => maxFila, maxColumna => maxColumna,
Xini => Xini, xMax => xMax, yMax => yMax);

Envia_Datos: ctrEnviaDatos generic map(widthWord => widthWord, widthAddr => widthAddr)
port map(clk => clk, clk_RAM => clk_RAM, rst => rst,
en_ctrEnvio_in => en_ctrEnvio_in, en_ctrEnvio_out => en_ctrEnvio_out,
ce_tx_out => ce_tx_out, ce_tx_in => ce_tx_in, totalPixeles => totalPixeles,
addr_re_out => addr_re_out, dato_tx => dato_tx, DO_out => DO_out);

Transmisor: txRS232 port map(clk => clk, rst => rst, dato_tx => dato_tx, ce_tx_in =>
ce_tx_in,
ce_tx_out => ce_tx_out, tx => tx);

SalidaLEDs: Muestra port map(clk => clk, LEDs => LEDs, salLEDs => salLEDs);
end arq_CNN;

```

B2. Modulo Receptor (rxRS232.vhd)

```
-- Diseñador: José de Jesús Morales Romero
```

```

-- VLSI-SEES-CINVESTAV-IPN
-- Receptor RS232
-- Velocidad: 115200 bauds
-- Descripción: Receptor de datos via RS232,
-- 8 bits de dato, 1 bit de stop
-- Puertos
-- clk:         Señal de reloj 100MHz
-- rst:         Reset general
-- rx:          Entrada de recepción
-- ce_rx:       Bandera de recepción
-- inRS232:     Dato recibido
-- Banderas:    ce_rx: '0' No esta recibiendo, '1' Recibiendo datos
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity rxRS232 is
    port(clk, rst, rx: in std_logic;
         ce_rx: out std_logic;
         inRS232: out std_logic_vector(7 downto 0));
end rxRS232;

architecture arq_rxRS232 of rxRS232 is

    signal erx: std_logic := '0';
    signal regrx: std_logic_vector(7 downto 0);
    signal cntrx: std_logic_vector(8 downto 0);
    signal trx: std_logic_vector(4 downto 0);
    constant baudrx: std_logic_vector(8 downto 0) := "100100001";

begin

    process(rst, rx, trx) begin
        if(rst = '1')then erx <= '0'; ce_rx <= '0';
        else
            if(rx = '0')then erx <= '1'; ce_rx <= '1';
            else
                if(trx = "00000")then erx <= '0'; ce_rx <= '0';
                elsif(trx >= "11101")then erx <= '0'; ce_rx <= '0';
                else erx <= '1'; ce_rx <= '1';
                end if;
            end if;
        end if;
    end process;

    process(clk, rst, cntrx, trx, erx) begin
        if(rst = '1')then cntrx <= (others => '0'); trx <= (others => '0');
        else
            if(erx = '0')then cntrx <= (others => '0'); trx <= (others => '0');
            else
                if(clk'event and clk = '1')then cntrx <= cntrx + 1;
                if(cntrx = baudrx)then trx <= trx + 1; cntrx <= (others => '0');
                end if;
            end if;
        end if;
    end process;

    process (clk,rst,trx,rx,regrx) begin
        if rst='1' then regrx <= (others=>'0');
        elsif (clk'event and clk='1') then
            case (trx) is
                when "00100" => regrx(0) <= rx; when "00111" => regrx(1) <= rx;
                when "01010" => regrx(2) <= rx; when "01101" => regrx(3) <= rx;
                when "10000" => regrx(4) <= rx; when "10011" => regrx(5) <= rx;
                when "10110" => regrx(6) <= rx; when "11001" => regrx(7) <= rx;
                when others => null;
            end case;
        end if;
    end process;
end architecture arq_rxRS232;

```

```

process(erx, rst, regrx) begin
  if(rst = '1') then inRS232 <= "00000000";
  else
    if(erx = '0') then inRS232 <= regrx;
    else inRS232 <= "00000000";
    end if;
  end if;
end process;

end arq_rxRS232;

```

B3. Modulo pila de datos (pila.vhd)

```

-- Diseñador: José de Jesús Morales Romero
-- VLSI-SEES-CINVESTAV-IPN
-- Pila de datos
-- Descripción: Pila que almacena los parámetros de la CNN
-- Puertos
-- clk:          Señal de reloj 100MHz
-- rst:          Reset general
-- rstInt:       Reset interno
-- inRS232:      Entrada de datos
-- pixel:        Pixel de la imagen de entrada
-- Banderas
-- ce_rx: '0' No está recibiendo , '1' Recibiendo datos
-- en_parametros: '0' pila llenándose, '1' pila llena
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity pila is
  port(clk, rst, rstInt: in std_logic;
        inRS232: in std_logic_vector(7 downto 0);
        ce_rx: in std_logic := '0';
        en_parametros: out std_logic := '0';
        xMax, yMax: out std_logic_vector(7 downto 0);
        A11, A12, A13, A21, A22, A23, A31, A32, A33: out std_logic_vector(17 downto 0);
        B11, B12, B13, B21, B22, B23, B31, B32, B33: out std_logic_vector(17 downto 0);
        I: out std_logic_vector(35 downto 0);
        pixel: out std_logic_vector(7 downto 0));
end pila;
architecture arq_pila of pila is

  signal reg01, reg02, reg03, reg04, reg05, reg06: std_logic_vector(7 downto 0) := (others => '0');
  signal reg07, reg08, reg09, reg10, reg11, reg12: std_logic_vector(7 downto 0) := (others => '0');
  signal reg13, reg14, reg15, reg16, reg17, reg18: std_logic_vector(7 downto 0) := (others => '0');
  signal reg19, reg20, reg21, reg22, reg23, reg24: std_logic_vector(7 downto 0) := (others => '0');
  signal reg25, reg26, reg27, reg28, reg29, reg30: std_logic_vector(7 downto 0) := (others => '0');
  signal reg31, reg32, reg33, reg34, reg35, reg36: std_logic_vector(7 downto 0) := (others => '0');
  signal reg37, reg38, reg39, reg40, reg41, reg42: std_logic_vector(7 downto 0) := (others => '0');
  signal reg43, reg44: std_logic_vector(7 downto 0) := (others => '0');

  signal contReg: std_logic_vector(5 downto 0) := (others => '0');
  signal tem_ce_rx: std_logic;

begin

  process(clk, rst, rstInt, ce_rx) begin
    if(rst = '1' or rstInt = '1') then tem_ce_rx <= '0';
    else
      if(clk'event and clk = '1') then tem_ce_rx <= ce_rx; end if;
    end if;
  end process;

  process(rst, rstInt, tem_ce_rx, contReg) begin
    if(rst = '1' or rstInt = '1') then contReg <= (others => '0');
    else
      if(tem_ce_rx'event and tem_ce_rx = '0') then

```

```

        if(contReg <= "101010")then contReg <= contReg + 1;
        else contReg <= "101011";
        end if;
    end if;
end if;
end process;

process(contReg, tem_ce_rx, inRS232) begin
    if(tem_ce_rx'event and tem_ce_rx = '0')then
        case contReg is
            when "000000" => reg01 <= inRS232; when "000001" => reg02 <= inRS232;
            when "000010" => reg03 <= inRS232; when "000011" => reg04 <= inRS232;
            when "000100" => reg05 <= inRS232; when "000101" => reg06 <= inRS232;
            when "000110" => reg07 <= inRS232; when "000111" => reg08 <= inRS232;
            when "001000" => reg09 <= inRS232; when "001001" => reg10 <= inRS232;
            when "001010" => reg11 <= inRS232; when "001011" => reg12 <= inRS232;
            when "001100" => reg13 <= inRS232; when "001101" => reg14 <= inRS232;
            when "001110" => reg15 <= inRS232; when "001111" => reg16 <= inRS232;
            when "010000" => reg17 <= inRS232; when "010001" => reg18 <= inRS232;
            when "010010" => reg19 <= inRS232; when "010011" => reg20 <= inRS232;
            when "010100" => reg21 <= inRS232; when "010101" => reg22 <= inRS232;
            when "010110" => reg23 <= inRS232; when "010111" => reg24 <= inRS232;
            when "011000" => reg25 <= inRS232; when "011001" => reg26 <= inRS232;
            when "011010" => reg27 <= inRS232; when "011011" => reg28 <= inRS232;
            when "011100" => reg29 <= inRS232; when "011101" => reg30 <= inRS232;
            when "011110" => reg31 <= inRS232; when "011111" => reg32 <= inRS232;
            when "100000" => reg33 <= inRS232; when "100001" => reg34 <= inRS232;
            when "100010" => reg35 <= inRS232; when "100011" => reg36 <= inRS232;
            when "100100" => reg37 <= inRS232; when "100101" => reg38 <= inRS232;
            when "100110" => reg39 <= inRS232; when "100111" => reg40 <= inRS232;
            when "101000" => reg41 <= inRS232; when "101001" => reg42 <= inRS232;
            when "101010" => reg43 <= inRS232; when others => reg44 <= inRS232;
        end case;
    end if;
end process;

process(clk, rst, rstInt, contReg) begin
    if(rst = '1' or rstInt = '1')then en_parametros <= '0';
    elsif(clk'event and clk = '1')then
        if(contReg <= "101010")then en_parametros <= '0';
        else en_parametros <= '1';
        end if;
    end if;
end process;

xMax <= reg01; yMax <= reg02;
A11 <= reg03 & reg04 & "00"; A12 <= reg05 & reg06 & "00"; A13 <= reg07 & reg08 & "00";
A21 <= reg09 & reg10 & "00"; A22 <= reg11 & reg12 & "00"; A23 <= reg13 & reg14 & "00";
A31 <= reg15 & reg16 & "00"; A32 <= reg17 & reg18 & "00"; A33 <= reg19 & reg20 & "00";
B11 <= reg21 & reg22 & "00"; B12 <= reg23 & reg24 & "00"; B13 <= reg25 & reg26 & "00";
B21 <= reg27 & reg28 & "00"; B22 <= reg29 & reg30 & "00"; B23 <= reg31 & reg32 & "00";
B31 <= reg33 & reg34 & "00"; B32 <= reg35 & reg36 & "00"; B33 <= reg37 & reg38 & "00";
I <= reg39 & reg40 & reg41 & reg42 & reg43(7 downto 4); pixel <= reg44;

end arq_pila;

```

B.4 Modulo de Memoria RAM (moduloRAM.vhd)

```

-- Diseñador: José de Jesús Morales Romero
-- VLSI-SEES-CINVESTAV-IPN
-- Memoria RAM
-- Descripción: RAM de dos módulos
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity moduloRAM is
    generic(
        widthWord, widthAddr: integer; sizeRAM: integer := 256);
    port(clk, we_in, en_in, we_out, en_out: in std_logic;

```

```

    addr_re_in, addr_we_in: in std_logic_vector(widthAddr-1 downto 0);
    DI_in, DI_out: in std_logic_vector(widthWord-1 downto 0);
    DO_in, DO_out: out std_logic_vector(widthWord-1 downto 0);
    addr_re_out, addr_we_out: in std_logic_vector(widthAddr-1 downto 0));
end moduloRAM;

architecture arq_moduloRAM of moduloRAM is

type ram_t is array (0 to sizeRAM-1) of std_logic_vector(widthWord-1 downto 0);
signal RAM_In, RAM_Out: ram_t;

begin

process(clk) begin
    if(clk'event and clk='1')then
        if(en_in = '1')then
            if(we_in = '1')then RAM_In(conv_integer(addr_we_in)) <= DI_in; end if;
            DO_in <= RAM_In(conv_integer(addr_re_in));
            end if;
        end if;
    end process;

process(clk) begin
    if(clk'event and clk='1')then
        if(en_out = '1')then
            if(we_out = '1')then RAM_Out(conv_integer(addr_we_out)) <= DI_out; end if;
            DO_out <= RAM_Out(conv_integer(addr_re_out));
            end if;
        end if;
    end process;

end arq_moduloRAM;

```

B5. Módulo de control de la memoria RAM (ctrWeRe.vhd)

```

-- Diseñador: José de Jesús Morales Romero
-- VLSI-SEES-CINVESTAV-IPN
-- Modulo de control de RAM
-- Descripción: Modulo de control de lectura y escritura de la memoria RAM
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity ctrWeRe is
    generic(widthWord, widthAddr: integer);
    port(clk, rst, rstInt, ce_rx, en_imagenIn: in std_logic;
        en_imagenGuardada, we_in: out std_logic;
        pixel: in std_logic_vector(7 downto 0);
        addr_we_in: out std_logic_vector(widthAddr-1 downto 0);
        DI_in: out std_logic_vector(widthWord-1 downto 0);
        totalPixeles: in std_logic_vector(widthAddr-1 downto 0)
    );
end ctrWeRe;

architecture arq_ctrWeRe of ctrWeRe is

signal contAddr: std_logic_vector(widthAddr-1 downto 0) := (others => '0');
signal tem_ce_rx: std_logic;

begin

process(clk, rst, rstInt, ce_rx) begin
    if(rst = '1' or rstInt = '1')then tem_ce_rx <= '0';
    else
        if(clk'event and clk = '1')then tem_ce_rx <= ce_rx; end if;
    end if;
end process;

process(en_imagenIn, tem_ce_rx, contAddr) begin

```

```

    if(en_imagenIn = '0')then contAddr <= (others => '0');
    else
        if(tem_ce_rx'event and tem_ce_rx = '0')then contAddr <= contAddr + 1; end if;
    end if;
end process;

process(clk, contAddr, en_imagenIn, totalPixeles) begin
    if(en_imagenIn = '0')then we_in <= '0';
    elsif(clk'event and clk = '1')then
        if(contAddr <= (totalPixeles-1))then we_in <= '1';
        else we_in <= '0';
        end if;
    end if;
end process;

process(clk, contAddr, totalPixeles) begin
    if(contAddr <= totalPixeles)then addr_we_in <= contAddr-1; en_imagenGuardada <= '0';
    else addr_we_in <= (others => '1'); en_imagenGuardada <= '1';
    end if;
end process;

DI_in <= pixel;

end arq_ctrWeRe;

```

B6. Módulo de control del sistema completo de la CNN multiplexada (ctrCNN.vhd)

```

-- Diseñador: José de Jesús Morales Romero
-- VLSI-SEES-CINVESTAV-IPN
-- Módulo de control del sistema completo
-- Descripción: Modulo de control del multiplexado de la CNN
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity ctrCNN is
    generic(widthWord: integer := 8; widthAddr: integer := 8 );
    port(clk, rst, en_parametros, en_imagenGuardada: in std_logic;
        en_leeSubImg_out, en_conv8a18_out, en_Next: in std_logic;
        rstInt, en_imagenIn, en_leeSubImg_in: out std_logic;
        en_conv8a18_in, en_ctrCNN4x4_in, en_conv18a8_in: out std_logic;
        en_ctrCNN4x4_out, en_conv18a8_out: in std_logic;
        en_guardaSalida_in, en_Calcula_in: out std_logic;
        en_guardaSalida_out, en_Calcula_out, en_ctrEnvio_out: in std_logic;
        en_ctrEnvio_in, en_in, en_out: out std_logic;
        salLEDs: out std_logic_vector(7 downto 0));
end ctrCNN;

architecture arq_ctrCNN of ctrCNN is

    type estados is (edo_ReadParam, edo_ReadImgIn, edo_leeSubImg, edo_Conv8a18bits, edo_CNN,
        edo_Conv18a8bits, edo_GuardaSalida, edo_Calcula, edo_EnviaDato, edo_Wait);
    signal edo_pres, edo_fut: estados;

    signal contRst: std_logic_vector(2 downto 0) := (others => '0');
    signal banRst: std_logic := '0';

begin
    proceso_Estados: process(edo_pres, en_parametros, en_imagenGuardada, en_leeSubImg_out,
        en_conv8a18_out, en_conv8a18_out, en_ctrCNN4x4_out, en_conv18a8_out, en_guardaSalida_out,
        en_Calcula_out, en_Next, en_ctrEnvio_out, banRst) begin
        case edo_pres is
            when edo_ReadParam =>
                if(en_parametros = '1')then edo_fut <= edo_ReadImgIn;
                else edo_fut <= edo_ReadParam;
                end if;
            when edo_ReadImgIn =>
                if(en_imagenGuardada = '1')then edo_fut <= edo_leeSubImg;
                else edo_fut <= edo_ReadImgIn;

```

```

        end if;
    when edo_leeSubImg =>
        if(en_leeSubImg_out = '1')then edo_fut <= edo_Conv8a18bits;
        else edo_fut <= edo_leeSubImg;
        end if;
    when edo_Conv8a18bits =>
        if(en_conv8a18_out = '1')then edo_fut <= edo_CNN;
        else edo_fut <= edo_Conv8a18bits;
        end if;
    when edo_CNN =>
        if(en_ctrCNN4x4_out = '1')then edo_fut <= edo_Conv18a8bits;
        else edo_fut <= edo_CNN;
        end if;
    when edo_Conv18a8bits =>
        if(en_conv18a8_out = '1')then edo_fut <= edo_GuardaSalida;
        else edo_fut <= edo_Conv18a8bits;
        end if;
    when edo_GuardaSalida =>
        if(en_guardaSalida_out = '1')then edo_fut <= edo_Calcula;
        else edo_fut <= edo_GuardaSalida;
        end if;
    when edo_Calcula =>
        if(en_Calcula_out = '1')then
            if(en_Next = '1')then edo_fut <= edo_EnviaDato;
            else edo_fut <= edo_leeSubImg;
            end if;
        else edo_fut <= edo_Calcula;
        end if;
    when edo_EnviaDato =>
        if(en_ctrEnvio_out = '1')then edo_fut <= edo_Wait;
        else edo_fut <= edo_EnviaDato;
        end if;
    when edo_Wait =>
        if(banRst = '1')then edo_fut <= edo_ReadParam;
        else edo_fut <= edo_Wait;
        end if;
    end case;
end process;

proceso_Avance: process(clk, rst) begin
    if(rst = '1')then edo_pres <= edo_ReadParam;
    elsif(rising_edge(clk))then edo_pres <= edo_fut;
    end if;
end process;

readImg: process(clk, rst, edo_pres) begin
    if(edo_pres = edo_ReadImgIn)then en_imagenIn <= '1';
    else en_imagenIn <= '0';
    end if;
end process;

activaRAMIn: process(clk) begin
    if(rising_edge(clk))then
        if(edo_pres = edo_ReadParam)then en_in <= '1';
        elsif(edo_pres = edo_ReadImgIn)then en_in <= '1';
        elsif(edo_pres = edo_leeSubImg)then en_in <= '1';
        else en_in <= '0';
        end if;
    end if;
end process;

activaLeeSubImg: process(clk) begin
    if(rising_edge(clk))then
        if(edo_pres = edo_leeSubImg)then en_leeSubImg_in <= '1';
        else en_leeSubImg_in <= '0';
        end if;
    end if;
end process;

activaCon8a18bits: process(clk) begin
    if(rising_edge(clk))then

```

```

        if(edo_pres = edo_Conv8a18bits) then en_conv8a18_in <= '1';
        else en_conv8a18_in <= '0';
        end if;
    end if;
end process;

activa_CNN: process(clk) begin
    if(rising_edge(clk)) then
        if(edo_pres = edo_CNN) then en_ctrCNN4x4_in <= '1';
        elsif(edo_pres = edo_Conv18a8bits) then en_ctrCNN4x4_in <= '1';
        else en_ctrCNN4x4_in <= '0';
        end if;
    end if;
end process;

activaConv18a8bits: process(clk) begin
    if(rising_edge(clk)) then
        if(edo_pres = edo_Conv18a8bits) then en_conv18a8_in <= '1';
        else en_conv18a8_in <= '0';
        end if;
    end if;
end process;

activaRAMOut: process(clk, edo_pres) begin
    if(edo_pres = edo_GuardaSalida) then en_out <= '1';
    elsif(edo_pres = edo_EnviaDato) then en_out <= '1';
    elsif(edo_pres = edo_Wait) then en_out <= '1';
    else en_out <= '0';
    end if;
end process;

activaGuardaSalida: process(clk) begin
    if(rising_edge(clk)) then
        if(edo_pres = edo_GuardaSalida) then en_guardaSalida_in <= '1';
        else en_guardaSalida_in <= '0';
        end if;
    end if;
end process;

activaCalc: process(clk) begin
    if(rising_edge(clk)) then
        if(edo_pres = edo_Calcula) then en_Calcula_in <= '1';
        else en_Calcula_in <= '0';
        end if;
    end if;
end process;

activaEnvio: process(clk) begin
    if(rising_edge(clk)) then
        if(edo_pres = edo_EnviaDato) then en_ctrEnvio_in <= '1';
        else en_ctrEnvio_in <= '0';
        end if;
    end if;
end process;

reinicia: process(clk) begin
    if(rising_edge(clk)) then
        if(edo_pres = edo_Wait) then contrSt <= contrSt + 1;
        if(contrSt <= "010") then rstInt <= '0'; banRst <= '0';
        elsif(contrSt >= "011" and contrSt <= "101") then rstInt <= '1'; banRst <= '0';
        else rstInt <= '0'; banRst <= '1';
        end if;
        else rstInt <= '0'; banRst <= '0';
        end if;
    end if;
end process;

process(clk) begin
    if(rising_edge(clk)) then
        if(edo_pres = edo_ReadParam) then salLEDs <= "00000011";
        elsif(edo_pres = edo_ReadImgIn) then salLEDs <= "00000110";
    end if;
end process;

```



```

    elsif(edo_pres = edo_leeSubImg) then salLEds <= "00001100";
    elsif(edo_pres = edo_Conv8a18bits) then salLEds <= "00011000";
    elsif(edo_pres = edo_CNN) then salLEds <= "00110000";
    elsif(edo_pres = edo_Conv18a8bits) then salLEds <= "01100000";
    elsif(edo_pres = edo_GuardaSalida) then salLEds <= "11000000";
    elsif(edo_pres = edo_Calcula) then salLEds <= "10000001";
    elsif(edo_pres = edo_EnviaDato) then salLEds <= "10000011";
    else salLEds <= "11111111";
    end if;
end if;
end process;

end arq_ctrCNN;

```

B7. Módulo de lectura de los bloques de la imagen de entrada (leeSubImg.vhd)

```

-- Diseñador: José de Jesús Morales Romero
-- VLSI-SEES-CINVESTAV-IPN
-- Modulo de Lectura de bloques
-- Descripción: Modulo de control de la lectura de los bloques de la imagen de entrada.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity leeSubImg is
    generic(widthWord: integer := 8; widthAddr: integer := 8);
    port(clk_RAM, en_leeSubImg_in: in std_logic;
        en_leeSubImg_out: out std_logic;
        xMax, Xini: in std_logic_vector(widthWord-1 downto 0);
        addr_re_in: out std_logic_vector(widthAddr-1 downto 0);
        reg01, reg02, reg03, reg04, reg05, reg06: out std_logic_vector(widthWord-1 downto
0);
        reg07, reg08, reg09, reg10, reg11, reg12: out std_logic_vector(widthWord-1 downto
0);
        reg13, reg14, reg15, reg16: out std_logic_vector(widthWord-1 downto 0);
        DO_in: in std_logic_vector(widthWord-1 downto 0));
end leeSubImg;

architecture arq_leeSubImg of leeSubImg is

    signal Xi: std_logic_vector(4 downto 0) := (others => '0');

    signal tem01,tem02,tem03,tem04,tem05,tem06,tem07: std_logic_vector(widthWord-1 downto 0);
    signal tem08, tem09,tem10,tem11,tem12,tem13,tem14: std_logic_vector(widthWord-1 downto 0);
    signal tem15,tem16: std_logic_vector(widthWord-1 downto 0);

begin

proce_LecRAM: process(clk_RAM, en_leeSubImg_in) begin
    if(rising_edge(clk_RAM)) then
        if(en_leeSubImg_in = '1') then
            addr_re_in <= Xini+((xMax - 1)*Xi(3 downto 2)) + Xi(1 downto 0) + Xi(3 downto 2);
            Xi <= Xi + 1;
            if(Xi <= "01111") then en_leeSubImg_out <= '0';
            else en_leeSubImg_out <= '1';
            end if;
        else en_leeSubImg_out <= '0'; Xi <= (others => '0');
        end if;
    end if;
end process;

proc_reg_in: process(clk_RAM, en_leeSubImg_in) begin
    if(falling_edge(clk_RAM)) then
        if(en_leeSubImg_in = '1') then
            case Xi is
                when "00000" => tem01 <= DO_in; when "00001" => tem01 <= DO_in;
                when "00010" => tem02 <= DO_in; when "00011" => tem03 <= DO_in;
                when "00100" => tem04 <= DO_in; when "00101" => tem05 <= DO_in;
                when "00110" => tem06 <= DO_in; when "00111" => tem07 <= DO_in;
            end case;
        end if;
    end if;
end process;

```

```

        when "01000" => tem08 <= DO_in; when "01001" => tem09 <= DO_in;
        when "01010" => tem10 <= DO_in; when "01011" => tem11 <= DO_in;
        when "01100" => tem12 <= DO_in; when "01101" => tem13 <= DO_in;
        when "01110" => tem14 <= DO_in; when "01111" => tem15 <= DO_in;
        when "10000" => tem16 <= DO_in; when others => null;
    end case;
else
    tem01 <= tem01; tem02 <= tem02; tem03 <= tem03; tem04 <= tem04; tem05 <= tem05;
    tem06 <= tem06; tem07 <= tem07; tem08 <= tem08; tem09 <= tem09; tem10 <= tem10;
    tem11 <= tem11; tem12 <= tem12; tem13 <= tem13; tem14 <= tem14; tem15 <= tem15;
    tem16 <= tem16;
end if;
end if;
end process;

reg01 <= tem01; reg02 <= tem02; reg03 <= tem03; reg04 <= tem04; reg05 <= tem05;
reg06 <= tem06; reg07 <= tem07; reg08 <= tem08; reg09 <= tem09; reg10 <= tem10;
reg11 <= tem11; reg12 <= tem12; reg13 <= tem13; reg14 <= tem14; reg15 <= tem15;
reg16 <= tem16;
end arq_leeSubImg;

```

B8. Módulo convertidor de 8 a 18 bits (conv8a18bits.vhd)

```

-- Diseñador: José de Jesús Morales Romero
-- VLSI-SEES-CINVESTAV-IPN
-- Módulo: conv8a18bits
-- Descripción: Convertidor de 8 a 18 bits
-- 1 de signo, 6 de entero y 11 de fracción
-- Se utiliza la siguiente formula:  $Y = 1 - ((2*X)/255)$ 
-- Banderas
-- en_conv18a8_in: '1' Habilita conversión de 8 a 18 bits, '0' No habilita la conversión
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity conv8a18bits is
    port(clk, rst, rstInt, en_conv8a18_in: in std_logic;
          en_conv8a18_out: out std_logic;
          reg01, reg02, reg03, reg04, reg05, reg06, reg07: in std_logic_vector(7 downto 0);
          reg08, reg09, reg10, reg11, reg12, reg13, reg14: in std_logic_vector(7 downto 0);
          reg15, reg16: in std_logic_vector(7 downto 0);
          inN11, inN12, inN13, inN14, inN21, inN22, inN23: out std_logic_vector(17 downto
0);
          inN24, inN31, inN32, inN33, inN34, inN41, inN42: out std_logic_vector(17 downto
0);
          inN43, inN44: out std_logic_vector(17 downto 0));
end conv8a18bits;

architecture arq_conv8a18bits of conv8a18bits is

    signal temIn11, temIn12, temIn13, temIn14, temIn21, temIn22: std_logic_vector(17 downto 0);
    signal temIn23, temIn24, temIn31, temIn32, temIn33, temIn34: std_logic_vector(17 downto 0);
    signal temIn41, temIn42, temIn43, temIn44: std_logic_vector(17 downto 0);
    signal inTem: std_logic_vector(7 downto 0) := (others => '0');
    signal conv: std_logic_vector(17 downto 0) := (others => '0');
    signal multTem: std_logic_vector(35 downto 0) := (others => '0');
    signal contIn: std_logic_vector(5 downto 0) := (others => '0');

begin

process(clk, rst, rstInt, en_conv8a18_in) begin
    if(rst = '1' or rstInt = '1')then contIn <= (others => '0'); en_conv8a18_out <= '0';
    else
        if(clk'event and clk = '1')then
            if(en_conv8a18_in = '1')then
                if(contIn < "010001")then contIn <= contIn + 1; en_conv8a18_out <= '0';
                else contIn <= "010001"; en_conv8a18_out <= '1';
                end if;
            else contIn <= (others => '0'); en_conv8a18_out <= '0';
            end if;
        end if;
    end if;
end process;

```

```

    end if;
  end if;
end process;

process(clk, rst, rstInt, en_conv8a18_in, contIn) begin
  if(rst = '1' or rstInt = '1')then inTem <= (others => '0');
  else
    if(clk'event and clk = '0')then
      if(en_conv8a18_in = '1')then
        case contIn is
          when "000000" => inTem <= reg01; when "000001" => inTem <= reg02;
          when "000010" => inTem <= reg03; when "000011" => inTem <= reg04;
          when "000100" => inTem <= reg05; when "000101" => inTem <= reg06;
          when "000110" => inTem <= reg07; when "000111" => inTem <= reg08;
          when "001000" => inTem <= reg09; when "001001" => inTem <= reg10;
          when "001010" => inTem <= reg11; when "001011" => inTem <= reg12;
          when "001100" => inTem <= reg13; when "001101" => inTem <= reg14;
          when "001110" => inTem <= reg15; when "001111" => inTem <= reg16;
          when others => inTem <= (others => '0');
        end case;
      else inTem <= (others => '0');
      end if;
    end if;
  end if;
end process;

process(clk, rst, rstInt, en_conv8a18_in) begin
  if(rst = '1' or rstInt = '1')then mulTem <= (others => '0');
  else
    if(clk'event and clk = '1')then
      if(en_conv8a18_in = '1')then mulTem <= ("0000000000"&inTem)*"000000010000000100";
      else mulTem <= (others => '0');
      end if;
    end if;
  end if;
end process;

process(clk, rst, rstInt, en_conv8a18_in) begin
  if(rst = '1' or rstInt = '1')then conv <= (others => '0');
  else
    if(clk'event and clk = '0')then
      if(en_conv8a18_in = '1')then
        if(inTem = "00000000")then conv <= "00000001000000000000";
        elsif(inTem = "11111111")then conv <= "11111111000000000000";
        else conv <= "00000001000000000000" - mulTem(23 downto 6);
        end if;
      else conv <= (others => '0');
      end if;
    end if;
  end if;
end process;

process(clk, rst, rstInt, contIn, en_conv8a18_in) begin
  if(rst = '1' or rstInt = '1')then
    temIn11 <= (others => '0'); temIn12 <= (others => '0'); temIn13 <= (others => '0');
    temIn14 <= (others => '0'); temIn21 <= (others => '0'); temIn22 <= (others => '0');
    temIn23 <= (others => '0'); temIn24 <= (others => '0'); temIn31 <= (others => '0');
    temIn32 <= (others => '0'); temIn33 <= (others => '0'); temIn34 <= (others => '0');
    temIn41 <= (others => '0'); temIn42 <= (others => '0'); temIn43 <= (others => '0');
    temIn44 <= (others => '0');
  else
    if(clk'event and clk = '1')then
      if(en_conv8a18_in = '1')then
        case contIn is
          when "000001" => temIn11 <= conv; when "000010" => temIn12 <= conv;
          when "000011" => temIn13 <= conv; when "000100" => temIn14 <= conv;
          when "000101" => temIn21 <= conv; when "000110" => temIn22 <= conv;
          when "000111" => temIn23 <= conv; when "001000" => temIn24 <= conv;
          when "001001" => temIn31 <= conv; when "001010" => temIn32 <= conv;
          when "001011" => temIn33 <= conv; when "001100" => temIn34 <= conv;
          when "001101" => temIn41 <= conv; when "001110" => temIn42 <= conv;
        end case;
      end if;
    end if;
  end if;
end process;

```

```

        when "001111" => temIn43 <= conv; when "010000" => temIn44 <= conv;
        when others => null;
    end case;
    else
        temIn11 <= temIn11; temIn12 <= temIn12; temIn13 <= temIn13; temIn14 <= temIn14;
        temIn21 <= temIn21; temIn22 <= temIn22; temIn23 <= temIn23; temIn24 <= temIn24;
        temIn31 <= temIn31; temIn32 <= temIn32; temIn33 <= temIn33; temIn34 <= temIn34;
        temIn41 <= temIn41; temIn42 <= temIn42; temIn43 <= temIn43; temIn44 <= temIn44;
    end if;
end if;
end if;
end process;

inN11 <= temIn11; inN12 <= temIn12; inN13 <= temIn13; inN14 <= temIn14; inN21 <= temIn21;
inN22 <= temIn22; inN23 <= temIn23; inN24 <= temIn24; inN31 <= temIn31; inN32 <= temIn32;
inN33 <= temIn33; inN34 <= temIn34; inN41 <= temIn41; inN42 <= temIn42; inN43 <= temIn43;
inN44 <= temIn44;

end arq_conv8a18bits;

```

B9. Módulo de control de la CNN de 4x4 neuronas (ctrCNN4x4.vhd)

```

-- Diseñador: José de Jesús Morales Romero
-- VLSI-SEES-CINVESTAV-IPN
-- ctrCNN4x4: Controlador de la CNN de 4x4 neuronas
-- Descripción: Controla el procesamiento de la CNN de 4x4 neuronas
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity ctrCNN4x4 is
    port(clk, rst, rstInt, en_ctrCNN4x4_in: in std_logic;
        ctrNeurona: out std_logic_vector(5 downto 0);
        banN11, banN12, banN13, banN14, banN21, banN22, banN23, banN24: in std_logic;
        banN31, banN32, banN33, banN34, banN41, banN42, banN43, banN44: in std_logic;
        en_ctrCNN4x4_out: out std_logic);
end ctrCNN4x4;

architecture arq_ctrCNN4x4 of ctrCNN4x4 is

    signal tempBan: std_logic := '0';
    signal tem_ctrNeurona: std_logic_vector(5 downto 0) := (others => '0');

begin

    process(clk, rst, rstInt, en_ctrCNN4x4_in) begin
        if(rst = '1' or rstInt = '1') then tem_ctrNeurona <= (others => '0');
        else
            if(en_ctrCNN4x4_in = '0') then tem_ctrNeurona <= (others => '0');
            else
                if(clk'event and clk = '1') then
                    if(tem_ctrNeurona < "011001") then tem_ctrNeurona <= tem_ctrNeurona + 1;
                    else tem_ctrNeurona <= (others => '0');
                    end if;
                end if;
            end if;
        end if;
    end process;

    ctrNeurona <= tem_ctrNeurona;
    tempBan <= banN11 and banN12 and banN13 and banN14 and banN21 and banN22 and banN23 and
        banN24 and banN31 and banN32 and banN33 and banN34 and banN41 and banN42 and
        banN43 and banN44;

    process(clk, rst, rstInt, en_ctrCNN4x4_in) begin
        if(rst = '1' or rstInt = '1') then en_ctrCNN4x4_out <= '0';
        else
            if(en_ctrCNN4x4_in = '1') then
                if(clk'event and clk = '1') then

```

```

        if(tempBan = '1')then en_ctrCNN4x4_out <= '1';
        else en_ctrCNN4x4_out <= '0';
        end if;
    end if;
    else en_ctrCNN4x4_out <= '0';
    end if;
end if;
end process;

end arq_ctrCNN4x4;

```

B10. Módulo de la Neurona (Neurona.vhd)

```

-- Diseñador: José de Jesús Morales Romero
-- VLSI-SEES-CINVESTAV-IPN
-- Módulo de Neurona
-- Descripción: Neurona que recibe los templates A y B, además de los valores de entrada y
-- salida.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity Neurona is
    port(clk, en_ctrCNN4x4_in: in std_logic;
        banN: out std_logic;
        vy22: out std_logic_vector(17 downto 0);
        A11, A12, A13, A21, A22, A23, A31, A32, A33: in std_logic_vector(17 downto 0);
        B11, B12, B13, B21, B22, B23, B31, B32, B33: in std_logic_vector(17 downto 0);
        vu11, vu12, vu13, vu21, vu22, vu23, vu31, vu32: in std_logic_vector(17 downto 0);
        vu33, vy11, vy12, vy13, vy21, vy23, vy31, vy32: in std_logic_vector(17 downto 0);
        vy33, dt: in std_logic_vector(17 downto 0);
        I: in std_logic_vector(35 downto 0);
        ctrNeurona: in std_logic_vector(5 downto 0));
end Neurona;

architecture arq_Neurona of Neurona is

    signal X: std_logic_vector(17 downto 0) := (others => '0');
    signal Y: std_logic_vector(17 downto 0) := (others => '0');
    signal XY: std_logic_vector(35 downto 0) := (others => '0');

    signal sumA: std_logic_vector(35 downto 0) := (others => '0');
    signal sumB: std_logic_vector(35 downto 0) := (others => '0');
    signal sumTotal: std_logic_vector(35 downto 0) := (others => '0');

    signal preEdo: std_logic_vector(17 downto 0) := (others => '0');
    signal mulPreEdo: std_logic_vector(35 downto 0) := (others => '0');
    signal edo: std_logic_vector(35 downto 0) := (others => '0');
    signal edoSat: std_logic_vector(17 downto 0) := (others => '0');
    signal outN: std_logic_vector(17 downto 0) := (others => '0');

begin

process(clk, ctrNeurona, X, Y, en_ctrCNN4x4_in) begin
    if(clk'event and clk = '1')then
        if(en_ctrCNN4x4_in = '1')then
            case ctrNeurona is
                when "000000" => X <= A11; Y <= vy11; when "000001" => X <= A12; Y <= vy12;
                when "000010" => X <= A13; Y <= vy13; when "000011" => X <= A21; Y <= vy21;
                when "000100" => X <= A22; Y <= outN; when "000101" => X <= A23; Y <= vy23;
                when "000110" => X <= A31; Y <= vy31; when "000111" => X <= A32; Y <= vy32;
                when "001000" => X <= A33; Y <= vy33; when "001001" => X <= B11; Y <= vu11;
                when "001010" => X <= B12; Y <= vu12; when "001011" => X <= B13; Y <= vu13;
                when "001100" => X <= B21; Y <= vu21; when "001101" => X <= B22; Y <= vu22;
                when "001110" => X <= B23; Y <= vu23; when "001111" => X <= B31; Y <= vu31;
                when "010000" => X <= B32; Y <= vu32; when "010001" => X <= B33; Y <= vu33;
                when others => X <= (others => '0'); Y <= (others => '0');
            end case;
        end if;
    end if;
end process;

```

```

    end if;
end process;

process(clk, en_ctrCNN4x4_in) begin
    if(clk'event and clk = '1')then
        if(en_ctrCNN4x4_in = '1')then XY <= X * Y;
        else XY <= (others => '0');
        end if;
    end if;
end process;

process(clk, ctrNeurona, XY, en_ctrCNN4x4_in) begin
    if(clk'event and clk='0')then
        if(en_ctrCNN4x4_in = '1')then
            if(ctrNeurona < "010011")then sumA <= sumA + XY; sumTotal <= sumTotal;
            elsif(ctrNeurona < "010100")then sumTotal <= sumA + I - edo;
            else sumA <= (others => '0'); sumTotal <= sumTotal;
            end if;
        else sumA <= (others => '0'); sumTotal <= (others => '0');
        end if;
    end if;
end process;

process(clk, sumTotal, preEdo, ctrNeurona, edo, en_ctrCNN4x4_in, vu22)
begin
    if(clk'event and clk = '1')then
        if(en_ctrCNN4x4_in = '1')then
            if(ctrNeurona < "010100")then
                if(sumTotal(10) = '1')then
                    if(sumTotal(35) = '0')then preEdo <= (sumTotal(35)&sumTotal(27 downto 11)) + 1;
                    else preEdo <= (sumTotal(35)&sumTotal(27 downto 11)) - 1;
                    end if;
                else preEdo <= sumTotal(35) & sumTotal(27 downto 11);
                end if;
            elsif(ctrNeurona < "010101")then mulPreEdo <= preEdo * dt;
            elsif(ctrNeurona < "010110")then edo <= edo + mulPreEdo;
            else mulPreEdo <= mulPreEdo; edo <= edo;
            end if;
        else preEdo <= vu22;
        if(vu22(17) = '1')then edo <= "1111111" & vu22 & "000000000000";
        else edo <= "0000000" & vu22 & "000000000000";
        end if;
    end if;
end process;

process(ctrNeurona, clk, edo, en_ctrCNN4x4_in, vu22) begin
    if(clk'event and clk = '1') then
        if(en_ctrCNN4x4_in = '1')then
            if(ctrNeurona < "010111")then
                if(edo(10) = '1')then edoSat <= (edo(35) & edo(27 downto 11)) + 1;
                else edoSat <= edo(35) & edo(27 downto 11);
                end if;
            elsif(ctrNeurona < "011000")then
                if(edoSat < "000000100000000000") then
                    if(edoSat > "111111100000000000") then outN <= edoSat; banN <= '0';
                    else outN <= "111111100000000000"; banN <= '1';
                    end if;
                else outN <= "000000100000000000"; banN <= '1';
                end if;
            end if;
        else banN <= '0'; edoSat <= vu22; outN <= vu22;
        end if;
    end if;
end process;

vy22 <= outN;

end arq_Neurona;

```

B11. Módulo convertidor de 18 a 8 bits (conv18a8bits.vhd)

```

-- Diseñador: José de Jesús Morales Romero
-- VLSI-SEES-CINVESTAV-IPN
-- Modulo de convertidor de 18 a 8 bits
-- Descripción: Convertidor de 18 a 8 bits además de los valores de entrada y salida.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity conv18a8bits is
    port(clk, rst, rstInt, en_conv18a8_in: in std_logic;
          en_conv18a8_out: out std_logic;
          outN11, outN12, outN13, outN14, outN21, outN22: in std_logic_vector(17 downto 0);
          outN23, outN24, outN31, outN32, outN33, outN34: in std_logic_vector(17 downto 0);
          outN41, outN42, outN43, outN44: in std_logic_vector(17 downto 0);
          sal11, sal12, sal13, sal14, sal21, sal22, sal23: out std_logic_vector(7 downto
0);
          sal24, sal31, sal32, sal33, sal34, sal41, sal42: out std_logic_vector(7 downto
0);
          sal43, sal44: out std_logic_vector(7 downto 0));
end conv18a8bits;

architecture arq_conv18a8bits of conv18a8bits is

    signal out11, out12, out13, out14, out21: std_logic_vector(7 downto 0) := (others => '0');
    signal out22, out23, out24, out31, out31: std_logic_vector(7 downto 0) := (others => '0');
    signal out33, out34, out41, out42, out43: std_logic_vector(7 downto 0) := (others => '0');
    signal out44, outX: std_logic_vector(7 downto 0) := (others => '0');
    signal cont: std_logic_vector(4 downto 0);
    signal outNX: std_logic_vector(17 downto 0);

begin

process(clk, rst, rstInt) begin
    if(rst = '1' or rstInt = '1')then cont <= (others => '0'); en_conv18a8_out <= '0';
    else
        if(clk'event and clk = '1')then
            if(en_conv18a8_in = '1')then
                if(cont <= "10001")then cont <= cont + 1; en_conv18a8_out <= '0';
                else cont <= "10010"; en_conv18a8_out <= '1';
                end if;
            else cont <= (others => '0'); en_conv18a8_out <= '0';
            end if;
        end if;
    end if;
end process;

process(clk, rst, rstInt, cont, en_conv18a8_in) begin
    if(rst = '1' or rstInt = '1')then outNX <= (others => '0');
    else
        if(clk'event and clk = '1')then
            if(en_conv18a8_in = '1')then
                case cont is
                    when "00000" => outNX <= outN11; when "00001" => outNX <= outN12;
                    when "00010" => outNX <= outN13; when "00011" => outNX <= outN14;
                    when "00100" => outNX <= outN21; when "00101" => outNX <= outN22;
                    when "00110" => outNX <= outN23; when "00111" => outNX <= outN24;
                    when "01000" => outNX <= outN31; when "01001" => outNX <= outN32;
                    when "01010" => outNX <= outN33; when "01011" => outNX <= outN34;
                    when "01100" => outNX <= outN41; when "01101" => outNX <= outN42;
                    when "01110" => outNX <= outN43; when "01111" => outNX <= outN44;
                    when others => outNX <= outNX;
                end case;
            else outNX <= (others => '0');
            end if;
        end if;
    end if;
end process;

process(clk, rst, rstInt, outNX) begin
    if(rst = '1' or rstInt = '1')then outX <= "00000000";

```

```

else
  if(clk'event and clk = '0')then
    if(en_conv18a8_in = '1')then
      if(outNX = "000000100000000000")then outX <= X"30";
      elsif(outNX = "111111100000000000")then outX <= X"31";
      else outX <= "10101010";
      end if;
    else outX <= "01010101";
    end if;
  end if;
end if;
end process;

process(clk, cont, en_conv18a8_in) begin
  if(clk'event and clk = '1')then
    if(en_conv18a8_in = '1')then
      case cont is
        when "00001" => out11 <= outX; when "00010" => out12 <= outX;
        when "00011" => out13 <= outX; when "00100" => out14 <= outX;
        when "00101" => out21 <= outX; when "00110" => out22 <= outX;
        when "00111" => out23 <= outX; when "01000" => out24 <= outX;
        when "01001" => out31 <= outX; when "01010" => out32 <= outX;
        when "01011" => out33 <= outX; when "01100" => out34 <= outX;
        when "01101" => out41 <= outX; when "01110" => out42 <= outX;
        when "01111" => out43 <= outX; when others => out44 <= outX;
      end case;
    else
      out11 <= out11; out12 <= out12; out13 <= out13; out14 <= out14;
      out21 <= out21; out22 <= out22; out23 <= out23; out24 <= out24;
      out31 <= out31; out32 <= out32; out33 <= out33; out34 <= out34;
      out41 <= out41; out42 <= out42; out43 <= out43; out44 <= out44;
    end if;
  end if;
end process;

sal11 <= out11; sal12 <= out12; sal13 <= out13; sal14 <= out14; sal21 <= out21;
sal22 <= out22; sal23 <= out23; sal24 <= out24; sal31 <= out31; sal32 <= out32;
sal33 <= out33; sal34 <= out34; sal41 <= out41; sal42 <= out42; sal43 <= out43;
sal44 <= out44;

end arq_conv18a8bits;

```

B12. Módulo para guardar la salida (guardaSalida.vhd)

```

-- Diseñador: José de Jesús Morales Romero
-- VLSI-SEES-CINVESTAV-IPN
-- Modulo de Guardar salida
-- Descripción: Realiza el guardado de los datos procesados en memoria RAM
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity guardaSalida is
  generic(widthWord: integer; widthAddr: integer);
  port(clk_RAM, en_guardaSalida_in: in std_logic;
    en_guardaSalida_out, we_out: out std_logic;
    Xini: in std_logic_vector(widthAddr-1 downto 0);
    xMax: in std_logic_vector(widthWord-1 downto 0);
    fila, columna, maxFila, maxColumna: in std_logic_vector(widthWord-2 downto 0);
    addr_we_out: out std_logic_vector(widthAddr-1 downto 0);
    DI_out: out std_logic_vector(widthWord-1 downto 0);
    sal11, sal12, sal13, sal14: in std_logic_vector(widthWord-1 downto 0);
    sal21, sal22, sal23, sal24: in std_logic_vector(widthWord-1 downto 0);
    sal31, sal32, sal33, sal34: in std_logic_vector(widthWord-1 downto 0);
    sal41, sal42, sal43, sal44: in std_logic_vector(widthWord-1 downto 0));
end guardaSalida;

architecture arq_guardaSalida of guardaSalida is

```



```

signal Xo: std_logic_vector(4 downto 0);

begin

weRAM_Salida: process(clk_RAM, Xo) begin
  if(rising_edge(clk_RAM)) then
    if(en_guardaSalida_in = '1') then
      addr_we_out <= Xini + ((xMax - 1) * Xo(3 downto 2)) + Xo(1 downto 0) + Xo(3 downto 2);
      Xo <= Xo + 1;
      if(Xo <= "01111") then en_guardaSalida_out <= '0';
      else en_guardaSalida_out <= '1';
      end if;
    else en_guardaSalida_out <= '0'; Xo <= (others => '0');
    end if;
  end if;
end process;

process_we_out: process(Xo, maxFila, maxColumna, columna, fila, en_guardaSalida_in) begin
  if(en_guardaSalida_in = '1') then
    if(maxFila = "0000001" and maxColumna = "0000001") then we_out <= '1';
    elsif(maxFila = "0000001" and maxColumna > "0000001") then
      if(columna <= "0000001") then
        if(Xo = "00100" or Xo = "01000" or Xo = "01100" or Xo > "10000") then we_out <= '0';
        else we_out <= '1';
        end if;
      elsif(columna >= maxColumna) then
        if((Xo >= "00000" and Xo <= "00001") or Xo = "00101" or
        Xo = "01101" or Xo > "10000") then we_out <= '0';
        else we_out <= '1';
        end if;
      else
        if(Xo = "00010" or Xo = "00011" or Xo = "00110" or Xo = "00111" or Xo = "01010" or
        Xo = "01011" or Xo = "01110" or Xo = "01111") then we_out <= '1';
        else we_out <= '0';
        end if;
      end if;
    elsif(maxFila > "0000001" and maxColumna <= "0000001") then
      if(fila <= "0000001") then
        if(Xo > "01100") then we_out <= '0';
        else we_out <= '1';
        end if;
      elsif(fila >= maxFila) then
        if(Xo <= "00100" or Xo > "10000") then we_out <= '0';
        else we_out <= '1';
        end if;
      else
        if(Xo <= "00100" or Xo > "01101") then we_out <= '0';
        else we_out <= '1';
        end if;
      end if;
    else
      if(fila <= "0000001") then
        if(columna <= "0000001") then
          if((Xo >= "00000" and Xo <= "0011") or (Xo >= "00101" and Xo <= "00111") or
          (Xo >= "01001" and Xo <= "01011")) then we_out <= '1';
          else we_out <= '0';
          end if;
        elsif(columna >= maxColumna) then
          if((Xo >= "00010" and Xo <= "00100") or (Xo >= "00110" and Xo <= "01000") or (Xo >= "01010"
and
          Xo <= "01100")) then we_out <= '1';
          else we_out <= '0';
          end if;
        else
          if((Xo >= "00010" and Xo <= "00011") or (Xo >= "00110" and Xo <= "00111") or (Xo >=
"01010" and
          Xo <= "01011")) then we_out <= '1';
          else we_out <= '0';
          end if;
        end if;
      elsif(fila >= maxFila) then

```

```

        if(columna <= "0000001")then
            if((Xo>="00101"andXo<="00111") or (Xo>="01001" and Xo<="01011") or (Xo>="01101"
and
            Xo <= "01111"))then we_out <= '1';
            else we_out <= '0';
            end if;
        elsif(columna >= maxColumna)then
            if((Xo>="00110"andXo<="01000") or (Xo>="01010" and Xo<="01100") or (Xo>="01110"
and
            Xo <= "10000"))then we_out <= '1';
            else we_out <= '0';
            end if;
        else
            if((Xo>="00110"andXo<="00111") or (Xo>="01010" and Xo<="01011") or (Xo>="01110"
and
            Xo <= "01111"))then we_out <= '1';
            else we_out <= '0';
            end if;
        end if;
    else
        if(columna <= "0000001")then
            if((Xo>="00101"andXo<="00111") or (Xo>="01001" and Xo<="01011"))then we_out <=
'1';
            else we_out <= '0';
            end if;
        elsif(columna >= maxColumna)then
            if((Xo>="00110"andXo<="01000") or (Xo>="01010" and Xo<="01100")) then we_out <=
'1';
            else we_out <= '0';
            end if;
        else
            if((Xo>="00110"andXo<="00111") or (Xo>="01010" and Xo<="01011"))then we_out <=
'1';
            else we_out <= '0';
            end if;
        end if;
    end if;
    else we_out <= '0';
    end if;
end process;

```

```

proc_we_RAM: process(clk_RAM, en_guardaSalida_in, Xo) begin
    if(falling_edge(clk_RAM))then
        if(en_guardaSalida_in = '1')then
            case Xo is
                when "00000" => DI_out <= sal11; when "00001" => DI_out <= sal11;
                when "00010" => DI_out <= sal12; when "00011" => DI_out <= sal13;
                when "00100" => DI_out <= sal14; when "00101" => DI_out <= sal21;
                when "00110" => DI_out <= sal22; when "00111" => DI_out <= sal23;
                when "01000" => DI_out <= sal24; when "01001" => DI_out <= sal31;
                when "01010" => DI_out <= sal32; when "01011" => DI_out <= sal33;
                when "01100" => DI_out <= sal34; when "01101" => DI_out <= sal41;
                when "01110" => DI_out <= sal42; when "01111" => DI_out <= sal43;
                when "10000" => DI_out <= sal44; when others => null;
            end case;
            else DI_out <= (others => '0');
            end if;
        end if;
    end process;
end arq_guardaSalida;

```

B13. Módulo que calcula los valores para el siguiente bloque (calculaValores.vhd)

```

-- Diseñador: José de Jesús Morales Romero
-- VLSI-SEES-CINVESTAV-IPN
-- Modulo de Cálculo de valores
-- Descripción: Calcula los valores para apuntar al siguiente bloque de la imagen.
library IEEE;

```

```

use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity calculaValores is
    generic(widthWord: integer; widthAddr: integer);
    port(clk, clk_RAM, rst, rstInt, en_Calcula_in: in std_logic;
        en_Calcula_out, en_Next: out std_logic;
        totalPixeles, Xini: out std_logic_vector(widthAddr-1 downto 0);
        fila, columna: out std_logic_vector(widthWord-2 downto 0);
        maxFila, maxColumna: out std_logic_vector(widthWord-2 downto 0);
        xMax, yMax: in std_logic_vector(widthWord-1 downto 0));
end calculaValores;

architecture arq_calculaValores of calculaValores is

    type estados is (edo_Espera, edo_Bandera_Next, edo_CalculaFilCol, edo_CalcXini, edo_FinCal);
    signal edo_pres, edo_fut: estados;

    signal temFila: std_logic_vector(widthWord-1 downto 0);
    signal temColumna: std_logic_vector(widthWord-1 downto 0);
    signal tem_Xini: std_logic_vector(widthAddr-1 downto 0) := (others => '0');
    signal tem_Fila: std_logic_vector(widthWord-2 downto 0) := "0000001";
    signal tem_Columna: std_logic_vector(widthWord-2 downto 0) := "0000001";
    signal conta_NextIni: std_logic_vector(2 downto 0) := (others => '0');
    signal tem_MaxFila: std_logic_vector(widthWord-2 downto 0);
    signal tem_MaxColumna: std_logic_vector(widthWord-2 downto 0);
    signal tem_en_Next: std_logic := '0';
    signal banCalFilCol: std_logic := '0';
    signal banXini: std_logic := '0';
    signal ban_Next: std_logic := '0';
    signal tem_Fila_Dos: std_logic_vector(widthWord-1 downto 0) := "00000001";
    signal tem_Fila_Tres: std_logic_vector(widthWord-1 downto 0) := "00000001";

begin

    totalPixeles <= xMax * yMax;

    maxFila_maxColumna: process(clk) begin
        if(rising_edge(clk)) then
            temFila <= yMax - 1; temColumna <= xMax - 1; tem_MaxFila <= temFila(widthWord-1 downto
1);
            tem_MaxColumna <= temColumna(widthWord-1 downto 1);
        end if;
    end process;

    proceso_Estados: process(edo_pres, en_Calcula_in, ban_Next, banCalFilCol, banXini) begin
        case edo_pres is
            when edo_Espera =>
                if(en_Calcula_in = '1') then edo_fut <= edo_Bandera_Next;
                else edo_fut <= edo_Espera;
                end if;
            when edo_Bandera_Next =>
                if(ban_Next = '1') then edo_fut <= edo_CalculaFilCol;
                else edo_fut <= edo_Bandera_Next;
                end if;
            when edo_CalculaFilCol =>
                if(banCalFilCol = '1') then edo_fut <= edo_CalcXini;
                else edo_fut <= edo_CalculaFilCol;
                end if;
            when edo_CalcXini =>
                if(banXini = '1') then edo_fut <= edo_FinCal;
                else edo_fut <= edo_CalcXini;
                end if;
            when edo_FinCal =>
                if(en_Calcula_in = '1') then edo_fut <= edo_FinCal;
                else edo_fut <= edo_Espera;
                end if;
        end case;
    end process;
end process;

```

```

proceso_Avance: process(clk, rst, rstInt) begin
    if(rst = '1' or rstInt = '1')then edo_pres <= edo_Espera;
    elsif(rising_edge(clk))then edo_pres <= edo_fut;
    end if;
end process;

proceso_Bandera: process(clk_RAM, rst, rstInt) begin
    if(rst = '1' or rstInt = '1')then tem_en_Next <= '0'; ban_Next <= '0';
    else
        if(rising_edge(clk_RAM))then
            if(edo_pres = edo_Bandera_Next)then
                if(tem_Fila >= tem_MaxFila and tem_Columna >= tem_MaxColumna)then tem_en_Next <=
'1';
                else tem_en_Next <= '0';
                end if;
                ban_Next <= '1';
            else ban_Next <= '0'; tem_en_Next <= tem_en_Next;
            end if;
        end if;
    end if;
end process;

proceso_CalFilCol: process(clk_RAM, rst, rstInt) begin
    if(rst = '1' or rstInt = '1')then
        tem_Columna <= "0000001"; tem_Fila <= "0000001"; banCalFilCol <= '0';
    else
        if(rising_edge(clk_RAM))then
            if(edo_pres = edo_CalculaFilCol)then
                if(tem_Columna >= tem_MaxColumna)then tem_Columna<="0000001"; tem_Fila <=
tem_Fila+1;
                else tem_Columna <= tem_Columna + 1; tem_Fila <= tem_Fila;
                end if;
                banCalFilCol <= '1';
            else tem_Columna <= tem_Columna; tem_Fila <= tem_Fila; banCalFilCol <= '0';
            end if;
        end if;
    end if;
end process;

tem_Fila_Dos <= ('0'&tem_Fila) + ('0'&tem_Fila); tem_Fila_Tres <= tem_Fila_Dos - "00000010";

proceso_CalXini: process(clk_RAM, rst, rstInt) begin
    if(rst = '1' or rstInt = '1')then tem_Xini <= (others => '0'); banXini <= '0';
    else
        if(rising_edge(clk_RAM))then
            if(edo_pres = edo_CalcXini)then
                if(tem_Columna <= "0000001")then tem_Xini <= tem_Fila_Tres * xMax;
                else tem_Xini <= tem_Xini + 2;
                end if;
                banXini <= '1';
            else tem_Xini <= tem_Xini; banXini <= '0';
            end if;
        end if;
    end if;
end process;

proceso_Fin: process(clk, rst, rstInt) begin
    if(rst = '1' or rstInt = '1')then en_Calcula_out <= '0';
    else
        if(rising_edge(clk))then
            if(edo_pres = edo_FinCal)then en_Calcula_out <= '1';
            else en_Calcula_out <= '0';
            end if;
        end if;
    end if;
end process;

Xini <= tem_Xini; fila <= tem_Fila; columna <= tem_Columna;
maxFila <= tem_MaxFila; maxColumna <= tem_MaxColumna; en_Next <= tem_en_Next;

end arq_calculaValores;

```

B14. Módulo de control de envío de datos (ctrEnviaDatos.vhd)

```
-- Diseñador: José de Jesús Morales Romero
-- VLSI-SEES-CINVESTAV-IPN
-- Modulo de Control de envio de datos
-- Descripción: Modulo que controla en envio de datos desde la memoria RAM hacia el puerto
-- RS232.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity ctrEnviaDatos is
    generic(widthWord: integer := 8; widthAddr: integer := 5);
    port(clk, clk_RAM, rst, en_ctrEnvio_in, ce_tx_out: in std_logic;
         en_ctrEnvio_out, ce_tx_in: out std_logic;
         totalPixeles, addr_re_out: in std_logic_vector(widthAddr-1 downto 0);
         dato_tx: out std_logic_vector(7 downto 0);
         DO_out: in std_logic_vector(widthWord-1 downto 0));
end ctrEnviaDatos;

architecture arq_ctrEnviaDatos of ctrEnviaDatos is

    type SendData is (edo_EsperaEnvio, edo_EnviaByte, edo_SiguienteByte,
                     edo_CompruebaUltimoByte, edo_Espera);
    signal edo_pres_Envia, edo_fut_Envia: SendData;

    signal cont: std_logic_vector(widthAddr-1 downto 0) := (others => '0');
    signal contSigByte: std_logic_vector(1 downto 0) := (others => '0');
    signal ban_SigByte, ban_EnvioTotal: std_logic := '0';

    constant CR: std_logic_vector(7 downto 0) := X"0D";

begin

    process(edo_pres_Envia, en_ctrEnvio_in, ce_tx_out, ban_SigByte, ban_EnvioTotal) begin
        case edo_pres_Envia is
            when edo_EsperaEnvio =>
                if(en_ctrEnvio_in = '1') then edo_fut_Envia <= edo_EnviaByte;
                else edo_fut_Envia <= edo_EsperaEnvio;
                end if;
            when edo_EnviaByte =>
                if(ce_tx_out = '1') then edo_fut_Envia <= edo_SiguienteByte;
                else edo_fut_Envia <= edo_EnviaByte;
                end if;
            when edo_SiguienteByte =>
                if(ban_SigByte = '1') then edo_fut_Envia <= edo_CompruebaUltimoByte;
                else edo_fut_Envia <= edo_SiguienteByte;
                end if;
            when edo_CompruebaUltimoByte =>
                if(ban_EnvioTotal = '1') then edo_fut_Envia <= edo_Espera;
                else edo_fut_Envia <= edo_EnviaByte;
                end if;
            when edo_Espera =>
                if(en_ctrEnvio_in = '1') then edo_fut_Envia <= edo_Espera;
                else edo_fut_Envia <= edo_EsperaEnvio;
                end if;
        end case;
    end process;

    process(clk, rst) begin
        if(rst = '1') then edo_pres_Envia <= edo_EsperaEnvio;
        else
            if(rising_edge(clk)) then edo_pres_Envia <= edo_fut_Envia;
            end if;
        end if;
    end process;

    enviaByte: process(clk) begin
```

```

    if(clk'event and clk = '1') then
        if(edo_pres_Envia = edo_EnviaByte) then ce_tx_in <= '1';
        else ce_tx_in <= '0';
        end if;
    end if;
end process;

siguienteByte: process(clk_RAM, cont) begin
    if(clk_RAM'event and clk_RAM = '0') then
        if(edo_pres_Envia = edo_SiguienteByte) then cont <= cont + 1; ban_SigByte <= '1';
        elsif(edo_pres_Envia = edo_Espera) then cont <= (others => '0'); ban_SigByte <= '0';
        else cont <= cont; ban_SigByte <= '0';
        end if;
    end if;
    addr_re_out <= cont;
end process;

compruebaSiguienteByte: process(clk, cont) begin
    if(clk'event and clk = '0') then
        if(edo_pres_Envia = edo_CompruebaUltimoByte) then
            if(cont <= totalPixeles) then ban_EnvioTotal <= '0';
            else ban_EnvioTotal <= '1';
            end if;
        else ban_EnvioTotal <= ban_EnvioTotal;
        end if;
    end if;
end process;

DatoAEnviar: process(clk) begin
    if(clk'event and clk = '1') then
        if(cont < totalPixeles) then dato_tx <= DO_out;
        else dato_tx <= CR;
        end if;
    end if;
end process;

envioOut: process(clk) begin
    if(clk'event and clk = '1') then
        if(edo_pres_Envia = edo_Espera) then en_ctrEnvio_out <= '1';
        else en_ctrEnvio_out <= '0';
        end if;
    end if;
end process;

end arq_ctrEnviaDatos;

```

B15. Módulo de envío de datos (txRS232.vhd)

```

-- Transmisor RS232
-- Autor: José de Jesús Morales Romero
-- Descripción: Transmisor RS232 a 115200 Bauds
-- Puertos
-- ce_tx_in: habilita la transmisión de dato_txs
-- '1' envia dato_txs, '0' espera
-- ce_tx_out: Indica si termino de enviar el dato_tx
-- '1' en espera el tx, '0' en uso el tx
-- dato tx: dato tx a ser enviado de 8 bits
-- clk : señal de reloj 100MHz
-- rst : reset general
-- tx: salida serial RS232 a 115200 bauds
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity txRS232 is
    port(clk, rst, ce_tx_in: in std_logic;
        dato_tx: in std_logic_vector(7 downto 0);
        ce_tx_out, tx: out std_logic);
end txRS232;

```

```

architecture arq_txRS232 of txRS232 is

constant baudtx: std_logic_vector(9 downto 0) := "1101100100"; -- "1101100101" 115200 bauds
constant limBits: std_logic_vector(3 downto 0) := "1011";
signal contBaud: std_logic_vector(9 downto 0) := (others => '0');
signal contBits: std_logic_vector(3 downto 0) := (others => '0');

begin

cont_baud_tx: process(clk, rst, ce_tx_in, contBaud) begin
  if(rst = '1')then contBaud <= (others => '0'); contBits <= (others => '0');
  else
    if(ce_tx_in = '1')then
      if(rising_edge(clk))then
        if(contBaud > baudtx)then
          contBaud <= (others => '0');
          if(contBits < limBits)then contBits <= contBits + 1;
          else contBits <= limBits;
          end if;
        else contBaud <= contBaud + 1; contBits <= contBits;
        end if;
      end if;
    else contBaud <= (others => '0'); contBits <= (others => '0');
    end if;
  end if;
end process;

ban_salida: process(clk, rst, contBits) begin
  if(rst = '1')then ce_tx_out <= '1';
  else
    if(rising_edge(clk))then
      if(contBits < limBits)then ce_tx_out <= '0';
      else ce_tx_out <= '1';
      end if;
    end if;
  end if;
end process;

with contBits select
  tx <= '1' when "0000",
        '0' when "0001",          -- Bit de start
        dato_tx(0) when "0010", -- Inicia transmisión de dato_tx
        dato_tx(1) when "0011", -- LSB
        dato_tx(2) when "0100", dato_tx(3) when "0101", dato_tx(4) when "0110",
        dato_tx(5) when "0111", dato_tx(6) when "1000",
        dato_tx(7) when "1001",  -- MSB
        '1' when "1010",        -- Bit de stop
        '1' when others;

end arq_txRS232;

```

B16. Módulo que muestra el estado del Sistema (Muestra.vhd)

```

-- Diseñador: José de Jesús Morales Romero
-- VLSI-SEES-CINVESTAV-IPN
-- Modulo de Mostrar
-- Descripción: Muestra el estado del sistema
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity muestra is
  port(clk: in std_logic;
        LEDs: out std_logic_vector(7 downto 0); salLEDs: in std_logic_vector(7 downto
0));
end muestra;

architecture arq_Muestra of muestra is

begin

```

```

process(clk) begin
    if(clk'event and clk='1')then LEDs <= salLEDs;
    end if;
end process;

end arq_Muestra;

```

C. Código en MATLAB para la comunicación con el FPGA.

C1. Programa principal (main.m)

```

imagen = imread('./imagenes/Celulas/proteina_BW.jpg'); % Carga la imagen
imagenCNNTemporal = im2imCNN(imagen,1); % Convierte a escala de grises
imagenA = imagenCNNTemporal;
imagenCNN = uint8(compruebaTamano(imagenCNNTemporal)); %Comprueba el tamaño de la imagen

%Plantillas para Bordes
templateA = [0,0,0; 0,4,0; 0,0,0];
templateB = [0,-2,0;-2,5,-2;0,-2,0];
templateI = -1;
% Prueba de la CNN con el FPGA
tic %Empieza a medir el tiempo
A = template2bits(templateA);
B = template2bits(templateB);
I = I2bits(templateI);
%Parametros del serial y habre el puerto
s = serial('COM4');
set(s,'DataBits',8);
set(s,'StopBits',1);
set(s,'BaudRate',115200);
set(s,'Parity','none');
set(s,'Terminator','CR');
set(s,'InputBufferSize', 65536);
set(s,'Timeout', 30);
set(s,'OutputBufferSize', 65536);
% Calcula el tamaño de la imagen
[XFPGA, YFPGA, tamX_FPGA, tamY_FPGA] = calculaBloquesFPGA(imagenCNN);
% Preguntar cuantas veces se procesará la imagen
nVeces = input('Ingrese el valor de la variable: ');
% Comienza a procesar la imagen
if(nVeces < 1) disp('No es posible procesar la imagen'); toc;
else
    for mVeces=1:nVeces
        if((XFPGA == 1) && (YFPGA == 1))

```



```

    [imagenSalida, imagenAEnviar] = enviaRecibeImagen(imagenCNN, A, B, I, s);
else
    imagenSalida = imagenCNN;
    for j=1:YFPGA
        for i=1:XFPGA
            [imagenTemporal] = obtieneSubImgFPGA(imagenCNN, XFPGA, YFPGA,...
                tamX_FPGA, tamY_FPGA, i, j);
            [imagenTemporal] = enviaRecibeImagen(imagenTemporal, A, B, I, s);
            [imagenTemporal] = compruebaTamano(imagenTemporal);
            [imagenSalida] = guardaSubImgFPGA(imagenTemporal,...
                imagenSalida, XFPGA, YFPGA, tamX_FPGA, tamY_FPGA, i, j);
        end
    end
    imagenCNN = imagenSalida;
end
elapsedTime1 = toc; % Finaliza de medir el tiempo de ejecución del FPGA
mensaje = ['El tiempo de ejecución con FPGA fue de: ', num2str(elapsedTime1), ' s.'];
disp(mensaje);
mensaje0 = ['Respuesta FPGA, t = ', num2str(elapsedTime1), ' s'];
figure('Name','CNN multiplexada','NumberTitle','off')
subplot(1,2,1); %Muestra imagen original
imshow(imagenA);
title('Imagen original');
subplot(1,2,2); %Muestra imagen del FPGA
imshow(imagenSalida);
title(mensaje0);
end

```

C2. Función convertidor a escala de grises (im2imCNN)

```

function [imagenSalida] = im2imCNN(imagenEntrada, bandera)
%bandera = 1: Escala de grises
%bandera = 2: Escala de blanco y negros
%imagenSalida = imagenEntrada;
imagenSalida = rgb2gray(imagenEntrada);
if(bandera == 2)
    level = graythresh(imagenSalida); %Convierte a escala de blanco y negro
    imagenSalidaTemporal = im2bw(imagenSalida, level);
    [m,n] = size(imagenSalidaTemporal);
    for i=1:m
        for j=1:n
            if(imagenSalidaTemporal(i,j) == 1)
                imagenSalida(i,j) = 255; %Es blanco el pixel
            else
                imagenSalida(i,j) = 0; %Es negro el pixel
            end
        end
    end
end
imagenSalida = uint8(imagenSalida);
end

```

C3. Función para comprobar el tamaño de la función (compruebaTamano.m)

```

function [imagenCNN] = compruebaTamano(imagenCNN)
[m,n] = size(imagenCNN);
if (rem(m,2)==1)
    m = m + 1;
    imagenCNNtemp = zeros(m,n);
    for i=1:(m-1)
        for j=1:n
            imagenCNNtemp(i,j) = imagenCNN(i,j);
        end
    end
    for i=1:n
        imagenCNNtemp(m,i) = imagenCNNtemp(m-1,i);
    end
    imagenCNN = imagenCNNtemp;
end

```

```

end
if(rem(n,2)==1)
    n = n + 1;
    imagenCNNtemp = zeros(m,n);
    for i=1:m
        for j=1:(n-1)
            imagenCNNtemp(i,j) = imagenCNN(i,j);
        end
    end
    for i=1:m
        imagenCNNtemp(i,n) = imagenCNNtemp(i,n-1);
    end
    imagenCNN = imagenCNNtemp;
end
end

```

C4. Función para convertir los templetas a valores de bits (template2bits.m)

```

function [templateSalida] = template2bits(templateEntrada)
    temporal = zeros(1, 18); k = 1;
    for i=1:3
        for j=1:3
            [a,b] = dec2bits(templateEntrada(i,j));
            temporal(k+k-1) = a;
            temporal(k+k) = b;
            k = k + 1;
        end
    end
    templateSalida = uint8(temporal);
end

```

C5. Función auxiliar para convertir los templetas a valores de bits (template2bits.m)

```

function [Byte1, Byte2] = dec2bits(decimal)
    cantBits = 24;
    temporal = zeros(1, cantBits);
    p = 5;
    residuo = abs(decimal); %Convierte el valor en positivo
    for i=2:cantBits %Convierte a binario
        resultado = 2^(p);
        if(residuo >= resultado)
            temporal(i) = 1;
            residuo = residuo - resultado;
        else
            temporal(i) = 0;
        end
        p = p - 1;
    end
    if(decimal < 0) %Si decimal es valor negativo hace la conversión a complemento a 2
        for i=1:cantBits %Complemento a 1
            if(temporal(i) == 1)
                temporal(i) = 0;
            else
                temporal(i) = 1;
            end
        end
        p = cantBits; %Complemento a 2
        C = 0;
        BB = zeros(1,cantBits);
        BB(1,cantBits) = 1;
        for i=1:cantBits
            A = temporal(p);
            B = BB(p);
            temporal(p) = xor(C,xor(A,B));
            C = or(and(A,B), and(C, xor(A,B)));
            p = p-1;
        end
    end
    % Coloca Byte1 y Byte2

```

```

tem1 = temporal(1:8);
tem2 = temporal(9:16);
tem3 = temporal(17:24);
p = 7; %Calcula el valor decimal
Byte1 = 0;
Byte2 = 0;
Byte3 = 0;
for i=1:8
    Byte1 = Byte1 + tem1(i)*2^p;
    Byte2 = Byte2 + tem2(i)*2^p;
    Byte3 = Byte3 + tem3(i)*2^2;
    p = p-1;
end
%Convierte a uint8
Byte1 = uint8(Byte1);
Byte2 = uint8(Byte2); %Byte3 = uint8(Byte3);
end

```

C6. Función para convertir el umbral a valores de bits (I2bits.m)

```

function [Byte] = I2bits(decimal)
% Convierte el valor a positivo
residuo = abs(decimal);
temporal = zeros(1,40);
p = 12;
%Convierte a binario
for i=2:40
    resultado = 2^(p);
    if(residuo >= resultado)
        temporal(i) = 1;
        residuo = residuo - resultado;
    else
        temporal(i) = 0;
    end
    p = p - 1;
end
%Si decimal es valor negativo hace la conversión a complemento a 2
if(decimal < 0)
    %Complemento a 1
    for i=1:40
        if(temporal(i) == 1)
            temporal(i) = 0;
        else
            temporal(i) = 1;
        end
    end
    %Complemento a 2
    p = 40;
    C = 0;
    BB = zeros(1,40);
    BB(1,40) = 1;
    for i=1:40
        A = temporal(p);
        B = BB(p); temporal(p) = xor(C,xor(A,B));
        C = or(and(A,B), and(C, xor(A,B)));
        p = p-1;
    end
end
end
% Coloca Byte1 y Byte2
tem1 = temporal(1:8);
tem2 = temporal(9:16);
tem3 = temporal(17:24);
tem4 = temporal(25:32);
tem5 = temporal(33:40);
%Calcula el valor decimal
p = 7;
Byte1 = 0;
Byte2 = 0;
Byte3 = 0;
Byte4 = 0;

```

```

Byte5 = 0;
for i=1:8
    Byte1 = Byte1 + tem1(i)*2^p;
    Byte2 = Byte2 + tem2(i)*2^p;
    Byte3 = Byte3 + tem3(i)*2^p;
    Byte4 = Byte4 + tem4(i)*2^p;
    Byte5 = Byte5 + tem5(i)*2^p; p = p-1;
end
%Convierte a uint8
Byte1 = uint8(Byte1);
Byte2 = uint8(Byte2);
Byte3 = uint8(Byte3);
Byte4 = uint8(Byte4);
Byte5 = uint8(Byte5);
Byte = [Byte1, Byte2, Byte3, Byte4, Byte5];
End

```

C7. Función que calcula la cantidad de bloques (calculaBloquesFPGA.m)

```

function [BloqueX_FPGA, BloqueY_FPGA, tamX, tamY] = calculaBloquesFPGA(imagenCNN)
[m,n] = size(imagenCNN);
tamY = 254;
cont = 0;
res = m;
ban = 0;
while (ban == 0)
    cont = cont + 1;
    if ((res == 0) || ((res >= 4) && (res <= tamY)))
        ban = 1;
    else
        ban = 0;
        if (res < 0)
            tamY = tamY - 2;
            res = m;
            cont = 0;
        else
            res = res - tamY + 2;
        end
    end
end
BloqueY_FPGA = cont;
tamX = 254;
cont = 0;
res = n;
ban = 0;
while (ban == 0)
    cont = cont + 1;
    if ((res == 0) || ((res >= 4) && (res <= tamX)))
        ban = 1;
    else ban = 0;
        if (res < 0)
            tamX = tamX - 2;
            res = n;
            cont = 0;
        else
            res = res - tamX + 2;
        end
    end
end
BloqueX_FPGA = cont;
end

```

C8. Función que calcula la cantidad de bloques (enviaRecibeImagen.m)

```

function [imagenSalida, imagenAEnviar] = enviaRecibeImagen(imagenEntrada, A, B, I, s)
[fil, col] = size(imagenEntrada);
imagenEntrada = uint8(imagenEntrada);
imagenAEnviar = zeros(1,fil*col);
k = 1;

```

```

for i=1:fil
    for j=1:col
        imagenAEnviar(1,k) = imagenEntrada(i,j); k = k + 1;
    end
end
end
A = uint8(A);
B = uint8(B);
I = uint8(I);
imagenAEnviar = uint8(imagenAEnviar);
xMax = uint8(col);
yMax = uint8(fil);
% Abre el puerto
fopen(s);
% Envia los parametros
fwrite(s, xMax);
fwrite(s, yMax);
fwrite(s, A);
fwrite(s, B);
fwrite(s,I);
% Envia la imagen
fwrite(s,imagenAEnviar);
% Escribe el fin de datos
fwrite(s,4);
% Obtiene la imagen de salida
[a, count] = fgetl(s);
% Cierra el puerto
fclose(s);
% Crea una subimagen inicial
imagenSalida = uint8(ones(fil,col));
% Convierte a imagen leible por MATLAB
if (count < (fil*col+1))
    disp('Imagen recibida fuera de rango')
else
    k = 1;
    for i=1:fil
        for j=1:col
            if (a(k) == '1')
                imagenSalida(i,j) = 255;
            elseif (a(k) == '0')
                imagenSalida(i,j) = 0;
            else
                imagenSalida(i,j) = uint8(a(k));
            end
            k = k + 1;
        end
    end
end
end
end
end
end

```

C9. Función para obtener una subimagen de máximo 254x254 pixeles (obtieneSubImgFPGA.m)

```

function [imagenSalida] = obtieneSubImgFPGA(imagenEntrada, BloqueX_FPGA, BloqueY_FPGA, tamX,
tamY, xBloque, yBloque)
[m,n] = size(imagenEntrada);
if((BloqueX_FPGA == 1) && (BloqueY_FPGA > 1))
    xi = 1;
    xo = n;
    if(yBloque == 1)
        yi = 1;
        yo = tamY;
    elseif(yBloque == BloqueY_FPGA)
        yi = (yBloque-1)*tamY-(2*(yBloque-1)) + 1;
        yo = m;
    else
        yi = (yBloque-1)*tamY-(2*(yBloque-1)) + 1;
        yo = yi + tamY - 1;
    end
elseif((BloqueY_FPGA == 1) && (BloqueX_FPGA > 1))
    yi = 1;
    yo = m;

```

```

    if(xBloque == 1)
        xi = 1;
        xo = tamX;
    elseif(xBloque == BloqueX_FPGA)
        xi = (xBloque-1)*tamX-(2*(xBloque-1)) + 1;
        xo = n;
    else
        xi = (xBloque-1)*tamX-(2*(xBloque-1)) + 1;
        xo = xi + tamX - 1;
    end
else
    if(xBloque == 1)
        xi = 1;
        xo = tamX;
        if(yBloque == 1)
            yi = 1;
            yo = tamY;
        elseif(yBloque == BloqueY_FPGA)
            yi = (yBloque-1)*tamY-(2*(yBloque-1)) + 1;
            yo = m;
        else
            yi = (yBloque-1)*tamY-(2*(yBloque-1)) + 1;
            yo = yi + tamY - 1;
        end
    elseif(xBloque == BloqueX_FPGA)
        xi = (xBloque-1)*tamX-(2*(xBloque-1)) + 1;
        xo = n;
        if(yBloque == 1)
            yi = 1;
            yo = tamY;
        elseif(yBloque == BloqueY_FPGA)
            yi = (yBloque-1)*tamY-(2*(yBloque-1)) + 1;
            yo = m;
        else
            yi = (yBloque-1)*tamY-(2*(yBloque-1)) + 1;
            yo = yi + tamY - 1;
        end
    end
else
    xi = (xBloque-1)*tamX-(2*(xBloque-1)) + 1;
    xo = xi + tamX - 1;
    if(yBloque == 1)
        yi = 1;
        yo = tamY;
    elseif(yBloque == BloqueY_FPGA)
        yi = (yBloque-1)*tamY-(2*(yBloque-1)) + 1;
        yo = m;
    else
        yi = (yBloque-1)*tamY-(2*(yBloque-1)) + 1;
        yo = yi + tamY - 1;
    end
end
end
imagenSalida = imagenEntrada(yi:yo, xi:xo);
end

```

C10. Función para guardar una subimagen de máximo 254x254 pixeles (guardaSubImgFPGA.m)

```

function [imagenSalida] = guardaSubImgFPGA(imagenEntrada, imagenSalida, BloqueX_FPGA,
BloqueY_FPGA, tamX, tamY, xBloque, yBloque)
[mo,no] = size(imagenSalida);
if((BloqueX_FPGA == 1) && (BloqueY_FPGA > 1))
    xoi = 1;
    xof = no;
    xii = 1;
    xif = no;
    if(yBloque == 1)
        yoi = 1;
        yof = tamY - 1 ;
        yii = 1;
        yif = tamY - 1;
    end
end

```

```

elseif(yBloque == BloqueY_FPGA)
    yoi = ((yBloque-1)*tamY - (2*(yBloque - 1))) + 2;
    yof = mo; yii = 2;
    yif = mo - yoi + 2;
else
    yoi = ((yBloque-1)*tamY-(2*(yBloque-1)))+2;
    yof = yoi+tamY-3; yii = 2;
    yif = tamY - 1;
end
elseif((BloqueX_FPGA > 1) && (BloqueY_FPGA == 1))
    yoi = 1;
    yof = mo;
    yii = 1;
    yif = mo;
    if(xBloque == 1)
        xoi = 1;
        xof = tamX - 1;
        xii = 1;
        xif = tamX - 1;
    elseif(xBloque == BloqueX_FPGA)
        xoi = ((xBloque-1)*tamX-(2*(xBloque-1)))+2;
        xof = no;
        xii = 2;
        xif = no-xoi+2;
    else
        xoi = ((xBloque-1)*tamX-(2*(xBloque-1)))+2;
        xof = xoi+tamX-3;
        xii = 2;
        xif = tamX-1;
    end
else
    if(xBloque == 1)
        xoi = 1;
        xof = tamX - 1;
        xii = 1;
        xif = tamX - 1;
        if(yBloque == 1)
            yoi = 1;
            yof = tamY - 1;
            yii = 1;
            yif = tamY - 1;
        elseif(yBloque == BloqueY_FPGA)
            yoi = ((yBloque-1)*tamY-(2*(yBloque-1)))+2;
            yof = mo;
            yii = 2;
            yif = mo-yoi+2;
        else
            yoi = ((yBloque-1)*tamY-(2*(yBloque-1)))+2;
            yof = yoi+tamY-3;
            yii = 2;
            yif = tamY-1;
        end
    elseif(xBloque == BloqueX_FPGA)
        xoi = ((xBloque-1)*tamX-(2*(xBloque-1)))+2;
        xof = no;
        xii = 2;
        xif = no-xoi+2;
        if(yBloque == 1)
            yoi = 1;
            yof = tamY - 1;
            yii = 1;
            yif = tamY - 1;
        elseif(yBloque == BloqueY_FPGA)
            yoi = ((yBloque-1)*tamY-(2*(yBloque-1)))+2;
            yof = mo;
            yii = 2;
            yif = mo - yoi + 2;
        else
            yoi = ((yBloque-1)*tamY-(2*(yBloque-1)))+2;
            yof = yoi+tamY-3;
            yii = 2;
        end
    end
end

```

```

    yif = tamY - 1;
end
else
if(yBloque == 1)
    xoi = ((xBloque - 1)*tamX - (2*(xBloque - 1))) + 2;
    xof = xoi + tamX - 3;
    yoi = 1;
    yof = tamY - 1;
    xii = 2;
    xif = tamX - 1;
    yii = 1;
    yif = tamY - 1;
elseif(yBloque == BloqueY_FPGA)
    xoi = ((xBloque - 1)*tamX - (2*(xBloque - 1))) + 2;
    xof = xoi + tamX - 3;
    yoi = ((yBloque - 1)*tamY - (2*(yBloque - 1))) + 2;
    yof = mo;
    xii = 2;
    xif = tamX - 1;
    yii = 2;
    yif = mo - yoi + 2;
else
    xoi = ((xBloque - 1)*tamX - (2*(xBloque - 1))) + 2;
    xof = xoi + tamX - 3;
    yoi = ((yBloque - 1)*tamY - (2*(yBloque - 1))) + 2;
    yof = yoi + tamY - 3;
    xii = 2;
    xif = tamX - 1;
    yii = 2;
    yif = tamY - 1;
end
end
end
imagenSalida(yoi:yof, xoi:xof) = imagenEntrada(yii:yif, xii:xif);
end

```