UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

PROGRAMA DE MAESTRIA Y DOCTORADO EN INGENIERIA

FACULTAD DE INGENIERÍA

"INVESTIGACIÓN, DESARROLLO E IMPLEMENTACIÓN DIGITAL DE UNA RED NEURONAL PARA RECONOCIMIENTO DE IMÁGENES"

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

DOCTOR EN INGENIERÍA

INGENIERÍA ELÉCTRICA – INSTRUMENTACIÓN

P R E S E N T A :

VEGA RAMÍREZ ALEJANDRO ANTONIO

TUTOR:
Dra. TETYANA BAYDYK

2011

JURADO ASIGNADO:

Presidente: Dr. Felipe de Jesús Lara Rosano

Secretario: Dr. Marco Antonio Arteaga Pérez

1er. Vocal: Dra. Tetyana Baydyk Mykolaivna

2do. Vocal: Dr. Ernst Mikhailovich Kussul

3er. Vocal: Dra. Graciela Velasco Herrera

Lugar o lugares donde se realizó la tesis:

CCADET

TUTOR DE TESIS:
Dra. Tetyana Baydyk Mykolaivna

_____

FIRMA

Parte de este trabajo fue apoyado por los proyectos:
PAPIIT IN 110510-3
PAPIIT IN 119610
ICyTDF 330/2009

Índice

Investigación, desarrollo e implementación digital de una red neuronal para reconocimiento de imágenes

Tesis Doctoral

Resumen

Alejandro Antonio Vega Ramírez
Facultad de Ingeniería
Centro de Ciencias Aplicadas y Desarrollo Tecnológico
Universidad Nacional Autónoma de México

El uso de redes neuronales en el reconocimiento y clasificación de imágenes muestra resultados y rendimientos, ambos buenos. Esta es una razón muy importante por la que se han convertido en objeto de análisis y discusión dentro del área de control, recibiendo una considerable atención.

La implementación de redes neuronales en dispositivos FPGA (*Field Programmable Gate Array*) es una forma de optimizar y economizar los sistemas de control. Estos dispositivos pueden ser programados a partir de lenguajes de programación de hardware, como Verilog y VHDL (acrónimo de VHSIC, *Very High Speed Integrated Circuit*, y HDL, *Hardware Description Language*), los cuales son independientes de los distintos modelos y productores de dichos dispositivos.

Los FPGAs tienen características, como las altas velocidades de operación, para responder adecuadamente con los requerimientos definidos por los algoritmos de trabajo de las redes neuronales. La alta integración de los dispositivos FPGA permite la implementación de estos algoritmos en un solo dispositivo, lo cual reduce espacios y costos en la implantación de estos dispositivos en el control de procesos. Los FPGAs pueden soportar los cálculos paralelos de las redes neuronales de forma masiva, por lo que existen aplicaciones sustentadas en ambos temas, FPGAs y redes neuronales.

Se propone la implementación del clasificador neuronal de área receptiva limitada, LImited Receptive Area (LIRA), en un dispositivo programable con el objetivo de implantarse en un proceso de clasificación de imágenes. Se implementa para 2 y 5 clases respectivamente, utilizándose dos plataformas de desarrollo y dispositivos diferentes para tal efecto, una tarjeta de versión anterior con capacidades pequeñas, y una de nueva generación con las capacidades que la alta integración ofrece actualmente. Este clasificador neuronal será utilizado en el reconocimiento de imágenes.

Research, development and digital implementation of a neural network for image recognition
Doctoral thesis
Abstract

Alejandro Antonio Vega Ramírez
Faculty of Engineering
Center of Applied Science and Technological Development
National Autonomous University of Mexico

The use of neural networks in image recognition and classification shows good results and performance. This is an important reason why they became the object of analysis and discussion within the control area, receiving considerable attention.

The implementation of neural networks in FPGAs (Field Programmable Gate Array) is a way to optimize and economize the control systems. These devices can be programmed using programming languages such as Verilog and VHDL (an acronym for VHSIC, Very High Speed Integrated Circuit, and HDL, Hardware Description Language), which are independent of the different models and manufacturers of these devices.

FPGAs have features such as high operating speeds, to respond adequately to the requirements defined by the neural networks algorithms. The high integration of the FPGA devices allows the implementation of these algorithms in a single device, reducing space and costs of control systems. The FPGAs can support the massive parallel computations of neural networks, so there are applications supported on both issues, FPGAs and neural networks.

The implementation of the LImited Receptive Area (LIRA) neural classifier in a programmable device is proposed for an image classification process. It is implemented for 2 and 5 classes respectively, using two different development platforms and devices for this purpose, a previous version card with small capacities and another one of new generation with the high integration capabilities currently offered. This neural classifier will be used in image recognition.

Introducción

Los sistemas de control adaptivo en general implican la capacidad de acomodarse a modificaciones no predecibles del medio, sean esos cambios internos o externos. Un sistema de control adaptivo mide las características dinámicas en forma continua y automática de la planta, y usa la diferencia para variar los parámetros ajustables al sistema, o genera una señal actuante para mantener el desempeño óptimo, independientemente de las modificaciones ambientales. Es decir, el sistema mide continuamente su desempeño propio de acuerdo con alguna referencia establecida y modificar, en su momento, sus parámetros para mantener un desempeño óptimo independientemente de las modificaciones externas. Estas características de auto-organización le otorgan el nominativo de adaptivo.

La base del control adaptivo se fundamenta sobre el hecho de que hay alguna condición de operación o desempeño del sistema, mucho mejor que cualquiera otra, para lo cual es necesario definir qué construye ese desempeño óptimo. En sistemas de control adaptivo tal desempeño está definido en función del índice de desempeño, que se debe fijar al establecer los objetivos, que pueden ser tan diversos como los sistemas a los cuales se van a aplicar, pero en general se hacen extensibles al objetivo de la optimización, al minimizar el costo de operación o maximizar el beneficio económico.

El uso de redes neuronales, como controladores no lineales, en los procesos de control muestra que los resultados obtenidos y los rendimientos son los deseados, con lo cual se observan diversas ventajas con respecto a los que se logran con los controladores convencionales [1]. Esta es una razón muy importante por la que se han convertido en objeto de análisis y discusión dentro del área de control, recibiendo una considerable atención dentro de los temas involucrados en el campo del control automático [2] – [8]. Su eficacia como compensadores en el rechazo de los efectos producidos por perturbaciones, mejorando la precisión en el seguimiento para eliminar el par por perturbaciones no lineales en el lazo estabilizador del buscador [9], y en el control de velocidad en estado ocioso del motor de un auto [10], así como en la compensación a las fricciones y perturbaciones producidas en el proceso de lectura y escritura de un disco duro [1], son ejemplos de aplicación de redes neuronales al control de procesos.

Por consecuencia, los algoritmos basados en redes neuronales para el control de procesos se han involucrado en diversos campos, de tal manera que existen diseños de controladores neuronales con diferentes aplicaciones [11] – [14]. También, utilizando estas redes neuronales y sus algoritmos, se pueden desarrollar e implementar técnicas y métodos de diagnostico para detección de fallas y protección de motores eléctricos, garantizando una optima confiabilidad, seguridad máxima, mantenimiento oportuno y preventivo, tanto en aplicaciones de baja potencia como en alta potencia [15].

A partir de la aparición de la lógica difusa, respaldada por la teoría del conjunto difuso de Lotfi Zadeh [16], donde existe un cierto grado de veracidad en una afirmación, que va mas allá de los valores lógicos clásicos “0” y “1”, muchos campos la han utilizado, tales como la inteligencia artificial y la teoría de control, estableciéndose, en esta ultima, un nuevo tipo de control conocido como difuso, el cual está ideado para controlar procesos sustentado en los principios de dicha lógica.

La utilidad mostrada por esta lógica es tal que ha provocado que las redes neuronales hayan sido involucradas dentro de este campo difuso, implantándose algoritmos de trabajo con el objetivo de controlar procesos, cuyo comportamiento no puede ser definido por la lógica proposicional clásica. Por ello, existen redes neuronales difusas, las cuales combinan la capacidad de razonar de forma difusa para manipular información dudosa, y la capacidad de aprender de los procesos, como las redes neuronales artificiales. Además, las redes neuronales difusas tienen la habilidad de aproximarse a los sistemas dudosos y no lineales [17] – [20]. Por tanto, muchas investigaciones fueron realizadas usando redes neuronales difusas para representar plantas complejas y construir controladores avanzados [21] – [27].

Los algoritmos basados en redes neuronales tienen la ventaja de poder ser simplificados en algoritmos de trabajo menos complejos, con lo que se reducen las demandas de equipos de cómputo robustos, y mejoran las tolerancias en cuanto a fallas [28].

La implementación de redes neuronales en dispositivos FPGA (*Field Programmable Gate Array*) es una forma de optimizar y economizar los sistemas de control, debido a que estos dispositivos se pueden programar, y volver a programar, buscando que el diseño del sistema sea el requerido por el usuario, en concordancia con sus necesidades de tamaño, velocidad y precio [28].

Estos dispositivos pueden ser programados a partir de lenguajes de programación de hardware, como Verilog y VHDL (acrónimo de VHSIC, *Very High Speed Integrated Circuit*, y HDL, *Hardware Description Language*), los cuales son independientes de los distintos modelos y productores de dichos dispositivos, además de que su compatibilidad con las diversas plataformas de desarrollo es tal, que los diseños pueden ser emigrados hacia las diferentes opciones de desarrollo presentes en el mercado.

Por otro lado, los FPGA's tienen características, como las altas velocidades de operación, para responder adecuadamente con los requerimientos definidos por los algoritmos de trabajo de las redes neuronales. La alta integración de estos dispositivos FPGA permite la implementación total de estos algoritmos de trabajo en un solo dispositivo, lo cual reduce espacios y costos en la implantación de estos dispositivos en el control de procesos [28] – [34]. Los FPGAs pueden soportar los cálculos paralelos de las redes neuronales de forma masiva, por lo que existen aplicaciones sustentadas en ambos temas, FPGAs y redes neuronales [35] – [38].

La estructura interna de los FPGAs permite la implementación de un procesador de propósito general, el cual se utilice como un controlador de procesos. Esto es, un dispositivo FPGA se puede convertir en un procesador de propósito general, al desarrollársele una arquitectura de control y procesamiento de datos internamente, que posteriormente sea utilizado como un controlador digital en procesos de control. Esto es realizable debido a que dentro de un FPGA se puede implementar toda la circuitería necesaria para procesar, de forma rápida y simple, los datos necesarios para el control, y también los algoritmos de control pueden ser ejecutados, de forma fácil y sencilla [38]. Se logra entonces una mezcla entre software y hardware dentro de un solo dispositivo, cuyo resultado incrementa la programabilidad y flexibilidad en el diseño de sistemas digitales de

control, mejorando el rendimiento del sistema a través del procesamiento paralelo, y reduciendo los tiempos de desarrollo [39] – [42].

Las redes neuronales se caracterizan por sus pesados algoritmos de cálculo paralelo, los cuales emplean grandes cantidades de recursos computacionales, y son menos útiles para implementaciones con sistemas digitales baratos [43]. Debido a las grandes cantidades de recursos digitales que se ven envueltos en los diseños de controladores neuronales, la necesidad de dispositivos FPGAs con capacidades mayores provoca una gran demanda de estos y un incremento en los costos de los mismos, lo cual hace que no sean considerados una opción inmediata para la implementación de controladores neuronales [43].

A partir de lo anterior, a continuación se propone la implementación de un clasificador neuronal en un dispositivo programable con el objetivo de implantarse en un proceso de clasificación de imágenes. Este clasificador neuronal será implementado en el reconocimiento de imágenes de micropiezas, cifras escritas, así como texturas y rostros humanos, entre otros.

En el primer Capitulo se resalta la importancia de la adquisición de datos para ser procesados por sistemas electrónicos, particularmente los sistemas computacionales digitales, y se describen un poco los métodos existentes usualmente empleados para el reconocimiento y clasificación de imágenes.

En el Capitulo 2 se mencionan las ventajas que ofrecen las redes neuronales como herramientas en el proceso de reconocimiento y clasificación de imágenes, y se presenta el clasificador neuronal LIRA, parte central de este trabajo de investigación. Se describe su algoritmo de trabajo y los procesos que lleva a cabo en los procesos de reconocimiento.

En el tercer Capitulo se describen de manera general los dispositivos programables existentes para ser utilizados en el desarrollo e implementación de clasificadores neuronales. Posteriormente se realiza la implementación de una neurona a partir de las condiciones de operación que se pretende esta tenga.

El Capitulo cuarto describe el desarrollo del clasificador neuronal LIRA para su implementación en un dispositivo programable, desde las especificaciones que debe tener dicho clasificador, hasta su implementación física en el dispositivo programable. Cabe señalar que en este capitulo se implementa para 2 y 5 clases respectivamente, con la salvedad de que se utilizan dos plataformas de desarrollo y dispositivos diferentes para tal efecto, una tarjeta de versión anterior con capacidades pequeñas, y una de nueva generación con las capacidades que la alta integración ofrece actualmente.

Finalmente se hace un recuento de los resultados obtenidos al utilizar estos dispositivos programables para la implementación de clasificadores neuronales para el reconocimiento de imágenes y de las ventajas que pueden aportar en las tareas de reconocimiento a las cuales sean aplicados.

Dados los alcances obtenidos con estos dispositivos utilizados, se proponen nuevas líneas de trabajo para aprovechar al máximo las capacidades de los dispositivos programables, y en particular el utilizado para este trabajo, tratando de cubrir totalmente las expectativas

para las cuales se pretende implementar este clasificador neuronal LIRA en un dispositivo programable que sea capaz de realizar todo el proceso de reconocimiento y clasificación de imágenes para cualquier tarea que le sea asignada.

Capítulo 1

Reconocimiento y clasificación de imágenes

1.1. Introducción

El reconocimiento de patrones es un proceso fundamental que se encuentra en casi todas las acciones humanas, este involucra la identificación de figuras y el reconocimiento de formas Estas figuras y formas son parte de un conjunto de señales de fácil interpretación para el ser humano cuya capacidad le permite distinguir y clasificar bajo criterios específicos. Este es, sin duda, un motivo muy importante de análisis para el campo de la computación puesto que pretende emular dichos criterios humanos bajo el sustento de un sistema computacional, es decir, una maquina que aprende, a partir de la toma de datos directos, y en función de estos, realizar una acción acorde a su categoría.

1.1.1. Clasificación de imágenes

Basado en los conceptos anteriores, se define una parte esencial del reconocimiento de patrones: la clasificación. Una señal se clasifica a partir de sus características. Las señales, sus características y las clases a las que corresponden pueden ser de cualquier índole o naturaleza, un ejemplo de ello puede ser la clasificación de imágenes binarias de números [45], o la clasificación de ruidos de cantos de los pájaros en clases de órdenes aviares en función de las frecuencias de los sonidos emitidos [46].

La clasificación de patrones se basa en un conocimiento previo obtenido de los datos que se pretende involucrar en este proceso. Estos patrones a clasificar pueden ser grupos de medidas, observaciones, muestras, recopilaciones.

Un sistema diseñado para la clasificación puede consistir de elementos tales como un sensor que recoge las observaciones a clasificar, un sistema de extracción de características que transforma la información observada en valores numéricos o simbólicos, y un sistema de clasificación que, basado en las características extraídas, clasifica la medición.

El proceso de clasificación normalmente se basa en un conjunto de patrones ya clasificados y descritos previamente. Este conjunto es denominado de entrenamiento, y la actividad desarrollada a partir de este criterio es el aprendizaje supervisado. Cuando el sistema no tiene un conjunto de entrenamiento para su "aprendizaje", es decir, no hay datos previos, el proceso es no supervisado, ya que el propio sistema se ajusta a las clases basado en un criterio estadístico para los patrones.

La clasificación utiliza habitualmente uno de los siguientes procedimientos [44]:

- clasificación estadística (o teoría de la decisión)
- clasificación sintáctica (o estructural)

El reconocimiento estadístico de patrones está basado en las características estadísticas de los patrones, asumiendo que han sido generados por un sistema probabilístico. El reconocimiento estructural de patrones está basado en las relaciones estructurales de las características de dichos patrones [44].

Una amplia gama de algoritmos son desarrollados para la clasificación, desde los sencillos clasificadores bayesianos hasta las más poderosas redes neuronales. Un problema de sumo interés es la relación que existe entre el problema a resolver, es decir, los datos a clasificar, y el uso de diversos algoritmos de clasificación.

Entre las aplicaciones de la clasificación de patrones están el reconocimiento de voz, la clasificación de documentos en sus diversos requerimientos, por ejemplo la diferenciación entre spam y no spam, el reconocimiento de escritura, reconocer el código postal escrito en los sobres, el reconocimiento de imágenes, caras humanas como un ejemplo, y muchas más [44 - 50]. Los dos últimos ejemplos operan con imágenes digitales como entradas del sistema de clasificación.

1.1.2. Adquisición de datos

La adquisición de datos consiste en tomar muestras del mundo real, convirtiéndolas en datos los cuales pueden ser operados y manipulados por una computadora. A estos datos, tales como señales y formas de onda, todos ellos analógicos, se les procesa para obtener información necesaria [51]. Los sistemas que se encargan de la adquisición de datos incluyen sensores de capacidades apropiadas para convertir cualquier parámetro medido a una señal eléctrica. Los datos obtenidos a través de este proceso pueden ser mostrados, analizados, y almacenados en una computadora a partir de la digitalización de dichas señales a través de un convertidor analógico digital.

El proceso de adquisición de datos continúa con la medición de las características y propiedades físicas del objeto bajo investigación. Para obtener estos valores físicos se utiliza un transductor, el cual convierte todos estos datos físicos a señales eléctricas medibles, tales como voltajes, corrientes, variaciones de resistencia o capacitancia, entre otras. La capacidad de un sistema de adquisición de datos para medir diferentes fenómenos físicos depende de los transductores, ya que estos deben convertirlos en señales medibles por la electrónica del sistema. Existen diversos tipos de transductores para diferentes aplicaciones, así, los hay para medir temperatura, presión o flujo. Posteriormente se utiliza un convertidor analógico digital para digitalizar las diferentes señales eléctricas obtenidas, y que puedan ser operadas y manipuladas por la computadora [51].

1.1.3. Cámaras web como medios de adquisición de imágenes

Visión computacional es el conjunto de todas aquellas técnicas y modelos que permitan el procesamiento, análisis y explicación de cualquier tipo de información espacial obtenida a través de imágenes digitales.

El objetivo de la visión computacional es tomar decisiones útiles acerca de los objetos físicos reales del mundo, es decir, de las escenas presentes, con base en las imágenes digitales adquiridas. Por lo tanto, su tarea consiste en la construcción de descriptores de la escena en función de las características relevantes contenidas en una imagen.

Debido a que la información visual es una de las principales fuentes de datos del mundo real, resulta útil el proveer a una computadora digital del sentido de la vista, (a partir de

imágenes tomadas con cámaras digitales o analógicas) que, junto con otros mecanismos como el "aprendizaje", hagan de esta una herramienta capaz de detectar y ubicar objetos en el mundo real, parte del objetivo principal de la visión por computadora. En la Figura 1.1 se presentan algunos ejemplos de cámaras digitales.

a *b* *c*

Figura 1.1. Cámaras digitales: a) Creative Labs Video Blaster WebCam NX; b) Creative Labs Video Blaster WebCam NX PRO; c) D-Link DCS – 5300

La cámara Creative Labs Video Blaster WebCam NX (Figura 1.1.*a*) tiene un costo de entre $22 y $39 dólares americanos, y se puede utilizar con el puerto USB, tiene una resolución de: 352 x 288 y 640 x 480 píxeles . La cámara Creative Labs Video Blaster WebCam NX PRO (Figura 1.1.*b*) tiene un costo mayor, de entre $39 y $62 dólares, tiene conexión para el puerto USB, y su resolución es de: 1024 x 768 píxeles. La cámara D-Link DCS – 5300 (Figura 1.1.*c*) tiene un costo de $377.54 dólares, con capacidad para conexión con Ethernet y Fast Ethernet; su resolución es de 380 líneas de televisión. Puede rotarse y capturar imagines y video en 270 grados.

1.2. Métodos para el reconocimiento y clasificación de imágenes

El sistema nervioso humano recibe aproximadamente 10^9 bits de datos externos por segundo, adquiridos y procesados la mayoría de estos por el sentido de la vista [52]. De manera análoga, la mayoría de los datos a ser procesados automáticamente aparecen en forma de imágenes.

El objetivo del reconocimiento de imágenes es lograr una descripción concisa y representativa del universo bajo observación. El interés en esta información incluye características detalladas, modos de comportamiento, etc. que involucran a dichos elementos, tales como objetos, fenómenos, conceptos. Por ello, estos se perciben como patrones y los procesos que llevan a su comprensión son llamados *procesos perceptuales*. Por lo tanto, el reconocimiento de patrones es una herramienta esencial para la interpretación automática de datos, ya que se encarga del etiquetado, es decir, la clasificación y la asignación de nombres de esos elementos. La clasificación es la etapa de toma de decisiones del sistema. Su función consiste en asignar a la categoría apropiada los patrones de clase desconocida de forma previa.

A continuación se muestran varios enfoques para el reconocimiento de patrones:

1.2.1. Reconocimiento estadístico: Esta opción se basa en la teoría de probabilidad y estadística y supone que se tiene un conjunto de medidas numéricas con

distribuciones de probabilidad conocidas, a partir de las cuales se hace el reconocimiento [53].

Las diferencias en patrones de una misma clase puede deberse a ruido, deformaciones y otras variaciones, por lo cual se debe tomar en cuenta esta variabilidad en dichos patrones. Por ello, el proceso asociado a la generación de patrones puede describirse mediante un modelo probabilístico adecuado. Con lo anterior se puede asumir que cada patrón $x = [x_1, x_2,\ldots, x_n]$ es un vector aleatorio n-dimensional que pertenece a una de m posibles clases $w_i i = (1,\ldots, m)$ donde cada clase w_i tiene una probabilidad de ocurrencia supuesta igual a $P(w_i)$. La distribución de probabilidad del vector patrón x de la clase w_i se caracteriza por la función densidad de probabilidad condicional para la i-esima clase $p(x \mid w_i)$.

Tomando en cuenta que las probabilidades predefinidas suman uno:

$$\sum_{i=1}^{m} P(w_i) = 1 \tag{1.1}$$

y que la densidad conjunta o función de densidad de probabilidad no condicional $p(x)$ está dada por:

$$p(x) = \sum_{i=1}^{m} p(x \mid w_i)\, P(w_i) \tag{1.2}$$

Entonces, desde el punto de vista práctico, el interés se centra en la probabilidad posterior al evento, es decir cuando el patrón x ya ha ocurrido, para cada clase w_i. Esto se obtiene por la *Fórmula de Bayes* relacionando las probabilidades condicionales [52]:

$$p(w_i \mid x) = \frac{p(x \mid w_i) P(w_i)}{p(x)} = \frac{p(x \mid w_i) P(w_i)}{\sum_{j=1}^{m} p(x \mid w_j) P(w_j)} \tag{1.3}$$

1.2.2. Reconocimiento sintáctico: Este enfoque se fundamenta en encontrar las relaciones estructurales que guardan los objetos de estudio, utilizando la teoría de lenguajes formales [53]. El objetivo es construir una gramática que describa la estructura del universo de objetos.

El reconocimiento estadístico de patrones no toma en cuenta el contexto del patrón que se pretende etiquetar, esto es, la relación entre los diferentes patrones que pudieran existir. En ciertas situaciones, los patrones complejos pueden ser descompuestos en elementos más simples de manera recursiva hasta llegar a componentes básicos. El sistema de reconocimiento que hace uso de esta información de contexto se le denomina sintáctico. En este tipo de reconocimiento sintáctico, cada patrón se describe en términos de sus elementos básicos, es decir, sus elementos terminales, y de ciertas reglas sintácticas, la gramática bajo la cual se rige el proceso [54].

El reconocimiento del lenguaje natural implica la representación de patrones mediante una estructura jerárquica, la cual involucra interacciones entre primitivas. La relación entre la estructura de patrones y la sintaxis del lenguaje consiste en:

- Los patrones, que son las sentencias en el lenguaje establecido.
- Las primitivas, que son el alfabeto de dicho lenguaje.
- Las sentencias generadas a partir de una gramática definida.

El reconocimiento consiste en comprobar que el patrón, en este caso la sentencia, se puede generar mediante las primitivas, los elementos del alfabeto, a partir de las reglas definidas por la gramática especificada.

Posteriormente, se aplica un análisis sintáctico que se encarga de decidir si una oración es gramaticalmente correcta, por lo que depende de utilizar una gramática muy extensa formada por todas las reglas del lenguaje.

1.2.3. Reconocimiento lógico combinatorio: Esta alternativa se sustenta en la idea de que la modelación del problema debe ser lo más cercana posible a la realidad del mismo, sin hacer suposiciones que no estén fundamentadas. Uno de los aspectos esenciales del enfoque es que las características utilizadas para describir a los objetos de estudio deben ser tratadas cuidadosamente [53].

Existen dos maneras de representación de objetos en el reconocimiento de patrones: representación en términos de ciertos alfabetos de las partes primitivas de los objetos, forma típica de la aproximación sintáctica; o la representación en términos de ciertos conjuntos de características, o variables, típica del reconocimiento de patrones estadístico. El reconocimiento de patrones lógico combinatorio usa esta ultima forma de representar objetos [55].

Los objetos se representan usualmente a través de una secuencia de valores numéricos o de valores categóricos exclusivamente. Sin embargo, cuando se analizan problemas reales en reconocimiento de patrones, con selección de características y clasificación supervisada o no supervisada, en muchos casos, la descripción de los objetos por los especialistas en estas áreas no es meramente una descripción en términos de características categóricas o numéricas exclusivamente.

Existen muchas instancias donde la descripción de los objetos no es clásica, es decir, las características no son exclusivamente numéricas o categóricas. Ambos tipos de valores pueden aparecer simultáneamente, y algunas veces, incluso es necesario utilizar un símbolo especial que denote la ausencia de valores, que son valores perdidos. Una descripción mezclada e incompleta de objetos se utiliza en este caso. Mezclada en el sentido de que hay características numéricas y categóricas simultáneamente; incompleta porque hay valores faltantes en las descripciones del objeto. A partir de esto, los objetos son solo vectores *n*-arios; elementos de un producto cartesiano simple sin alguna propiedad topológica, lógica o algebraica asumida. Otra alternativa para los vectores *n*-arios es el uso de una codificación de todas las características categóricas.

En este caso uno puede caer en la falacia ingenua de trabajar solo con números y entonces tener la natural posibilidad de usar un operador aritmético especifico. No se puede hacer una interrelación directa entre códigos y números, es decir, una mezcla o combinación, cuando se usan estas clases de operadores. Una vez que un problema concreto surge, estos tipos de descripción de objetos tienen varias implicaciones. Es imposible, desde un punto de vista metodológico, emplear un numero real o un espacio booleano para la representación de objetos, o usar algunos operadores aritméticos, o funciones de distancia para medir la similitud entre dos descripciones de un objeto. Muchos clasificadores, supervisados y no supervisados, así como muchas técnicas de selección de características se construyeron con la convicción de que los datos eran solo números reales. Como consecuencia, estos clasificadores y técnicas no serían empleados en la solución de problemas con datos mezclados. La mayoría de ellos asume distribuciones concretas conocidas de los valores de característica, lo que casi nunca coincide con la realidad. Por estas razones, muchos de los métodos presentados para resolver problemas tradicionales en reconocimiento de patrones no tienen sentido [55], o a lo mucho dan una pista vaga sobre la posible solución del problema en el caso de datos incompletos mezclados. Aun así, estas posibles pistas son difíciles de considerar.

Existen problemas de selección de características y clasificación cuyas descripciones de objetos se basan en datos mezclados e incompletos, y bajo estas condiciones, es probable que el reconocimiento de patrones lógico combinatorio trabaje mejor que las aproximaciones tradicionales debido a su capacidad de capturar la realidad y al mismo tiempo procesar estos datos por un modelado matemático dentro de una adecuada estructura metodológica. Aun cuando esta opción provee de mejoras significativas en los casos de datos mezclados incompletos, también puede ser usada en problemas donde las otras opciones han dado buenos resultados.

El reconocimiento lógico combinatorio no es solo una combinación de métodos y técnicas para la solución de problemas tradicionales de reconocimiento, sino que también es una aproximación metodológica para la solución de estos mismos problemas reales. Esto es, su tarea principal es iniciar con las condiciones del problema concreto a fin de obtener el modelo apropiado para su solución. En cualquier caso, no se puede partir del modelo matemático del problema general y adaptarlo al problema concreto. El modelo llega solo después de que el proceso de modelado adecuado ha sido totalmente estructurado [55].

El reconocimiento lógico combinatorio se originó en la antigua Unión Soviética [56]. La principal tarea de la publicación fue encontrar una solución para pronosticar recursos minerales. Para resolver este problema fue necesario tomar en cuenta, además de la medición geofísica de diferentes fenómenos, el tipo de suelo, la presencia o ausencia de fallas, y otras características que tienen un carácter cualitativo. En realidad, en dicha publicación, los autores consideran que todas las características son booleanas, y el problema se resolvió en una estructura cualitativa de manera muy simple. La aproximación fue transformar las características en una estructura cualitativa. Más tarde, otros trabajos consideraron características de

naturaleza diferente, en lugar de las primeras características booleanas solas o las cualitativas solas sin transformación [57 - 63]. Es importante resaltar que esta aproximación se originó debido a la necesidad práctica de modelar problemas reales en una forma más adecuada.

1.2.4. Redes neuronales: Este método supone que se tiene una estructura de neuronas interconectadas que se estimulan unas a otras, las cuales pueden ser "entrenadas" para dar una cierta respuesta cuando se le presentan determinados valores [53].

Una definición muy interesante sobre lo que es una red neuronal, con la intención de ser lo más completa posible pudiera ser:

Una red neuronal artificial es un procesador distribuido en paralelo de forma masiva que tiene una tendencia natural para almacenar conocimiento de forma experimental y lo hace disponible para su uso [64].

De acuerdo con la anterior definición, estas redes neuronales artificiales, formadas por elementos simples y con organización paralela, están interconectadas masivamente en paralelo, cuya intención es interactuar con los objetos del mundo real del mismo modo que lo hace el ser humano a través del sistema nervioso biológico.

Las redes neuronales artificiales son estructuras pensadas para simular las estructura y el comportamiento del sistema nervioso, por esta razón una neurona artificial posee entradas, salidas y un estado, además de estar conectadas a otras neuronas siendo las salidas de estas entradas de las demás [65].

Una red neuronal tiene una similitud con el cerebro humano en dos aspectos:

- Su conocimiento lo adquiere utilizando un proceso de entrenamiento para lograr su "aprendizaje".

- Esta información es almacenada en las interconexiones entre las neuronas de la red, esto es, en los denominados pesos sinápticos.

La aplicación de las redes neuronales en la tarea del reconocimiento de patrones consiste en establecer un número de categorías dentro de las cuales los patrones de entrada deben ser clasificados. Para esto se considera una fase de entrenamiento en la cual se presenta a la red los patrones que debe aprender y la categoría dentro de la cual están clasificados. Después de esta fase, se presenta un patrón nuevo y desconocido, pero que pertenece a alguna de las categorías aprendidas, y la red debe decidir a cuales de estas categorías se asemeja mejor. La ventaja del uso de redes neuronales estriba en la posibilidad de que se pueden separar regiones no lineales de decisión tan complicadas como se desee, dependiendo del número de neuronas de la red y de las capas. Por lo anterior, el reconocimiento de patrones con redes neuronales artificiales sirve para resolver problemas de clasificación cuya complejidad es alta [64].

Capítulo 2

Clasificadores neuronales

2.1. Introducción

En los sistemas de reconocimiento de patrones con redes neuronales, las más utilizadas son aquellas redes con múltiples capas que funcionan hacia delante. La red neuronal, o perceptron, de Frank Rosenblatt [66] es un ejemplo de red con múltiples capas. Este tipo de red está compuesta por una capa de entrada, una o más capas intermedias y una capa de salida, todas estas capas tienen un conjunto de neuronas que las constituyen. La información de entrada se propaga entonces hacia adelante desde la capa de entrada a través de la intermedia hasta la salida; este tipo de configuración se conoce como red neuronal multicapa, o "*MultiLayer Perceptrons*" (MLP) [67 - 69]. Este tipo de red es muy aplicada para resolver con éxito multitud de problemas en reconocimiento de patrones debido a que utiliza el algoritmo de retropropagación, o "*back propagation*" [64]. Básicamente, este proceso consiste en dos recorridos a través de las diferentes capas de la red, uno hacia adelante y uno hacia atrás. En el recorrido hacia adelante, se aplica en la capa de entrada un patrón, el cual propaga su efecto a través de las diferentes capas y como consecuencia produce un vector en la capa de salida. Durante este proceso, los pesos sinápticos entre las neuronas de la red son fijos y no se modifican. Durante el recorrido hacia atrás, en cambio, los pesos si se modifican de acuerdo con la regla de corrección del error, esta consiste en minimizar un error, comúnmente cuadrático, por medio de un gradiente descendente estocástico, por lo que la parte esencial del algoritmo es el cálculo de las derivadas parciales de dicho error con respecto a los parámetros de la red neuronal. La señal de salida real se compara con la señal deseada y como resultado se obtiene una señal de error, que se propaga en dirección contraria, es decir, hacia atrás, a través de la red modificando los pesos, de forma que, al volver a pasar el vector de entrada hacia adelante, la respuesta obtenida se asemeje más a la salida deseada. En concreto, una red neuronal multicapa:

1. Contiene una o más capas internas de neuronas no relacionadas directamente con la entrada y la salida. Estas neuronas "ocultas" capacitan a la red para "aprender" progresivamente cualquier correspondencia entre la entrada y la salida y almacenar internamente esta información.

2. Posee un gran número de conexiones, estas vienen determinadas por los pesos de la red. Un cambio en la conexión entre las neuronas equivale a un cambio en los pesos.

Estas características hacen que la habilidad para "aprender" de esta red, a partir del entrenamiento, sea muy potente [64].

El comportamiento de este tipo de redes multicapa hace que sea difícil conocer previamente la respuesta de la red. Esto debido a que el comportamiento no lineal de las neuronas, las cuales están muy interconectadas, hace difícil un análisis teórico de la red, y la existencia de neuronas "ocultas", que impide poder "ver" cómo se produce el "aprendizaje". Con lo que se pudiera determinar cuáles son las características que mejorarían dicho "aprendizaje" [64].

El desarrollo del algoritmo retropropagación, *back propagation*, proporciona un método eficiente para entrenar este tipo de redes. Aun cuando no es capaz de resolver todos los problemas, se ha demostrado que es uno de los mejores. La importancia está en su

capacidad de autoadaptar los pesos de las neuronas intermedias para aprender la relación que existe entre el conjunto de patrones de entrada y su correspondiente salida, y poder aplicar esa relación después del entrenamiento a nuevos patrones a la entrada con imperfecciones o con ruido. La red debe encontrar una representación interna que le permita generar las salidas deseadas durante la etapa de entrenamiento, y posteriormente, durante el funcionamiento, ser capaz de generar salidas para entradas que no le fueron mostradas durante el proceso de entrenamiento pero que se asemejan a alguna de las que si lo fueron [64].

2.2. El clasificador neuronal de área receptiva limitada: LIRA

Actualmente, las redes neuronales son muy utilizadas en las tareas de reconocimiento de imágenes [70 – 72]. De ellas se distinguen dos tipos: la primera usa un proceso de entrenamiento basado en gradientes, cuya regla consiste en varias capas de neuronas y todas ellas tienen características diferenciales [71 - 72]. En el proceso de entrenamiento, los pesos sinápticos de las conexiones entre neuronas se modifican. Para la segunda, también se tienen varias capas neuronales pero solo las capas finales tienen conexiones modificables [70, 73, 74]. Las capas de inicio contienen neuronas interconectadas con conexiones no modificables. Los pesos sinápticos de estas conexiones se determinan a partir de un proceso aleatorio. El primer grupo es bueno para el reconocimiento de propiedades generales en los objetos. Así, si se desea distinguir una "O" de una "L", esto se logra con buenos resultados. El segundo grupo es mejor para el reconocimiento de propiedades locales de los objetos. Para este caso, es bueno reconocer una "O" de una "Q". Debido a la necesidad de reconocimiento de propiedades locales en los objetos, se desarrolló una propuesta de reconocimiento denominada clasificador neuronal de área receptiva limitada LIRA (*LImited Receptive Area classifier*), perteneciente al segundo grupo de redes neuronales.

Este clasificador neuronal LIRA está fundamentado en los principios de la red neuronal presentada por Frank Rosenblatt, conocidos como el perceptron de Rosenblatt [66]. El clasificador LIRA tiene dos variantes: LIRA_binary y LIRA_grayscale. El primero utilizado para reconocer imágenes binarias, es decir, en blanco y negro, y el segundo para el reconocimiento de imágenes en escala de grises [75].

2.3. Estructura del clasificador neuronal LIRA

El clasificador LIRA, como el perceptron de Rosenblatt, contiene tres capas de neuronas. La primera de ellas, capa *S*, corresponde a lo que se denomina retina, debido a que es la forma en la cual se entran las imágenes al clasificador. La segunda capa, capa *A*, corresponde a la extracción de características de la imagen entrada, también se le denomina capa asociativa. La tercera capa, capa *R*, es la de salida. Cada neurona de esta capa corresponde a una de las clases de salida. Para la tarea de reconocimiento de dígitos manuscritos, una de las pruebas para verificar su efectividad, esta capa contenía 10 neuronas con correspondencia a los dígitos 0,…, 9. Las conexiones entre las capas *S* y *A* se establecieron a través de un proceso aleatorio y no pueden ser cambiadas durante el entrenamiento del perceptron. Estas tienen pesos de -1 o 1.

Cada neurona de la capa *A* tiene conexiones con todas las neuronas de la capa *R*. En principio, los pesos de las conexiones están puestos a 0, y se modifican durante el proceso de entrenamiento. La regla de modificación corresponde al algoritmo de entrenamiento, por ejemplo, de Hebb.

Para el desarrollo del clasificador LIRA se realizaron algunas alteraciones a la estructura del preceptron, así como a los algoritmos de entrenamiento y reconocimiento. También se modificó el procedimiento aleatorio del arreglo de las conexiones entre *S* y *A*, y la regla con la cual se selecciona a la neurona ganadora en la capa de salida *R*, además de la adaptación para el reconocimiento de imágenes en escala de grises. El clasificador LIRA fue probado en dos aplicaciones: reconocimiento de dígitos manuscritos, lo que puede ser útil en cheques bancarios donde se requiere el reconocimiento de cifras escritas [75], y en microensamble.

2.3.1. El algoritmo de trabajo del clasificador neuronal LIRA

Este algoritmo de trabajo consiste en las etapas que muestra la Figura 2.1.

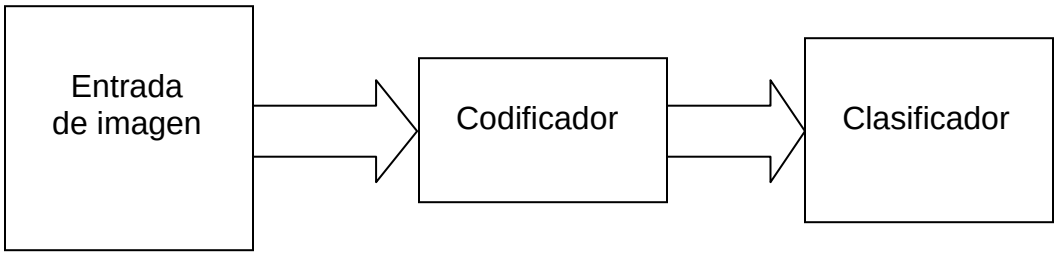

Figura 2.1. Diagrama a bloques del clasificador LIRA

A partir de dicha figura se analizan cada uno de los bloques de este algoritmo.

2.3.1.1. Proceso de entrada de imágenes al clasificador

La forma en que son entradas las imágenes al clasificador consiste en suponer que la imagen tiene la forma de un rectángulo cuyas dimensiones son $H \times W$, Figura 2.2.

Este concepto se asume partiendo de una analogía con la retina del ojo humano. Con base en ello, *S* corresponde a la imagen y es la capa de entrada del sistema. Esta capa, también llamada retina, cuenta con un conjunto de neuronas que son conectadas a la capa *A*, asociativa. Cada pixel de la imagen entrada corresponde a una neurona en la capa *S*, la cual tiene conexiones aleatorias con la capa *A*. Es necesario enumerar todas estas neuronas en *S*. El total de ellas es $N = H \times W$. Para determinar las conexiones de las neuronas de la ventana $h \times w$ con la capa *A* se hace una selección aleatoria de entre 1 y *M*, (donde *M* es el número de neuronas de la ventana $h \times w$).

Para la realización de experimentos se elige un número de conexiones aleatorias entre la ventana de la capa *S* y las neuronas de la capa *A*. En la Figura 2.2 se presentan 4 conexiones aleatorias a cada neurona de la capa *A*. Entre estas 4 conexiones hay dos conexiones de excitación (mostradas con flechas) y dos conexiones de inhibición (mostradas con círculos).

Las conexiones de excitación aumentan la activación de la neurona correspondiente, y las de inhibición disminuyen la activación de cada neurona a la cual están conectadas.

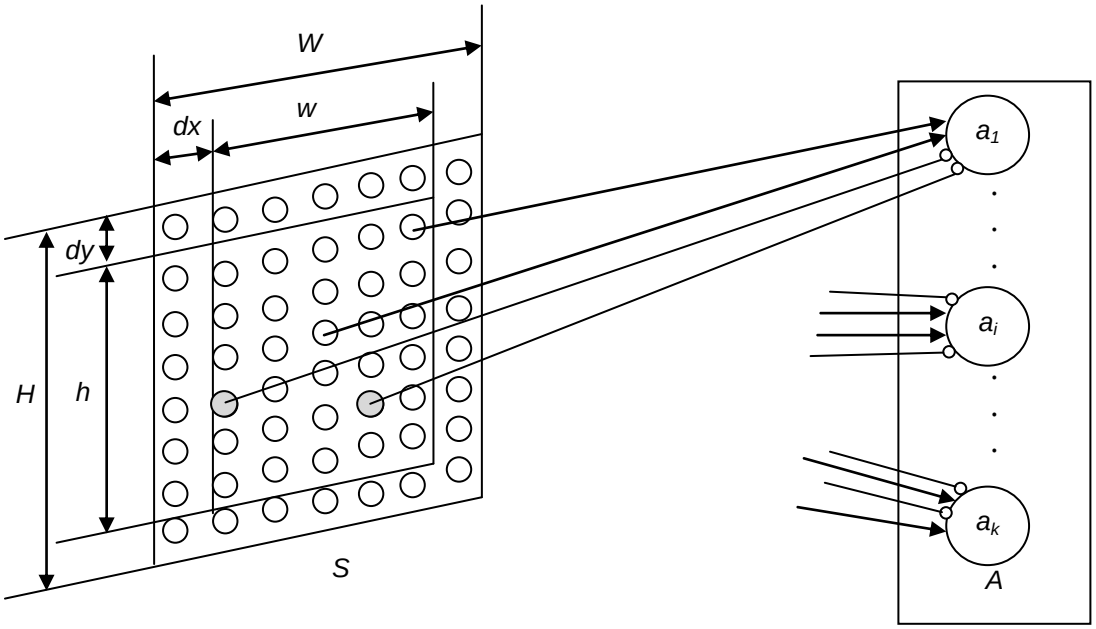

Figura 2.2. Conexiones entre la capa S, o retina, y la capa A

Las neuronas con los números de la ventana $w \times h$ en la capa *S* se conectarán con una neurona de la capa *A*. Este criterio se usa para generar todas las conexiones entre las neuronas de *S* y *A*. Particularmente, como ejemplo, todas las ventanas pueden tener el mismo tamaño ($h = w = 20$) y pueden estar sobrepuestas, Figura 2.3; forma con el cual se trabaja. Puede aplicarse un criterio distinto en cuanto al tamaño y la forma de las ventanas generadas en la retina, es decir, *h* y *w* pueden variar y diferir en sus tamaños, Figura 2.4.

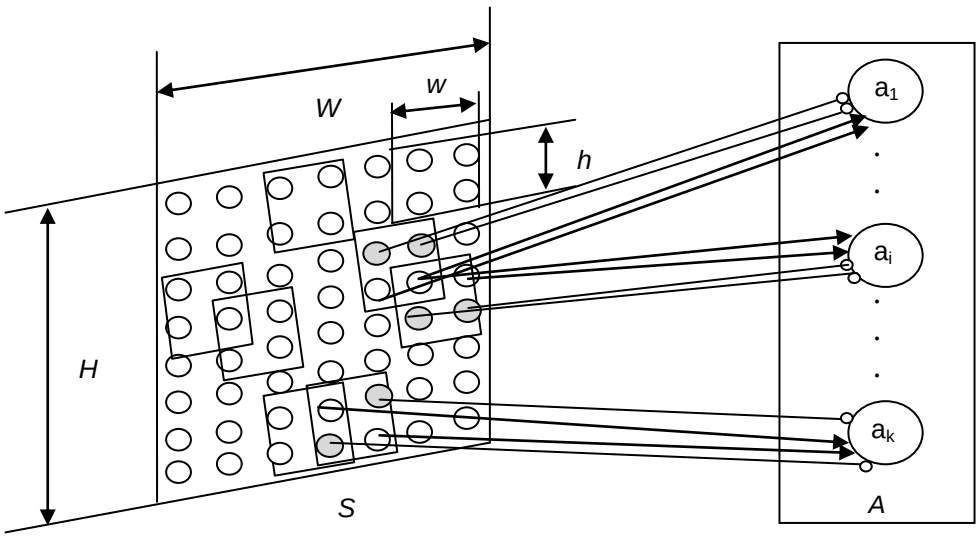

Figura 2.3. Conexión entre ventanas cuadradas (h = w = 20) de la capa S y la capa asociativa A

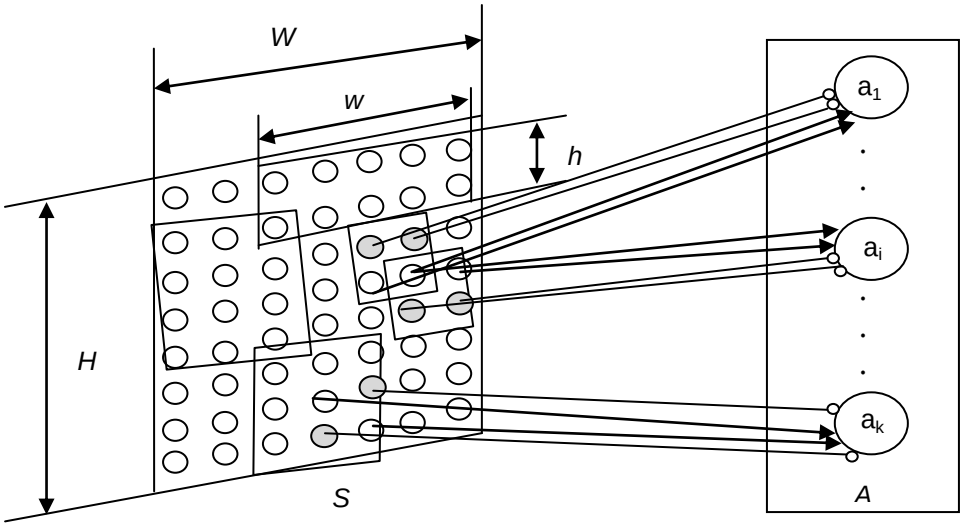

Figura 2.4. Conexión entre ventanas de diferentes tamaños (h ≠ w) de la capa S y la capa A

Para el caso del clasificador a utilizar, LIRA, las conexiones aleatorias entre *S* y *A* no se hacen con todas las neuronas de la capa *S*, sino a partir del rectángulo ($h \times w$), generado en *S*. De la Figura 2.2, las distancias *dx* y *dy* son números aleatorios seleccionados a partir de: *dx* de [0, $W - w$] y *dy* de [0, $H - h$], de donde *W* y *H* son el ancho y el alto de la capa *S*.

2.3.1.2. Proceso de codificación

Analizando ahora el segundo bloque de la Figura 2.1, se tiene el diagrama de flujo de la Figura 2.5, este muestra el proceso para codificar las imágenes hacia el clasificador. El primer paso consiste en generar una máscara. El siguiente es ingresar la imagen a codificar, representada como *x* en el diagrama. *N* es el número total de imágenes que son ingresadas a dicho clasificador. A partir de tal imagen se aplica la máscara, obteniendo las neuronas que están activas en este proceso. La máscara es un conjunto de conexiones excitatorias e inhibitorias entre la retina (capa *S*) y las neuronas del grupo *A*, Figura 2.2. Esta mascara se genera por un proceso aleatorio que selecciona dichas conexiones. Se elige aleatoriamente la ventana en la capa *S*, lo cual se logra a través de las siguientes expresiones:

$$dx_{ij} = random(W - w), \qquad (2.1)$$
$$dy_{ij} = random(H - h),$$

donde dx_{ij} y dx_{ij} es la posición de la ventana (punto superior izquierdo) en la capa *S*, *random(z)* es un número aleatorio de entre 0 y *z*. Posteriormente se eligen aleatoriamente las neuronas con conexiones excitatorias e inhibitorias. Cada posición de neurona dentro del rectángulo con conexiones excitatorias o inhibitorias se define por las expresiones:

$$x_{ij} = random_{ij}(w), \qquad (2.2)$$
$$y_{ij} = random_{ij}(h),$$

donde *i, j* es la posición de las neuronas con conexiones excitatorias o inhibitorias en la ventana.

Las coordenadas absolutas de las neuronas con conexiones excitatorias o inhibitorias en la retina se definen por:

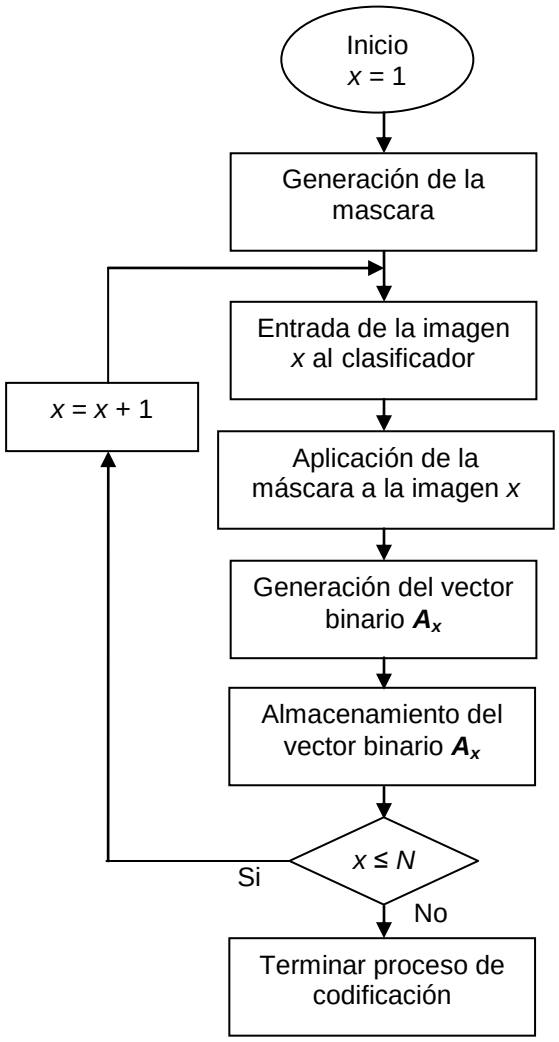

Figura 2.5. Diagrama de flujo del proceso de codificación

$$X_{ij} = x_{ij} + dx_{ij}, \qquad Y_{ij} = y_{ij} + dy_{ij}. \tag{2.3}$$

El proceso de codificación se realiza a partir de la máscara aplicada a la imagen entrada. Cada imagen de entrada define una actividad diferente de las neuronas en la capa *A*. El vector binario que corresponde a las actividades neuronales asociativas es denominado como código binario de la imagen $\boldsymbol{A} = a_1,\ldots, a_k$, donde *k* es el numero de neuronas en la capa *A*. El procedimiento que transforma la imagen de entrada en el vector binario se denomina codificación de la imagen.

Para este sistema particular la *i*-esima neurona de la capa *A* está activa solo si todas las conexiones excitatorias con la retina corresponden al objeto y todas las inhibitorias corresponden al fondo. En este caso, $a_i = 1$, de lo contrario, $a_i = 0$. Por el hecho de trabajar

con estos sistemas se tiene en conocimiento que el número de neuronas activas *m* en esta capa *A* es muchas veces menor que el número total de neuronas *k*. De acuerdo con datos neurofisiológicos, el número de neuronas activas en la corteza cerebral es cientos de veces menor que el número total de ellas. Por ello, la expresión $m = c\sqrt{n}$ es utilizada para hacer una relación con este hecho. Donde *c* es una constante cuyo valor está entre 1 y 5.

En experimentos realizados se utilizaron un total de neuronas *k* de entre 32000 hasta 512000. Por lo cual, los vectores tienen este tamaño. Si se trabajara con estos vectores, y se deseara multiplicarlos con esas magnitudes se necesitaría una gran capacidad de memoria para almacenar los resultados obtenidos. Por lo anterior, es conveniente entonces representar al vector binario *A* como una lista con los números de las neuronas activas, con lo que se reduce la demanda de espacio en memoria.

Con estos datos se obtienen los códigos de las imágenes, los cuales se guardan en forma compacta facilitando el cálculo de actividad de las neuronas en la capa de salida. Cada imagen tiene una lista propia que muestra las posiciones de las neuronas activas.

El proceso termina cuando se llega al número máximo de imágenes *N*, es decir, ya no existen más imágenes a codificar, por lo cual se continúa con la tarea de entrenamiento.

2.3.1.3. Clasificador

El clasificador es el tercer bloque a analizar de la Figura 2.1. En la Figura 2.6 se muestra este clasificador que incluye la interacción entre las capas *A* y *R* a través de las conexiones *W*.

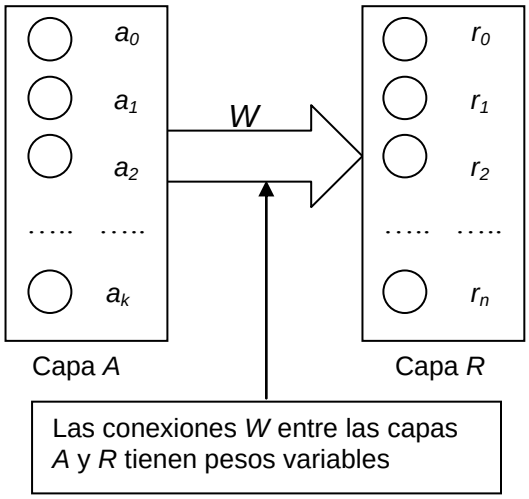

Figura 2.6. Interacción entre las capas A y R en el proceso de entrenamiento

La regla de conexión entre las capas *A* y *R* es de la siguiente manera:

Todas las neuronas de la capa *A* son conectadas con todas las neuronas de la capa *R*. Las conexiones generadas *W* forman una matriz de forma $k \times n$, donde *k* es el número de neuronas de la capa *A* y *n* es el número de neuronas de la capa *R*.

Antes de iniciar el proceso de entrenamiento, todos los pesos de las conexiones entre las neuronas de la capa *A* y la *R* están puestos a cero.

1. El proceso de entrenamiento empieza con la lectura desde la memoria del código de la primera imagen hacia el clasificador. Se calculan las excitaciones E_i de las neuronas de la capa *R*. Este valor E_i se obtiene de:

$$E_i = \sum_{j=1}^{k} a_j w_{ji} \tag{2.4}$$

   donde E_i es la excitación de la *i*-esima neurona de la capa *R*; a_j es la excitación de la *j*-esima neurona en la capa *A*; w_{ji} es el peso de la conexión entre dichas neuronas de las capas mencionadas.

2. La neurona con una excitación E_{max} es considerada como la ganadora.

3. El número de la neurona ganadora se denota con i_w, y el numero de la neurona que corresponde a la clase a la cual pertenece la imagen de entrada con i_c. Si $i_w = i_c$, nada ocurre, de lo contrario, si $i_w \neq i_c$

$$\begin{aligned} &(\forall j)(w_{ji_c}(t+1) = w_{ji_c}(t) + a_j) \\ &(\forall j)(w_{ji_w}(t+1) = w_{ji_w}(t) - a_j) \\ &\text{Si } (w_{ji_w}(t+1) < 0) \Rightarrow w_{ji_w}(t+1) = 0, \\ &\text{Si } (w_{ji_c}(t+1) > w_{max}) \Rightarrow w_{ji_c}(t+1) = w_{max}, \end{aligned} \tag{2.5}$$

   donde $w_{ji}(t)$ es el peso de la conexión entre la *j*-esima neurona de la capa *A* y la *i*-esima neurona de la capa *R* antes de ser modificado, es decir, en un tiempo *t* presente. $w_{ji}(t+1)$ es el peso ya modificado, en un tiempo consecuente $t+1$. w_{max} es el máximo peso que se puede establecer. Esta es la regla de entrenamiento de Hebb.

En la Figura 2.7 se observa el diagrama de bloques para la actividad de entrenamiento del clasificador. El valor *c* implica en número de ciclos de entrenamiento que debe realizar el clasificador, *y* indica el número de la imagen entrada, *N* es el número total de imágenes que son entradas, y *e* es el número de errores calculados en el proceso de entrenamiento.

El proceso de entrenamiento es iterativo, después de que todas las imágenes de entrenamiento *N* han sido entradas se calcula el número de errores *e* en este proceso. Si este es mayor a un porcentaje *p* predefinido, se realiza un nuevo proceso de entrenamiento. Si no es así, el entrenamiento termina; normalmente se requiere de un porcentaje del 0%.

El proceso de entrenamiento también puede ser detenido al ser fijada una cierta cantidad de ciclos de entrenamiento *c*, como se observa en el diagrama de la Figura 2.7.

2.3.1.4. Proceso de reconocimiento del clasificador

En el proceso de reconocimiento de imágenes, las capas *A* y *R* también tienen una interacción directa, como lo muestra la Figura 2.6, pero no hay cambio en los pesos de las conexiones entre estas capas. El cálculo de la actividad neuronal se hace con

$$E_i = \sum_{k=1}^{n} a_k w_{ki} \,, \tag{2.6}$$

donde E_i es la excitación de la *i*-esima neurona en la capa *R*; a_k es la excitación de la *k*-esima neurona de *A*; w_{ki} es el peso de la conexión entre la *k*-esima neurona de la capa *A* y la *i*-esima neurona de la capa *R*. Con la máxima excitación de la neurona en la capa *R* se obtiene el resultado del reconocimiento seleccionando la neurona ganadora.

En la Figura 2.8 se observa el diagrama a bloques del proceso de reconocimiento, *y* implica el número de la imagen a reconocer y *N* el número total de imágenes en el proceso, i_w denota al número de la neurona ganadora, e i_c al número de la neurona que corresponde a la clase a la cual pertenece la imagen de entrada. Como primera acción, se realiza una lectura del código de la imagen, se calcula la excitación, y posteriormente se elige la excitación máxima

$$i_w = E_{max} = max\,(E_i) \tag{2.7}$$

si $i_w \neq i_c$ se hace un cálculo de errores.

También se observa en el diagrama a *e* que denota el conteo del numero de errores.

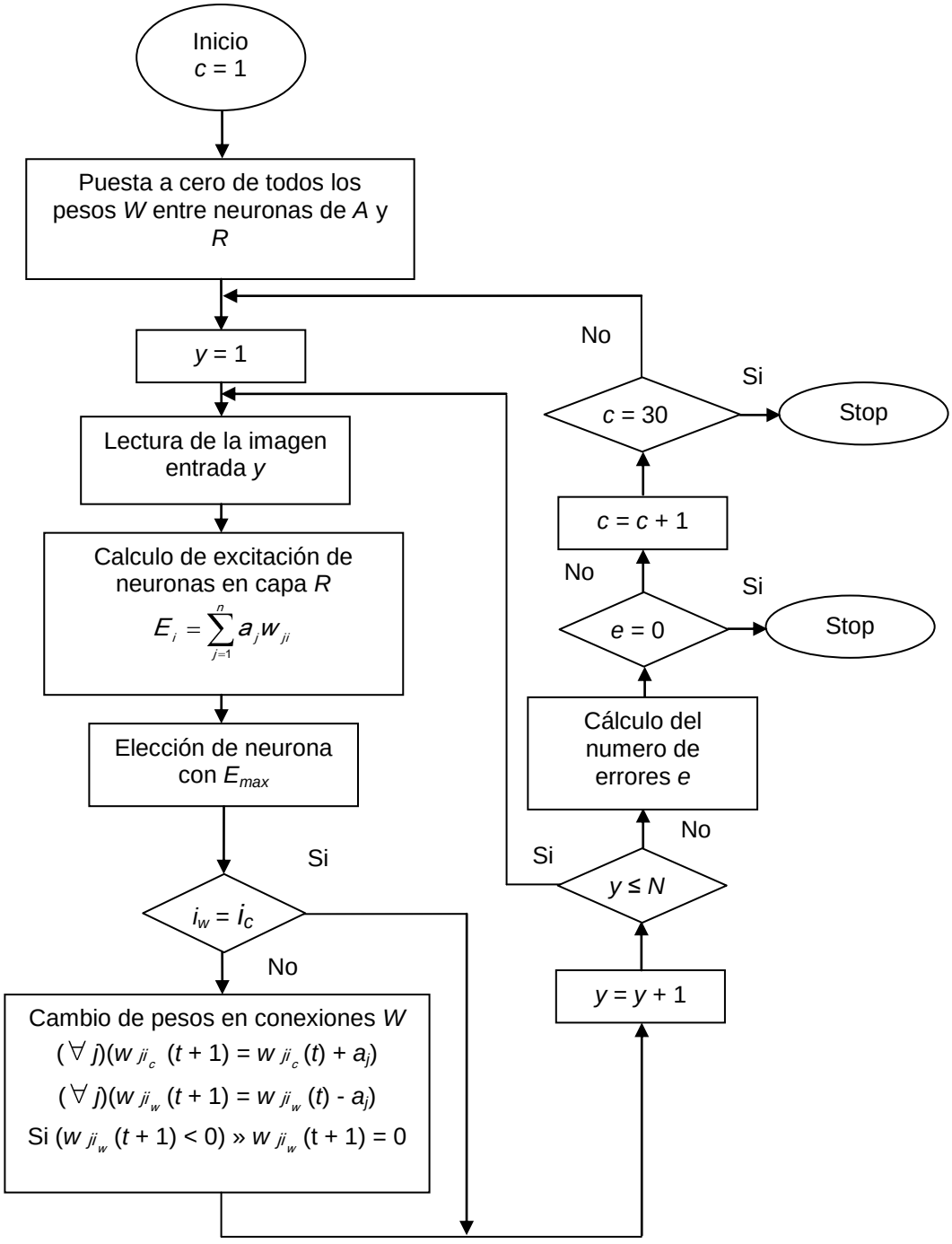

Figura 2.7. Diagrama de bloques del proceso de entrenamiento del clasificador

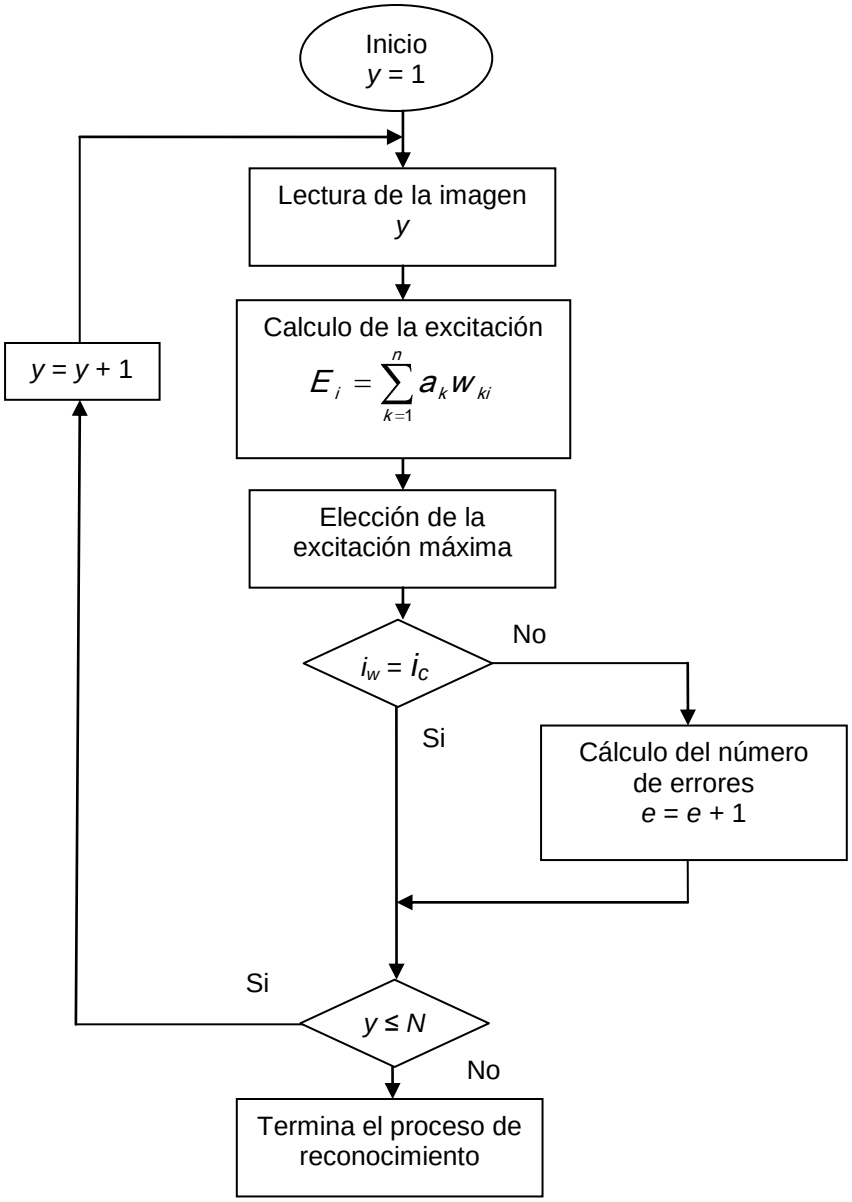

Figura 2.8. Diagrama de bloques para el proceso de reconocimiento

Capítulo 3

Dispositivos lógicos programables

3.1. Introducción

Debido a que la alta integración en dispositivos electrónicos cada día es mucho mayor, es que los dispositivos lógicos programables han acaparado la atención de los ingenieros desarrolladores [28 – 35, 76]. El numero de compuertas en un FPGA (*Field Programmable Gate Array*) es normalmente de 500 000, aunque esto se duplicará en poco tiempo, y su costo será cada día menor. Ambos aspectos llaman la atención de quienes se dedican a diseñar sistemas, puesto que el costo se reduce y la capacidad de operación aumenta, lo cual lleva a reconsiderar la relación entre procesadores y software, basando los diseños en aplicaciones de solo hardware.

Considerando esta opción, los diseñadores describen el comportamiento lógico de la aplicación en términos de alto nivel, es decir, a través de diferentes lenguajes de programación generan el comportamiento del sistema a desarrollar, y el resultado compilado es el que se graba o "baja" a la pieza de silicio. Debido a esta posibilidad de diseñar sistemas, existen diferentes tipos de lógica programable disponible. Esta oferta incluye todo lo necesario, desde los pequeños dispositivos con la capacidad de implementar ecuaciones lógicas hasta los poderosos FPGAs que pueden emular un procesador completo con todos sus elementos periféricos necesarios [36 - 38, 77].

3.1.1. Dispositivos lógicos programables complejos. CPLDs.

En un principio, el dispositivo muy utilizado con esta metodología era el PLD (*Programmable Logic Device*). Este dispositivo lógico programable permitía implementar en hardware diseños de lógica digital con una total flexibilidad [76]. Con ello, se podía evitar el uso de partes TTL (*Transistor-Transistor Logic*) sencillas, tales como ANDs, ORs o NOTs, y en su lugar utilizar un solo dispositivo PLD. El PLA (*Programmable Logic Array*) el PAL (*Programmable Array Logic*) y el GAL (*Generic Array Logic*) son dispositivos lógicos programables que pueden ser aplicados a diseños digitales. Utilizar alguno de estos dispositivos tiene como ventaja la reducción del espacio a utilizar, ya que íntegramente contiene todo el dispositivo lógico a manejar, además de que si la lógica requiere de algún cambio, este será internamente, mientras que el área designada para el dispositivo no tendrá alteración alguna. La estructura interna de un PLD está constituida por macroceldas interconectadas, las cuales están compuestas por elementos lógicos combinatorios, tales como compuertas OR y AND, y de un flip-flop. Esto implica que una ecuación lógica booleana pequeña puede ser implementada en una macrocelda. Dicha ecuación puede combinar algunas entradas binarias y generar un estado en una salida, la cual, si es necesario, puede mantenerse en el flip-flop hasta que sea aplicado un pulso de reloj. Según el fabricante, cada dispositivo tendrá un cierto número de macroceldas para las aplicaciones necesarias.

Dado que estos dispositivos son pequeños, es probable, y necesario, utilizar dispositivos con mayores capacidades, los cuales tiene una estructura similar a aquellos PLDs, por lo cual el método de implementación es el mismo [78]. Gracias al incremento en la densidad de los circuitos integrados, los productores de estos dispositivos presentaron sus productos en un mayor empaque al cual se le denominó CPLD (*Complex Programmable Logic Device*). Este dispositivo lógico programable complejo implica una combinación de varios

PLDs interconectados en un solo circuito integrado, quizás de mayor tamaño, que permite implementar más ecuaciones lógicas y más diseños complejos. Con esto, se pueden reemplazar docenas de dispositivos TTL sencillos con uno solo dispositivo de estos.

Debido a que los CPLDs pueden aceptar diseños más grandes que los sencillos PLDs, estos son muy utilizados para lógica de control de alto rendimiento o para maquinas de estados finitas complejas. Estos dispositivos tienen retardos más predecibles y normalmente más cortos [78].

3.1.2. Matrices de compuertas programables por campo. FPGAs.

Otro dispositivo programable es el FPGA (*Field Programmable Gate Array*) el cual puede aceptar cualquier diseño de hardware descrito para un CPLD. Normalmente se utiliza para el diseño de prototipos, cuyo diseño final estará implantado en un ASIC (*Application-Specific Integrated Circuit*), este dispositivo de propósito específico es el que será producido en masa para su aplicación final. La estructura interna de un FPGA es totalmente distinta a la de un CPLD y PLD, ya que este dispositivo puede tener bloques lógicos más simples y pequeños con respecto a las macroceldas del PLD, a lo que se le conoce como arquitectura de grano fino, o más grande y compleja. Sin embargo, no son nunca tan grandes como un PLD, o como los bloques lógicos de un CPLD. Los bloques lógicos del FPGA son un par de compuertas lógicas o una tabla de búsqueda y un flip-flop. Debido a los flip-flops excedentes, la arquitectura de un FPGA es mucho más flexible que la de un CPLD, lo que lo hace mejor en aplicaciones de interconexión y de registro. También son muy utilizados en lugar de procesadores, sobre todo cuando el procesamiento de paquetes de datos de entrada debe ser realizado a un ritmo muy rápido [39 – 43, 79]. Además de ello, los FPGAs son más densos, es decir, mas compuertas dentro de una área dada, y de costo menor que los CPLDs, por lo cual son la primera opción para diseños lógicos mayores.

3.1.3. Lenguajes de descripción de hardware

El proceso para crear la lógica digital se realiza a partir de la descripción de la estructura del hardware y del comportamiento, todo esto escrito en un lenguaje de descripción de hardware de alto nivel. Este código es compilado y grabado en el dispositivo programable. También se puede crear una lógica utilizando un esquema grafico del comportamiento deseado, es decir, el circuito lógico desarrollado que se pretende implementar, aunque esta práctica se ha hecho menos popular debido a que los diseños son más complejos y las herramientas basadas en lenguaje han sido mejoradas [76, 80].

Los desarrolladores de software tienden a describir los procesos de manera secuencial, incluso cuando se involucran aplicaciones multiproceso, por ello, el código fuente escrito es ejecutado en ese orden. Si hay un sistema operativo este se usa para crear la apariencia de paralelismo, pero hay solo un motor de ejecución. Durante la entrada del diseño, los diseñadores deben pensar y programar en paralelo. Todas las señales de entrada son procesadas en paralelo, así como se moverían a través de un conjunto de motores de ejecución, cada uno es una serie de macroceldas e interconexiones, hacia sus señales de salida destinadas. Además, las sentencias del lenguaje de descripción de hardware crean

estructuras, las cuales todas son ejecutadas al mismo tiempo. La transferencia entre macrocelda y macrocelda está sincronizada con otra señal, como lo puede ser un reloj [80].

El proceso de diseño es seguido por una simulación, el simulador ejecuta el diseño y confirma que las salidas correctas se producen a partir de las entradas propuestas. Con esto, el diseñador puede asegurarse que la lógica desarrollada funciona correctamente antes de continuar con el desarrollo.

La compilación solo se puede hacer cuando hay una representación correcta del hardware descrito, este proceso viene acompañado de una síntesis del diseño el cual es una representación previa de lo que posteriormente se grabará en el dispositivo programable. La grabación se realiza después que se ha definido el lugar y la ruta del proceso, es decir, las estructuras lógicas descritas en la representación previa son ubicadas en las macroceldas, así como las interconexiones, entradas y salidas definidas. Todo esto implica una designación de pines del dispositivo CPLD o FPGA a utilizar, esto es, el "circuito impreso" resultante es distribuido dentro del dispositivo, a fin del que el circuito integrado ejecute el diseño del hardware desarrollado.

Con lo anterior, solo resta probar el circuito integrado con datos de entrada reales, a fin de que el diseño interno genere las salidas deseadas, de ser así, el dispositivo está listo para ser implantado en el sistema de control de proceso correspondiente.

3.2. Tarjetas de Desarrollo de Altera

Altera Corporation es una compañía, fundada en 1983 y establecida en San José, California, pionera en soluciones de lógica programable. Su experiencia en este campo permite a las empresas, dedicadas a la aplicación de semiconductores en sistemas de proceso, a innovar, trascender y ganar dentro del mercado en que estas se encuentren, de forma rápida y efectiva. Altera ofrece dispositivos programables, tales como FPGAs, CPLDs y ASICs, además de soporte técnico, herramientas de software y desarrollos propios, para proveer al consumidor de soluciones de alto nivel dentro de su mercado de acción [81].

A la par de la evolución en cuanto a diseño electrónico se refiere, Altera sigue otorgando soluciones rápidas a los consumidores, además de ventajas sobre costos, desarrollos y otras necesidades, ya que abarca un mercado muy amplio, el cual surgió con los primeros productos lógicos programables, que siguen evolucionando.

Gracias a la amplia gama de productos que Altera ofrece, es que se considera como una de las mejores opciones para la solución de problemas, desde el punto de vista de la lógica programable, toda vez que proporciona los componentes electrónicos necesarios para la aplicación, además de la paquetería, o software, que permite la programación de dichos dispositivos, incluso la opción de tarjetas de desarrollo para hacer pruebas previas con los primeros desarrollos, lo cual implica una ventaja importante antes del proceso en serie del producto final. Otras ventajas son las hojas de especificaciones de cada uno de los dispositivos que produce, y los manuales de operación de dichas tarjetas de desarrollo. Incluso, a través de su página de internet se pueden obtener los diferentes programas de aplicación y dichas hojas de datos. Esto hace de Altera una buena opción para la solución de problemas con dispositivos lógicos programables como herramienta de solución.

3.2.1. MAX + Plus II de Altera

Altera, en su amplio rango de productos, ofrece la paquetería de aplicación para la programación de dispositivos lógicos programables. Esta paquetería, denominada MAX + Plus II, Figura 3.1, abarca todos los aspectos necesarios para lograr el objetivo antes citado.

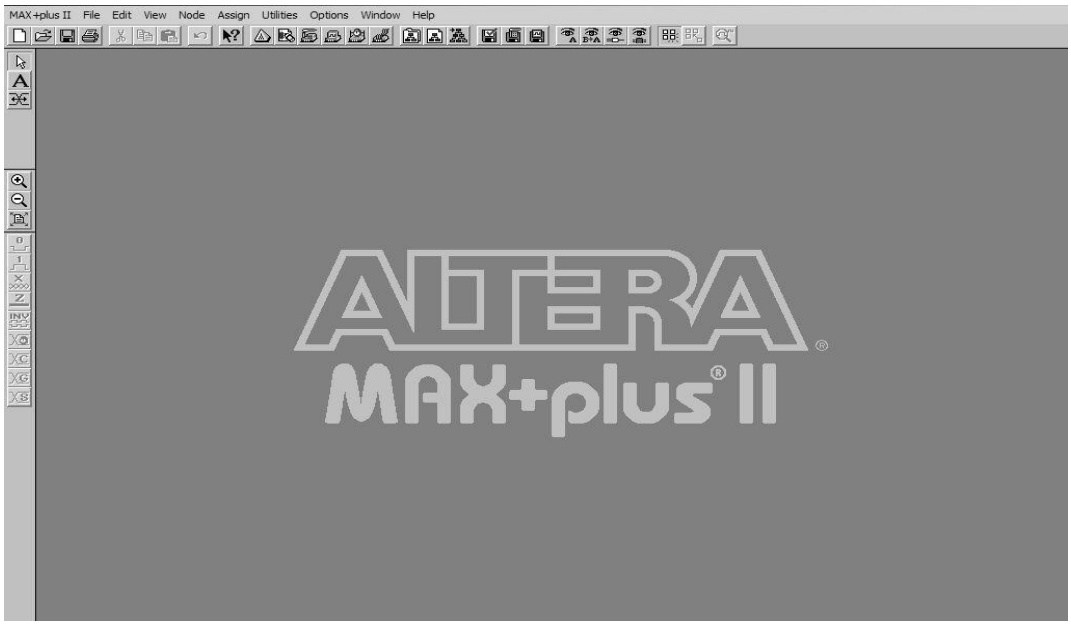

Figura 3.1. MAX + plus II. Paquetería de aplicación de Altera Corporation

A través de las diferentes opciones que brinda esta aplicación se puede generar el diseño que describa el comportamiento que se pretende implantar en el dispositivo a utilizar. La Figura 3.2 muestra las formas bajo las cuales se puede "crear" o describir el diseño a implementar.

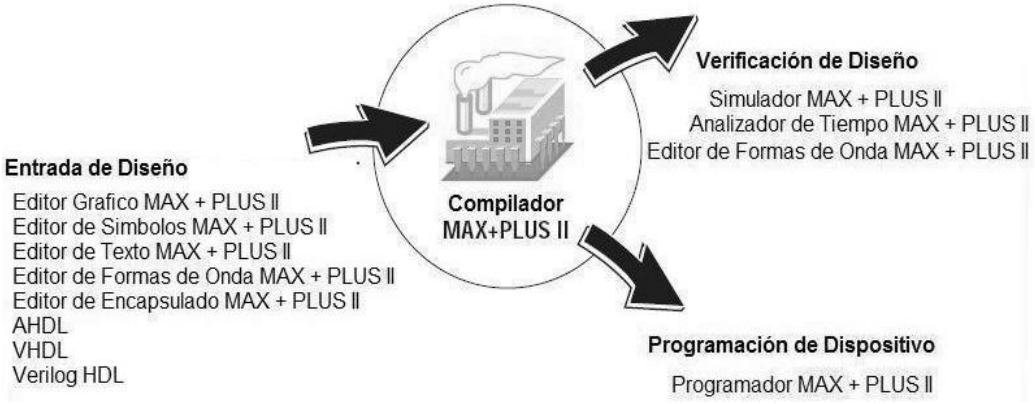

Figura 3.2. Herramientas proporcionadas por el ambiente MAX + plus II de Altera Corporation

La opción más simple y sencilla de usar es el manejador o editor de esquemáticos (*Graphic Editor*). Esta permite estructurar, como si se tratara de un circuito físico, el diseño resultante de la aplicación en cuanto a lógica booleana se refiere, de tal manera que se

pueda verificar el comportamiento definido para esa aplicación particular. En la Figura 3.3 se muestra un ejemplo de aplicación del editor de esquemáticos de MAX + plus II de Altera, se trata del circuito lógico que describe la estructura interna de un flip-flop tipo D.

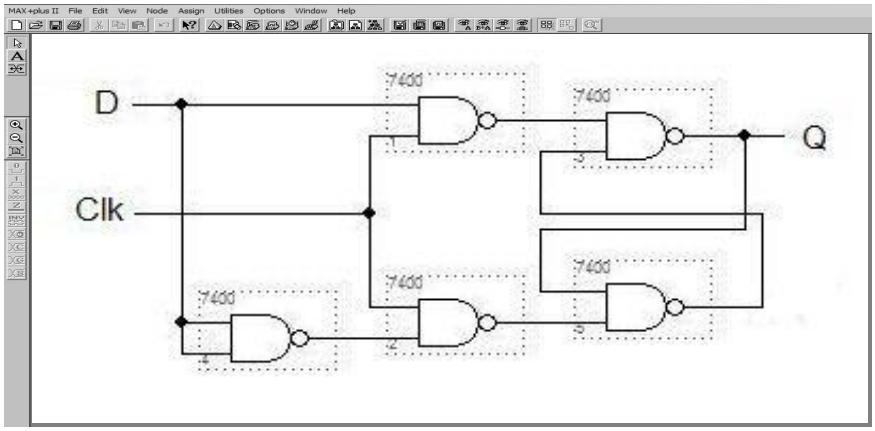

Figura 3.3. Circuito lógico de un flip-flop tipo D

En principio, el programa cuenta con las librerías relacionadas a los dispositivos digitales sencillos, esto es, cuenta con la mayoría de los elementos que describe el manual TTL de dispositivos digitales. También, cuenta con elementos digitales no descritos en el mencionado manual, pero que operan de la misma manera, es decir, existen dispositivos que, con pequeñas modificaciones, operan como los que si se encuentra en venta, Figura 3.4.

En la Figura 3.4 se aprecia el archivo donde se encuentran una gran cantidad de símbolos nombrados de acuerdo con su comportamiento. Así, por ejemplo, al ser seleccionada la componente "and2", implica invocar hacia el editor de esquemáticos una compuerta AND de dos entradas. Esta selección fue hecha en función del comportamiento del dispositivo que se quiere utilizar.

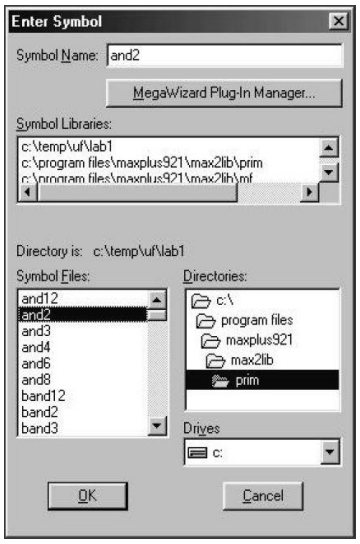

Figura 3.4. Perspectiva del manejador de símbolos en función de su comportamiento

Con las librerías que contienen a los dispositivos digitales, tanto la que los administra en función de su nominativo o comportamiento como la que lo hace de acuerdo con su número de encapsulado o de componente, como dos de las principales ventajas que brinda el manejador de símbolos de la aplicación Max plus II para la estructuración de circuitos lógicos, aunque no por ello son las únicas para la opción esquemática del editor grafico, se pueden "alambrar" toda una variedad de circuitos digitales en el editor de esquemáticos del paquete en cuestión. En la Figura 3.5 se muestra otro ejemplo de aplicación del editor grafico de MAX + plus II de Altera, se trata de un circuito lógico, con una mayor cantidad de compuertas sencillas, que describe el comportamiento del diseño de un sumador de dos números A y B, ambos de 8 bits.

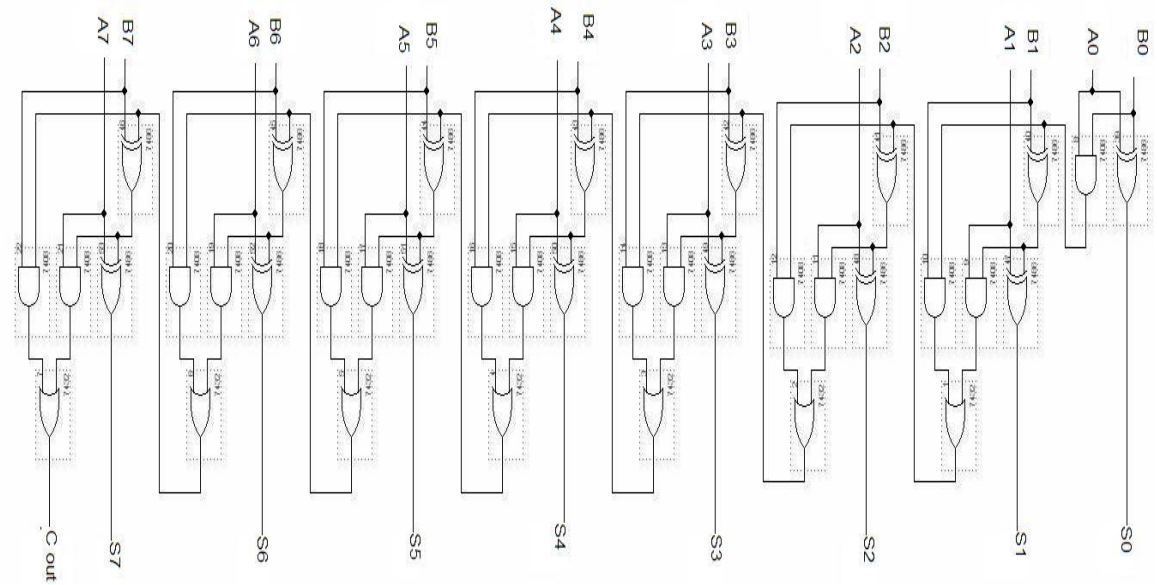

Figura 3.5. Circuito lógico de un sumador de dos números, A y B, de ocho bits cada uno

Después de realizar el circuito completo, el siguiente paso consiste en la compilación de dicho esquemático de tal manera que sea revisado en toda su estructura y, de ser necesario, la mención de los posibles errores u omisiones al momento de ser construido. En la Figura 3.6 se observan las diferentes etapas que debe satisfacer el diseño esquemático realizado. El compilador se encarga de verificar y validar cada una de esas etapas. El proceso de compilación termina cuando se ha cubierto cada uno de los requerimientos que se muestran en la imagen.

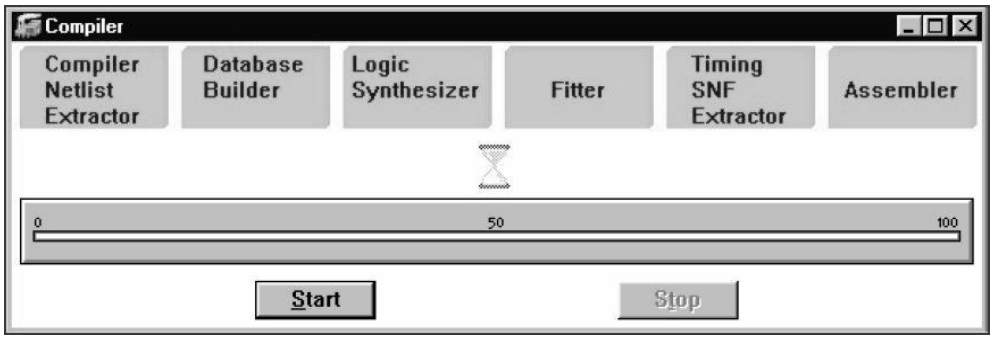

Figura 3.6. Etapas del proceso de compilación con el MAX + plus II Compiler

Si se han encontrado errores en la estructura del esquemático se generan los correspondientes mensajes de error o de advertencia para hacer las correcciones necesarias. Si no existen errores, se muestra entonces un mensaje de aprobación para la simulación o para la programación en el dispositivo seleccionado.

La última compilación exitosa genera un archivo el cual permite la simulación del circuito a partir del ingreso de las señales involucradas en el sistema, además de las de control y de temporización, en la Figura 3.7 se observa cómo se invoca al editor de formas de onda.

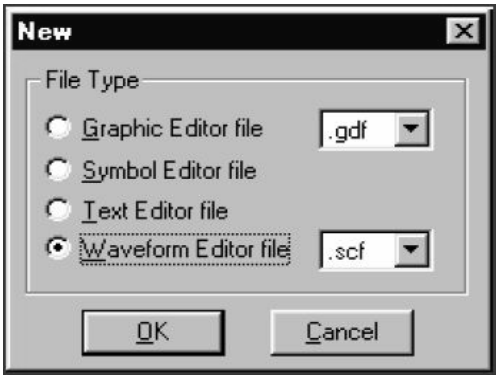

Figura 3.7. Selección del editor de formas de onda

En la Figura 3.8 se muestra un ejemplo de aplicación del editor de formas de onda del MAX + plus II de Altera, se observa el comportamiento de un circuito esquemático desarrollado. A partir de varias combinaciones en las entradas del circuito, se generan las salidas correspondientes que describen la función del esquema del diseño.

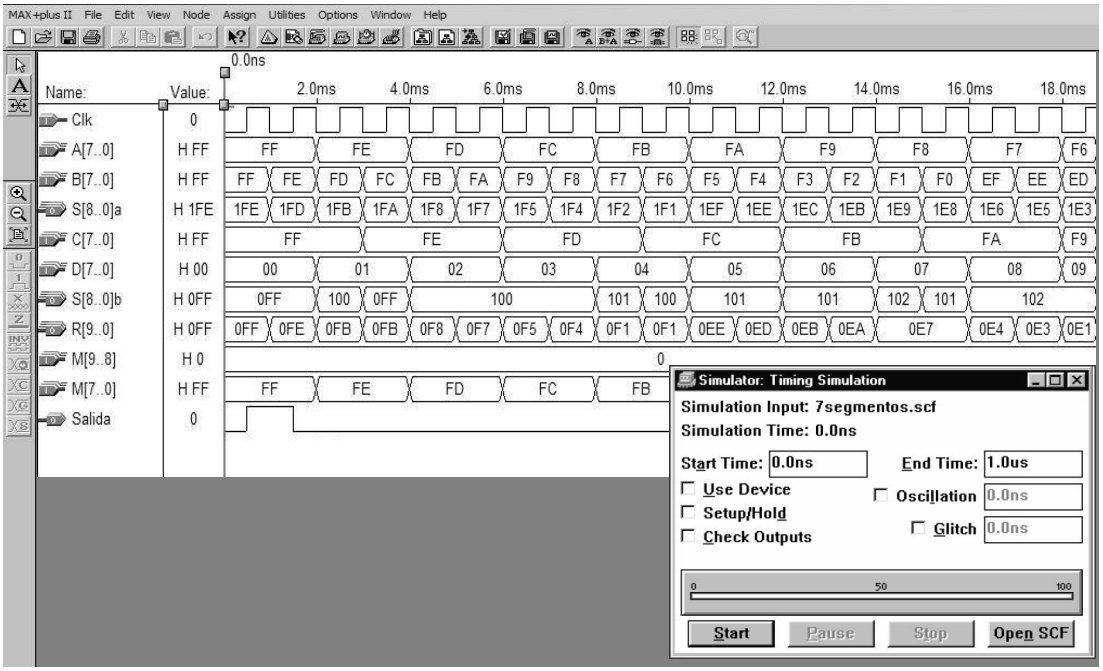

Figura 3.8. Aplicación del editor de formas de onda para verificar el comportamiento del circuito esquemático desarrollado

Con este editor se puede visualizar el comportamiento que tendrá el sistema, y por lo tanto, se puede verificar si ese es el que realmente se desea manejar, o si por el contrario, alguna de las rutinas no concuerda con el objetivo planteado. De ser así, se pueden realizar las correcciones necesarias al circuito esquemático, con lo cual se pueda mejorar la forma de ejecutar los procesos, y cubrir el objetivo bajo el cual fue diseñado el mencionado sistema.

Cuando ya no existe error alguno tanto en la estructura del circuito esquemático como en el comportamiento del mismo, el siguiente paso consiste en la programación del dispositivo. En primer lugar, se define el modelo y capacidad del mismo, como lo muestra la Figura 3.9, y posteriormente, se graba el archivo creado para este fin, en el espacio destinado para ello, de la forma que se observa en la Figura 3.10. Finalmente se realizan las pruebas de forma física con el dispositivo y con las señales de datos, de control y de tiempo, aplicadas de manera real. Para ello existen las tarjetas de desarrollo con las que la compañía Altera tiene en venta para los diseñadores de hardware.

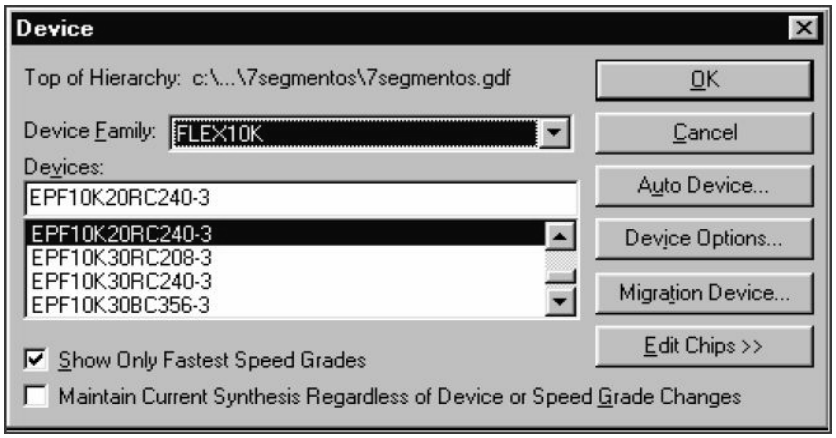

Figura 3.9. Selección del dispositivo para la implantación del diseño desarrollado

Figura 3.10. Proceso de grabación del diseño en el dispositivo seleccionado con el programador de MAX + plus II

La Figura 3.11 muestra un modelo de tarjeta de desarrollo. Con todos los aditamentos incluidos en dicha tarjeta se pueden hacer las pruebas físicas con el diseño implantado en los dispositivos programables con que cuenta.

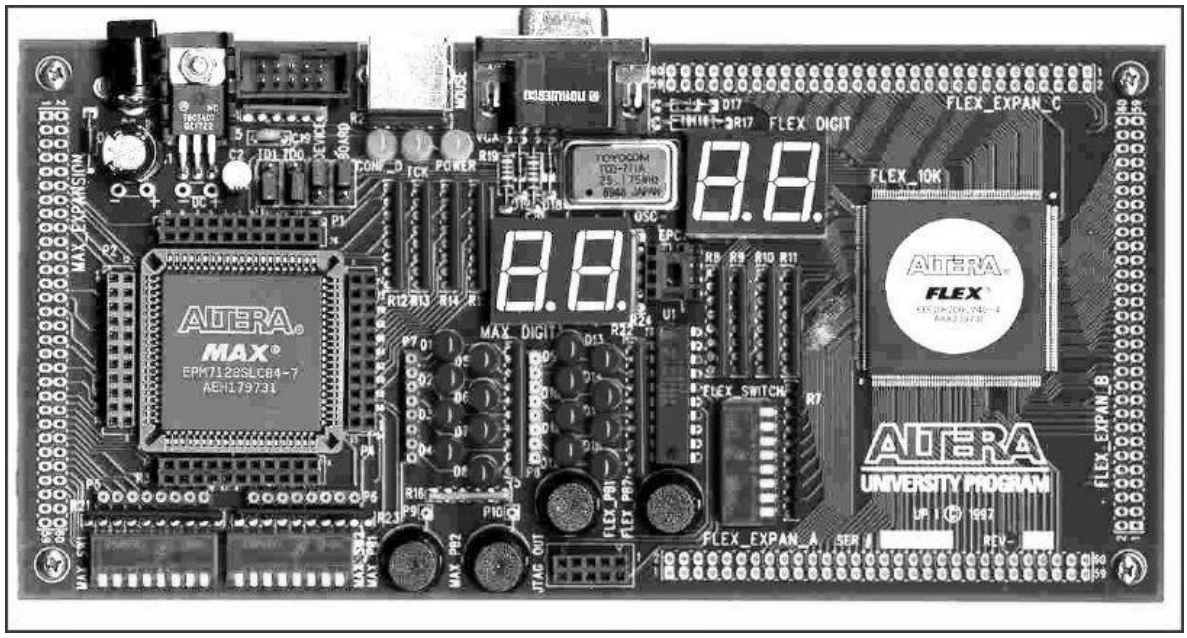

Figura 3.11. Tarjeta de desarrollo UP1/2 de Altera Corporation

3.3. Diseño e implementación de una neurona

El modelo de neurona que se muestra en la Figura 3.12 tiene dos grupos de entradas: dos del tipo excitatorias, y dos del tipo inhibitorias, además de un umbral *Th*, con lo cual se controla la actividad de la salida *S*.

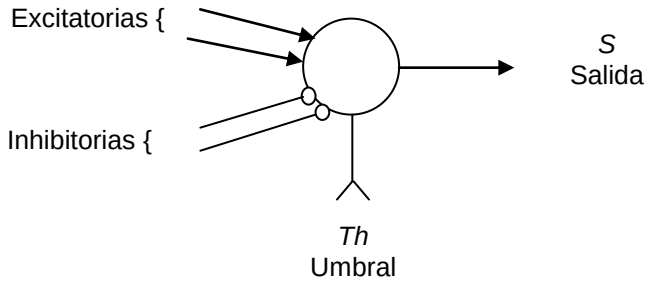

Figura 3.12. Modelo de neurona

Para implementar esta neurona se necesita diseñar un sumador que involucre las entradas excitadoras, y otro que considere las entradas inhibitorias, para ello, a continuación se muestra el proceso de diseño del sumador.

3.3.1. Sumador

El sumador que se utilizará es de un bit, ya que son dos entradas las que se pretende sumar, de acuerdo con el modelo que se muestra.

Para las conexiones excitatorias:

Proceso de suma de dos números de un bit.

$$\begin{array}{r} O_0 \\ +\ O_1 \\ \hline S_0 \\ C_0 \end{array}$$

Y la tabla de verdad (Tabla 3.1) que describe el comportamiento de la suma de los números de un bit es la siguiente:

O_0	O_1	S_0	C_0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabla 3.1. Tabla de verdad

Con el planteamiento anterior se obtiene un arreglo de compuertas básicas como lo muestra la Figura 3.13. Se utiliza una compuerta EX-OR, para la suma S_0, y una compuerta AND para el acarreo (carry), C_0, es decir, $S_0 = O_0$ XOR O_1, y $C_0 = O_0$ AND O_1.

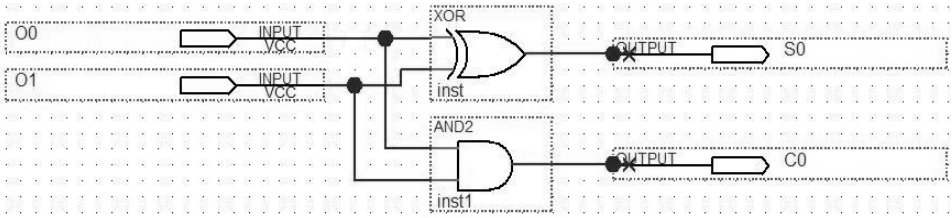

Figura 3.13. Circuito lógico de un sumador de dos números de un bit, para las conexiones excitatorias

Para las conexiones inhibitorias se sigue un proceso similar.

Proceso de Suma de dos números de un bit.

$$\begin{array}{r} O_2 \\ +\ O_3 \\ \hline S_1 \\ C_1 \end{array}$$

Con lo cual se obtiene la tabla de verdad (Tabla 3.2) que describe el comportamiento de la suma de los números de un bit:

O_2	O_3	S_1	C_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabla 3.2. Tabla de verdad

Cuyo arreglo de compuertas se observa en la Figura 3.14. El diseño es igual, solo se han cambiado los subíndices de las entradas y salidas del circuito. Esto es: $S_1 = O_2$ XOR O_3, y $C_1 = O_2$ AND O_3.

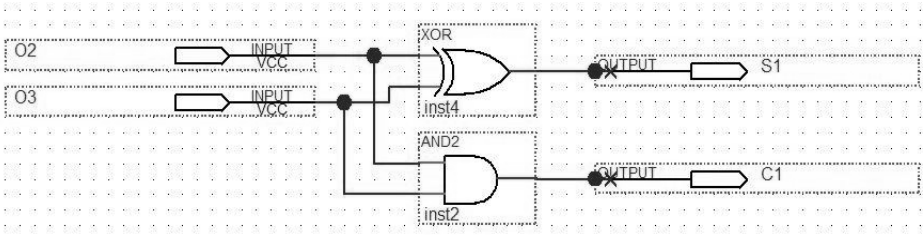

Figura 3.14. Circuito lógico de un sumador de dos números de un bit, para las conexiones inhibitorias

Después de lograr sumar las entradas excitatorias e inhibitorias, se continúa con el proceso de activación, de tal manera que la neurona muestre o no actividad a partir del valor establecido en el umbral *Th*. Para lograr esto se necesita diseñar un restador.

3.3.2. Restador

Para este caso se diseña un restador de dos números de dos bits, ya que los sumadores presentan dos valores en su resultado, uno de suma y otro de acarreo.

Proceso de resta de dos números de dos bits. Primera parte de la sustracción, $S_0 - S_1$:

$$\begin{array}{rr} & C_0\ S_0 \\ - & C_1\ S_1 \\ & C_2 \\ \hline & R_3\ R_2 \\ & C_3\ C_2 \end{array}$$

Por ello se tiene la tabla de verdad (Tabla 3.3) que describe el comportamiento de la resta de los primeros dos bits, S_0 menos S_1:

S_0	S_1	R_2	C_2
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Tabla 3.3. Tabla de verdad

Para la primera parte del restador, S_0 menos S_1, el arreglo de compuertas lo muestra la Figura 3.15. Es un circuito similar al sumador, solo que se ha negado una de las entradas, S_0, para el caso del acarreo (carry). $R_2 = S_0$ XOR S_1, y $C_2 = \hat{S}_0$ AND S_1, donde el símbolo "^" implica la negación de la variable.

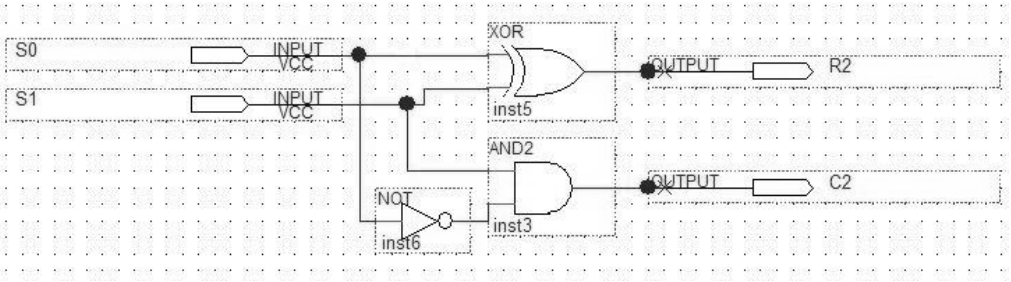

Figura 3.15. Circuito lógico de la primera parte del restador de dos números de dos bits

Para la segunda parte del restador, donde ya se involucra el acarreo C_2, se tiene que:

Proceso de resta de dos números de dos bits. Segunda parte con acarreo, $C_0 - C_1 - C_2$:

$$\begin{array}{rr} & C_0\ S_0 \\ - & C_1\ S_1 \\ & C_2 \\ \hline & R_3\ R_2 \\ & C_3\ C_2 \end{array}$$

Cuya tabla de verdad del comportamiento (Tabla 3.4) es:

C_0	C_1	C_2	R_3	C_3
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Tabla 3.4. Tabla de verdad

En la Figura 3.16 se muestra esta siguiente parte del restador, que ya involucra el acarreo en el proceso de substracción. Con lo que: $R_3 = C_0$ XOR (C_1 XOR C_2) y C_3 = [$\hat{C}_0$ AND (C_1 XOR C_2)] OR (C_1 AND C_2). Nuevamente, el símbolo "^" implica la negación de la variable.

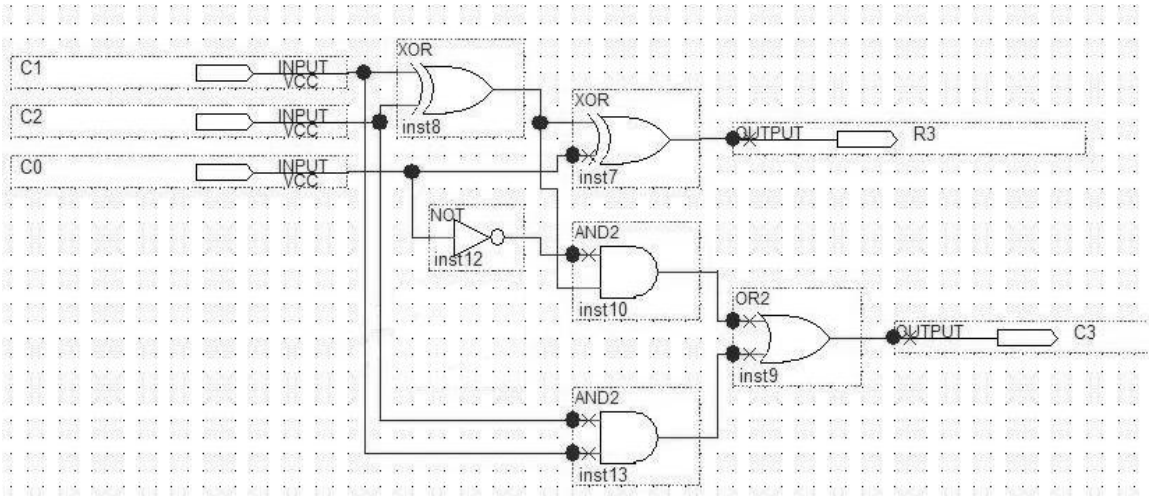

Figura 3.16. Circuito lógico de la segunda parte del restador de dos números de dos bits

Con este restador se logra un resultado que posteriormente será comparado con el umbral, *Th*, de tal manera que se active o desactive la salida *S* de la neurona. Para lograr esto es necesario entonces diseñar un circuito comparador que opere con el resultado del restador y el valor de umbral que se aplique en un momento *t*.

3.3.3. Comparador

Para el diseño del comparador se tiene que: R_3 y R_2 son las salidas del restador, las cuales se compararán con el valor de umbral, que se denota con *U*, compuesto por U_1 y U_0, por ser *R* un número de dos bits. Este comparador tiene una salida *S*, la cual será alta, $S = 1$, si el resultado de la resta es mayor o igual al valor de umbral, es decir, $R \geq U$, en caso contrario, será cero. En la siguiente tabla de verdad (Tabla 3.5) se describe dicho comportamiento.

La variable *S* muestra el resultado de la comparación, y toma valores de uno, $S = 1$, cuando el numero *R*, es decir R_3R_2, es mayor o igual al valor U, U_1U_0, $R \geq U$. En caso contrario, *S* es cero, $S = 0$, $R < U$. Para simplificar la salida *S* se aplica el método de reducción de Karnaugh [82], de la forma siguiente, Figura 3.17:

U_1U_0 \ R_3R_2	00	01	11	10
00	1	1	1	1
01	0	1	1	1
11	0	0	1	0
10	0	0	1	1

Figura 3.17. Mapa de Karnaugh

R_3	R_2	U_1	U_0	S
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Tabla 3.5. Tabla de verdad

Cuyo resultado es: S = [R_3 AND (R_2 OR $\hat{U}_0$)] OR [$\hat{U}_1$ AND (R_2 OR R_3)] OR ($\hat{U}_1$ AND $\hat{U}_0$), el símbolo "^" implica la negación de la variable. El circuito que define el proceso de comparación entre el valor resultante de la resta y el umbral, a partir de este resultado, se observa en la Figura 3.18. El arreglo de las compuertas básicas compara los datos de R y U.

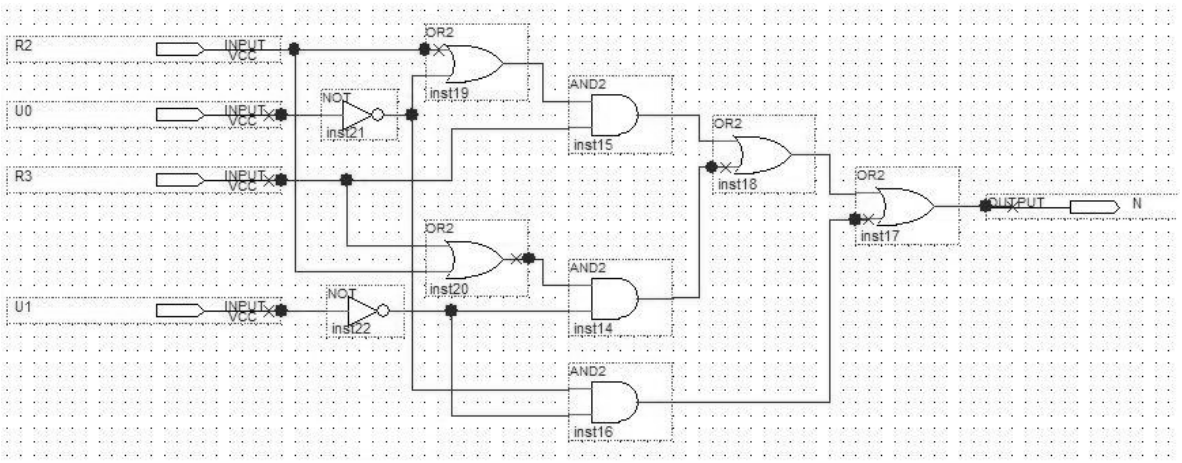

Figura 3.18. Circuito lógico del comparador de dos números de dos bits

Finalmente, al interconectar todos los elementos diseñados, esto es, los dos sumadores, el restador y el comparador, se tiene el circuito de la Figura 3.19, que describe el comportamiento de la neurona mostrada en la Figura 3.12.

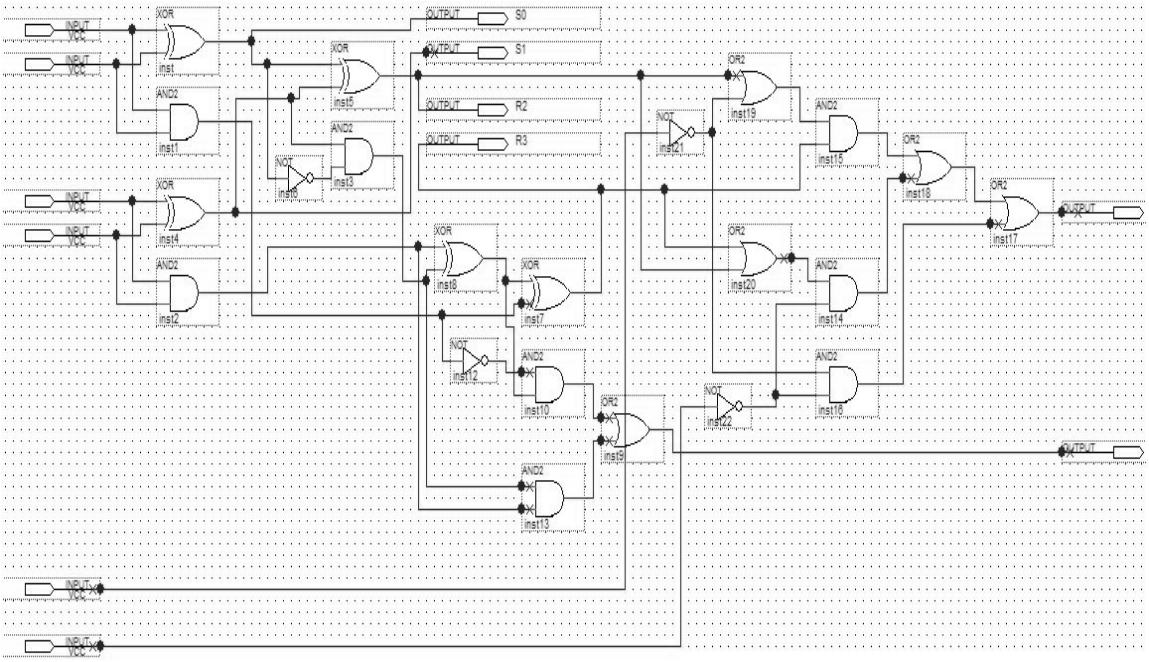

Figura 3.19. Circuito lógico del modelo de neurona (Figura 3.12)

Para comprobar el correcto comportamiento del circuito se procede a la simulación con lo cual se puedan verificar los resultados deseados.

3.3.4. Simulación

Las Figuras 3.20 y 3.21 muestran el comportamiento, a partir de la aplicación de valores en las entradas del circuito, que llevan a establecer que el desempeño del circuito lógico resultante del proceso de diseño es el definido por el modelo de neurona que se ha mostrado en la Figura 3.12.

La Figura 3.20 muestra un periodo de tiempo en el cual el desempeño del circuito describe la actividad de la neurona, esto es, los momentos en los cuales la neurona está activa o inactiva. El ovalo nos muestra el momento en el cual el valor de umbral, *U*, tiene un valor de cero, mientras que el valor de resta *R* varía entre 0 y 2. Dado que ambos resultados son igual y mayor que el umbral, respectivamente, se observa que la neurona está activa pues así lo demuestra con la salida *N*. Continuando con la línea de actividad dentro del ovalo, se observa como el valor de la resta *R* cambia a 0, con lo cual al ser comparado con el valor de umbral, que ahora es de 2, este es mayor que el primero, por lo cual la neurona sale de actividad, es decir, está inactiva como lo demuestra la salida *N* que está en un valor de cero. Se mantiene este estado de inactividad hasta que nuevamente el resultado de la resta *R*, con valor de 2 ahora, comparado con umbral *U*, todavía en 2, fuerza a la actividad ya que ambos valores son iguales cumpliéndose la condición de *R = U*. El proceso descrito continua aun fuera del ovalo, el cual se utiliza como referencia para describir el comportamiento deseado que el modelo de neurona de la Figura 3.12 define.

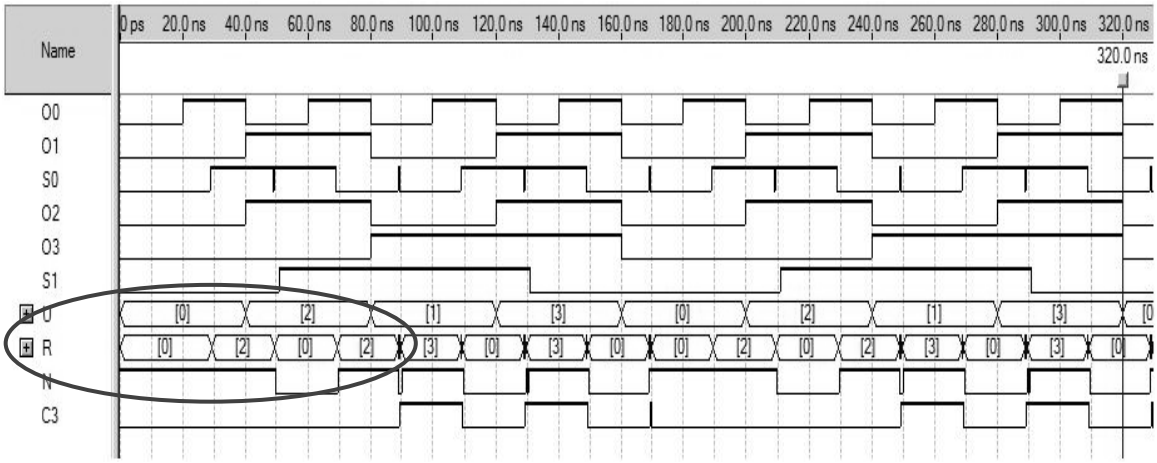

Figura 3.20. Grafica de comportamiento del circuito lógico del modelo de neurona

En la Figura 3.21 se muestra la continuación del desempeño que tiene el circuito lógico del modelo de neurona, pero en otro periodo de tiempo. Se observa un comportamiento similar al descrito en la Figura 3.20.

El ovalo muestra el momento en el que la neurona está inactiva cuando el resultado de la resta, $R = 0$, es menor que el valor de umbral, $U = 2$. Posteriormente la neurona entra en actividad cuando el resultado de la resta, $R = 2$, es igual que el valor del umbral, $U = 2$. Y se mantiene activa, pues posterior a esto $R = 3$, es ahora mayor al umbral, U = 1. Posteriormente $R = 0$, y como umbral se mantiene en 1, $U = 1$, la neurona se desactiva, hasta que resta, $R = 3$, es igual que umbral, $U = 3$, logrando que se active nuevamente la neurona. Terminando con el área marcada por el ovalo, se observa que la neurona se desactiva pues el valor de la resta R es cero, R = 0, menor que umbral, U = 3. El proceso continua más allá del ovalo, teniendo un comportamiento con características como las ya descritas.

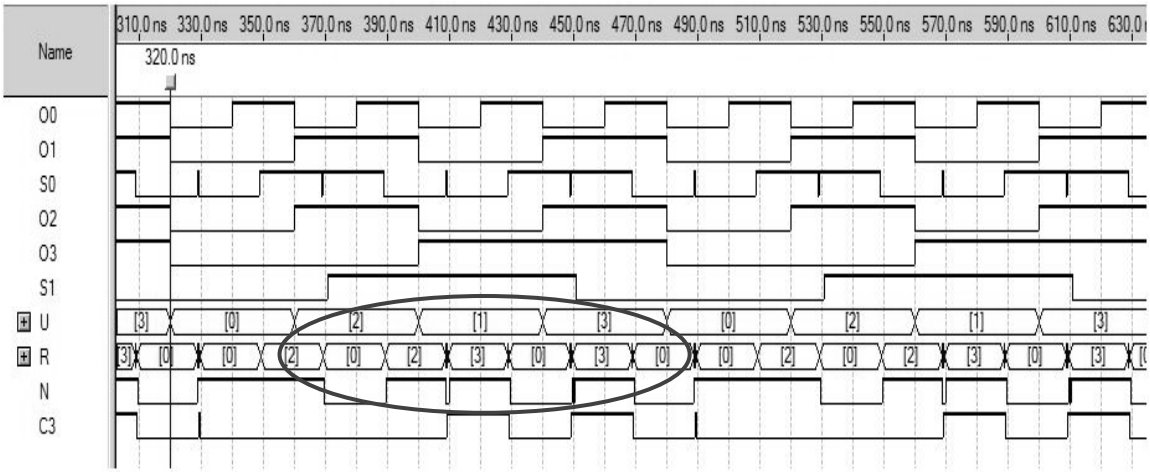

Figura 3.21. Grafica de comportamiento del circuito lógico del modelo de neurona

3.3.5. Resultados

Las Figuras 3.20 y 3.21 muestran que el desempeño del circuito lógico diseñado e implementado a partir del modelo de neurona de la Figura 3.12 es similar al planteado por dicho modelo, ya que como se puede apreciar en las simulaciones, al aplicarse las señales de entrada, tanto de las excitatorias como las inhibitorias, este arreglo lógico las opera para sumarlas, posteriormente las manipula con el restador, y les aplica a continuación el comparativo con el umbral, y así definir si la neurona sale de actividad o se mantiene activa, o por el contrario, si está inactiva y entra en actividad.

El programa de aplicación Max + plus II ha permitido la estructuración e implementación del circuito lógico que define el comportamiento de la neurona, y ha mostrado, a través de su modulo de simulación, el comportamiento que se espera debe tener dicha neurona. Con esto queda demostrada la utilidad de este paquete de aplicación para diseñar y estructurar sistemas lógicos que puedan ser implantados en sistemas de control de procesos.

Capítulo 4

Implementación del clasificador neuronal LIRA en un dispositivo lógico programable

4.1. Implementación del clasificador neuronal LIRA

4.1.1. Codificación

Después del ingreso de la imagen al clasificador a través del grupo de neuronas de la capa *S*, o retina, se continúa con las siguientes etapas en el proceso de trabajo del clasificador LIRA: codificación y clasificación, Figura 2.1.

La etapa de codificación consiste en el proceso de extracción de características de la imagen entrada. Para este propósito se genera una máscara. Esta máscara es un conjunto de conexiones de excitación y de inhibición entre las neuronas de la retina (capa *S*) y las neuronas de la capa *A*, o asociativa. En la Figura 4.1 se muestran estas conexiones, los círculos son conexiones inhibitorias, y las flechas son conexiones excitadoras. Las conexiones entre las neuronas de dichas capas son aleatorias, y, una vez generadas estas conexiones, no cambian durante los experimentos.

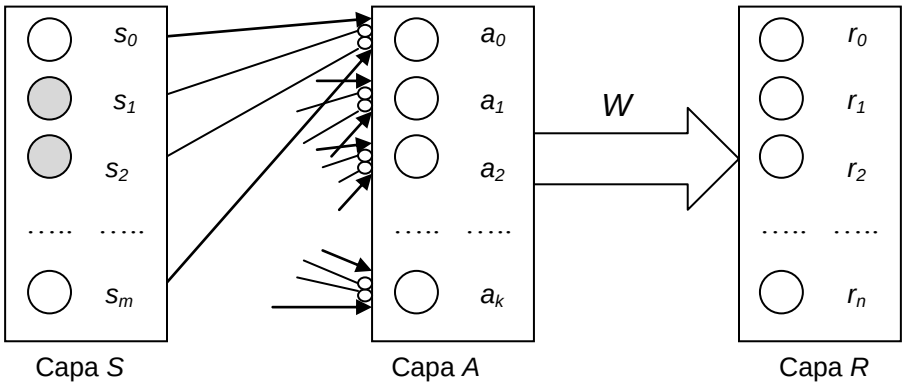

Figura 4.1. Proceso de generación de mascara y codificación

La posterior codificación se realiza a partir de la máscara aplicada a la imagen entrada. Cada imagen de entrada define una actividad diferente de las neuronas en la capa *A*. El vector binario que corresponde a las actividades neuronales asociativas es denominado como código binario de la imagen $\boldsymbol{A} = a_1,\ldots, a_k$, donde k es el numero de neuronas en la capa *A*. El procedimiento que transforma la imagen de entrada en el vector binario se denomina codificación de la imagen.

Para este sistema particular, la i-esima neurona, a_i, de la capa *A* está activa solo si todas las conexiones de excitación, de la retina, conectan las neuronas que corresponden al objeto, y todas las conexiones de inhibición conectan las neuronas que corresponden al fondo. En este caso, $a_i = 1$, de lo contrario, $a_i = 0$. Por el hecho de trabajar con estos sistemas se tiene en conocimiento que el número de neuronas activas p en esta capa *A* es muchas veces menor que el número total de neuronas k, esto es, $p << k$.

Con estos datos se obtienen los códigos de las imágenes, los cuales se guardan en forma compacta facilitando, mas adelante, el cálculo de actividad de las neuronas en la capa de salida *R*. Esto significa que, en lugar de guardar vectores binarios de gran tamaño, se guarda cada imagen como una lista propia de números de las posiciones de las neuronas activas, en el vector binario.

Las conexiones *W* entre las capas *A* y *R* tienen pesos variables, Figura 4.1, y cambian sus valores durante el proceso de entrenamiento. Estas conexiones entre neuronas de las mencionadas capas se generan con el criterio de "todas con todas", esto es, todas las neuronas de la capa *A* se conectan con todas las neuronas de la capa *R*.

Para almacenar la información sobre los pesos de las conexiones entre neuronas de la capa *A* y la capa *R*, se construye una matriz de la forma $k \times n$, Figura 4.2, donde *k* implica el número de neuronas de la capa *A*, y *n* es el numero de neuronas de la capa *R*, o número de clases a ser reconocidas.

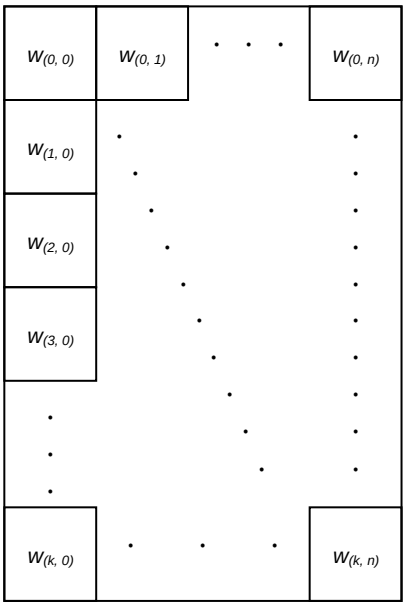

Figura 4.2. Modelo de la matriz k × n de los pesos entre las capas A y R

El proceso de reconocimiento entonces calcula la excitación de las neuronas de la capa *R*, con la Formula (2.4).

Si $a_k = 0$, entonces el producto $a_k \cdot w_{ki} = 0$ también, con lo cual el valor de excitación E_i proporciona un dato nulo. Por esto el interés de los bits que contienen unos, ya que únicamente los unos participan en los cálculos de las excitaciones en neuronas de la capa *R*.

La neurona que tiene la máxima excitación identifica a la clase reconocida (Capa *R*). Los unos en el vector binario *A* son distribuidos casi aleatoriamente, esto depende de las imágenes de entrada.

En aplicaciones de reconocimiento de imágenes, por ejemplo, se puede trabajar con un número *n* de clases, desde 2 hasta *n*, las cuales son presentadas en la capa *R*. Se define cada clase *n* con un tamaño de 16 bits, es decir, 2 Bytes para cada clase a ser reconocida. Para esto se utiliza un registro de 16 bits, y con un sumador-restador se calculan las excitaciones de las neuronas de la capa *R*.

Con el objeto de establecer cual es la clase de la imagen a ser reconocida, el menor Byte, de los dos que conforman el tamaño de cada clase *n*, desempeña la función de caracterizar la

actividad neuronal de la capa *R*, con la Formula (2.4). Por esta razón, existen muchos pesos, de acuerdo con el número de neuronas en la capa *A* y la capa *R*, que deben ser almacenados como una matriz de tamaño $k \times n$, los cuales son utilizados en las tareas de entrenamiento y reconocimiento del clasificador.

Por ello, es necesario utilizar una memoria de acceso aleatorio, la cual permita el almacenamiento de estos pesos w_{ki} en forma de Bytes. En la Figura 4.3 se muestra este modelo de memoria, donde *B* representa el peso w_{ki}, en Bytes, de las conexiones entre neuronas de las capas *A* y *R*. Los vectores de entrada de la matriz definen al grupo de neuronas que están activas en ese instante en la capa *A*, este mismo número de neuronas se encargará de direccionar cada uno de los espacios de almacenamiento con que cuenta dicha memoria.

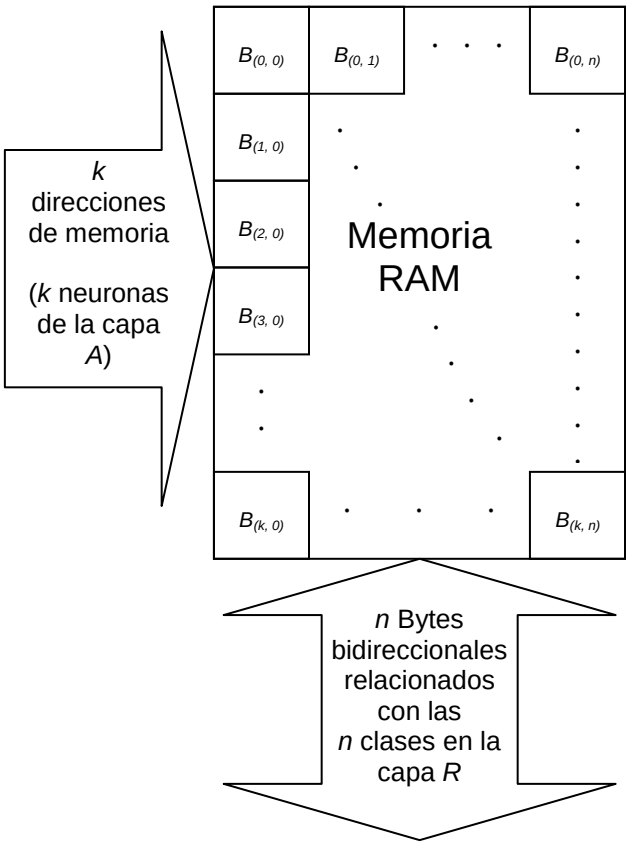

Figura 4.3. Modelo de memoria para implementar la matriz $k \times n$

En la Figura 4.4 se observa un ejemplo de un vector $\bar{A}$ de tamaño *k* que muestra los bits que corresponden a las neuronas de la capa *A*. Cada 1 habilita la lectura de los pesos almacenados en la localidad de memoria correspondiente. En la Figura 4.4, el bloque sombreado significa un uno del vector, el cual permite la lectura de los Bytes que representan los pesos de las conexiones, y también se aprecia el caso donde existen bits en cero, con lo cual esos pesos no son leídos, y por lo tanto, desechados en el cálculo de la excitación de las neuronas. De lo anterior se establece que, del número total *k* de neuronas

en la capa *A*, solo un número *p* de neuronas activas son involucradas en el cálculo de la actividad neuronal de la capa *R*, ($p << k$).

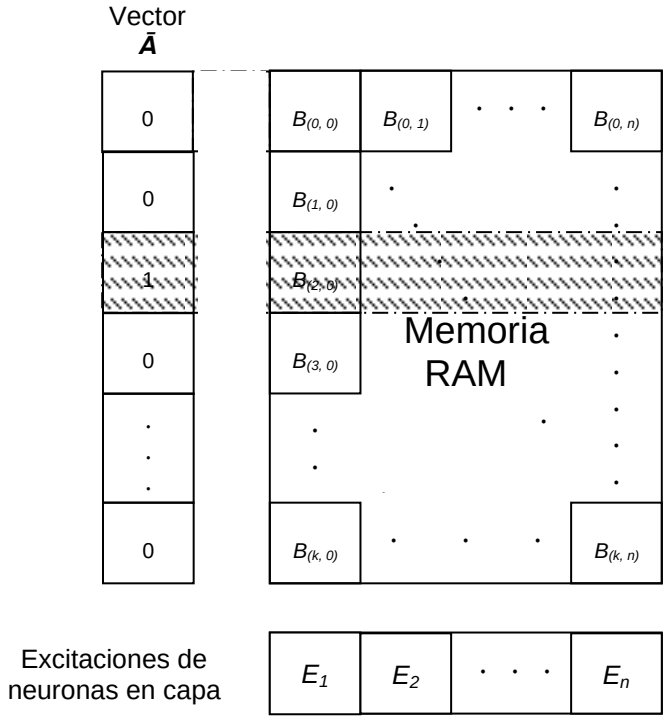

Figura 4.4. Ejemplo de operación del modelo de matriz k × n

Para representar la matriz de pesos, se utilizan varias memorias de acceso aleatorio, RAM (*Random Access Memory*), conectadas en forma paralela (Figura 4.5). Este grupo de memorias RAM tendrá relación con el número de clases *n* a reconocer, es decir, por cada clase existirá una memoria cuya salida, con un tamaño de un Byte, dará a conocer la suma de los pesos de las conexiones, de forma secuencial, relacionadas con las neuronas activas de la capa *A*. Cada memoria es entonces un conjunto de tamaño *k* de registros.

Para obtener el nivel de excitación de cada neurona de salida que corresponde a una clase, se tienen que sumar todas las líneas de memoria en la columna correspondiente. Los registros de almacenamiento de cada RAM_i suman las actividades, y posteriormente se elige la máxima actividad. De esta manera es como se realiza el proceso de reconocimiento. Para el entrenamiento, se utiliza algo similar, ya que el primer paso para ello es reconocer la imagen de entrada. Este proceso será explicado más adelante, en el proceso de entrenamiento.

Para realizar el trabajo con memorias RAM, y utilizando el ambiente de programación de Altera, se maneja el modelo de memoria con que cuenta dicha paquetería, con el fin de implementar la mencionada matriz sobre la cual se almacenarán los pesos de las conexiones entre la capas *A* y *R*, en los procesos de reconocimiento y de entrenamiento del clasificador LIRA.

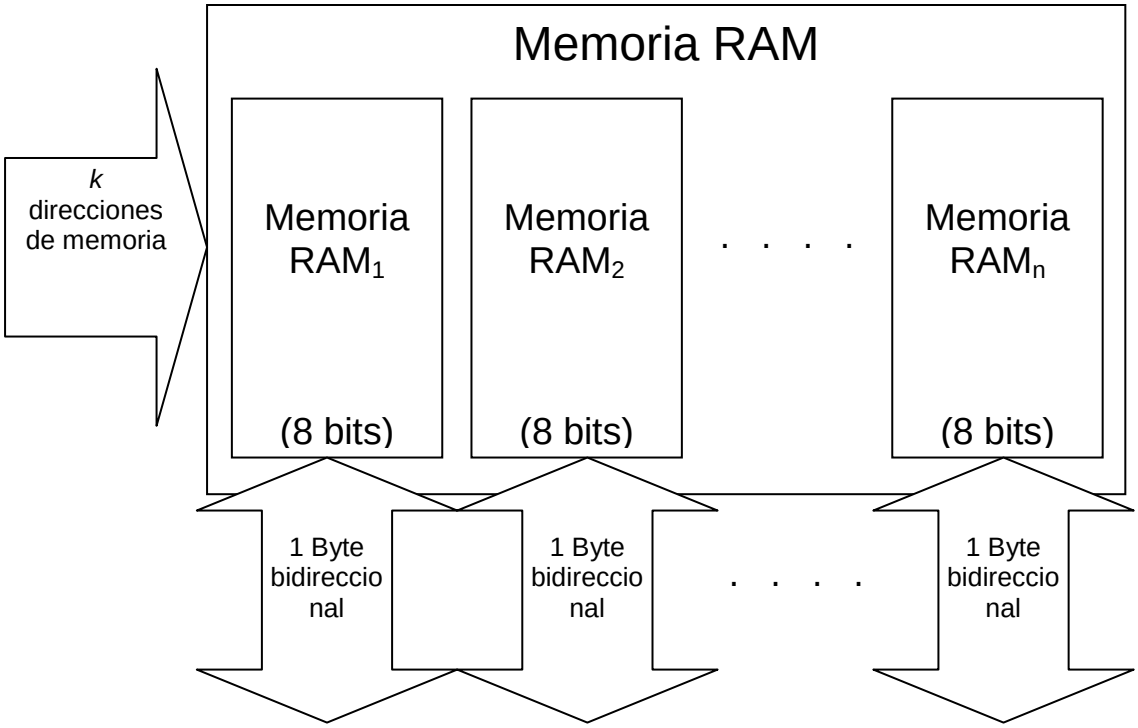

Figura 4.5. Grupo de memorias de acceso aleatorio (RAM)*, conectadas el paralelo*

4.1.2. Implementación de la matriz

El paquete de aplicación Max + plus II de Altera cuenta con una librería de módulos parametrizados, *Library of parameterized modules* (LPM), que ayuda a aligerar las tareas en la construcción y desarrollo de diseños esquemáticos, ya que su estructura interna se puede adecuar al objetivo particular del circuito en proceso de implementación. Se considera entonces que estos módulos parametrizados, al ser incorporados en diseños a ser desarrollados, y que requieran de dispositivos más complejos en su estructura, facilitan la construcción de dicho diseño.

Esta librería cuenta con módulos emuladores de memorias, lo que permite el uso de las mismas para estructurar las aplicaciones requeridas. En este sentido, el tipo de memoria RAM es el que se necesita para la implementación de la matriz en cuestión. En la Figura 4.6 se observa el esquema de una memoria RAM LPM configurada y estructurada para almacenar cuatro datos (pines Di en la Figura 4.6), cuya magnitud es de 2 bits (Di1 y Di0 en la Figura 4.6), es decir, cuenta con cuatro direcciones de memoria con dos bits de salida.

En esta Figura 4.6 se observa el grupo de entradas de datos, Di1 y Di0, a través de los cuales se ingresan los datos a las localidades de la memoria. Se muestran, además, el grupo de entradas que se encargan de direccionar el almacenamiento de dichos datos, A1 y A0, es decir, seleccionan en cual localidad se ha de almacenar cada uno de estos datos. También se puede apreciar el grupo de salidas, Do1 y Do0, que reflejan el dato almacenado en cada una de las localidades de la memoria. Para esto, es necesario que la entrada *we*, de control, esté en cero, esto es, en estado bajo. Por el contrario, cuando se guardan datos en las localidades correspondientes, la señal *we* está a uno, en estado alto. Finalmente, una entrada de reloj,

clk, es utilizada para sincronizar ambos procesos, escritura hacia, y lectura desde la memoria.

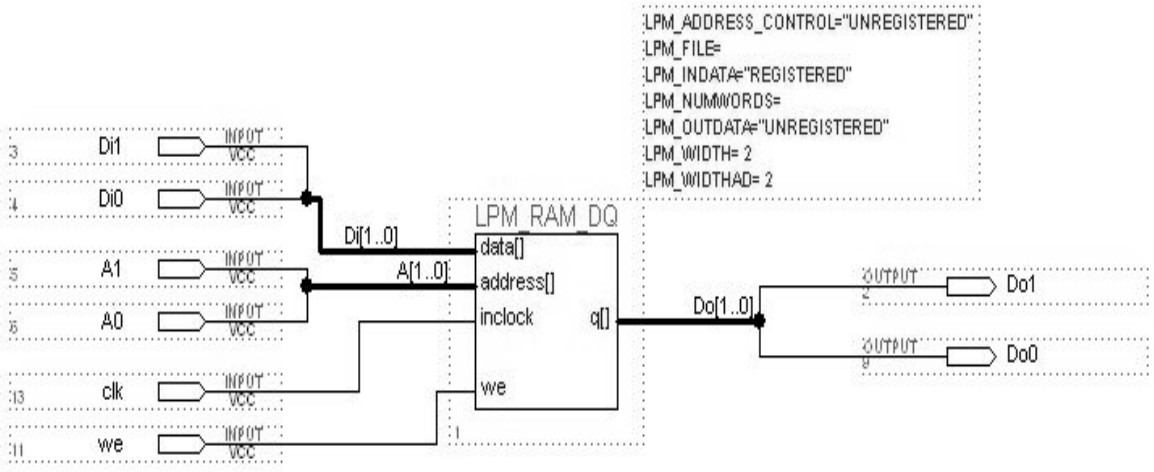

Figura 4.6. Esquema de una memoria RAM de 2 × 2

En la Figura 4.7 se observa el comportamiento, a través del simulador con que cuenta este programa de aplicación Max + plus II, del modelo de memoria RAM LPM de la Figura 4.6.

De esta Figura 4.7 se observa el reloj, *clk*, cuya función sincroniza, como ya se mencionó, el proceso de lectura y escritura de la memoria. Se observa también la entrada *we*, bajo la cual se rige el proceso de lectura y escritura, cuando *we* está en valor alto, uno, se escribe el dato presente en el grupo de entradas de datos, Di[1..0] (Di identifica a las entradas de datos, y [1..0] indica el número de entradas, es decir, el tamaño del dato en bits), y cuando *we* está en cero, el dato es exhibido en el grupo de salidas de datos, Do[1..0], (Do identifica a las salidas de datos, y [1..0] indica el número de salidas, esto es, el tamaño del dato en bits).

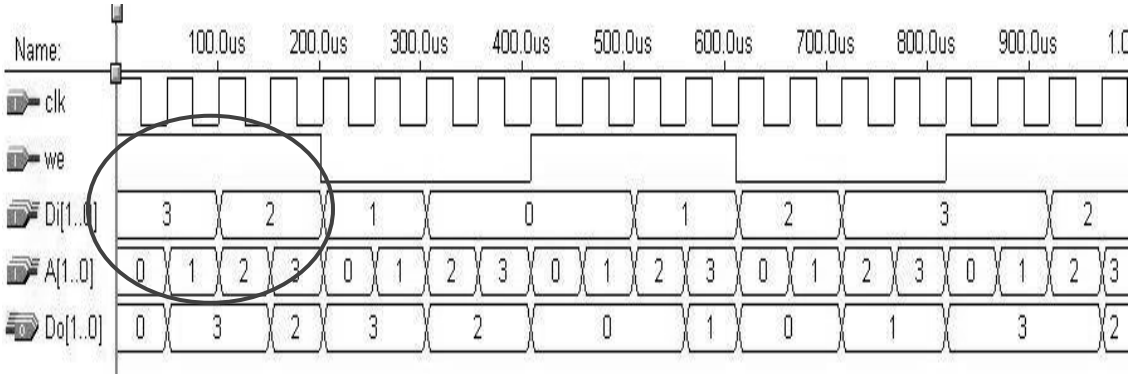

Figura 4.7. Comportamiento del esquema de la memoria RAM (escritura)

Según lo anterior, y observando el ovalo, *we* está en alto, por lo cual los datos de entrada en Di[1..0], datos 3 y 2, serán escritos, o almacenados, en la memoria. El dato 3 será almacenado en las direcciones A[1..0], 0 y 1, y el 2 lo será en las localidades A[1..0], 2 y 3, (A identifica a las entradas de direcciones, y [1..0] indica el número de entradas, es decir, el tamaño del dato en bits).

Posteriormente, la señal *we* cambia a un valor de cero, por lo cual ahora se lee lo que hay almacenado en las localidades A[1..0] de la memoria. En la siguiente Figura 4.8 se observa esta parte del proceso.

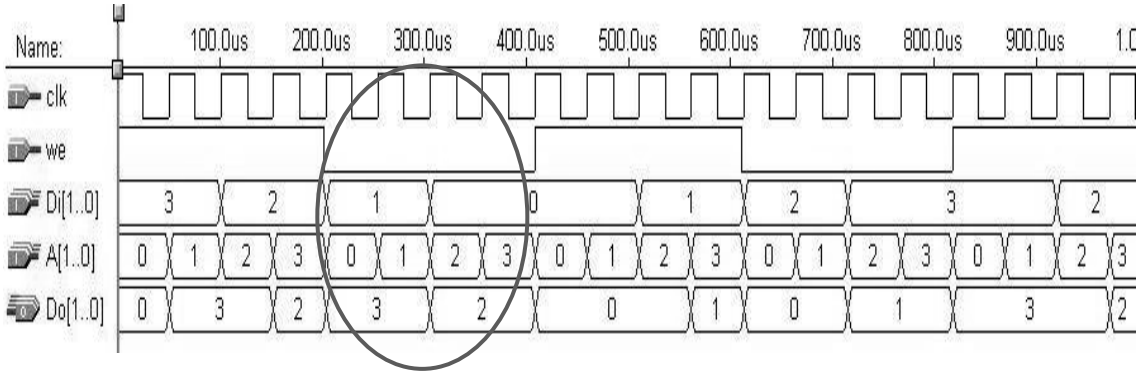

Figura 4.8. Comportamiento del esquema de la memoria RAM (lectura)

El ovalo muestra como, con la señal *we* en bajo, las localidades de memoria A[1..0] presentan el dato almacenado en cada una de ellas, a través de las salidas de datos Do[1..0]. En este sentido, las direcciones, A[1..0], 0 y 1 reflejan el dato guardado Do[1..0], 3, mientras que las otras dos, A[1..0], 2 y 3, lo hacen con el dato almacenado Do[1..0], 2.

La línea de operación mostrada en las Figuras 4.7 y 4.8 continúa, describiendo el proceso de lectura y escritura de la misma manera como el ya referido en los anteriores párrafos.

Para lograr una matriz de la forma $k \times n$ es necesario entonces agrupar un número n de este tipo de memorias RAM LPM, para poder establecer el número de clases a ser reconocidas por el clasificador LIRA. Se elige un número de clases a reconocer de 10, por lo cual son diez de estos elementos RAM los que serán utilizados para esta parte del proceso de reconocimiento del clasificador.

El número de bits por localidad de memoria será de ocho, y no de dos como lo muestra el modelo de memoria RAM mostrado en la Figura 4.6, además de que el número de localidades aumentará en función del número de neuronas a utilizar en la capa *S*, es decir, *k* indicará cuantas direcciones de memoria son necesarias para almacenar los pesos de las conexiones entre la capa *A* y la capa *R*, al realizar los procesos de reconocimiento y de entrenamiento del clasificador.

El reconocimiento empieza con la lectura desde la memoria de la primera imagen hacia el clasificador. Se calculan las excitaciones E_i de las neuronas de la capa *R*. Este valor E_i se obtiene de (2.4).

La neurona con máxima excitación E_{max} es considerada como la ganadora, obtenida con (2.7).

Después, se continua con la comparación entre la clase de la neurona ganadora, y la clase correcta a la que corresponde la imagen entrada, si son iguales, se continua el proceso con otra imagen, de lo contrario, se hace un calculo de errores. Esto es, si $i_w = i_c$, se continua

con la siguiente imagen a clasificar, de lo contrario se calculan los errores. i_w denota el número de la neurona ganadora, e i_c, el número de la neurona que corresponde a la clase a la cual pertenece la imagen de entrada.

El número de errores caracteriza la calidad del reconocimiento.

El proceso termina cuando ya no hay más imágenes a reconocer.

En el inicio del proceso de entrenamiento, todos los pesos de las conexiones entre las neuronas de la capa *A* y la *R* están puestos a cero, Figura 4.9. El entrenamiento empieza con la lectura desde la memoria de la primera imagen. Se calculan las excitaciones E_i de las neuronas de la capa *R*. Este valor E_i se obtiene de la Formula (2.4).

La neurona ganadora es la que tiene la máxima excitación E_{max}. Este valor E_{max} es obtenido con la Formula (2.7).

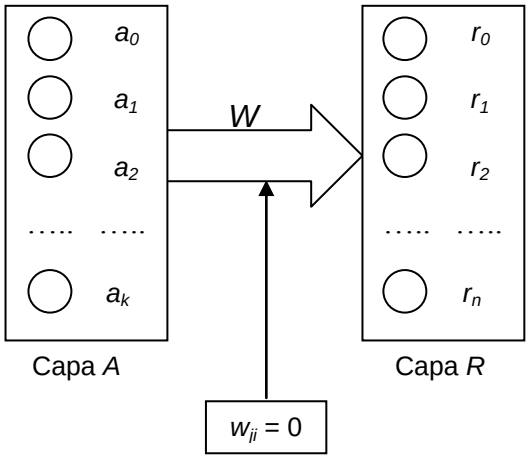

Figura 4.9. Inicialización del proceso de entrenamiento

Posteriormente, se hace una comparación entre la clase de la neurona ganadora, y la clase correcta a la que corresponde la imagen entrada. El número de la neurona ganadora se denota con i_w, y el número de la neurona que corresponde a la clase a la cual pertenece la imagen de entrada con i_c. Si $i_w = i_c$, nada ocurre puesto que no es necesario cambiar pesos, de lo contrario, si $i_w \neq i_c$ entonces se arreglan los pesos de acuerdo con las Formulas (2.5).

Lo que se hace es disminuir los pesos de la respuesta incorrecta, e incrementar los pesos de la respuesta correcta.

Para presentar el peso w_{max} se utiliza un Byte, por lo que el valor máximo que se puede almacenar es $w_{max} = 255$, puesto que $2^8 = 256$ niveles, de 0 a 255.

El proceso se repite algunas veces mas, a lo cual se le llama ciclos de entrenamiento. Los criterios para terminar el proceso de entrenamiento pueden ser:

1. Si el número de errores es menor que un cierto umbral establecido (el mejor de los casos es cuando el número de errores es cero).

2. Si el número de ciclos de entrenamiento ya se ha cumplido.

4.1.3. Esquema para simulación

De acuerdo con los anteriores procesos de reconocimiento y entrenamiento, es necesario entonces agregar ciertos componentes a la matriz, construida con memorias RAM, para cubrir los requerimientos definidos. En la Figura 4.10 se muestran estos arreglos a la matriz de pesos para trabajar en ambos procesos: reconocimiento y entrenamiento.

El primer componente a ser agregado consiste de un multiplexor (Mux) de dos entradas, una de ellas, de 8 bits, es para la salida de la memoria RAM, y la otra es para un valor constante "1", Figura 4.10. Ambos serán aplicados, de forma alterna a partir de una señal de control, al proceso de suma para el cálculo de la excitación E_i de la neurona de la capa R.

Para ello entonces, también se agrega un sumador-restador de s bits, en este caso 16, para obtener el valor de excitación, de tal manera que se satisfaga la Formula (2.4), Figura 4.10, y obtener las excitaciones de las neuronas de la capa R. Posteriormente se definirá cuál de ellas es la que tiene la mayor excitación, y por lo tanto, la que resulta ganadora en los procesos de reconocimiento y entrenamiento del clasificador neuronal LIRA.

Cabe señalar que cada memoria cuenta con un multiplexor de dos entradas, así como de un sumador-restador, por lo que, en función del número de clases a reconocer, existirá el mismo número de memorias con sus correspondientes multiplexores y sumadores-restadores, Figura 4.10.

Posteriormente, se aplica el resultado de la suma, o resta según sea el caso, a un registro de 16 bits, el cual conecta el mayor bit a un codificador de r bits, r está en función del número de clases a reconocer, que se encarga de mostrar el número, en binario, del mayor bit que está activo, es decir, que tenga un uno, Figura 4.10. Este número se relaciona con la neurona que muestra la mayor excitación E_i en ese instante, es decir, muestra la clase a la que pertenece el patrón entrado.

Este procedimiento permite definir qué neurona de la capa R tiene la mayor excitación, tanto en el reconocimiento como en el entrenamiento del clasificador, y presentar la clase a la que pertenece la imagen procesada. Esto es, activar la neurona de la capa R que se relaciona con la dicha clase reconocida.

También es necesario agregar un operador OR de n bits, igualmente está en función de las clases a reconocer, que se encargue de operar los bits de mayor valor de cada uno de los registros de 16 bits, Figura 4.10, que almacenan el resultado del sumador-restador. Este operador, cuando alguno de los bits de mayor valor presenta un uno, detiene el proceso de suma, o resta, de las excitaciones.

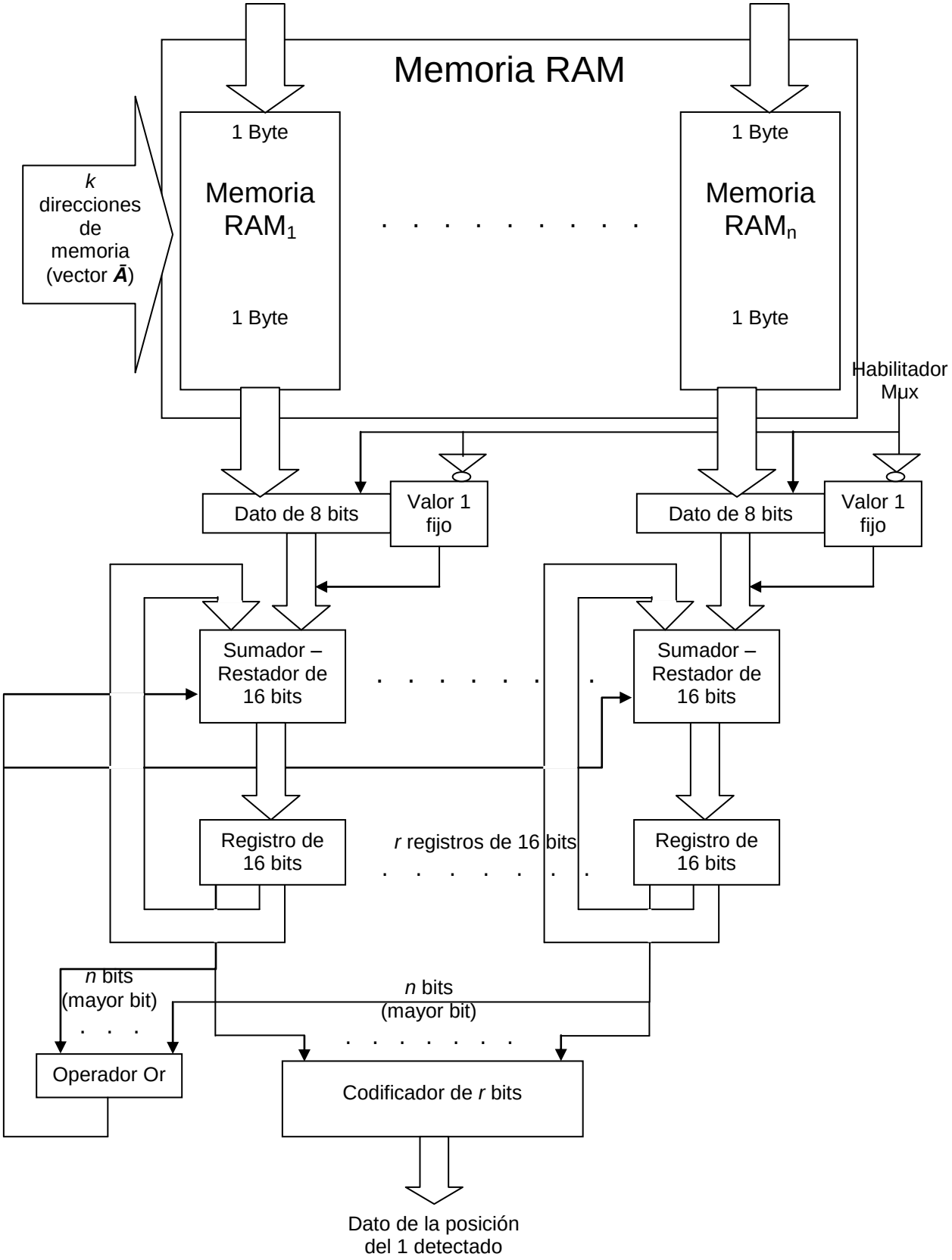

Figura 4.10. Esquema para simular la matriz k × n

4.1.4. Proceso de reconocimiento

Para el reconocimiento, este se inicia con el ingreso del código de la imagen. De la Figura 4.10, esto implica ingresar un vector $\bar{A}$ con un número *p* en los pines de direccionamiento *k* de las memorias. Este vector involucra una dirección de memoria que será aplicada en las líneas de direccionamiento de cada una de las memorias que conforman la matriz. Posteriormente, se realiza el calculo de excitación de cada una de las neuronas de la capa *R*, con la Formula (2.4). Esto se hace utilizando el sumador - restador de 16 bits que se encuentra acoplado a la salida de cada memoria RAM de la matriz, que contiene los pesos ya almacenados previamente en el proceso de entrenamiento. El resultado es guardado en el registro de 16 bits que tiene asignado cada uno de estos sumadores - restadores. En la Figura 4.10 están mostrados un número *n* de memorias con sus respectivos sumadores – restadores y registros que, juntos, se relacionan a cada una de las clases sobre las cuales el clasificador debe dar correspondencia con la imagen entrada. La primera etapa del cálculo de la excitación termina cuando todos los pesos almacenados en la memoria se han sumado.

Se inicia una siguiente etapa que consiste en deshabilitar la salida de la memoria, y habilitar un valor fijo de "1", a través del multiplexor, lo cual ayuda a obtener la excitación máxima E_i en cada una de las memorias. Para elegir la neurona ganadora se utiliza el codificador de *r* bits, que se encarga de operar el mayor bit de cada uno de los registros de 16 bits que recolectan el resultado de la suma de pesos de cada memoria (Figura 4.10). Este mayor bit, con un valor de 1, muestra la existencia de un valor máximo en el resultado de la suma de los pesos, lo cual se utiliza para detener el proceso de sumatoria, a través del operador Or, realizado para esta función.

Cuando el proceso de calculo de excitación está detenido, esto es, cuando los sumadores - restadores ya no hacen su operación de suma, el codificador de *r* bits realiza la combinación de cada uno de los bits de mayor valor, recolectados de los registros de 16 bits que almacenan el resultado de las sumas de los pesos, para determinar la neurona cuya clase es la ganadora. Este codificador muestra el número del bit de mayor valor que tiene el 1, valor máximo de la sumatoria de pesos. En función de esto, el codificador define la neurona ganadora, con lo que se establece la clase a la cual corresponde la imagen entrada.

Después, se continua con la comparación entre la clase de la neurona ganadora, y la clase correcta a la que corresponde la imagen entrada, si son iguales, se continua el proceso con otra imagen, de lo contrario, se hace un calculo de errores. Esto es, si $i_w = i_c$, se continua con la siguiente imagen a clasificar, de lo contrario se calculan los errores.

El proceso termina cuando ya no hay más imágenes a ser reconocidas.

4.1.5. Proceso de entrenamiento

El proceso se inicia con la puesta a cero de todos los pesos w_{ji} de las conexiones entre las neuronas de las capas *A* y *R*, esto es, en los registros de las memorias (Figura 4.10) los datos existentes son cambiados por datos en cero. El siguiente paso es la lectura de código de la imagen entrada, lo que consiste en aplicar un vector $\bar{A}$ con un número *p* en los pines de direccionamiento *k* de las memorias (Figura 4.10), esto es, ese vector presenta los números de las líneas de memoria que van a participar en el cálculo de excitación para cada

una de las neuronas de la capa *R*, con la Formula (2.4). Para este caso, el vector $\bar{A}$ tiene un número *p* de unos menor que 255, $p < 255$. El calculo de excitación se logra con el sumador - restador de 16 bits que se encuentra acoplado a la salida de la memoria que se encarga del almacenamiento de los pesos, el resultado es depositado en el registro de 16 bits que tiene asignado cada uno de estos sumadores - restadores. En la Figura 4.10 se muestra un número *n* de memorias con sus respectivos sumadores – restadores y registros que, en conjunto, representan las clases sobre las cuales el clasificador asigna la correspondencia de la imagen entrada. Cuando todas las líneas de direccionamiento de la RAM_i, que corresponde a un "1" en el vector $\bar{A}$, ya han sido analizadas, se bloquea la salida de la memoria y se habilita el registro con el valor fijo "1", utilizando el multiplexor, con lo cual se obtiene el valor máximo de excitación E_i de cada una de las memorias. Para la elección de la neurona ganadora se utiliza el codificador de *r* bits, el cual manipula el bit de más valor de cada uno de los registros de 16 bits que recolectan el resultado de la suma de pesos de cada memoria (Figura 4.10). Este bit de mayor valor, al estar en 1, implica que existe un valor máximo en el resultado de la sumatoria de los pesos, lo cual se utiliza para detener el proceso de suma de pesos, a través del operador Or, implementado para esta función.

Cuando el proceso de calculo de excitación está detenido, es decir, cuando los sumadores - restadores ya no hacen su operación de suma, para determinar la neurona cuya clase es la ganadora, el codificador de *r* bits realiza la combinación de cada uno de los bits con mayor valor, recolectados de los registros de 16 bits que almacenan el resultado de las sumas de los pesos. Este codificador recibe cada bit, y mantiene el número de la posición donde el 1 es detectado. Con esto, se define la neurona ganadora, con lo cual se establece la clase a la cual corresponde la imagen entrada.

Se hace la comparación entre la clase de la neurona ganadora, y la clase correcta a la que corresponde la imagen entrada, si son iguales, se continua el proceso con otra imagen, de lo contrario, se hacen ajustes en los pesos para que se cumpla la condición. Esto es, si $i_w = i_c$, nada ocurre con los pesos, de lo contrario los ajustes a estos pesos se realizan con (2.5). Este proceso implica entonces corregir los pesos en la matriz de memorias, a través de los sumadores – restadores de 16 bits, decrementando el peso de la neurona ganadora, e incrementado el peso de la neurona a la cual corresponde la imagen entrada.

El proceso continua hasta terminar con todas las imágenes a ser entradas al clasificador, o cuando se llegue al limite establecido.

4.1.6. Implementación en Altera

De acuerdo con la descripción y las consideraciones anteriores, se utiliza nuevamente el paquete de aplicación Max + plus II de Altera, en su ambiente de desarrollo de esquemáticos, para agregar entonces tales dispositivos al diseño esquemático de la Figura 4.6. Con ello, poder conocer el comportamiento de dicho arreglo esquemático y, a través del modulo de simulación con el que cuenta dicha aplicación, verificar que cumpla con los objetivos ya descritos en el proceso de entrenamiento y reconocimiento de imágenes del clasificador LIRA, y que se bosquejan en la Figura 4.10.

El la Figura 4.11 se muestra el diseño esquemático con la adición de los elementos mencionados. Se observa la memoria RAM (LPM_RAM), que se encarga de almacenar todos los pesos *w* involucrados en los procesos de reconocimiento y entrenamiento, a través de los pines Di1 y Di0, y direccionados por los pines A1 y A0.

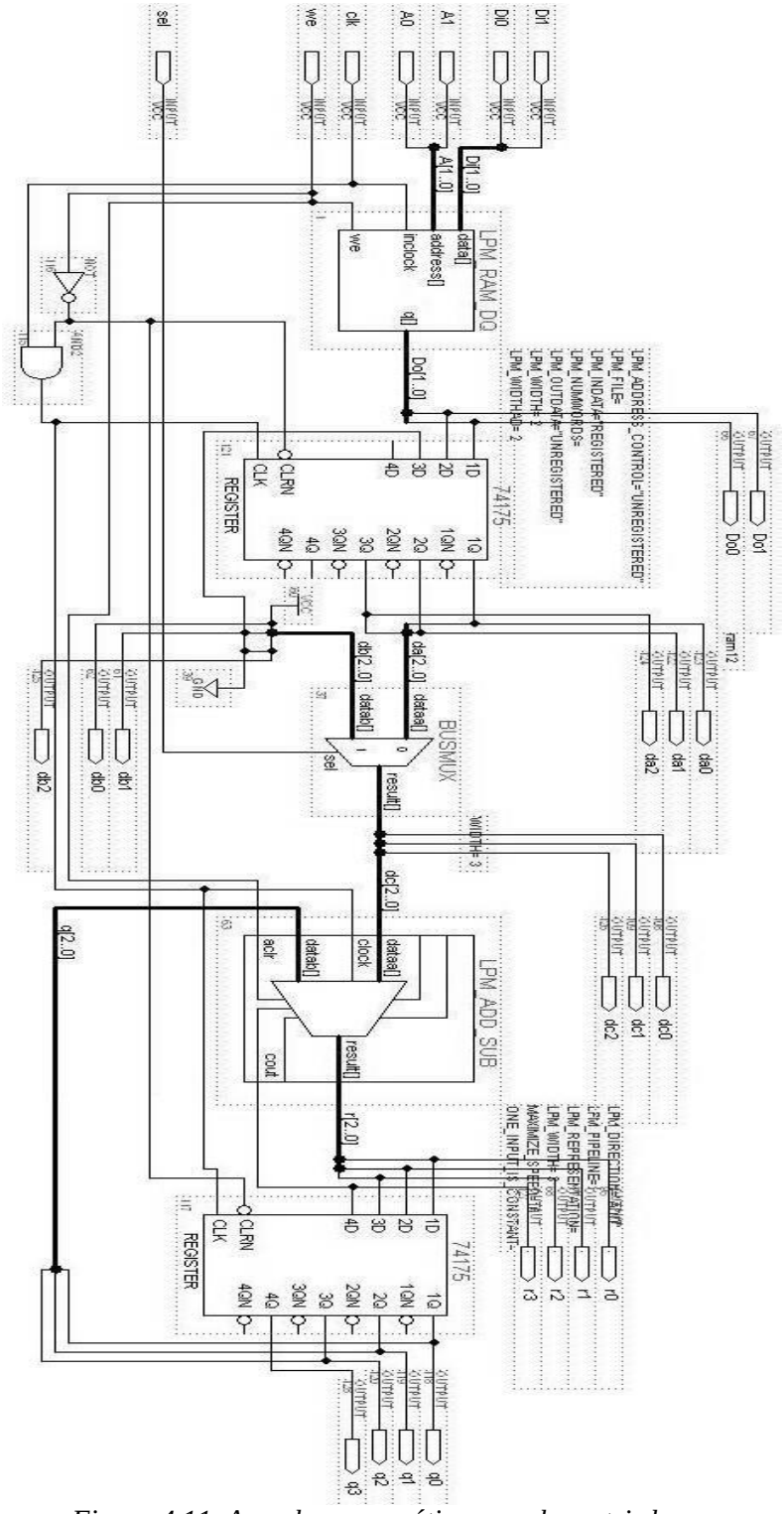

Figura 4.11. Arreglo esquemático para la matriz k × n

La salida (da[1..0]) de la RAM está acoplada a un registro (74175), que hace la función de bloquear esta salida, cuando se escriben datos a la memoria, cuando el pin *we* = 1, y deja pasar los datos, cuando se leen desde la memoria, pin *we* = 0, al multiplexor (BUSMUX). Este se encarga de seleccionar, con el pin *sel*, entre el Byte de salida de la memoria, *sel* = 0, que muestra los pesos *w* almacenados, y un valor fijo de "1" (db[2..0] en la Figura 4.11), con *sel* = 1. La salida del multiplexor (result []) se ingresa a una de las entradas (dataa[]) del sumador (LPM_ADD_SUB), el cual se encarga de sumar los pesos *w* contenidos en la memoria. La salida de este sumador (r[2..0]) se almacena en otro registro (74145) para mantener el dato por un cierto instante, mientras se realiza la siguiente operación de suma. La salida de este registro, q[2..0] junto con q3, muestra el resultado de la adición; q3 indica el acarreo del sumador, y q[2..0] es el resultado aplicado a la otra entrada del sumador, esto precisamente para el cálculo de la excitación máxima de cada una de las neuronas de la capa *R*, con lo que se define la clase ganadora. Este registro también se encarga de bloquear las salidas q, es decir, pone ceros en estas salidas cuando se habilita el proceso de almacenamiento de datos hacia la memoria, *we* = 1.

El mayor bit (q3) del registro 74175 es utilizado, junto con otros bits, para detener el proceso de suma, o calculo de excitación, además para indicar la neurona ganadora. Esta Figura 4.11 muestra el arreglo esquemático para una sola clase, por lo cual, por cada clase que se involucre en el proceso de reconocimiento y entrenamiento, se debe de incorporar una estructura igual a la mostrada, esto es, por cada clase se necesita un esquemático como el expuesto en la Figura 4.11.

La entrada de reloj, pin *clk*, es utilizada para sincronizar los procesos de escritura hacia, y lectura desde la memoria, y también para habilitar a los registros, permitiendo el paso de los datos hacia el otro extremo, o bien, evitar que pasen, escribiendo ceros.

Cabe recordar que este diseño esquemático está estructurado para direcciones de memoria y datos de 2 bits, con el objeto de simplificar y acelerar el proceso de simulación.

En la Figura 4.12 se muestra la simulación del comportamiento realizado por el arreglo esquemático de la Figura 4.11.

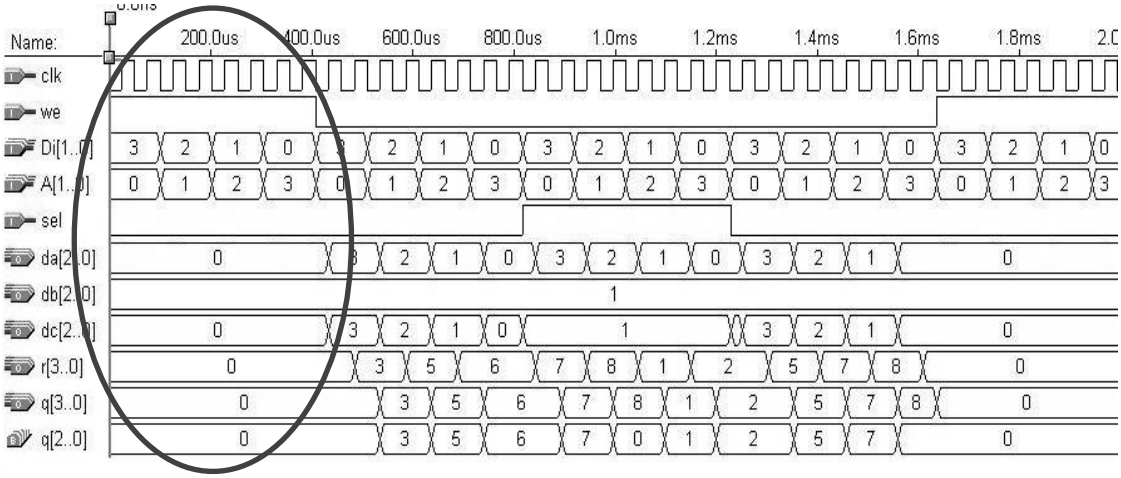

Figura 4.12. Simulación del comportamiento del arreglo esquemático (escritura)

La Figura 4.12 muestra varias señales, una de ellas es *clk*, que se encarga de sincronizar la operación de la memoria (LPM_RAM), y los registros (74175), esto es, dar una tiempo para que operen dichos dispositivos. *we* es otra señal de control que se encarga de controlar los procesos de escritura hacia y de lectura desde la memoria. El ovalo en la figura muestra el momento en que *we* está en uno, lo que implica que se están escribiendo datos hacia la memoria, *we* = 1. Di[1..0] indica los datos que son aplicados a las entradas de datos de la memoria, lo que significa que esas combinaciones son las que están siendo entradas a la memoria (3, 2, 1 y 0 en la Figura 4.12). A[1..0] muestra las direcciones de memoria, las cuales se encargan de seleccionar el número de registro dentro del cual se almacena cada dato aplicado en Di[1..0] (0, 1, 2 y 3 en la Figura 4.12). Se observa también la señal de control *sel*, cuya función es seleccionar alguna de las dos entradas del multiplexor (BUSMUX). En el ovalo, *sel* está en cero, lo cual indica que se está permitiendo el paso de la entrada cero, los datos que vienen de la memoria, hacia la salida de dicho multiplexor, *sel* = 0. Debido a que se están escribiendo datos en la memoria, los demás indicadores muestran cero, esto es, cuando se escriben datos a la memoria, no se realiza ninguna operación adicional en el arreglo esquemático, así lo demuestra el registro auxiliar da[2..0], que se encarga de mostrar los datos aplicados a la entrada 0 del multiplexor. Solo el registro auxiliar db[2..0], que muestra los datos aplicados a la entrada 1 del multiplexor, indica el valor fijo "1".

En la siguiente Figura 4.13 se describe otra parte del proceso de simulación del arreglo esquemático de la Figura 4.11. Se observan las mismas señales ya descritas anteriormente, *clk*, *we*, *sel*, así como los datos Di[1..0] y las direcciones de memoria A[1..0]. En esta parte, mostrada dentro del ovalo, se observan cambios en las señales. *we* tiene ahora un valor de cero, *we* = 0, lo que indica que se están leyendo datos desde la memoria. A[1..0] muestra las combinaciones para direccionar cada registro de la memoria (0, 1, 2 y 3 en la Figura 4.13). El registro auxiliar da[2..0], que se encarga de mostrar los datos aplicados a la entrada 0 del multiplexor, ahora expone los datos que vienen desde la memoria, puesto que se está leyendo información desde la misma (3, 2, 1 y 0 en la Figura 4.13).

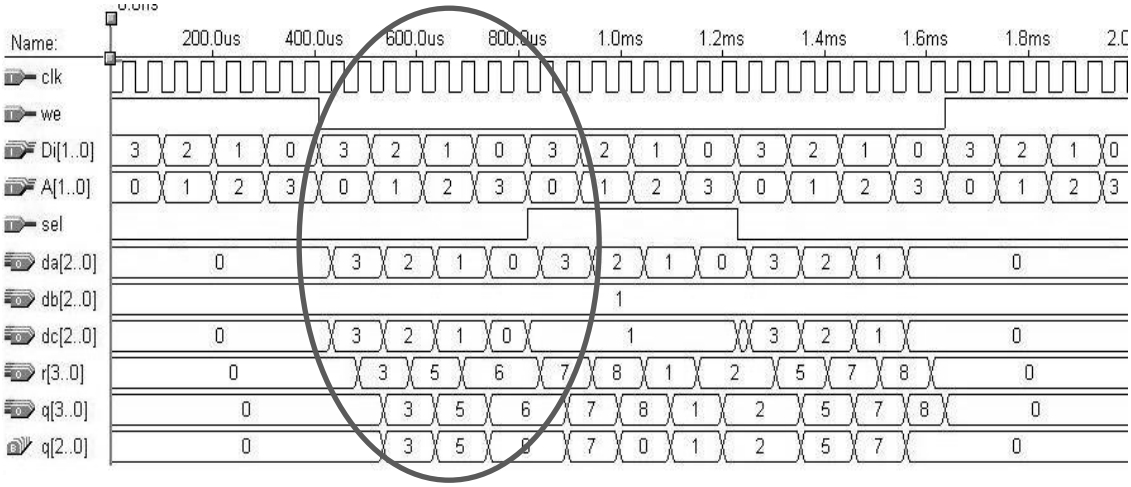

Figura 4.13. Simulación del comportamiento del arreglo esquemático (lectura)

La señal *sel* permanece en cero, con lo cual se permite el paso del dato proveniente de la memoria hacia el sumador, como lo muestra el registro auxiliar dc[2..0], que muestra la salida del multiplexor (los mismos datos 3, 2, 1 y 0). Como se ha mencionado, db[2..0] es un registro auxiliar que muestra el valor fijo "1", aplicado a la entrada 1 del multiplexor. A continuación se observa r[3..0], se trata también de un registro auxiliar que muestra el resultado de la operación de adición realizada por el sumador. La salida q[3..0] también muestra este resultado de la suma, pero que ya está guardado momentáneamente en el registro 74175, mientras se realiza la siguiente adición. Con q[3..0] se tiene el resultado de la suma junto con el acarreo de salida del sumador. Se observa otro registro auxiliar, q[2..0], que muestra el resultado de la suma, pero sin el acarreo de salida, que ha de ser aplicado a la otra entrada del sumador, para continuar calculando la excitación máxima de la neurona.

Existe un momento, mostrado también en el ovalo, en que *sel* cambia a uno, *sel* = 1, con lo que se permite el paso del valor fijo "1" hacia el sumador, como lo muestra el registro auxiliar dc[2..0]. Con esto, el resultado de la suma se ve incrementado en 1, como lo demuestran r[3..0] y q[3..0].

El proceso de simulación continúa mostrando que, cuando se leen datos desde la memoria, se realiza la adición con el sumador, para la obtención de la excitación de la neurona, mientras que cuando se escriben datos a la memoria, todo está inhabilitado.

Con lo mostrado por las Figuras 4.12 y 4.13 se comprueba que el arreglo esquemático de la Figura 4.11 opera adecuadamente, concordando parcialmente con los objetivos planteados en la Figura 4.10.

En función de las clases a ser reconocidas, se interconectarán varios de estos arreglos esquemáticos para lograr la matriz $k \times n$ (Figura 4.10), requerida para realizar los procesos de reconocimiento y entrenamiento con los algoritmos del clasificador neuronal LIRA.

Con el objetivo de realizar pruebas físicas utilizando el arreglo esquemático de este clasificador neuronal, la matriz $k \times n$ debe ser entonces construida con memorias, registros, multiplexores y sumadores, de 8 bits todos ellos, para poder obtener una respuesta acorde con los requerimientos establecidos en este apartado.

Por ello, en el siguiente apartado se muestra un arreglo esquemático estructurado para trabajar con dos clases, en el reconocimiento y entrenamiento de dicho clasificador neuronal LIRA.

4.2. Implementación del clasificador neuronal LIRA para dos clases

Debido a que el esquema electrónico involucra un dispositivo de almacenamiento de datos, lo cual implica el uso de muchos recursos dentro del dispositivo programable, reduciendo, por consecuencia, el espacio para estructurar otras funciones, en esta implementación se opta por no utilizar las memorias RAM_i. Los datos a ser operados por este esquema electrónico del clasificador son suministrados a través de una computadora.

En la siguiente Figura 4.14 se observa un esquema electrónico con dos estructuras, y cada una de ellas representa una clase sobre la cual el clasificador deberá clasificar las imágenes presentadas. Debido al número grande de componentes involucrados en cada estructura, y para reducir el tamaño del esquema electrónico en el editor grafico del paquete de desarrollo de Altera, cada una de ellas se muestra como un bloque sencillo, pero su comportamiento es el que se mostró en el apartado anterior.

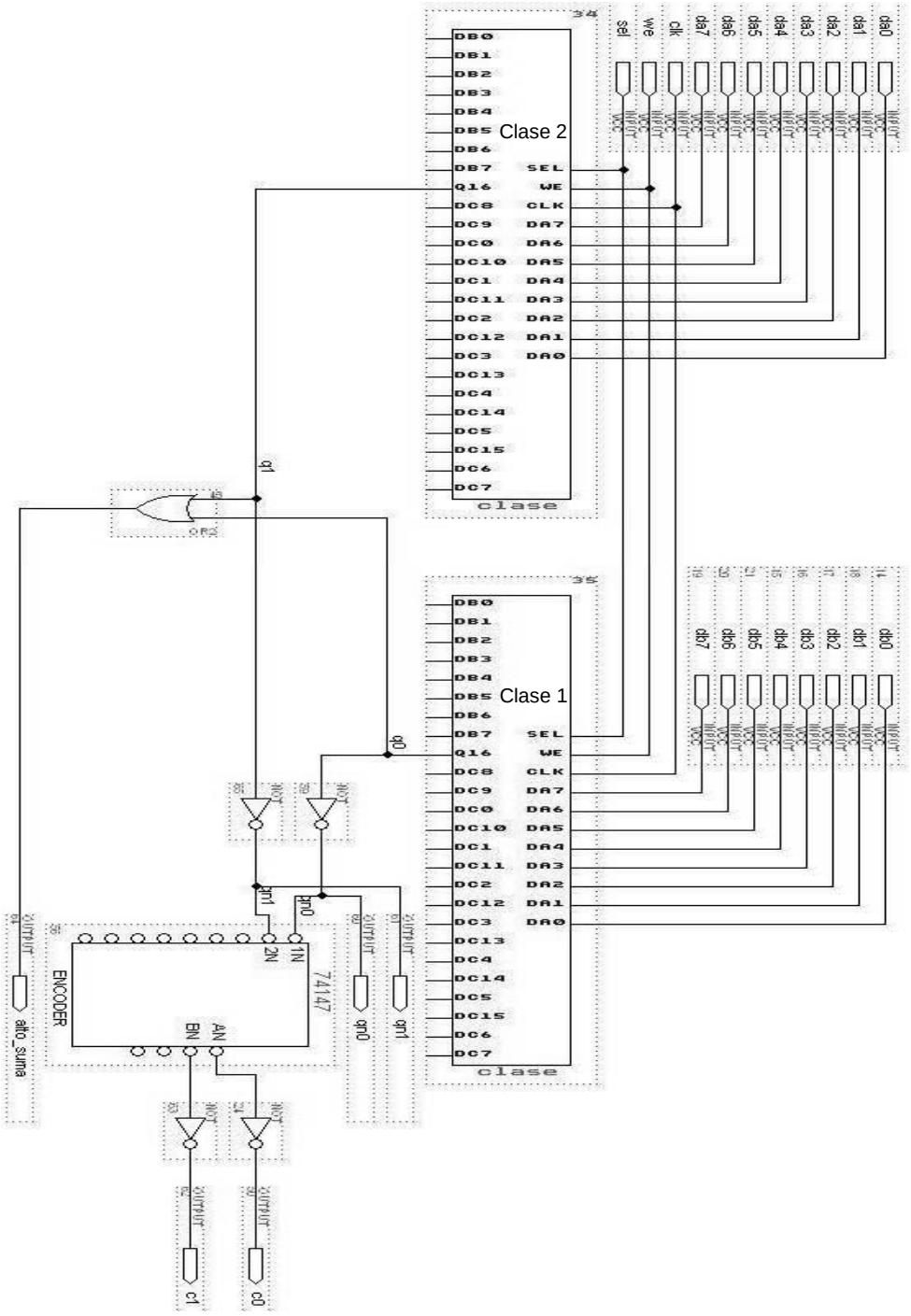

Figura 4.14. Esquema electrónico para dos clases

Ambas estructuras tienen una entrada de 8 bits (da[7..0] y db[7..0] en la Figura 4.14), que implicaría la inyección de los datos, provenientes de las memorias, para el proceso del cálculo de excitaciones E_i, con i = 0, 1, y de la excitación máxima E_{max}. Cada una de estas estructuras implica una neurona, que mostrará un cierto valor de excitación, que se asociará a la clase que esta representa.

Ambas, también, están regidas por una sola frecuencia de reloj, ya que al ser un proceso de cálculo paralelo, deben tener sincronía en las tareas de reconocimiento y entrenamiento, por ello el pin *clk* es aplicado a las dos estructuras. El pin *we* indica a ambas estructuras que se está haciendo un proceso de escritura de datos, cuando está en 1 (*we* = 1), lo cual les bloquea para realizar operaciones, en ese momento no se visualiza ningún valor, solo ceros a la salida. Cuando *we* está en cero (*we* = 0), esto habilita a las estructuras para realizar el calculo de las excitaciones en cada una de ellas, y determinar cual de ellas tiene la excitación máxima E_{max}. El pin *sel* se encarga de operar al multiplexor interno, de ambas estructuras, para continuar con el cálculo de la excitación, esto utilizando la entrada del multiplexor que tiene el valor fijo de "1". *sel* en cero (*sel* = 0) implica el paso del dato que viene de las memorias, en este caso, del exterior, y *sel* en uno (*sel* = 1) habilita el paso del valor fijo "1", predefinido internamente. Este pin *sel* es aplicado también a ambas estructuras.

De las salidas que estas estructuras muestran, solo son utilizadas aquellas que se relacionan con el bit de mayor valor, esto es, el bit que implica un resultado máximo en el cálculo de la excitación. De acuerdo con el diseño mostrado en el apartado anterior, la salida q16 se relaciona con este valor máximo, por lo que ambos pines de salida son lo utilizados para establecer cual de las dos estructuras tiene un valor máximo, es decir, cual de las dos clases está mostrando una excitación máxima E_{max}. El uso de estos dos pines de salida q16, ahora identificados como q0 y q1, permite detener el proceso de suma, cuando ya se ha obtenido un valor máximo de excitación, todo esto a través del operador OR, cuyas entradas corresponden a q0 y q1, y la salida implica una señal de detención del proceso, *alto_suma*.

Por otro lado, estas mismas salidas q16, ahora identificadas como q0 y q1, ayudan a establecer cual estructura es la que tiene la mayor excitación, esto significa que, muestran cual de las neuronas tiene la excitación máxima, indicando la clase ganadora; q0 representa la clase 1 y q1 la clase 2.

Para establecer cual estructura tiene la mayor excitación, se utiliza un ENCODER que muestra el numero de la clase, a partir de los datos aplicados por ambas q's, a dicho dispositivo, que se encarga de mostrar quien de ellas, q0 y q1, tiene la mayor excitación, y a que estructura, o neurona, corresponde dicha excitación, definiéndose así la clase ganadora.

Las Figuras 4.15 y 4.16 muestran simulaciones que permiten comprobar el adecuado comportamiento de este esquema electrónico, y la satisfacción del objetivo planteado por los algoritmos de trabajo para el reconocimiento y entrenamiento del clasificador neuronal LIRA.

En la Figura 4.15 se observa como la entrada de datos da[7..0] muestra un valor constante FF, el cual es sumado de forma continua, debido a que *sel* está en cero, como se observa en

la figura, lo que implica que se toma el dato de las entradas para el calculo de excitación. Para los datos en db[7..0] se aprecia que existe un valor constante en 00. La constante sumatoria del valor FF lleva a tener un resultado de esta que excede el tamaño del sumador, lo cual indica que hay un valor máximo, habilitando el pin q1 en este caso, asociado con la clase 2, con lo que se provoca que la salida *alto_suma* muestre un valor en "1", además de que el ENCODER indica que la clase que tiene una excitación máxima es la 2, es decir, la estructura que representa la neurona que define la clase 2 es la que tiene la mayor excitación.

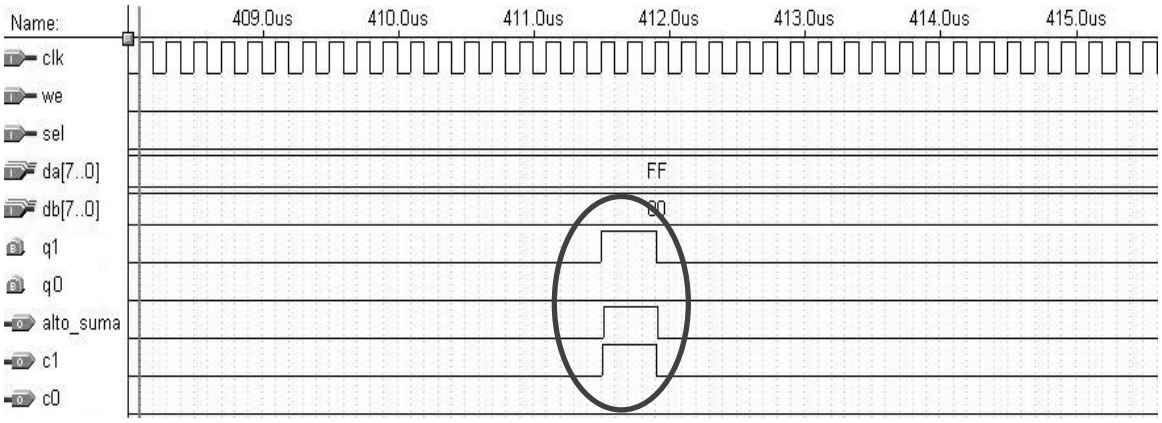

Figura 4.15. Reconocimiento de clase por el clasificador, Clase 2

En esta Figura 4.15 se aprecia como el proceso de sumatoria continua, de ahí que no se mantenga el identificador de la clase ganadora en un valor constante de 2. Esto se debe a que en la simulación no se detiene el proceso de sumatoria con la salida de control *alto_suma*, lo cual si debe de ocurrir en el proceso, ya que se ha obtenido una excitación máxima E_{max}. Posteriormente se procede con la siguiente etapa, el cálculo de errores en reconocimiento, o ajuste de pesos en entrenamiento.

Para la Figura 4.16 se observa algo similar a la anterior, pero ahora la entrada de datos db[7..0] es la que muestra un valor constante FF, que también se está sumado de forma continua, otra vez *sel* está en cero, como se puede apreciar en la figura, lo que implica que se toma el dato de las entradas para el calculo de excitación.

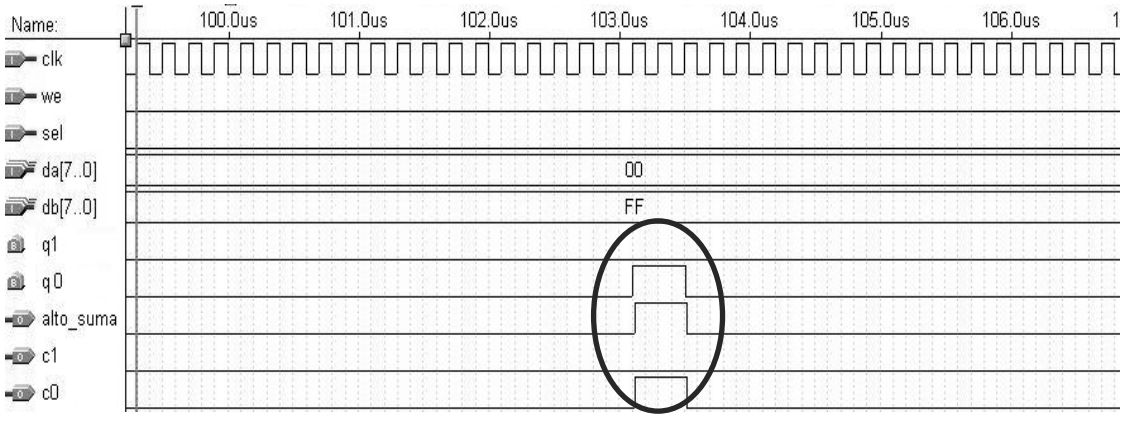

Figura 4.16. Reconocimiento de clase por el clasificador, Clase 1

Ahora, para los datos en da[7..0] se observa que existe un valor constante en 00. La constante sumatoria del valor FF lleva a tener un resultado de esta que excede nuevamente el tamaño del sumador, lo cual indica que hay un valor máximo, habilitándose, en este caso, al pin q0 que está asociado con la clase 1, lo que provoca que la salida *alto_suma* muestre un valor en "1" nuevamente, y además de que el ENCODER indique que la clase que tiene una excitación máxima es la 1, es decir, la estructura que representa la neurona que define la clase 1 es la que ahora tiene la mayor excitación.

También en esta Figura 4.16 se aprecia como el proceso de sumatoria continua, por lo que no se mantiene el identificador de la clase ganadora en un valor constante de 1. Esto es debido a que nuevamente en la simulación no se detiene el proceso de sumatoria con la salida de control *alto_suma*, pero que si debe de ocurrir en el proceso, puesto que se ha obtenido una excitación máxima E_{max}, continuándose con la siguiente etapa, el calculo de errores en reconocimiento, o ajuste de pesos en entrenamiento, del clasificador neuronal LIRA.

Una vez comprobado el correcto funcionamiento de este arreglo esquemático del clasificador neuronal LIRA para 2 clases, se procede con la implantación de dicho esquemático en el dispositivo programable, para comprobar su capacidad de operación, que ya ha mostrado en estos procesos de simulación.

4.2.1. Implantación física del esquema electrónico para dos clases

Para verificar y comprobar que el comportamiento del esquema electrónico mostrado, en modo de simulación, se puede desarrollar de forma física, dentro del dispositivo lógico programable con el que cuenta la tarjeta de desarrollo de Altera, Figura 4.17, a continuación se procede a programar dicho dispositivo, para posteriormente probar su funcionamiento adecuado.

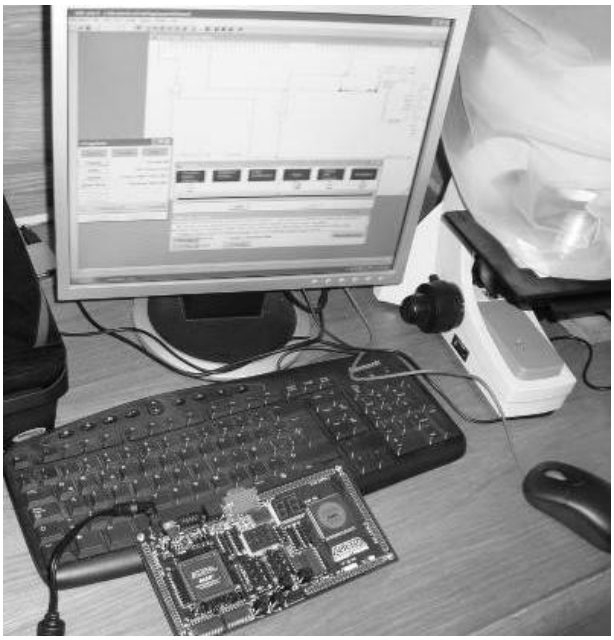

Figura 4.17. Programación del dispositivo en la tarjeta de desarrollo de Altera

La prueba consiste en aplicar valores en ambos grupos de pines de entrada, con el objeto de hacer la suma de los datos presentes en dichas entradas, los cuales se pueden considerar como los pesos que deben ser sumados para obtener la excitación de cada una de las neuronas, que representan las clases involucradas en los procesos de reconocimiento y entrenamiento, para posteriormente definir cual de ellas tiene la excitación máxima E_{max}, y con ello establecer la clase ganadora.

4.2.2. Resultados de la prueba para dos clases

En la Figura 4.18 se observa una de las pruebas, que consiste en aplicar datos en da[7..0] y en db[7..0], el primero es mayor en magnitud con respecto al segundo (Swtch7 y Swtch1 activos), por lo cual el indicador de la clase ganadora muestra que la clase que tiene una excitación máxima E_{max} es la clase 2.

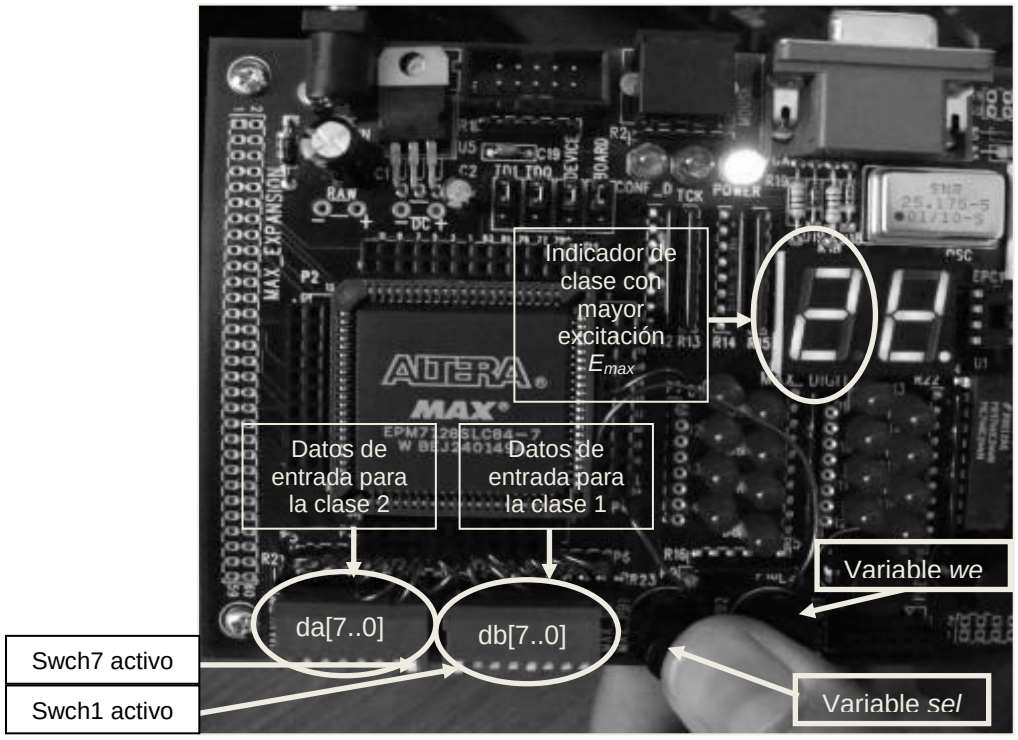

Figura 4.18. Prueba para la clase 2

Algo similar se puede apreciar en la Figura 4.19, pero ahora cambiando los datos de entrada. Ahora da[7..0] tiene un valor menor en magnitud con respecto al aplicado en db[7..0] (Swtch1 y Swtch7 activos), con esto, el indicador de la clase ganadora muestra que en esta ocasión quien tiene la excitación máxima E_{max} es la clase 1.

En ambas figuras, 4.18 y 4.19, se muestra que los interruptores de control *we* y *sel* están siendo presionados, esto permite que el proceso de suma se realice de forma adecuada.

Cuando *we* esta en nivel "0", no hay proceso de escritura en las memorias, lo que permite la sumatoria de los pesos almacenados en las mismas, de lo contrario, *we* = 1, el esquema electrónico se inhibe y no realiza operación alguna. Con *sel* en "0" se permite el paso del

dato proveniente de la memoria, si *sel* = 1, entonces se permite el paso del dato fijo "1", para determinar qué clase tiene la excitación máxima E_{max}.

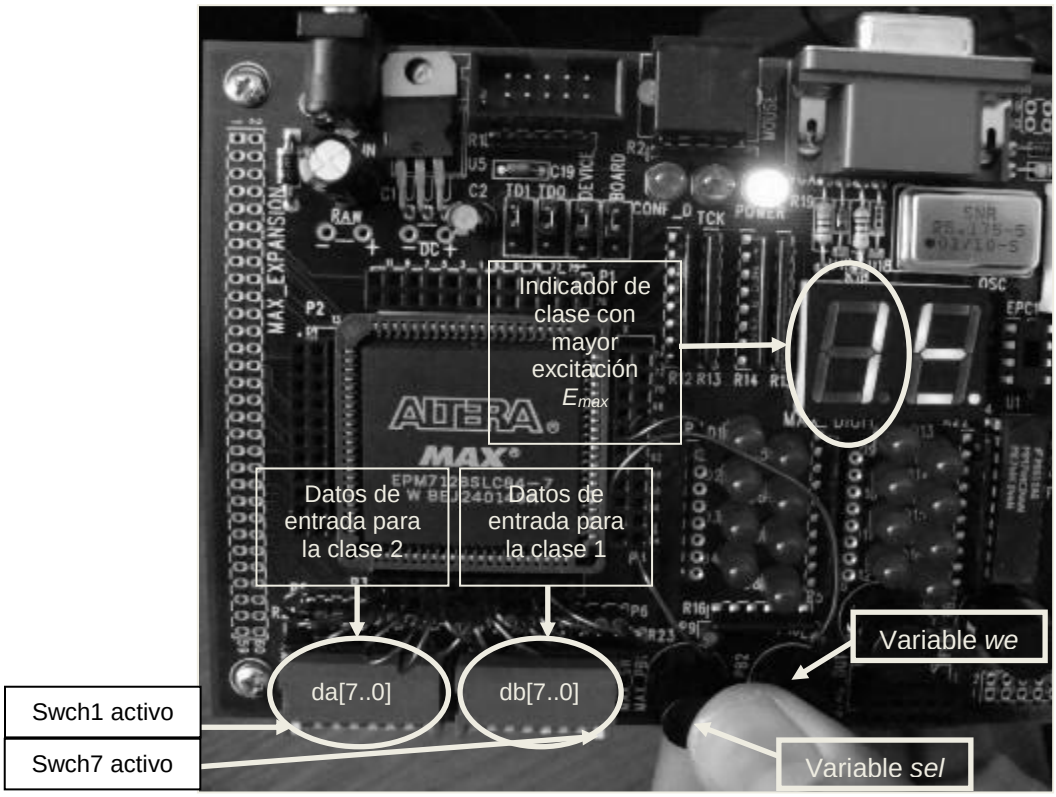

Figura 4.19. Prueba para la clase 1

Además de esto, se pretende medir el tiempo en el que este esquema realiza el cálculo de la suma, utilizando el cristal oscilador con el que cuenta dicha tarjeta de desarrollo, cuya frecuencia es de 25.175 MHz, Figura 4.20. Para realizar esta otra prueba se aplica un "1" en el pin de menor valor, solo en una de las dos clases del esquema, da[7..0] en la figura (Swtch1 activo). Considerando que el sumador es de 16 bits, más el bit de acarreo, se tiene un total de 17 bits. Con esto, se necesitan entonces 2^{16} = 65 536 eventos más 1, para lograr que el bit de mayor valor, el bit de acarreo del sumador, tenga un valor de "1", que es cuando se ha excedido el valor máximo de los 16 bits del sumador; este bit detiene el proceso de sumatoria, manteniéndose el dato de la clase que tiene la mayor excitación. El periodo de cada ciclo del reloj es de aproximadamente 39.73 ns, por lo que para lograr un valor en "1" en el acarreo del sumador es necesario un tiempo de 2.61 ms. En esta prueba se consideró el peor caso, que es el de sumar 65 537 veces un "1" para verificar qué tan rápido es el proceso. En este sentido, es claro que el tiempo, para el cálculo de la excitación de cada neurona por cada clase, es del orden de milisegundos.

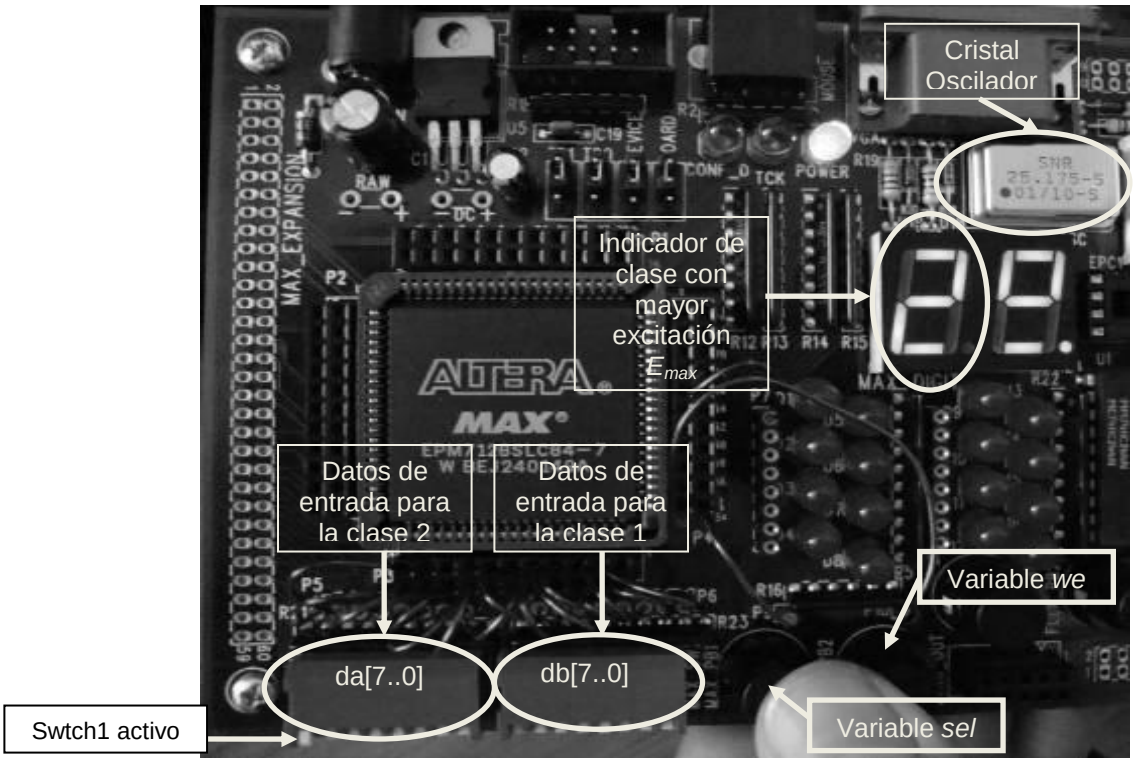

Figura 4.20. Prueba para medición del tiempo en el cálculo de excitación

Ahora, para el caso en el que los pesos a ser sumados tienen valores mayores a "1", y que solo algunos cuantos de ellos participan en el cálculo, pues todo esto está en función de ciertas combinaciones de las neuronas que están activas en la capa *A*, resulta evidente que la rapidez se ve incrementada, dado que ya no se suma un "1", sino valores de mayor peso a este, y solo un cierto número de ellos.

Si, además, agregamos que el calculo de la excitación se realiza de forma paralela para cada una de las clases involucradas en los procesos de reconocimiento y entrenamiento, entonces este tiempo de calculo de excitación es común para cada clase, con lo que se garantiza una instantaneidad en el calculo de las excitaciones de cada neurona, y por lo tanto, en la determinación de la clase ganadora, que es la que cuenta con la excitación máxima, E_{max}.

Cabe señalar que el proceso de sumatoria se realiza con una frecuencia de reloj de 25.175 MHz, que es la que maneja el cristal que Altera suministra en la tarjeta de desarrollo. En tarjetas de desarrollo más recientes los cristales que se suministran tienen frecuencias de reloj aun mayores, lo cual implica la posibilidad de mejorar los tiempos para realizar las sumatorias, es decir, el cálculo de excitación de cada neurona puede ser todavía más rápido.

El mayor tiempo consumido por el clasificador neuronal LIRA es en el proceso de entrenamiento, cuando los pesos de las conexiones entre las neuronas de las capas *A* y *R* cambian [83]. Este proceso necesita multiplicar el vector binario *A* con la matriz *W* para obtener el vector *R*, donde *k* es el tamaño del vector *A*. La matriz tiene ($k \times n$) elementos, donde *n* es el número de neuronas en la capa *R* y es igual al número de clases a ser reconocidas. La implementación de este algoritmo en una computadora implica realizar (*k*

× *n*) sumas. Utilizando una computadora Intel Pentium 4 de 32 bits a 1.5 GHz, la implementación de cada instrucción requiere de al menos 8 pulsos, se requieren 187.5 millones de instrucciones por segundo, es decir, $(1.5 * 10^9)/8$ instrucciones por segundo. La tarjeta de Altera, utilizada en esta investigación, requiere de un pulso por cada operación. Seleccionando *n* procesadores de Altera en paralelo, se realizan *n* operaciones de suma en paralelo [84]. Considerando la nueva tarjeta de desarrollo *Cyclone III LS FPGA* y sus diferentes frecuencias de oscilación, Apéndice A.1, y basados en un análisis con el procesador Cortex-M1, que permite conocer el espacio de uso de un FPGA pequeño, conservando la compatibilidad con el código del cualquier procesador ARM a partir del ARM7TDMI®, este proporciona 0.8 DMIPS por MHz. Con ello, se tienen 80 millones de instrucciones por segundo, es decir, 0.8 × 100 MHz. Estos cálculos muestran que si K = 2.34, es decir, menos de 3 clases a reconocer, el proceso en una computadora común es mas útil. Pero si K se incrementa a mas de 3, entonces la opción de la tarjeta de desarrollo de Altera es mas deseable para la implementación del clasificador neuronal LIRA. Al incrementar el número de clases se obtienen mejores condiciones para la implementación de los algoritmos de trabajo del clasificador LIRA con esquemas electrónicos. Por esta razón se implementa un nuevo clasificador para 5 clases en el nuevo kit de desarrollo *Cyclone III LS FPGA* de Altera, para acelerar los procesos de entrenamiento y reconocimiento [85].

Por lo anterior, a continuación se implementa el clasificador neuronal LIRA con la nueva tarjeta de desarrollo y para cinco clases.

4.3. Implementación del clasificador neuronal LIRA para cinco clases

En la siguiente Figura 4.21 se observa la implementación del esquema electrónico con cinco estructuras con la misma arquitectura, y cada una de ellas representa una clase sobre la cual el clasificador deberá clasificar las imágenes presentadas. De nuevo, debido al gran número de componentes involucrados en cada estructura, y para reducir el tamaño del esquema electrónico en el editor grafico del paquete de desarrollo de Altera, cada una de ellas se muestra como un bloque sencillo, pero su comportamiento es el que se mostró en el apartado anterior. Los datos a ser operados por este esquema electrónico del clasificador son suministrados a través de una computadora.

Todas estas estructuras tienen una entrada de 8 bits (da[7..0], db[7..0], dc[7..0], dd[7..0] y de[7..0] en la Figura 4.21), que implicaría la inyección de los datos, provenientes de las memorias, para el proceso del cálculo de excitaciones E_i, con i = 0, 1, 2, 3, 4, y de la excitación máxima E_{max}. Cada una de estas estructuras implica una neurona, que mostrará un cierto valor de excitación, que se asociará a la clase que esta representa.

También, están regidas por una sola frecuencia de reloj, ya que al ser un proceso de cálculo paralelo, deben tener sincronía en las tareas de reconocimiento y entrenamiento, por ello el pin *clk* es aplicado a todas las estructuras. El pin *we* indica a cada una de las estructuras que se está haciendo un proceso de escritura de datos, cuando está en 1 (*we* = 1), lo cual les bloquea para realizar operaciones, en ese momento no se visualiza ningún valor, solo ceros a la salida. Cuando *we* está en cero (*we* = 0), esto habilita a las estructuras para realizar el

calculo de las excitaciones en cada una de ellas, y determinar cual de ellas tiene la excitación máxima E_{max}.

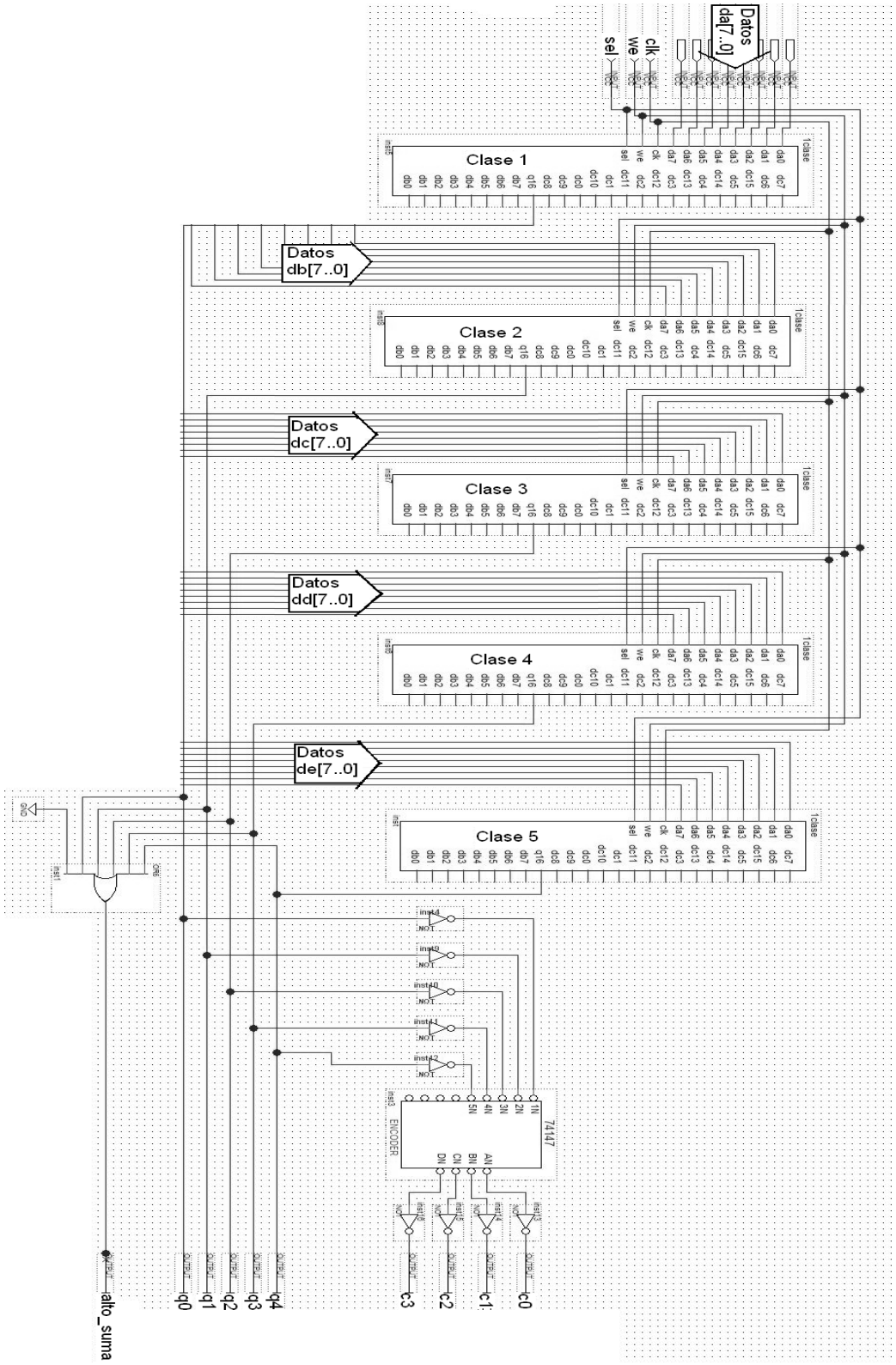

Figura 4.21. Esquema electrónico para cinco clases

El pin *sel* se encarga de operar al multiplexor interno, en cada una de las estructuras, para continuar con el cálculo de la excitación, esto utilizando la entrada del multiplexor que tiene el valor fijo de "1". *sel* en cero (*sel* = 0) implica el paso del dato que viene de las memorias, en este caso, del exterior, y *sel* en uno (*sel* = 1) habilita el paso del valor fijo "1", predefinido internamente. Este pin *sel* es aplicado también a las cinco estructuras.

De las salidas que estas estructuras muestran, solo son utilizadas aquellas que se relacionan con el bit de mayor valor, esto es, el bit que implica un resultado máximo en el cálculo de la excitación. De acuerdo con el diseño mostrado en el apartado anterior, la salida q16 se relaciona con este valor máximo, por lo que dichos pines de salida son los utilizados para establecer cual de las cinco estructuras tiene un valor máximo, es decir, cual de las cinco clases está mostrando una excitación máxima E_{max}. El uso de estos cinco pines de salida q16, ahora identificados como q0, q1, q2, q3 y q4 permite detener el proceso de suma, cuando ya se ha obtenido un valor máximo de excitación, todo esto a través del operador OR, cuyas entradas corresponden a q0, q1, q2, q3 y q4, y la salida implica una señal de detención del proceso, *alto_suma*.

Por otro lado, estas mismas salidas q16, ahora identificadas como q0, q1, q2, q3 y q4, ayudan a establecer cual estructura es la que tiene la mayor excitación, esto significa que, muestran cual de las neuronas tiene la excitación máxima, indicando la clase ganadora; q0 representa la clase 1, q1 la clase 2, q2 la clase 3, q3 la clase 4, y q4 la clase 5.

Para establecer cual estructura tiene la mayor excitación, se utiliza un ENCODER que muestra el numero de la clase, a partir de los datos aplicados por cada q, a dicho dispositivo, que se encarga de mostrar quien de ellas, q0, q1, q2, q3 y q4, tiene la mayor excitación, y a que estructura, o neurona, corresponde dicha excitación, definiéndose así la clase ganadora.

La Figura 4.22 muestra la simulación que permite comprobar el adecuado comportamiento de este esquema electrónico, y la satisfacción del objetivo planteado por los algoritmos de trabajo para el reconocimiento y entrenamiento del clasificador neuronal LIRA.

En esta Figura 4.22 se observa como la entrada de datos da[7..0] muestra un valor constante 1111 1111 (FF_{hex}), la entrada db[7..0] tiene un valor constate de 1111 0000 ($F0_{hex}$), la entrada dc[7..0] tiene un dato constante de 0000 1111 ($0F_{hex}$), la entrada dd[7..0] tiene un dato de 1000 0000 (80_{hex}), y la entrada de[7..0] tiene un dato de 0000 0001 (01_{hex}), todos ellos en formato binario. Cada valor es sumado de forma continua, debido a que *sel* está en cero, como se observa en la figura, lo que implica que se toma el dato de las entradas para el calculo de excitación. La constante sumatoria de cada valor lleva a tener un resultado de esta que excede el tamaño del sumador, lo cual indica que hay un valor máximo. En primer lugar, se habilita al pin q0, asociado con la clase 1, como se observa en la Figura 4.23 a más detalle.

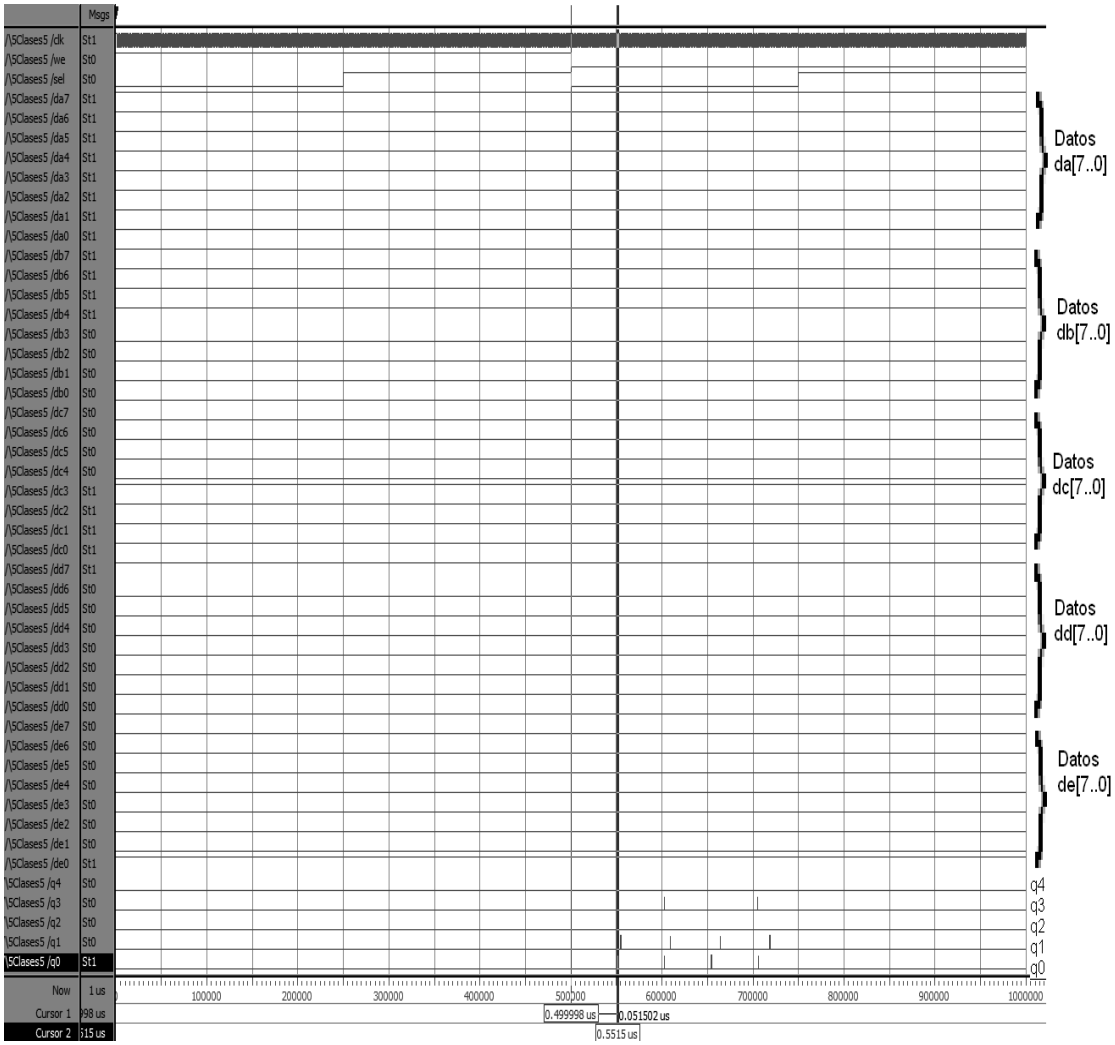

Figura 4.22. Reconocimiento de clases por el clasificador

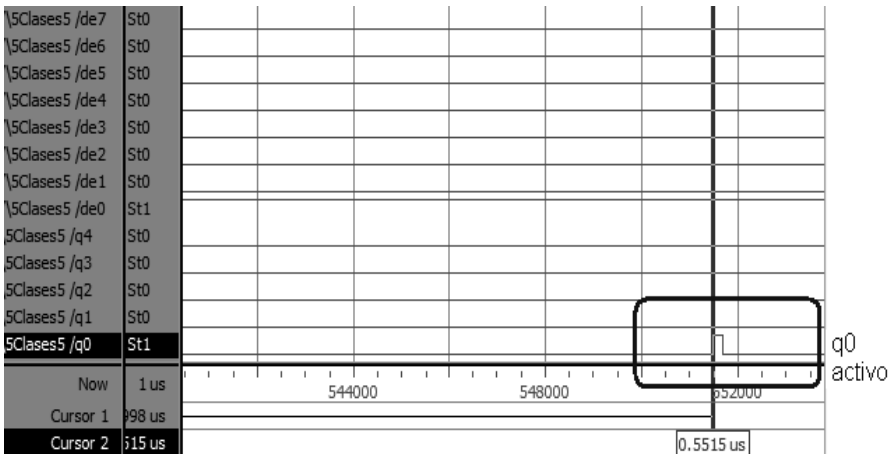

Figura 4.23. Reconocimiento de clase, clase 1

Posteriormente, se habilita el pin q1, correspondiente a la clase 2, como se puede apreciar a detalle en la Figura 4.24.

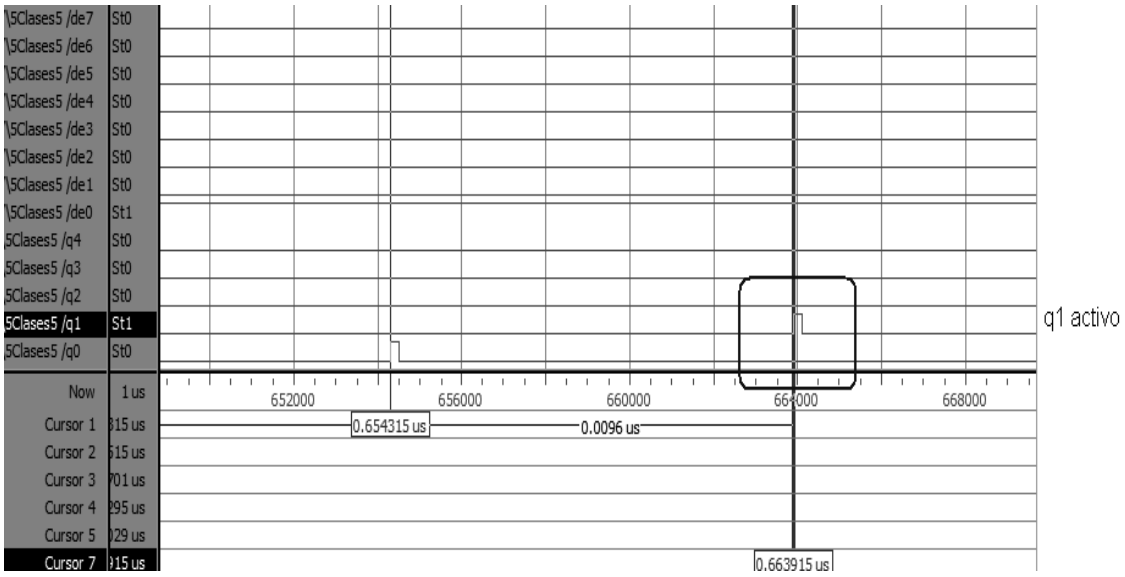

Figura 4.24. Reconocimiento de clase, clase 2

La siguiente clase que se habilita es la 4, que se corresponde con el pin q3, lo cual se observa en la Figura 4.25. También se observa como nuevamente el pin q0 se activa, puesto que el proceso de sumatoria se mantiene constante durante todo el periodo de la simulación.

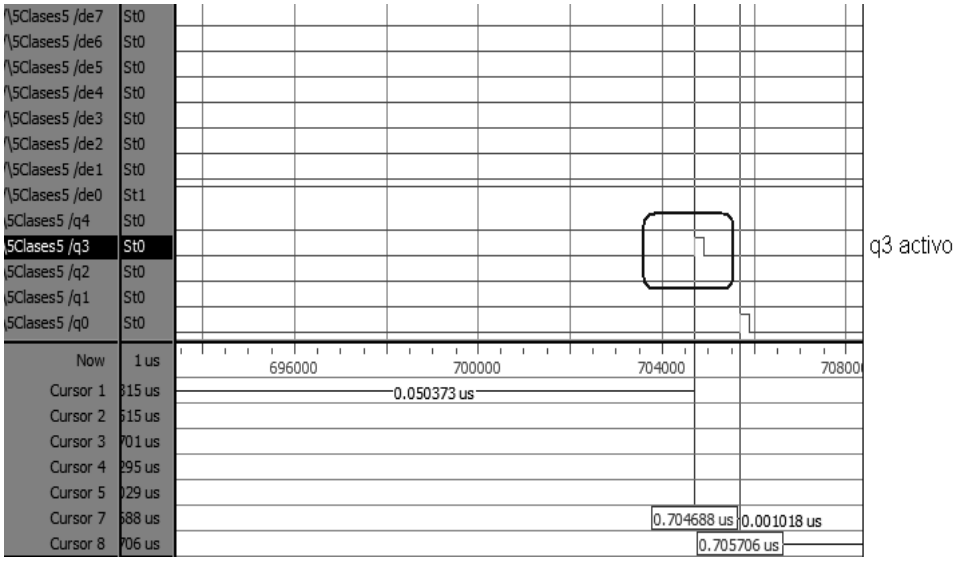

Figura 4.25. Reconocimiento de clase, clase 4

En la Figura 4.26 se observa un periodo de tiempo en el cual no existe activación de alguno de los pines de cada clase, esto responde al hecho de que el pin *sel*, que se encarga de operar al multiplexor interno dentro de cada una de las estructuras, en un periodo se encuentra en uno (*sel* = 1), lo que habilita el paso del valor fijo "1", predefinido

internamente para continuar con el cálculo de la excitación, esto provoca que se requiera mas tiempo para lograr activar alguno de los pines q. En la figura este tiempo es insuficiente para lograr estas activaciones. Si *sel* está en cero (*sel* = 0) implica el paso del dato que viene del exterior, lo que produce la actividad de los pines q.

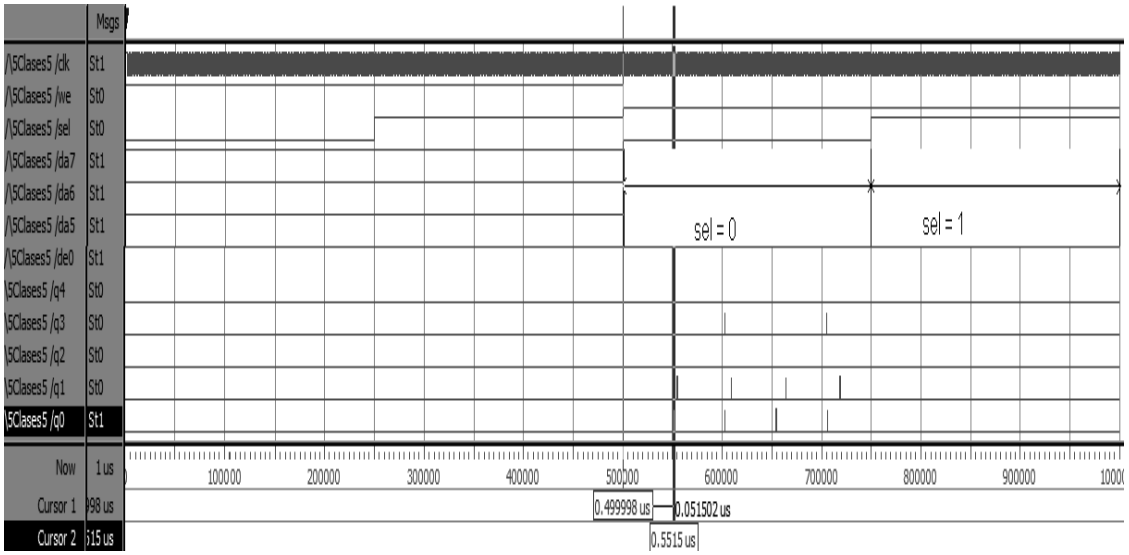

Figura 4.26. Comportamiento de sel

Para lograr la activación del pin q2 fue necesario cambiar los estados de las líneas de control en la simulación, es decir, *we* y *sel* fueron puestas a cero para garantizar que la constante sumatoria de 0F lograra activar al pin q2, correspondiente a la clase 3, como se muestra en la Figura 4.27.

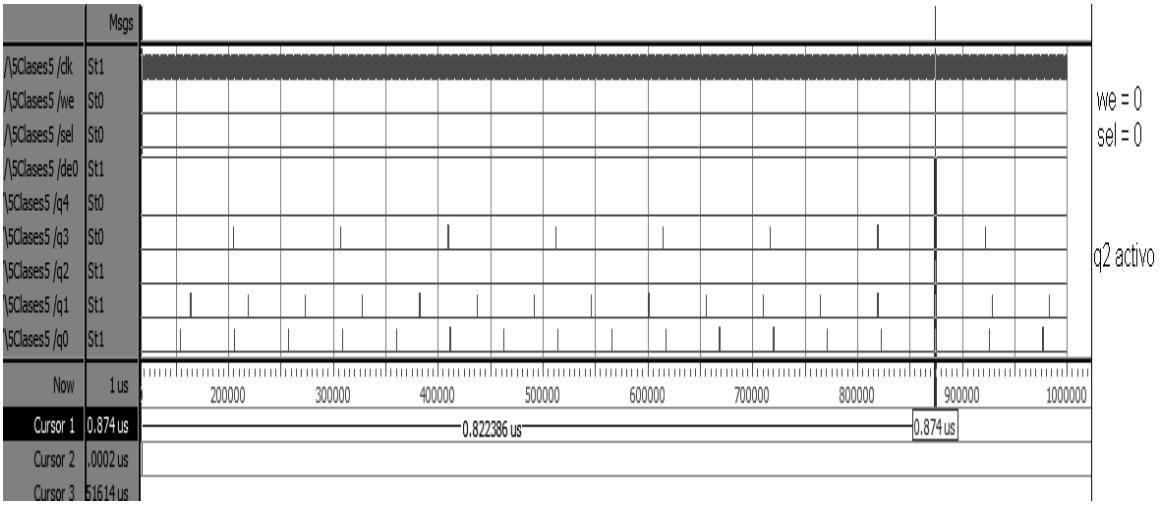

Figura 4.27. Reconocimiento de clase, clase 3

En esta Figura 4.27 se puede observar que el periodo requerido para poder activar el pin q2 es mucho mayor que para los pines q0, q1, y q3, ya que estos tienen una actividad más

frecuente, como se aprecia en la figura. En la Figura 4.28 se observa más a detalle la actividad del pin q2.

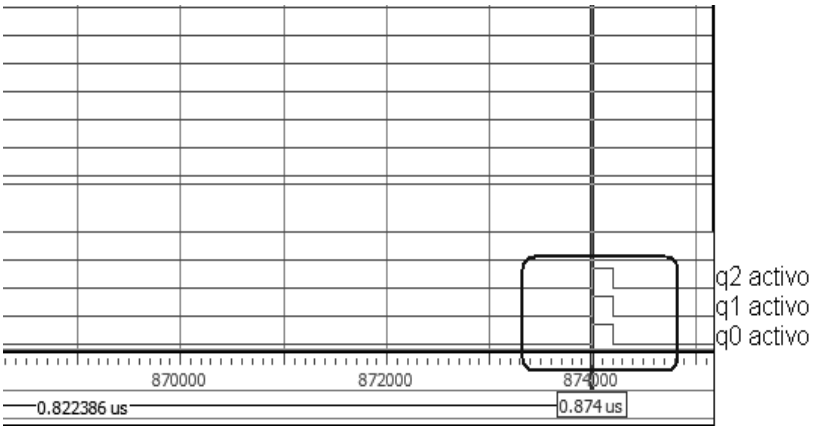

Figura 4.28. Reconocimiento de clase, clase 3

Finalmente, para poder activar al pin q4 fue necesario cambiar, además de los estados de las líneas de control en la simulación, el periodo de la simulación, es decir, *we* y *sel* fueron puestas a cero, y se incrementó el tiempo de simulación para garantizar que la constante sumatoria de 01 lograra activar al pin q4, correspondiente a la clase 5, como se muestra en la Figura 4.29.

Es notorio el hecho de que el periodo requerido para poder activar el pin q4 es mucho mayor que para los pines q0, q1, q2 y q3, que como se aprecia tienen una frecuencia de activación mayor, como se observa en la Figura 4.29. En la Figura 4.30 se observa más a detalle la actividad del pin q4.

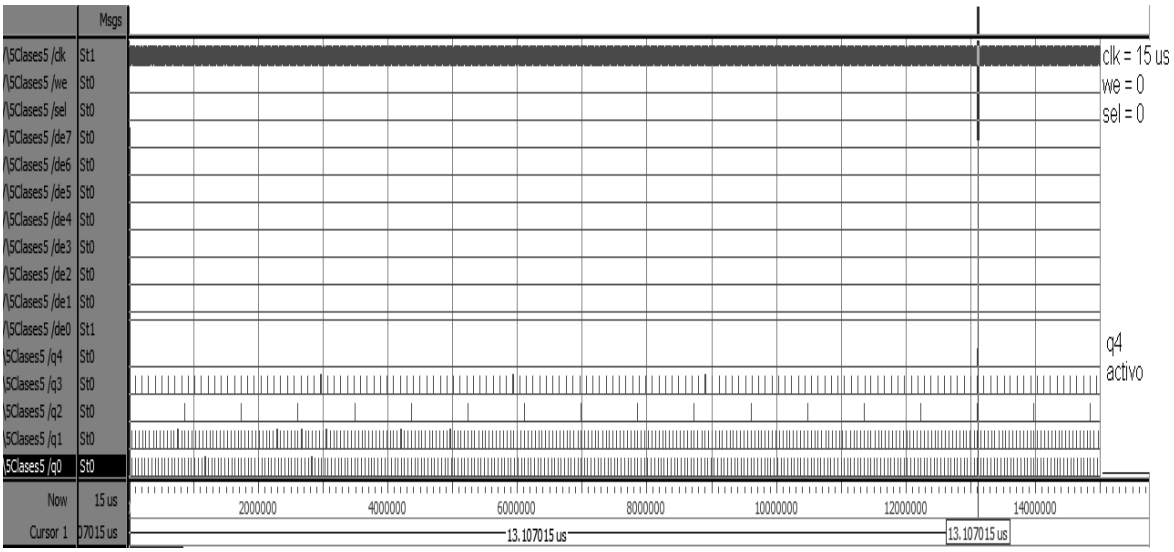

Figura 4.29. Reconocimiento de clase, clase 5

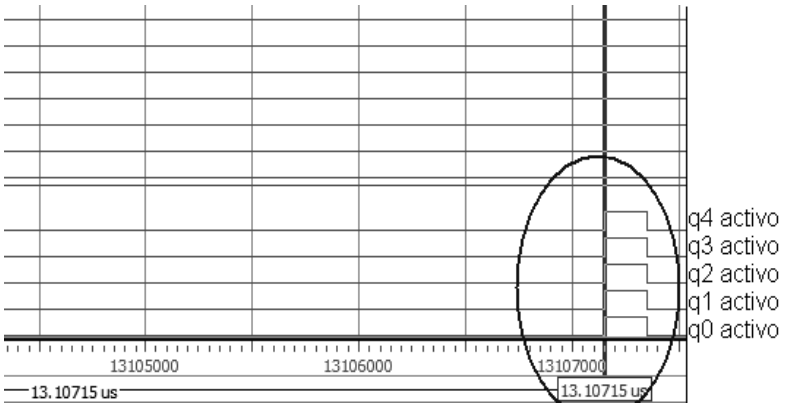

Figura 4.30. Reconocimiento de clase, clase 5

En la Figura 4.21 se observa también una compuerta OR de 6 entradas, de las cuales solo se utilizan 5. Esta compuerta opera la salida *alto_suma*, la cual muestra un valor en "1" cuando alguna de sus entradas, q0, q1, q2, q3 o q4, está en alto, es decir, cuando alguna de las neuronas presenta un valor máximo. En la Figura 4.31 se observa como al activarse las salidas q3, q0 y q2, respectivamente, hacen que la salida *alto_suma* tome un valor de "1".

Por otro lado, el ENCODER, mostrado también en la Figura 4.21, indica cual de las clases del esquema electrónico tiene la excitación máxima, es decir, indica el número de la estructura, que representa la neurona, que tiene la mayor excitación. Nuevamente, la Figura 4.31 muestra el número, en binario, de la clase, o neurona, que tiene la excitación en ese momento, representado con las salidas c3, c2, c1 y c0. Para este caso, es necesario mencionar que q0 se relaciona con la clase 1, q1 con la clase 2, q2 con la 3, q3 con la 4, y q4 con la clase 5. De lo anterior se establece entonces que cuando q3 se activa, c2 también lo hace, es decir, hay un 0100; la clase 4 está activa. Cuando q0 está activa, también lo está c0, esto es, 0001; la clase 1 está activa. Y cuando q1 se activa, también c1 lo hace, en este caso, 0010; la clase 2 se activa.

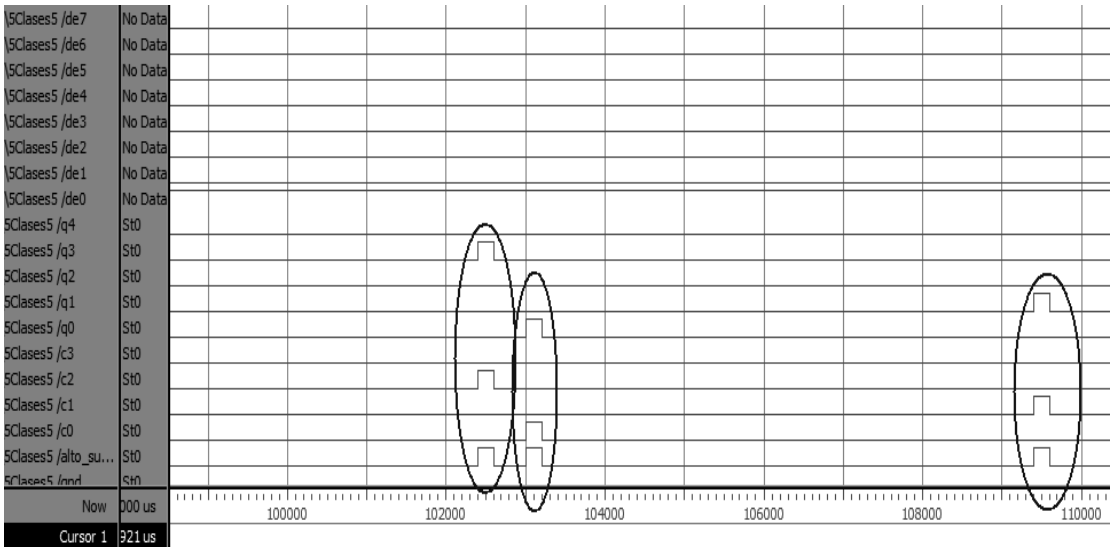

Figura 4.31. Activación de salidas del codificador y de la compuerta OR

En la Figura 4.32 se aprecia cómo el proceso de sumatoria continúa, de ahí que no se mantenga el identificador de la clase ganadora en un valor constante. Esto se debe a que en la simulación no se detiene el proceso de sumatoria con la salida de control *alto_suma*, lo cual si debe de ocurrir en el proceso, ya que se ha obtenido una excitación máxima E_{max}. Posteriormente se procede con la siguiente etapa, el cálculo de errores en reconocimiento, o ajuste de pesos en entrenamiento.

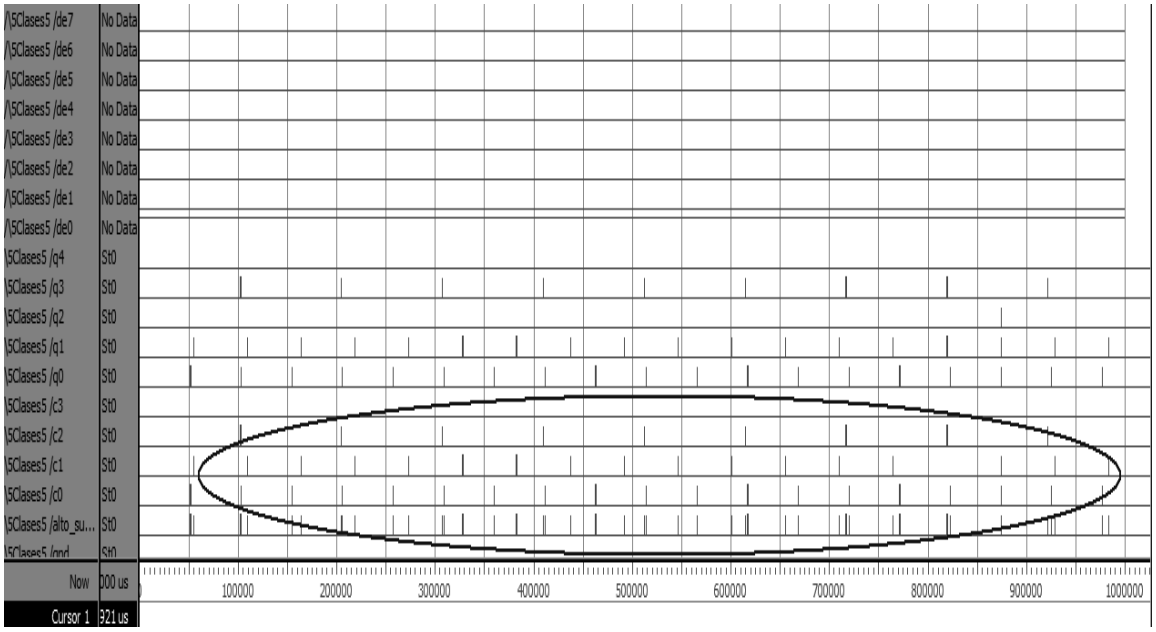

Figura 4.32. Proceso continuo de sumatoria. Variación de datos en compuerta OR y ENCODER

Una vez comprobado el correcto funcionamiento de este arreglo esquemático del clasificador neuronal LIRA ahora para 5 clases, se procede con la implantación de dicho esquemático en el dispositivo programable, para comprobar su capacidad de operación, mostrada ya en estos procesos de simulación.

4.3.1. Implantación física del esquema electrónico para cinco clases

Para verificar y comprobar que el comportamiento del esquema electrónico mostrado, en modo de simulación, se puede desarrollar de forma física, dentro del dispositivo lógico programable con el que cuenta la tarjeta de desarrollo de Altera, Figura 4.33, a continuación se procede a programar dicho dispositivo, para posteriormente probar su funcionamiento adecuado.

La prueba consiste en aplicar valores a los grupos de pines de entrada, con el objeto de hacer la suma de los datos presentes en dichas entradas, los cuales se pueden considerar como los pesos que deben ser sumados para obtener la excitación de cada una de las neuronas, que representan las clases involucradas en los procesos de reconocimiento y entrenamiento, para posteriormente definir cual de ellas tiene la excitación máxima E_{max}, y con ello establecer la clase ganadora.

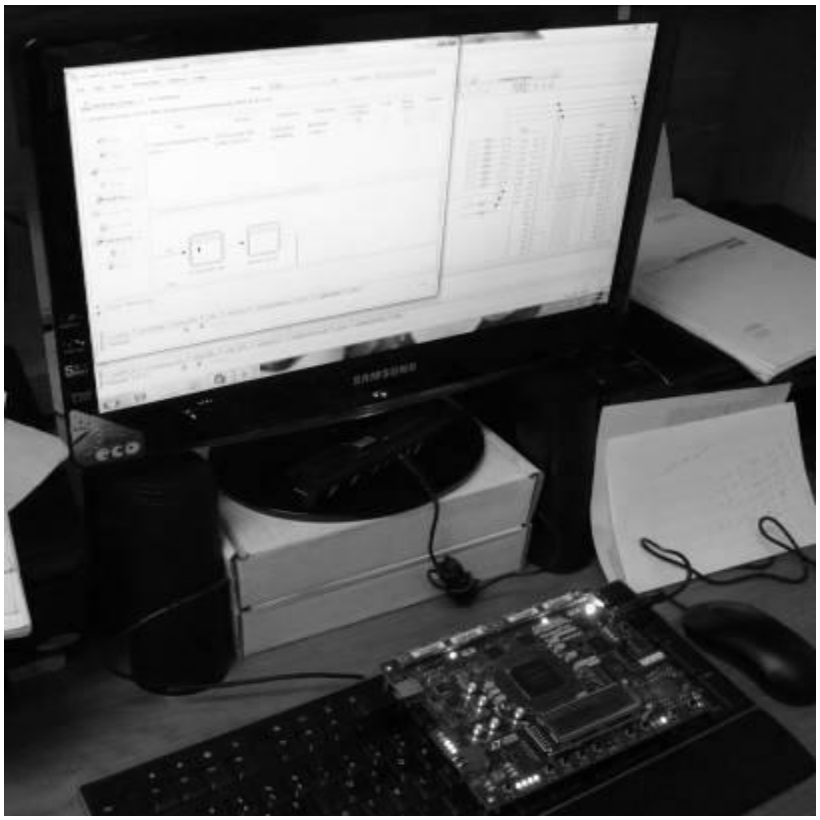

Figura 4.33. Programación del dispositivo en la tarjeta de Altera

En la Figura 4.34 se muestran los elementos utilizados para esta prueba. Se tienen 4 LEDs, de los cuales 3 muestran el número de la clase cuya neurona tiene la excitación máxima, E_{max}. Y el otro LED muestra que el proceso de suma se ha detenido pues se tiene una excitación máxima.

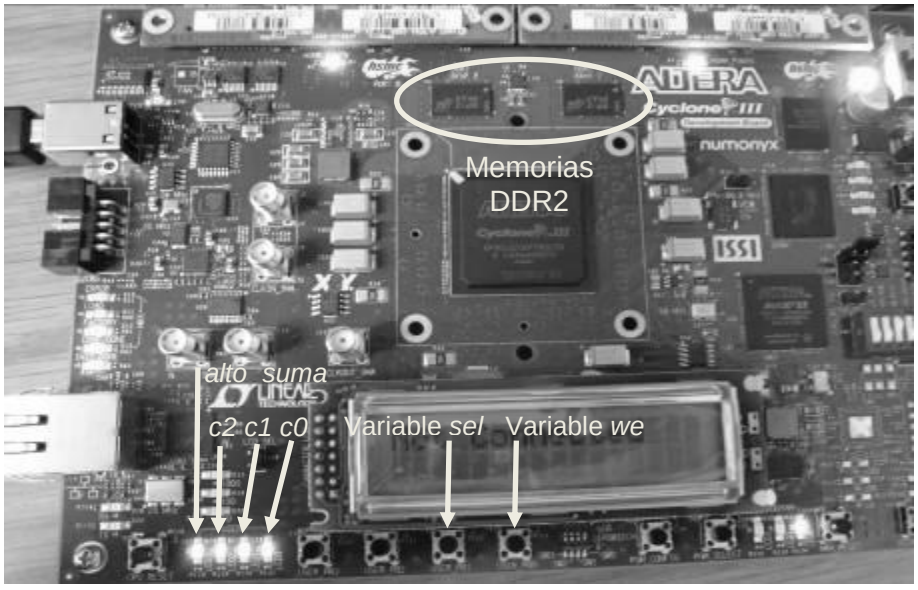

Figura 4.34. Elementos utilizados en la prueba

Cabe señalar que estos LEDs son activos bajos, es decir, indican los resultados en forma inversa, con valores de cero o apagado. También se muestran los switches que corresponden a las variables *sel* y *we*, que se encargan de inyectar el uno constante para el cálculo de la excitación máxima, y de deshabilitar el sistema cuando se están escribiendo datos en las memorias, respectivamente. También se observan las memorias DDR2 que se encargarán de almacenar, en *we* = 1, e inyectar, en *we* = 0, los datos a ser operados por el dispositivo programable para definir la neurona con la mayor excitación, y determinar la clase ganadora.

4.3.2. Resultados de la prueba para cinco clases

La prueba consiste en aplicar datos en da[7..0], db[7..0], dc[7..0], dd[7..0] y de[7..0] a través del grupo de memorias DDR2. Esto se logra a través de la conexión entre la computadora y la tarjeta, pues existe la posibilidad de manipular los datos dentro de las memorias utilizando el programa QUARTUS II, en su aplicación de programador de dispositivos.

En las siguientes figuras se observan los cambios en las salidas c2, c1 y c0, de acuerdo con los datos, provenientes de las memorias, que son ingresados a cada estructura electrónica para el cálculo de las excitaciones E_i, y muestran el numero de la neurona que tiene la excitación máxima, E_{max}, que indica a la neurona ganadora. Además de esto, el LED de *alto_suma* revela si se detiene el proceso de suma pues se ha llegado a un valor máximo de excitación.

En la Figura 4.35 se observa la primer situación, en la cual ningún LED esta apagado, esto es porque no hay datos a ser operados, y por lo tanto, no hay valor máximo de excitación. Cabe recordar que es necesario presionar los switches *sel* y *we* para que el circuito electrónico opere.

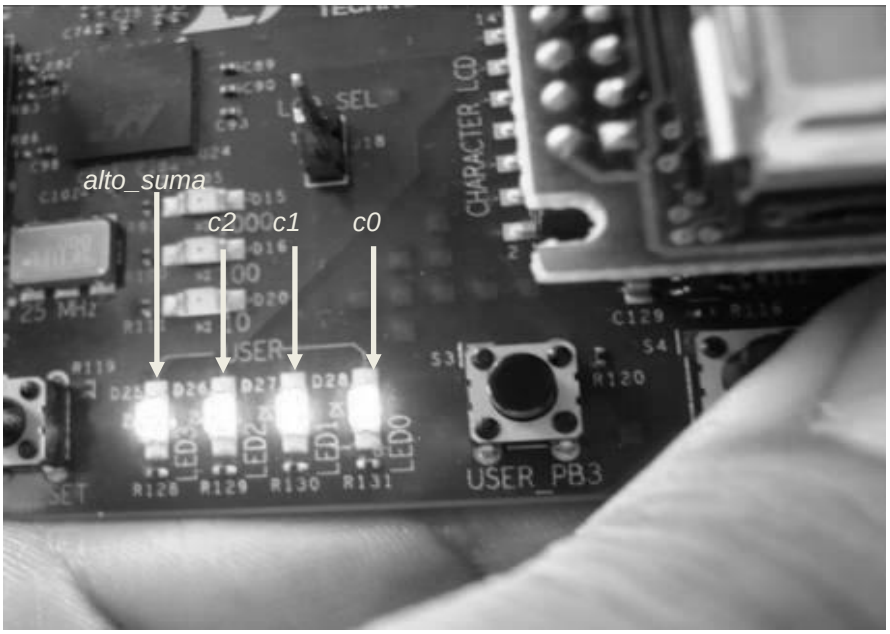

Figura 4.35. No hay excitación máxima ni clase ganadora

En las Figuras 4.36, 4.37, 4.38, 4.39 y 4.40, se observan las variaciones del número de la clase ganadora, es decir, de la neurona con la excitación máxima, E_{max}. Esto se logra adecuando los datos en las memorias que van a ser operados, de tal manera que se logre la excitación de una neurona en particular.

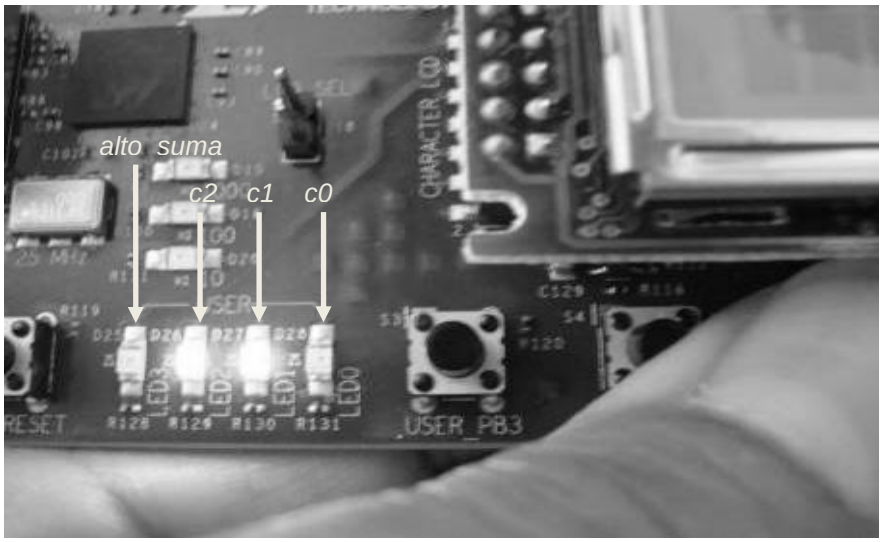

Figura 4.36. Prueba para la clase 1

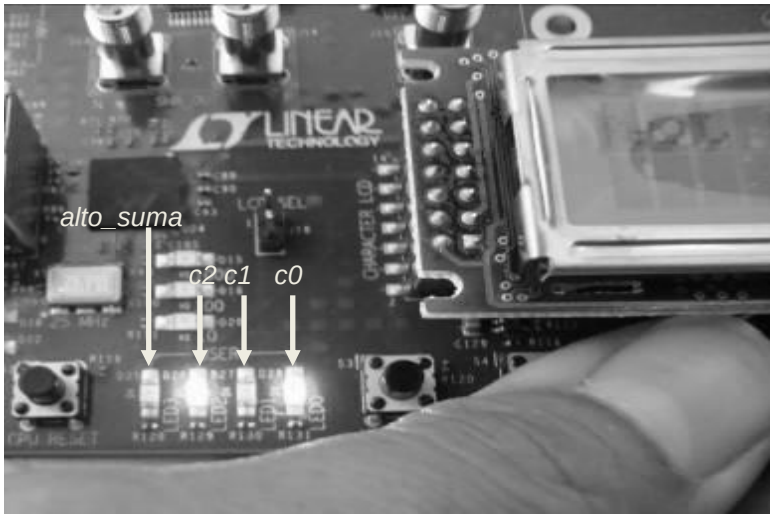

Figura 4.37. Prueba para la clase 2

Nuevamente se remarca el hecho de que los LEDs son activos bajos, es decir, indican los resultados en forma inversa, con valores de cero o apagado. Por esta razón el LED de *alto_suma* se apaga cuando se detiene la suma pues se ha llegado a un valor máximo. Y para los LEDs que representan el número de la clase ganadora, c2, c1 y c0, se apagan para hacer la combinación binaria correspondiente, en estado bajo.

En todas las figuras antes mencionadas se muestra que los interruptores de control *we* y *sel* están siendo presionados, esto permite que el proceso de suma se realice de forma

adecuada. Cuando *we* esta en nivel "0", no hay proceso de escritura en las memorias, lo que permite la sumatoria de los pesos almacenados en las mismas, de lo contrario, *we* = 1, el esquema electrónico se inhibe y no realiza operación alguna. Con *sel* en "0" se permite el paso del dato proveniente de la memoria, si *sel* = 1, entonces se permite el paso del dato fijo "1", para determinar qué clase tiene la excitación máxima E_{max}.

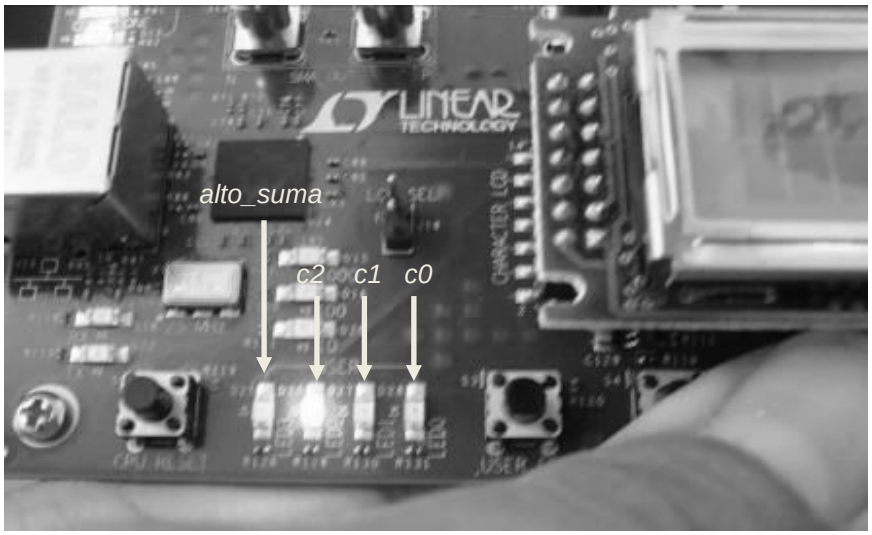

Figura 4.38. Prueba para la clase 3

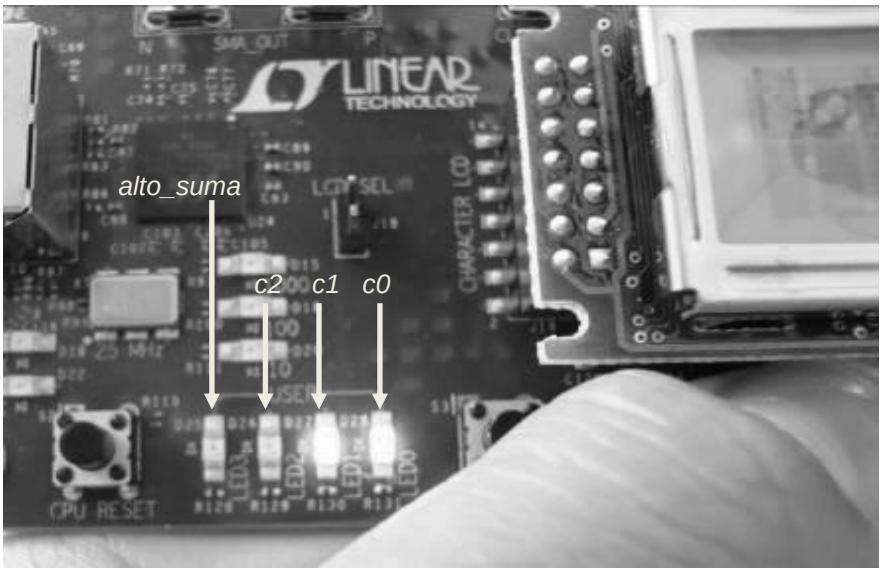

Figura 4.39. Prueba para la clase 4

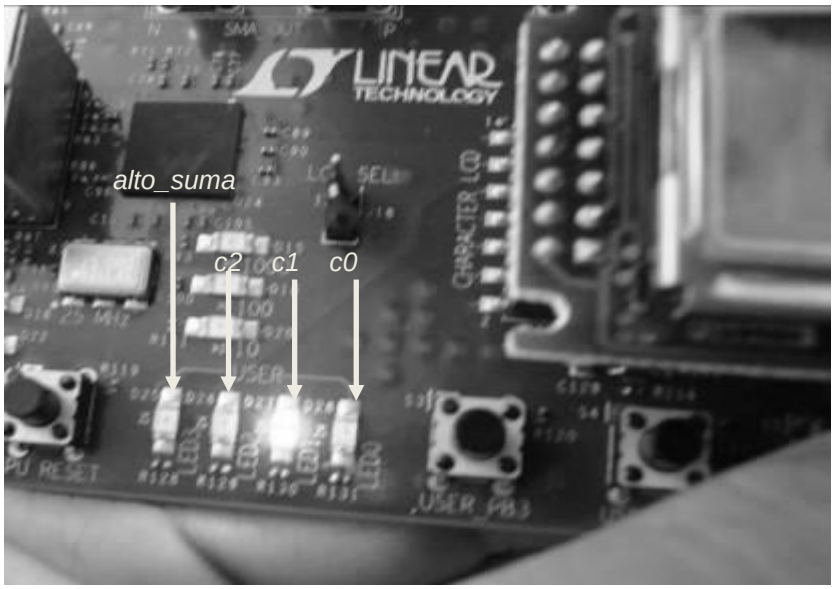

Figura 4.40. Prueba para la clase 5

Dadas las características novedosas de esta tarjeta de desarrollo, se utiliza uno de los osciladores con los que cuenta. Cabe señalar que son varios de estos dispositivos con diferentes frecuencias, 4 de 50, 66.6, 100 MHz, y uno mas programable, de entre 100, 125, 150 y 156.5 MHz, con una frecuencia básica de 125 MHz. Con esto, se pretende medir el tiempo en el que este esquema realiza el cálculo de la suma, utilizando el cristal oscilador de 100 MHz. Para realizar esta otra prueba se aplica un "1" en el pin de menor valor de[0] de forma manual, solo en la clase 5, Figura 4.41.

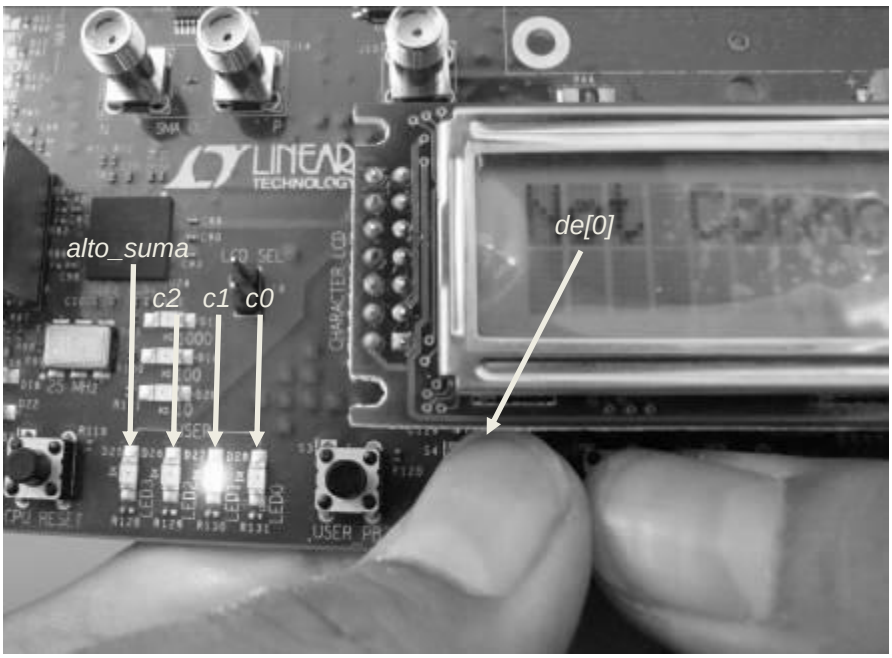

Figura 4.41. Prueba de rapidez de operación

Considerando que el sumador es de 16 bits, más el bit de acarreo, se tiene un total de 17 bits. Con esto, se necesitan entonces 2^{16} = 65 536 eventos más 1, para lograr que el bit de mayor valor, el bit de acarreo del sumador, tenga un valor de "1", que es cuando se ha excedido el valor máximo de los 16 bits del sumador; este bit detiene el proceso de sumatoria, manteniéndose el dato de la clase que tiene la mayor excitación. El periodo de cada ciclo del reloj es de aproximadamente 10 ns, por lo que para lograr un valor en "1" en el acarreo del sumador es necesario un tiempo de 0.655 ms. En esta prueba se consideró el peor caso, que es el de sumar 65 537 veces un "1" para verificar qué tan rápido es el proceso. En este sentido, es claro que el tiempo, para el cálculo de la excitación de cada neurona por cada clase, es del orden de microsegundos.

Ahora, para el caso en el que los pesos a ser sumados tienen valores mayores a "1", y que solo algunos cuantos de ellos participan en el cálculo, pues todo esto está en función de ciertas combinaciones de las neuronas que están activas en la capa *A*, resulta evidente que la rapidez se ve incrementada, dado que ya no se suma un "1", sino valores de mayor peso a este, y solo un cierto número de ellos.

Si, además, agregamos que el calculo de la excitación se realiza de forma paralela para cada una de las clases involucradas en los procesos de reconocimiento y entrenamiento, entonces este tiempo de calculo de excitación es común para cada clase, con lo que se garantiza una instantaneidad en el calculo de las excitaciones de cada neurona de la salida, y por lo tanto, en la determinación de la clase ganadora, que es la que cuenta con la excitación máxima, E_{max}.

4.4. Resultados

De acuerdo con los resultados obtenidos en la realización de los anteriores experimentos con los clasificadores para dos y cinco clases, se puede establecer que un clasificador neuronal implementado con alguno de estos dispositivos programables tiene una alta posibilidad de acelerar el proceso de reconocimiento, ya que sus estructuras internas, iguales y en paralelo, les permiten operar los datos que les son aplicados para poder obtener la excitación de cada neurona, E_i, y posteriormente definir cual de ellas tiene el valor máximo de excitación, E_{max}. Esto lo permite precisamente el procesamiento en paralelo de la información suministrada a cada estructura, y además operando todas a la misma velocidad de trabajo.

Ambos clasificadores tienen tal capacidad de operación y calculo, tanto de excitaciones individuales para cada clase, como del valor máximo total. La diferencia entre ellos es solamente generacional, puesto que la tarjeta de desarrollo del programa universitario de años anteriores cuenta con componentes electrónicos programables, con una escala de integración adecuada, que aun les permite realizar tareas de acuerdo con las ventajas que implican tales tarjetas. Pero esto no implica que el clasificador no realice adecuadamente su tarea, puesto que sí es capaz de alojar un clasificador neuronal de unas cuentas clases, específicamente dos, y de hacer las operaciones requeridas para la clasificación. Los pequeños inconvenientes son solo la poca interconectividad y la única opción de temporización que tiene, con un oscilador de 25.175 MHz, lo cual implica un periodo de

operación de 2.61 ms, para la prueba realizada considerando el peor caso sumando un "1" consecutivamente, en el calculo de la excitación máxima.

Por otro lado, la tarjeta de nueva generación cuenta con una mayor conectividad, pues puede interactuar con el programa de aplicación y su ambiente de trabajo, que es totalmente nuevo y mejorado, adecuado a las nuevas tarjetas de desarrollo y las nuevas tecnologías de los dispositivos programables que el proveedor tiene en el mercado, y en tiempo real, además cuenta con varios osciladores que permiten realizar las operaciones del clasificador en un menor periodo, específicamente con un tiempo de 0.655 ms, a 100 MHz, para el mismo caso de suma continua de un "1". Cabe señalar que en este ultimo experimento, 5 clases, el porcentaje de uso del dispositivo es de un poco menos del 1%, lo que implica e invita a explotarlo aun mas, realizando mas pruebas con un numero mayor de clases a reconocer, es decir, un numero mas grande estructuras internas que permitan una mejor precisión en la clasificación de imágenes. En este mismo sentido, para el caso del clasificador de dos clases en la tarjeta de versión anterior, el porcentaje de uso es del 3%, lo que sustenta el hecho de que esta tarjeta todavía, y también, es útil para seguir realizando pruebas con la implantación de clasificadores neuronales en el dispositivo programable con el que cuenta.

La estructura del clasificador es la misma, el diseño y los elementos utilizados también, solo se cambió la plataforma de trabajo, tanto de tarjeta de desarrollo como de programas de aplicación, para poder evaluar la capacidad que tienen los nuevos dispositivos programables de alojar un clasificador neuronal con el objetivo de implementarlo completamente, con sus algoritmos de trabajo, sus capacidades y aplicaciones, con una frecuencia de operación mucho mayor y un trabajo totalmente paralelizado.

Conclusiones

La implementación de clasificadores neuronales es uno de los quehaceres en el desarrollo de tareas relacionadas al reconocimiento de patrones, particularmente en la clasificación de imágenes. Existen y son investigados diferentes tipos de redes neuronales, de acuerdo con su constitución interna y por consecuencia por su modo de operación, empleados en diversas aplicaciones.

La implementación de estos clasificadores neuronales ha sido presentada, la cual se lleva a cabo a través de los dispositivos lógicos programables, los FPGAs. Estos FPGAs cuentan con arquitecturas internas que permiten el desarrollo de cualquier elemento que sea necesario para realizar operaciones lógicas a escalas muy grandes con frecuencias de trabajo cada vez más altas. Esto permite el alojamiento de un clasificador en un dispositivo de esta naturaleza.

Este trabajo de investigación se enfocó en dos conceptos. Por un lado, el clasificador neuronal de área receptiva limitada, LIRA, basado en los principios del perceptron de Frank Rosenblatt, que tiene características que permiten su implementación sin utilizar operaciones complejas, es decir, no demasiados recursos, y por otro lado, los dispositivos lógicos programables, tanto los complejos, CPLDs, como los arreglos de compuertas programables por campo, FPGAs, los cuales cuentan con arquitecturas internas que permiten la implementación de un clasificador neuronal de las características del LIRA.

Los resultados obtenidos a lo largo de este trabajo muestran la posibilidad de implementar un clasificador con un número fijo de clases a ser reconocidas, lo cual precisamente permite la arquitectura de estos dispositivos programables. Además de ello, y gracias al continuo avance que tiene el campo de la tecnología desde la perspectiva de las altas escalas de integración en referencia a componentes electrónicos, este número de clases a reconocer se puede incrementar, de acuerdo con las necesidades particulares de cada tarea de clasificación. Otro elemento importante es la velocidad con la cual estos dispositivos electrónicos de nueva generación pueden operar, ya que esto permite acelerar las tareas de entrenamiento y reconocimiento del clasificador.

Ambas características antes mencionadas dan a este nuevo clasificador neuronal implementado electrónicamente la posibilidad de ser considerado como un nuevo instrumento en las tareas de clasificación de imágenes.

Cada clase a ser reconocida implica una neurona de salida, es decir, una sección para calcular la excitación E_i de dicha neurona. Un clasificador neuronal es mas preciso si se tienen mas clases a ser reconocidas, es decir, muchas clases relacionadas a las características que se requieren clasificar. De acuerdo con esto, se puede decir que una clase, o neurona de salida, implica un proceso, por lo que para reconocer muchas clases, se requieren muchos procesos, esto significa entonces que un dispositivo programable permite paralelizar los procesos de entrenamiento y reconocimiento. Y si además agregamos que estos dispositivos han incrementado sus frecuencias de operación, esto lleva a tener procesos paralelos de reconocimiento y entrenamiento con frecuencias muy considerables, en este caso especifico del orden de los 100 MHz.

De acuerdo con las características de las tarjetas de desarrollo utilizadas en este trabajo, y a los resultados alentadores obtenidos, se vislumbra un futuro prometedor, puesto que ambos proyectan la posibilidad de acoplar los dispositivos, configurados como clasificadores neuronales, a las diferentes tareas de reconocimiento de imágenes. Cabe señalar que la tarjeta de última generación utilizada presenta muchas características que no han sido explotadas todavía, lo que hace pensar en la implementación total del clasificador neuronal LIRA en un solo dispositivo FPGA. La tarjeta de desarrollo de versión anterior tiene ciertas capacidades que pueden ser consideradas para continuar haciendo pruebas, tanto para implementar unas cuantas clases mas a reconocer, algunas neuronas o grupos de neuronas, como para lo que respecta al procesamiento paralelo, que es otra particularidad que puede ser útil en el trabajo de investigación, por los 25.175 MHz de su oscilador, todo esto gracias a la simplicidad del paquete de aplicación con el cual se diseñan y se graban tales dispositivos.

Estos resultados son solo el principio en la investigación para el desarrollo e implementación de redes neuronales en dispositivos programables. El siguiente paso es implementar totalmente el clasificador neuronal LIRA, junto con sus algoritmos de trabajo, en un solo encapsulado, que le permita ser mas autónomo, y que pueda hacer todos los arreglos y tareas necesarias que requiere hacer un controlador basado en redes neuronales.

Trabajo futuro

Los resultados obtenidos a lo largo de la realización de este trabajo muestran la utilidad que tiene el desarrollo e implementación de un clasificador neuronal para ser implantado en un dispositivo programable. Este último puede ser aplicado en las tareas de reconocimiento de patrones, particularmente en la clasificación de imágenes de cifras escritas, rostros humanos, texturas, micropiezas, entre otros.

Dadas las capacidades con las que cuenta la tarjeta de desarrollo *Cyclone III LS FPGA* y del dispositivo programable utilizado, existen muchas características de la misma que pueden ser explotadas todavía mas, con lo que se vislumbran posibilidades de seguir con la línea de investigación en la implementación del clasificador neuronal LIRA en este dispositivo FPGA programable.

Una primera línea a continuar desarrollando es explotar al máximo la capacidad de adecuación del dispositivo, esto para conocer cual será el número máximo de clases que pueden implementarse dentro del mismo. Entre mas clases pueda reconocer un clasificador, menor será el tiempo de la clasificación de imágenes.

Otra línea a seguir, relacionada con la primera de conocer ese máximo de capacidad, es la posibilidad de implementar el clasificador neuronal LIRA dentro del dispositivo programable, es decir, que no tenga dependencia con la computadora para realizar los procesos de entrenamiento y reconocimiento. Ello implicaría un sistema más independiente para la realización de las tareas a las cuales sea asignado este clasificador, y ya no sería necesario el uso de una computadora, solo para el caso de carga de información procesada y adecuada en las memorias de trabajo. Con esto, solo la tarjeta es la que se implantaría en el sistema al cual va a controlar.

Y una tercera línea a seguir es la de incorporar una cámara que interactúe con el dispositivo programable. La imagen de esta cámara puede ser usada en el caso del proceso de entrenamiento del clasificador neuronal, o de ser clasificada para la opción de reconocimiento del clasificador. Esto implica generar el vector con los pixeles que actúan como neuronas de la capa *S* de entrada, y con ello extraer las características de la imagen, para posteriormente generar los códigos, y la activación de las neuronas de la capa *A*, con lo cual se activen los pesos entre sus conexiones con la capa *R*, y con estos últimos calcular qué neurona tiene la mayor excitación, para así definir la clase a la que corresponde la imagen entrada.

Cabe destacar que cada una de estas líneas de trabajo mencionadas implica un proceso de noción y trabajo con la tarjeta, de tal manera que se conozcan todos sus alcances y explotar al máximo sus capacidades, y por ende llegar a conocer sus límites. A partir de estos resultados, obtener un clasificador neuronal funcional y de utilidad para acoplarlo a tareas en las cuales pueda ser totalmente confiable y adecuado.

Referencias

[1] C. Y. Lai, F. L. Lewis, V. Venkatakrishnan, X. Ren, S. S. Ge, and T. Liew, "Disturbance and friction compensations in hard disk drives using neural networks," in *IEEE Trans. Ind. Electron.*, vol. 57, no. 2, pp. 784 – 792, Feb. 2010.

[2] S. S. Ge, and C. Wang, "Adaptive neural control of uncertain MIMO nonlinear systems," *IEEE Trans. Neural Netw.*, vol. 15, no. 3, pp. 674 – 692, May 2004.

[3] E. B. Kosmatopoulos, M. M. Polycarpou, M. A. Christodoulou, and P. A. Ioannou, "High-order neural network structures for identification of dynamical systems," *IEEE Trans. Neural Netw.*, vol. 6, no. 2, pp. 422 – 431, Mar. 1995.

[4] F. L. Lewis, A. Yesildirek, and K. Liu, "Multilayer neural-net robot controller with guaranteed tracking performance," *IEEE Trans. Neural Netw.*, vol. 7, no. 2, pp. 388 – 399, Mar. 1996.

[5] F. L. Lewis, S. Jagannathan, and A. Yesildirek, *Neural Network Control of Robot Manipulators and Nonlinear Systems*. London, U. K., Taylor & Francis, 1999.

[6] M. M. Polycarpou, and M. J. Mears, "Stable adaptive tracking of uncertain systems using nonlinearly parameterized on-line approximator," *Int. J. Control*, vol. 70, no. 3, pp. 363 – 384, 1998.

[7] C. A. Hudson, N. S. Lobo, and R. Krishnan, "Sensorless control of single swith-based switched reluctance motor drive using neural network," *IEEE Trans. Ind. Electron.*, vol. 55, no. 1, pp. 321 – 329, Jan. 2008.

[8] J. Mazumdar, and R. G. Harley, "Recurrent neural networks trained with backpropagation through time algorithm to estimate nonlinear load harmonic currents," *IEEE Trans. Ind. Electron.*, vol. 55, no. 9, pp. 3484 – 3491, Sep. 2008.

[9] C. L. Lin, and Y. H. Hsiao, "Adaptive feedforward control for disturbance torque rejection in seeker stabilizing loop," *IEEE Trans. Control Syst. Technol.*, vo. 9, no. 1, pp. 108 - 121, Jan. 2001.

[10] D. Gorinevsky, and L. A. Feldkampf, "RBF network feedforward compensation of load disturbances in idle speed control," *IEEE Control Syst. Mag.*, vol. 16, no. 6, pp. 18 – 27, Dec. 1996.

[11] M. G. Simoes and B. K. Bose, "Neural network based estimation of feedback signals for a vector controlled induction motor drive," *IEEE Trans. Ind. Appl.*, vol. 31, no. 3, pp. 620 – 629, May/Jun. 1995.

[12] A. Bakhshai, J. Espinoza, G. Joos, and H. Jin, "A combined artificial neural network and DSP approach to the implementation of space vector modulation techniques," in *Conf. Rec. IAS Annu. Meeting*, 1996, pp. 934 – 940.

[13] J. O. Pinto, B. K. Bose, L. E. B. da Silva, and M. P. Kazmierkowski, "A neural-network-based space-vector PWM controller for voltage-fed inverter induction

motor drive," *IEEE Trans. Ind. Appl.*, vol. 36, no. 6, pp. 1628 – 1695, Nov./Dec. 2000.

[14] M. Cirsttea, A. Danu, M. McCormick, and D. Nicula, "A VHDL success story: Electric drive system using neural controller," in *Proc. IEEE VHDL Int. Users Forum Fall Workshop*, 2000, pp. 118 – 122.

[15] M. A. S. K. Khan, and M. A. Rahman, "Development and implementation of a novel fault diagnostic and protection technique for IPM motor drives," in *IEEE Trans. Ind. Electron.*, vol. 56, no. 1, pp. 85 – 92, Jan. 2009.

[16] L. A. Zadeh, "Fuzzy sets and systems," *System Theory*. Polytechnic Press, Brooklyn, NY, pp. 29 – 39, 1965.

[17] M. J. Er, and Y. Gao, "Robust adaptive control of robots manipulators using generalized fuzzy neural networks," in *IEEE Trans. Ind. Electron.*, vol. 50, no. 3, pp. 620 – 628, Jun. 2003.

[18] Z. Liu, and C. Li, "Fuzzy neural network quadratic stabilization output feedback control for biped robots via H_∞ approach," in *IEEE Trans. Syst. Man, Cybern., B, Cybern.*, vol. 33, no. 1, pp. 67 – 84, Feb. 2003.

[19] A. Chatterjee, K. Pulasinghe, K. Watanabe, and K. Izumi, "A particle-swarm-optimized fuzzy-neural network for voice-controlled robot systems," in *IEEE Trans. Ind. Electron.*, vol. 52, no. 6, pp. 1478 – 1489, Dec. 2005.

[20] F. J. Lin, and P. H. Shen, "Robust fuzzy neural network sliding-mode control for two-axis motion control system," in *IEEE Trans. Ind. Electron.*, vol. 53, no. 4, pp. 1209 – 1225, Aug. 2006.

[21] F. J. Lin, C. H. Lin, and P. K. Huang, "Recurrent fuzzy neural network controller design using sliding-mode control for linear synchronous motor drive," in *Proc. IEEE Control Theory Appl.*, vol. 151, no. 4, pp. 407 – 416, Jul. 2004.

[22] M. J. Er, and C. Deng, "Obstacle avoidance of a mobile robot using hybrid learning approach," in *IEEE Trans. Ind. Electron.*, vol. 52, no. 3, pp. 898 – 905, Jun. 2005.

[23] F. J. Lin, H. J. Shieh, P. K. Huang, and L. T. Teng, "Adaptive control with hysteresis estimation and compensation using RFNN for piezo-actuator," in *IEEE Trans. Ultrason., Ferroelectr., Freq., Control*, vol. 53, no. 9, pp. 1649 – 1661, Aug. 2006.

[24] T. Orlowska-Kowalska, and K. Szabat, "Control of the drive system with stiff and elastic couplings using adaptive neuro-fuzzy approach," in *IEEE Trans. Ind. Electron.*, vol. 54, no. 1, pp. 228 – 240, Feb. 2007.

[25] F. J. Lin, P. K. Huang, C. C. Wang, and L. T. Teng, "An induction generator system using fuzzy modeling and recurrent fuzzy neural network," in *IEEE Trans. Power Electron.*, vol. 22, no. 1, pp. 260 – 271, Jan. 2007.

[26] F. J. Lin, and P. H. Chou, "Adaptive control of two-axis motion control system using interval type-2 fuzzy neural network," in *IEEE Trans. Ind. Electron.*, vol. 56, no. 1, pp. 178 – 193, Jan. 2009.

[27] T. Orlowska-Kowalska, M. Dybkowski, and K. Szabat, "Adaptive sliding-mode neuro-fuzzy control of the two-mass induction motor drive without mechanical sensors," in *IEEE Trans. Ind. Electron.*, vol. 57, no. 2, pp. 553 – 564, Feb. 2010.

[28] L. Charaabi, E. Monmasson, and I. Slama-Belkhodja, "Presentation of an efficient design methodology for FPGA implementation of control systems. Application to the design of an antiwindup PI controller," *in Proc. IEEE-IECON Annu. Meeting*, 2002, vol. 3, pp. 1942 – 1947.

[29] S. Jayasoma, S. J. Dodds, and R. Perryman, "An FPGA implemented PMSM servo drive: Practical issues," in *Proc. Int. Universities Power Eng. Conf.*, 2004, vol. 1, pp. 499 – 503.

[30] Z. Zhou, T. Li, T. Takahashi, and E. Ho, "FPGA realization of a high-performance servo controller for PMSM," in *Proc. IEEE Appl., Power Electron., Conf.*, 2004, vol. 3, pp. 1604 – 1609.

[31] T. Takahashi, "FPGA based high performance AC servo motor drive-accelerator," International Rectifier, [Online]. Available: http://www.irf.com

[32] E. Monmasson and M. N. Cirstea, "FPGA design methodology for industrial control systems-A review," *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1824 – 1842, Aug. 2007.

[33] Y. F. Chan, M. Moallem, and W. Wang, "Design and implementation of modular FPGA-based PID controllers," *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1898 – 1906, Aug. 2007.

[34] Y. Tzou and T. Kuo, "Design and implementation of all FPGA-based motor control IC for permanent magnet AC servo motors," in *Proc. IEEE-IECON Annu. Meeting*, 1997, vol. 2, pp. 943 - 947.

[35] N. M. Botros and M. Abdul-Aziz, "Hardware implementation of an artificial neural network using field programmable gate arrays (FPGA's)," *IEEE Trans. Ind. Electron.*, vol. 41, no. 6, pp. 665 – 667, Dec. 1994.

[36] S. McBader, P. Lee, and A. Sartori, "The impact of modern FPGA architectures on neural hardware: A case study of the TOTEM neural processor," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, 2004, vol. 4, pp. 3149 – 3154.

[37] W. Qianyi and H. Nomura, "FPGA implementation of multi-valued 'and/or'-neural network," in *Proc. Int. Conf. Neural Netw. Signal Process.*, 2003, vol. 1, pp. 349 – 352.

[38] S. Jung and S. S. Kim, "Hardware implementation of a real-time neural network controller with a DSP and a FPGA for nonlinear systems," *IEEE Trans. Ind. Electron.*, vol. 54, no. 1, pp. 265 – 271, Feb. 2007.

[39] Y. S. Kung, R. F. Fung, and T. Y. Tai, "Realization of a motion control IC for X – Y table based on novel FPGA technology," in *IEEE Trans. Ind. Electron.*, vol. 56, no. 1, pp. 43 – 53, Jan. 2009.

[40] Y. S. Kung, P. G. Huang, and C. W. Chen, "Development of a SOPC for PMSM drives," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, 2004, vol. II, pp. II-329 – II-332.

[41] Y. S. Kung, and M. H. Tsai, "FPGA-based speed control IC for PMSM drive with adaptive fuzzy control," *IEEE Trans. Power Electron.*, vol. 22, no. 6, pp. 2476 – 2486, Nov. 2007.

[42] Y. S. Kung, K. H. Tseng, and T. Y. Tai, "FPGA-based servo control IC for X – Y table," in *Proc. IEEE ICIT*, 2006, pp. 2913 – 2918.

[43] D. Zhang, and H. Li, "A stochastic-based FPGA controller for an induction motor drive with integrated neural network algorithms," in *IEEE Trans. Ind. Electron.*, vol. 55, no. 2, pp. 551 – 561, Feb. 2008.

[44] R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification. John Wiley and Sons, Inc., New York, 2001, 680 pp.

[45] C. H. Chen, L. F. Pau, and P. S. P. Wang. Handbook of Pattern Recognition and Computer Vision. World Scientific, Singapore, 2nd edition, 1993.

[46] L. Rabiner, and B. H. Juang. Fundamentals of Speech Recognition. Prentice Hall, Englewood Cliffs, NJ, 1993.

[47] G. F. Luger. Cognitive Science: The Science of Intelligent Systems. Academic Press, New York, NY, 1994.

[48] D. R. Tveter. The Pattern Recognition basis of Artificial Intelligence. IEEE Press, New York, NY, 1998.

[49] L. Stark, and K. Bower. Generic Object Recognition using Form and Function. World Scientific, River Edge, NJ, 1996.

[50] J. W. Shavlink, and T. G. Dietterich. Readings in Machine Learning. Morgan Kaufmann, San Mateo, CA, 1990.

[51] M. Fischler, and O. Firschein. Readings in Computer Vision: Issues, Problems, Principles and Paradigms. Morgan Kaufmann, San Mateo, CA, 1987.

[52] J. Kittler. Notas del seminario de reconocimiento de patrones. Centro para el procesamiento de señales, voz y visión. Departamento de ingeniería eléctrica. Universidad de Surrey, Inglaterra, 2002, pp. 1-3. www.ee.surrey.ac.uk/Personal/J.Kittler/lecturenotes.

[53] J. A. Carrasco Ochoa. Curso de reconocimiento de patrones. Ciencias computacionales. Instituto de Nacional de Astrofísica, Óptica y Electrónica, México, 2006, pp. 2-3. http://ccc.inaoep.mx/~ariel/recpat.pdf.

[54] F. J. Cortijo Bon. Apuntes sobre reconocimiento de formas. Ingeniería informática. Departamento de ciencias de la computación e inteligencia artificial. Universidad de Granada, España, 2001, pp. 21-22. http://www-etsi2.ugr.es/depar/ccia/rf/www/tema1_00-01_www/tema1_00-01_www.html.

[55] J. Ruiz-Shulcloper, and M. A. Abidi, "Logical Combinational Pattern Recognition: A Review," *Transworld Research Networks*, Kerala, India, 2002, pp. 133-176.

[56] A. N. Dimitriev, Yu. I. Zhuravlev, and F. P. Krendelev, "About the mathematical principles of patterns and phenomena classification," *Journal Diskretnyi Analiz 7*, 3, 1966,

[57] A. N. Dimitriev, Yu. I. Zhuravlev, and F. P. Krendelev, "About a principle of classification and forecast of geological objects and phenomena," *Geologia I Geofizika 5*, 50, 1968,

[58] J. Gómez Herrera, O. Rodríguez Morán, S. Valladares Amaro, J. Ruiz Shulcloper, R. Pico Peña, and others, "Prognostic of gas-oil deposits in the Cuban ophiological association, applying mathematical modeling," *Geophysics International*, 33, 3, 447, 1995,

[59] M. R. Ortiz Posadas, J. F. Martínez Trinidad, and J. Ruiz Shulcloper, "A new approach to differential diagnosis of diseases," *International Journal of Bio-medical Computing*, 40, 179, 1996,

[60] M. R. Ortiz Posadas, and M. Lazo Cortés, "Evaluation of cleft palate patients' rehabilitation using pattern recognition techniques," *Proceedings of II Taller Iberoamericano de Reconocimiento de Patrones. Conferencia Internacional CIMAF'97*. La Habana. 231, 1997,

[61] M. R. Ortiz Posadas, J. Maya Behart, and M. Lazo Cortés, "Evaluation of lips and cleft palate chirurgery using logical combinatorial approach to pattern recognition theory," *Journal Revista Brasileira de Bioengenharia. Caderno de Engenharia Biomedica,*14, No. 1, 7, 1998,

[62] J. Ruiz Shulcloper, and A. Fuentes Rodríguez, "A cybernetic model for the youth delinquency," *Journal Revista Ciencias Matemáticas,* II, No. 1, 141, 1981,

[63] J. F. Martinez Trinidad, and A. Guzman Arenas, "The logical combinatorial approach to Pattern Recognition, an overview through selected Works," *Pattern Recognition,* Vol. 34, issue 4, 2001, pp. 1-13.

[64] Emiliano Aldabas-Rubira. Introducción al reconocimiento de patrones mediante redes neuronales. UPC-Campus Terrassa-DEE-EUETIT Colom, 1 08222 Terrassa Barcelona, España, 2002, pp. 1-3. http://www.jcee.upc.es/JCEE2002/Aldabas.pdf

[65] Guillermo Bien, Diego Krein, Alice Rambo. Redes Neuronales. Reconocimiento de Imágenes. Sistemas de Computación, Instituto Gastón Dachary, Posadas – Misiones, Argentina, 2006, pp. 1-2.

[66] F. Rosenblatt. Principles of Neurodynamics. Spartan Books, New York, 1962, 215 pp.

[67] J. R. Hilera, V. J. Martínez. Redes neuronales artificiales. Fundamentos, modelos y aplicaciones. Ed. Rama, 1995, 350 pp.

[68] S. Haykin. Neural networks. A comprehensive foundation. IEEE Press, 1994, 768 pp.

[69] Joan Cabestany Moncusí, Sergi Bermejo Sánchez; "Xarxes Neuronals". ETSETB-Departament D'Enginyeria Electronica, Universitat Politecnica de Catalunya, 2001. http://petrus.upc.es/~microele/neuronal/xn/docs/

[70] E. M. Kussul, T. N. Baidyk, "Neural random threshold classifier in OCR application," in *Proceedings of the Second All-Ukrainian International Conference UkrOBRAZ'94*, Ukraine, 1994, pp. 154-157.

[71] G. Ososkov, "Effective neural network approach to image recognition and control," in *Proceedings of International Conference on Physics and Control*, 1, 2003, pp. 242-246.

[72] S. Vitabile, A. Gentile, F. Sorbello, "A neural network based automatic road signs recognizer," in *Proceedings of the 2002 International Joint Conference on Neural Networks*, 3, 2002, pp. 2315-2320.

[73] T. J. W. Clarke, R. W. Prager, F. Fallside, "The modified Kanerva model: theory and results for real-time word recognition," *IEE Proceedings-F 138*, 1, 1991, pp. 25-31.

[74] E. M. Kussul, D. Rachkovskij, D. Wunsch, "The random subspace coarse coding scheme for real-valued vectors," in *Proceedings of IEEE International Joint Conference on Neural Networks*, Washington, 1999, pp. 450-455.

[75] E. Kussul, T. Baidyk, "Improved method of handwritten digit recognition tested on MNIST database," *Image and Vision Computing*, 22, 2004, pp. 971-981.

[76] M.Barr. Programmable Logic: What's it to Ya? Embedded Systems Programming. June 1999, pp. 75-84.

[77] R. C. Cofer and B. Harding. Rapid System Prototyping with FPGAs: Accelerating the design process. Newnes, Massachusetts, 2005, 320 pp.

[78] I. Grout. Digital Systems Design with FPGAs and CPLDs. Newnes, Massachusetts, 2008, 784 pp.

[79] S. Kilts. Advanced FPGA Design: Architecture, Implementation, and Optimization. Wiley-IEEE Press, New Jersey, 2007, 352 pp.

[80] P. Wilson. Design Recipes for FPGA: Using Verilog and VHDL. Newnes, Massachusetts, 2007, 320 pp.

[81] Altera Corporation. 2009. http://www.altera.com/corporate/about_us/abt-index.html.

[82] M. Karnaugh, "A Map Method for Synthesis of Combinational Logic Circuits," *Transactions of the AIEE, Communications and Electronics 72*, part I, November 1953, pp. 593-599.

[83] T. Baydyk, "Neural networks for the system of micro components recognition in micro mechanics," in *Proceedings of 1st. International Congress on Instrumentation and Applied Science (SOMI XXV)*, Cancun, Q. R. México, October 26 – 29, 2010, pp. 1 – 6.

[84] A. Vega, T. Baydyk, E. Kussul, and J. L. Perez, "Digital implementation of LIRA neural classifier," in *Proceedings of 1st. International Congress on Instrumentation and Applied Science (SOMI XXV)*, Cancun, Q. R. México, October 26 – 29, 2010, pp. 1 – 13.

[85] A. Vega, T. Baydyk, E. Kussul, and J. L. Pérez, "FPGA Realization of the LIRA Neural Classifier," *Optical Memory & Neural Networks* (Aceptado).

Anexos

A.1. Tarjeta de desarrollo FPGA Cyclone III LS

La *Cyclone III LS FPGA development board* es una tarjeta de desarrollo de prototipos de nueva generación, la cual provee de una plataforma física que permite desarrollar diseños con características de alta integración y bajo consumo de energía, entre otras, Figura A.1. Esta tarjeta provee de una amplia gama de elementos, tales como periféricos e interfaces de memoria, que permiten y facilitan la realización de diferentes diseños basados en el arreglo de compuertas programable por campo FPGA Cyclone III LS.

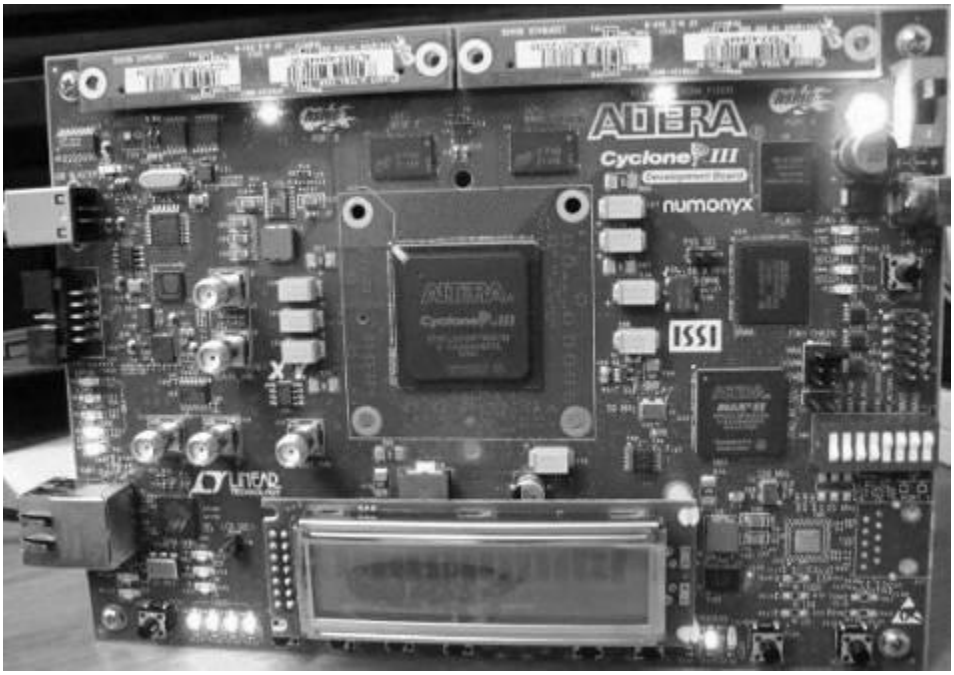

Figura A.1. Tarjeta de desarrollo Cyclone III LS FPGA

El elemento principal de esta tarjeta es el arreglo de compuertas programable por campo, FPGA, Cyclone III EP3CLS200F780 en su empaque de 780 pines. Su característica principal es su alta integración, pues cuenta con 198 464 elementos lógicos (LEs), y sus capacidades de operación, con 1.2 volts de suministro de energía, Figura A.2.

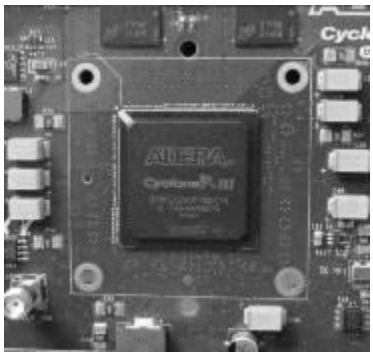

Figura A.2. Cyclone III LS FPGA

Además de ello, cuenta con un dispositivo lógico programable complejo, CPLD, MAX II EPM2210F256 en su empaque de 256 pines, que permite tener un control completo sobre cada uno de los dispositivos con los que cuenta esta tarjeta, Figura A.3.

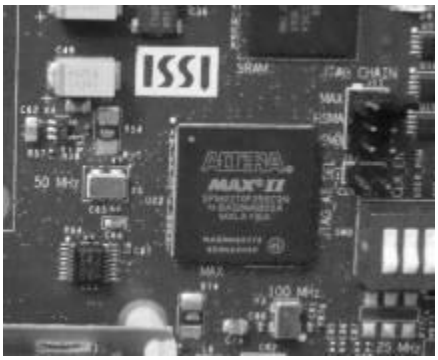

Figura A.3. MAX II CPLD

También cuenta con un conector USB-Blaster para su programación y su configuración a partir del programador Quartus II (Figura A.4), aunque no es la única forma de ser programado, ya que esto también puede ser realizado a través del MAX II y la memoria flash con la que cuenta esta tarjeta. Además de esta memoria de 64 Megabytes con un bus de datos de 16 bits, cuenta con dos bloques de memoria DDR2 de 512 Megabits y 16 bits de datos, haciendo un total de 1024 Megabits con 32 bits de datos, una memoria de acceso aleatorio estática síncrona de 2 Mbytes con 36 bits de datos, y una memoria EEPROM serie de 32 Kilobits, Figura A.5.

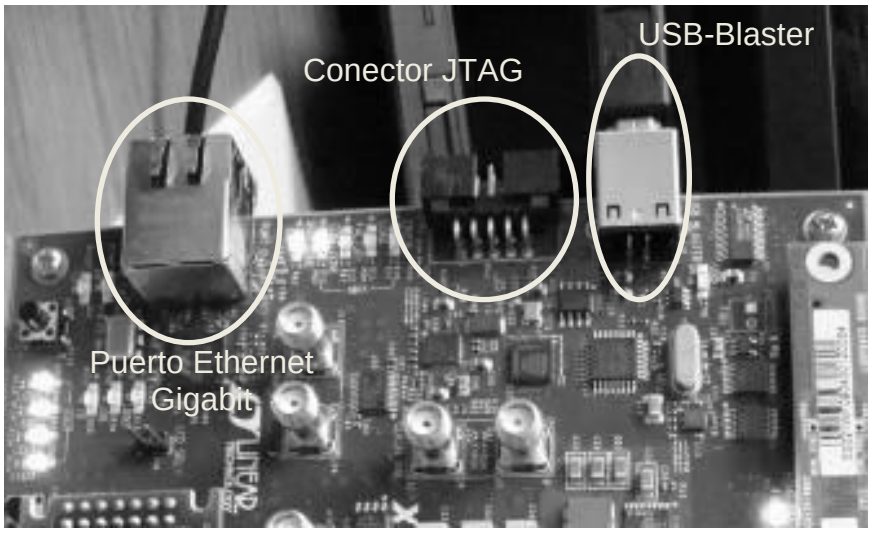

Figura A.4. Conectores para configuración y programación de la tarjeta

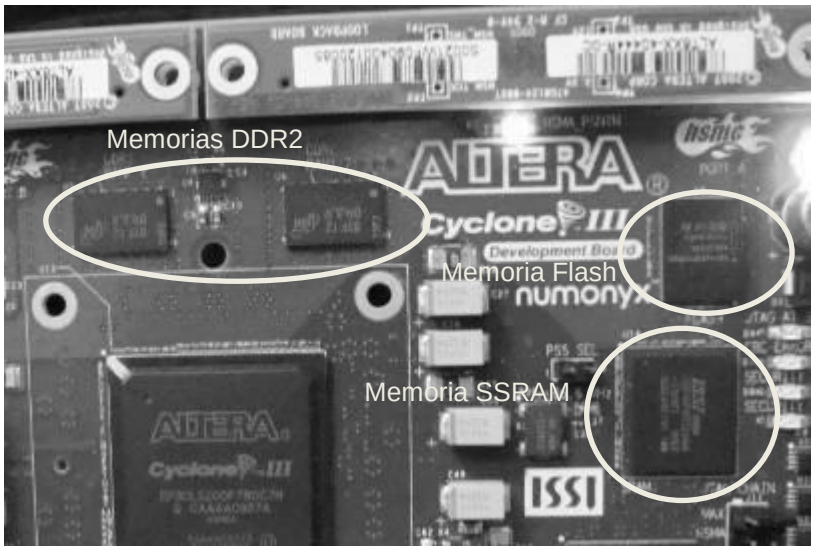

Figura A.5. Dispositivos de Memoria de la tarjeta de desarrollo

Cuenta también con 4 osciladores, de 50, 66.6 y 100 MHz, además de uno programable, de entre 100 125, 150 y 156.5 MHz, con una frecuencia básica de 125 MHz.

Además de los componentes anteriores, la tarjeta incluye switches de control para la operación y programación de la misma, así como para uso del usuario. Dispositivos LEDs indicadores de operación y programación, y de usuario, Figura A.6.

Otros dispositivos útiles para cualquier diseño son un display de cristal líquido de doble línea (Figura A.6), un puerto Ethernet Gigabit y un conector JTAG (Figura A.4).

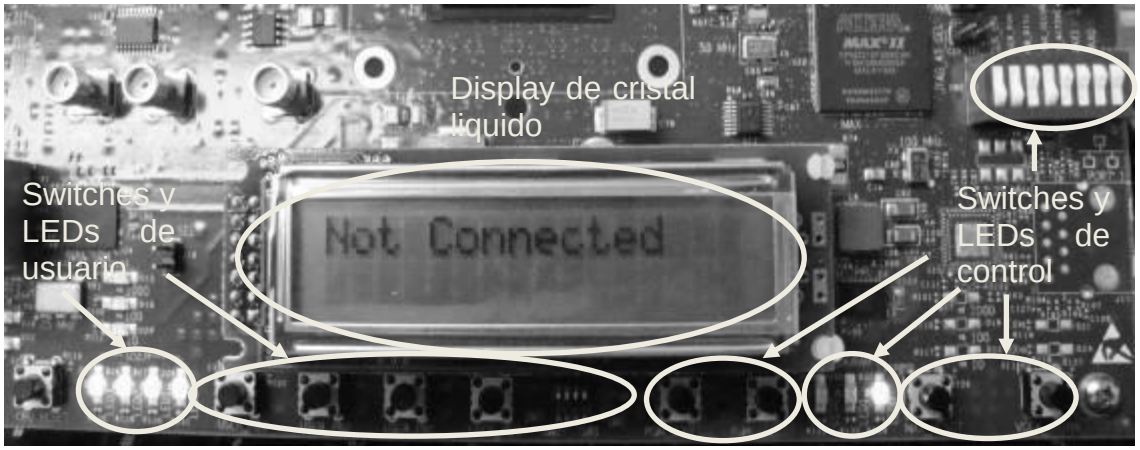

Figura A.6. Switches y LEDs de control y de usuario (detalle)

Todo esto en conjunto provee de los elementos necesarios para el diseño y prueba física de prototipos diseñados a partir de la lógica programable, desde diseños esquemáticos electrónicos, hasta diseños con lenguajes de descripción de hardware, como VHDL y Verilog.

A.2 Quartus II de Altera

La paquetería de diseño Quartus II de Altera es un ambiente totalmente completo para el diseño de sistemas en dispositivos programables, el cual proporciona una multiplataforma que se adapta a las necesidades del diseñador. El Quartus II provee de soluciones para cada fase de diseño en un FPGA o un CPLD.

Quartus II es un sofisticado sistema de diseño asistido por computadora, y como la mayoría de estas herramientas de diseño, está en constante y continuo mejoramiento y actualización, por lo que ha pasado por un gran numero de versiones. Actualmente se cuenta con la versión 10.0 con su paquete de servicio 10.1, existiendo una versión web y otra por subscripción, Figura A.7.

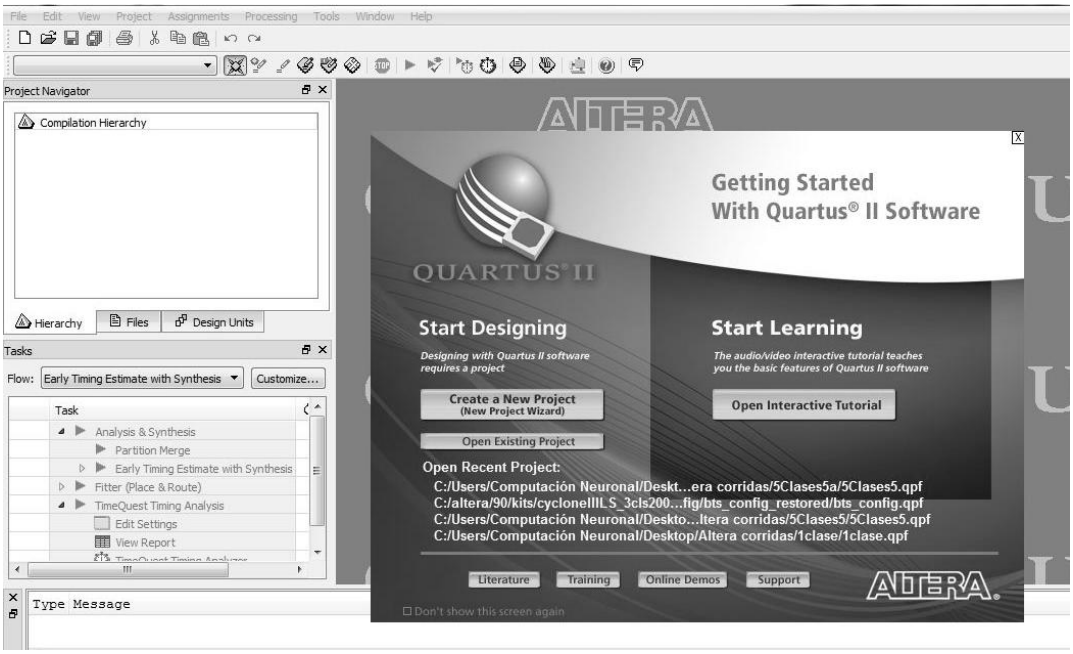

Figura A.7. Quartus II de Altera Corporation

El sistema de Quartus II incluye un soporte total para todos los métodos comunes de diseño de un circuito deseado, es decir, acepta el método de diseño por esquemáticos, y el método de diseño con lenguajes de descripción de hardware, como verilog y VHDL. Cabe señalar que Quartus II es la evolución de los programas de aplicación y diseño MAX + PLUS II y ASIC, por lo que tiene las mismas bases y estructuras, aunque se convierte en un paquete mas complejo, debido a la evolución de los dispositivos programables, CPLD y FPGA, y por consiguiente, de las tarjetas de desarrollo, que adquieren mas elementos incorporados, los cuales requieren de un ambiente con mas capacidades de control y operación, Figura A.8.

Figura A.8. Ambiente de trabajo del programa de aplicación Quartus II

El programa de aplicación Quartus II permite el diseño de sistemas a partir de:

Entrar el diseño, que consiste en especificar el diseño deseado ya sea por medio de un diagrama lógico esquemático, Figura A.9, o por el uso de un lenguaje de descripción de hardware, como Verilog o VHDL, Figura A.10.

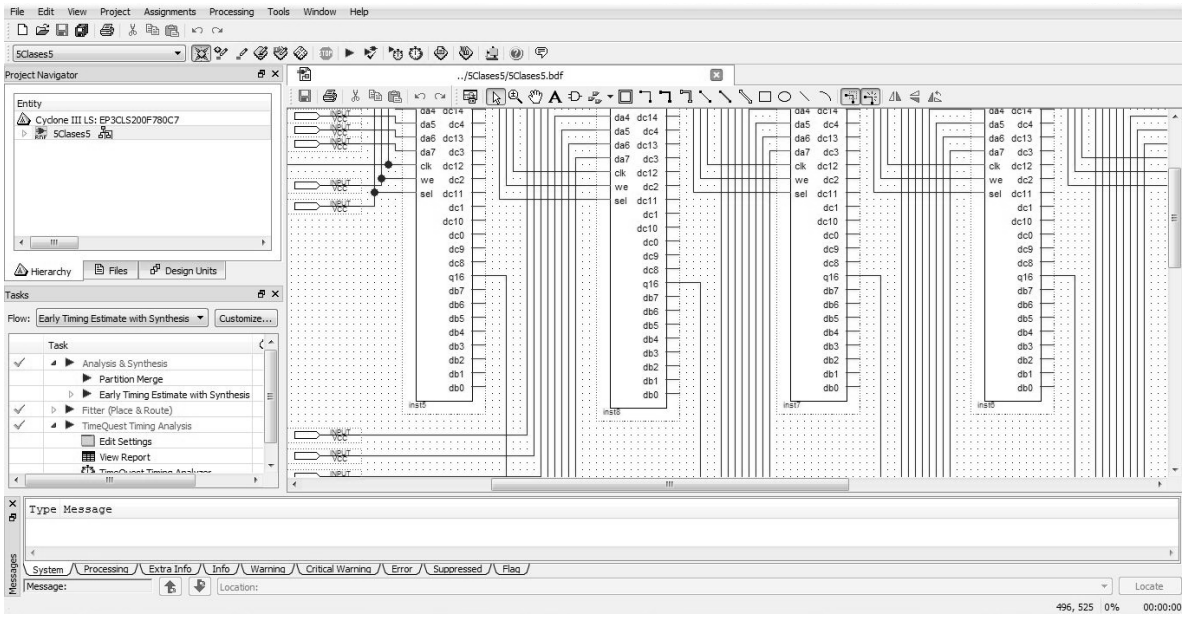

Figura A.9. Diseño por diagramas esquemáticos

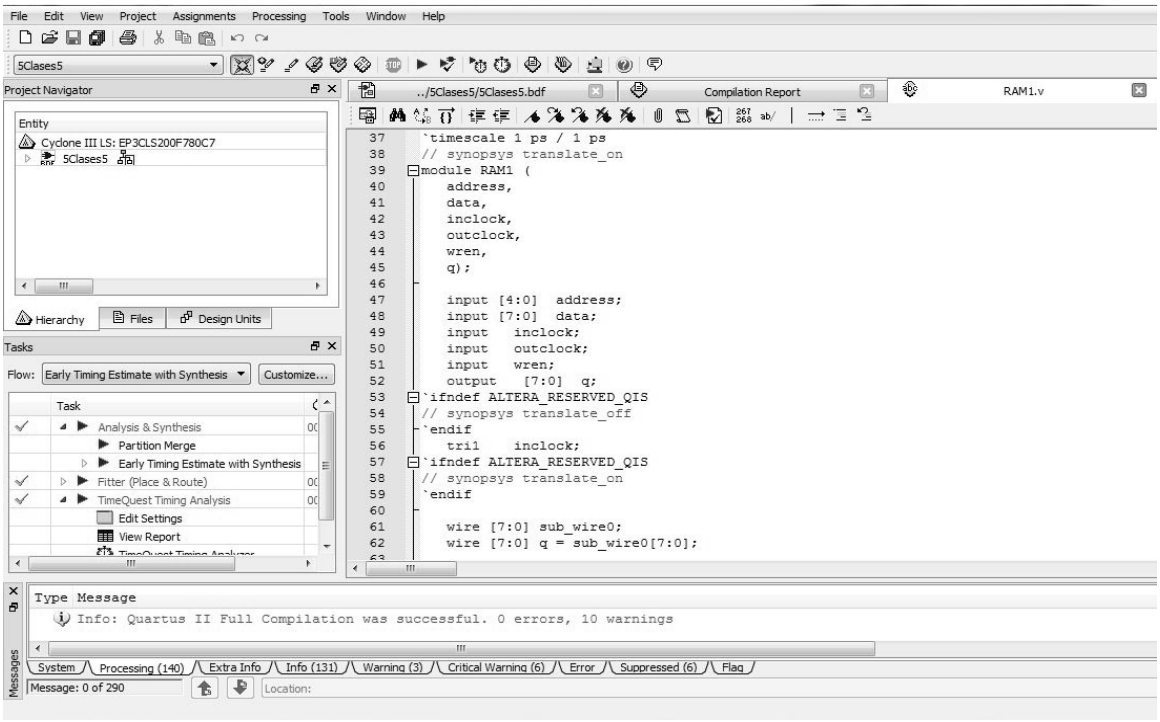

Figura A.10. Diseño por lenguajes de descripción de hardware

Sintetizar, que consiste en condensar el diseño especificado dentro de un circuito que consiste en un número de elementos lógicos contenidos en un dispositivo FPGA, Figura A.11.

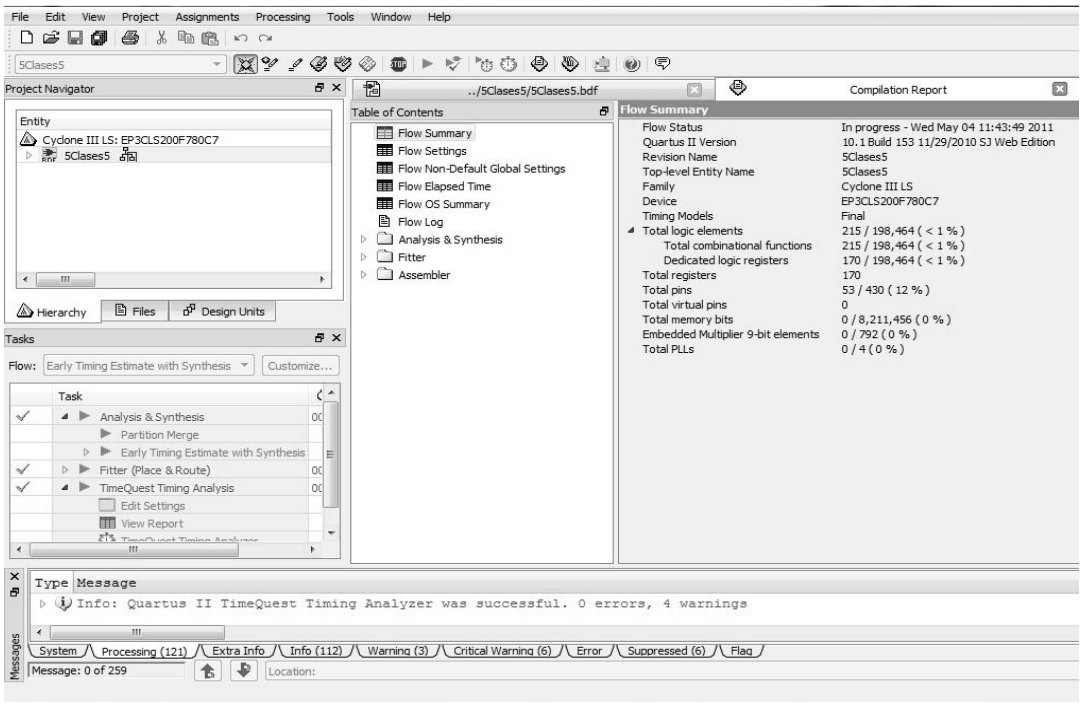

Figura A.11. Reporte del diseño compilado

Adecuación, que consiste en determinar, a través de una herramienta, la colocación de cada elemento lógico utilizado dentro de la red de elementos lógicos contenidos en el FPGA. También selecciona las líneas de conexión para relacionar los elementos lógicos utilizados, Figura A.11.

Análisis temporal, que consiste en analizar los retardos de propagación entre las diferentes vías del circuito condensado, para proveer de indicadores del rendimiento esperado por el circuito, Figura A.11.

Simulación temporal, que consiste en probar el circuito condensado para verificar su correcta funcionalidad y temporización, Figura A.11.

Programación y configuración, que consiste en implementar el circuito diseñado en un dispositivo programable físico, a través de programar los contactos de configuración de los elementos lógicos, y establecer las conexiones físicas requeridas, Figuras A.12 y A.13.

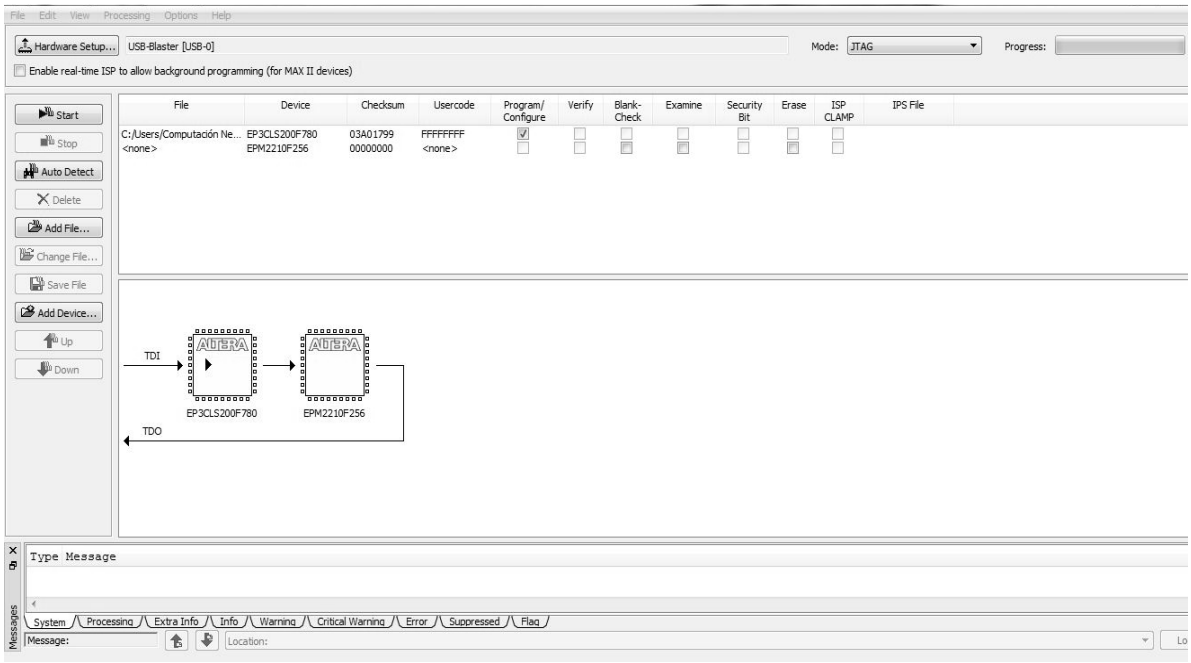

Figura A.12. Programador del Quartus II

Con esta secuencia de pasos se logra desarrollar e implementar en un dispositivo programable cualquier diseño que haya sido ideado para resolver alguna tarea en particular. La programación física del dispositivo implica que los demás pasos han sido cubiertos, pero sobre todo, que el diseño responde a las necesidades o requerimientos sobre los cuales haya sido concebido dicho sistema.

Por esta razón es muy útil el uso de un simulador que muestre el comportamiento del diseño desarrollado, sobre el cual se pueda definir si es adecuado o satisface los objetivos de sistema a implementar.

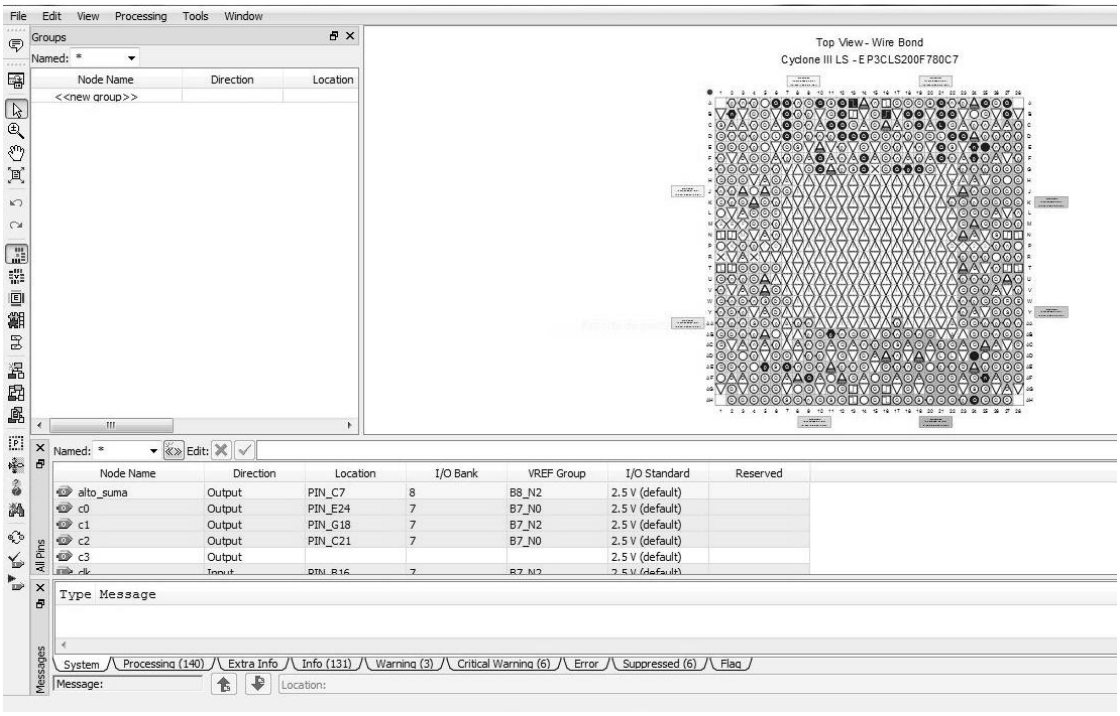

Figura A. 13. Ubicación y asignación de pines para el dispositivo utilizado

A.3 Modelsim de Mentor Graphics

Dada la alta cantidad de aplicaciones con las que cuenta la paquetería de diseño Quartus II versión 10.0, y su correspondiente complejidad, Altera ha optado por desincorporar de sus librerías la opción de simulación. Para cubrir esta necesidad, Mentor Graphics Corporation ha generando una nueva aplicación por separado conocida como Modelsim, y una especial para Altera Corporation, Figura A.14.

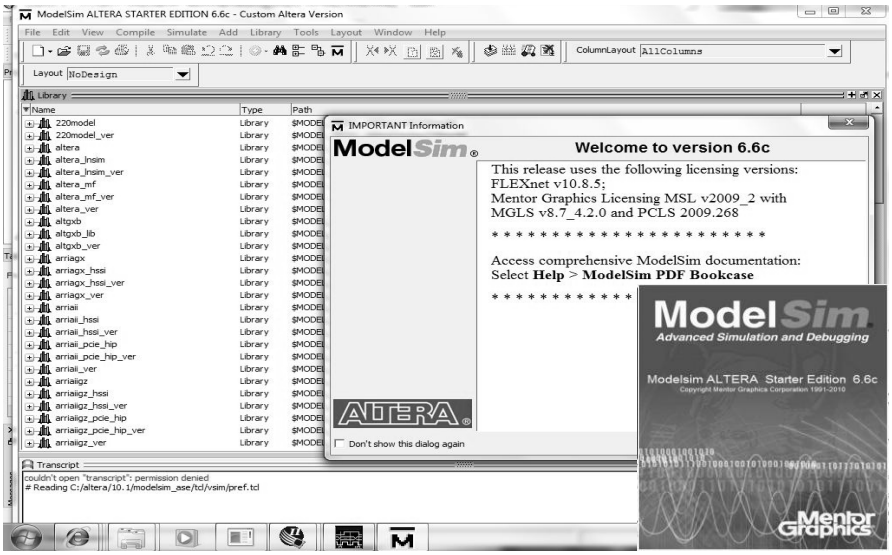

Figura. A.14. Modelsim de Mentor Graphics Corporation

Modelsim es una aplicación que se enfoca a la verificación y simulación de diseños generados en algún lenguaje de programación de hardware, tal como VHDL o Verilog, independientemente de la plataforma en la cual se hayan estructurado, es decir, no importa si el diseño se generó en otro paquete de aplicación ajeno a Quartus II. Con esto, solo es necesario generar un archivo de trabajo donde se sitúe el diseño sobre el cual se requiere conocer su comportamiento de manera simulada, Figura A.15. En este archivo de trabajo se compila dicho diseño, Figura A.16.

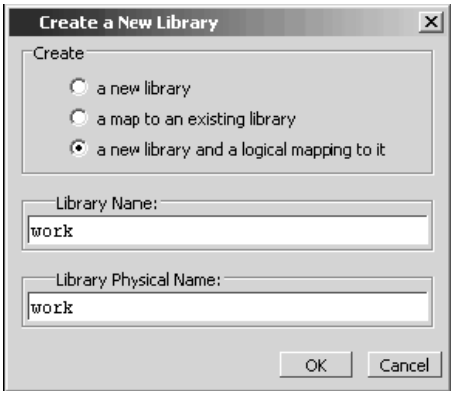

Figura A.15.Creacion del archivo de trabajo

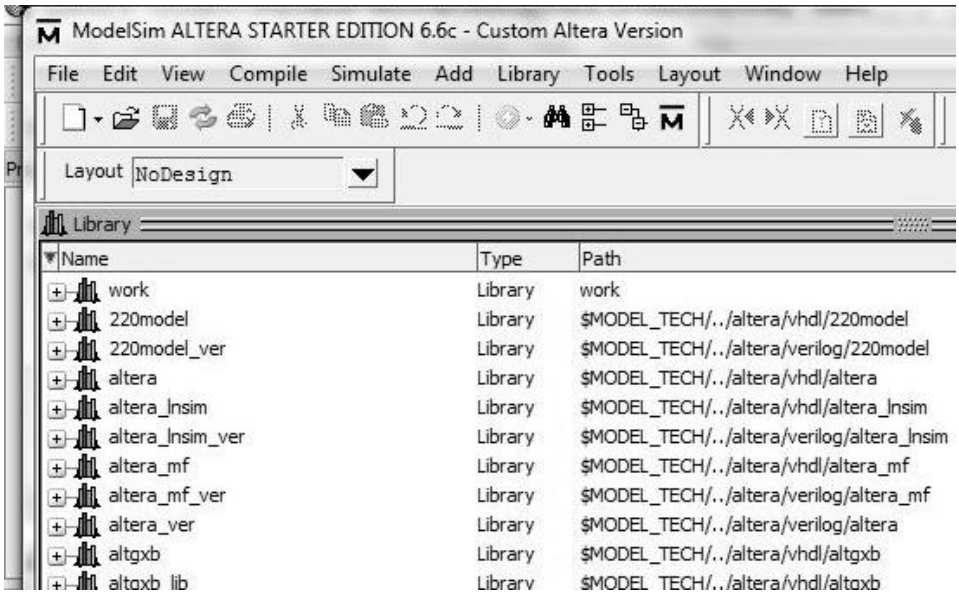

Figura A.16. Archivo de trabajo generado

A continuación, con Modelsim se hace el proceso de compilación para verificar o encontrar posibles errores en el diseño que se está implementando, Figura A.17. Este compilador ayuda a generar y estructurar completamente los archivos necesarios para la consecuente simulación. Si existe algún error de sintaxis, o si no ha sido declarado algún componente, este se encarga de mostrar las advertencias de los errores que deben corregirse, Figura A.18.

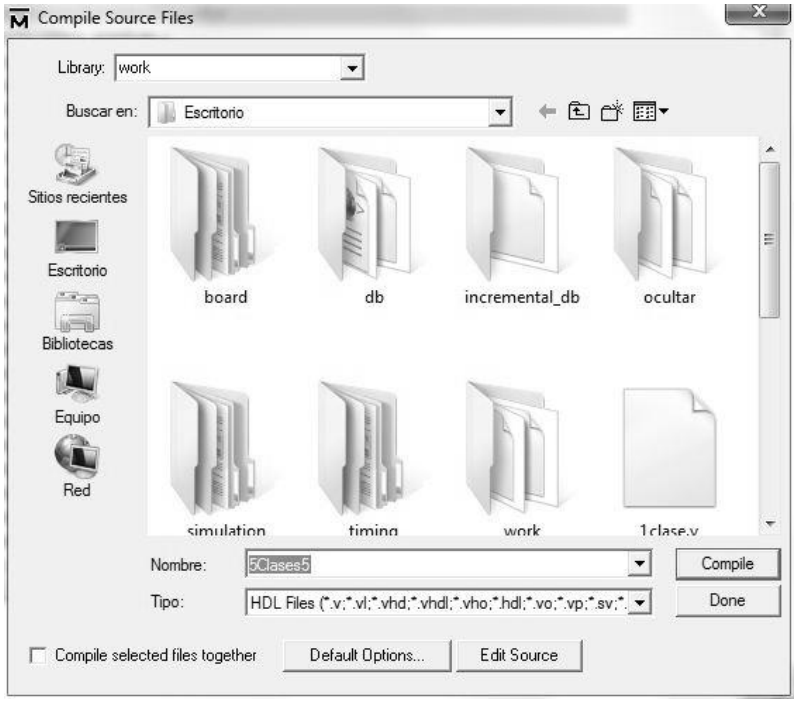

Figura A.17. Archivos a compilar

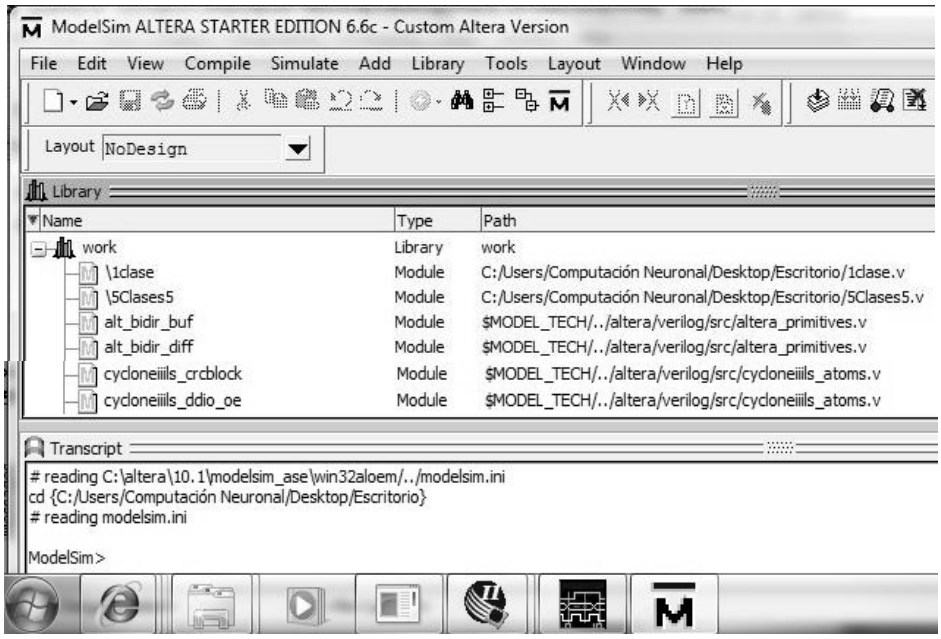

Figura A.18. Archivos compilados

El compilador permite también revisar o generar diseños en lenguaje de descripción de hardware a través de un editor de textos con el que cuenta, con lo que es posible corregir los errores en la estructura del diseño, Figura A.19.

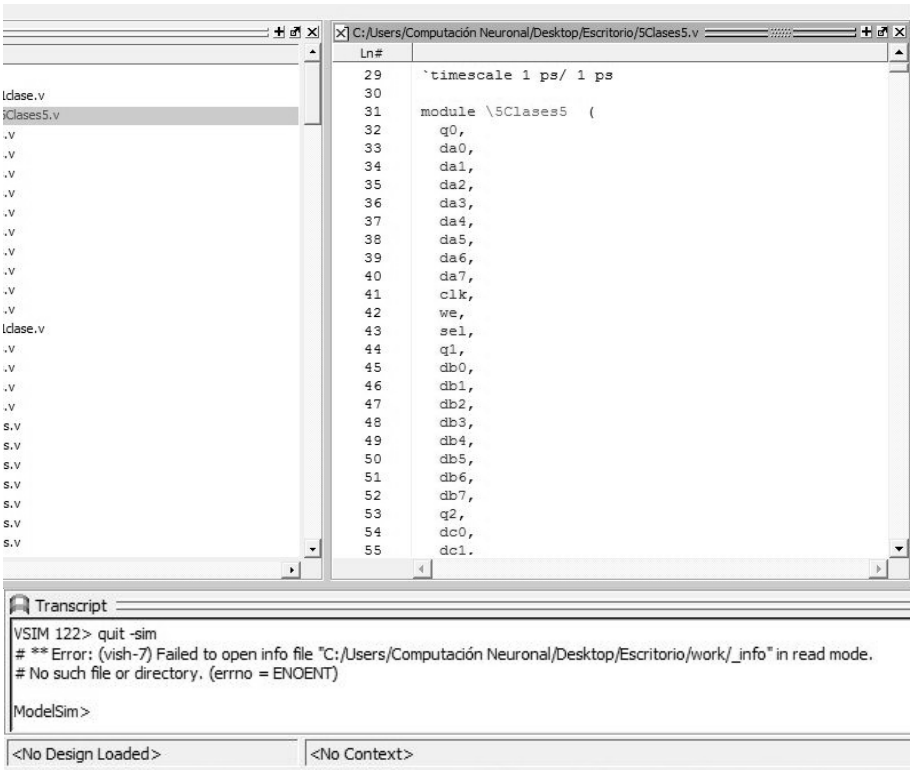

Figura A.19. Editor de texto para lenguajes de programación

Cuando todo ha sido corregido y estructurado adecuadamente, se invoca al simulador que inicia el proceso de construcción de la simulación, Figura A.20, consistente en el ingreso de todas las señales, tanto de entrada como de salida, involucradas en el diseño del sistema en desarrollo, Figura A.21.

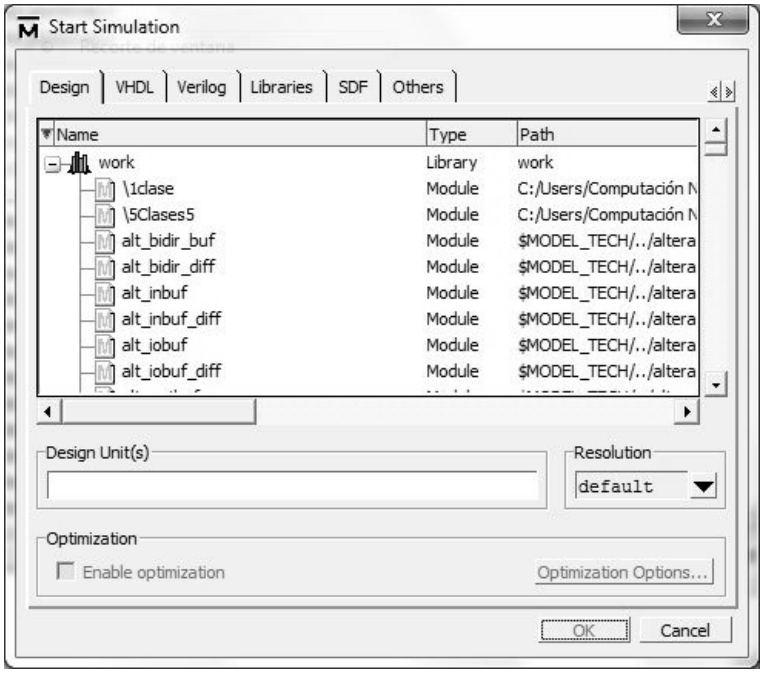

Figura A.20. Inicio del simulador

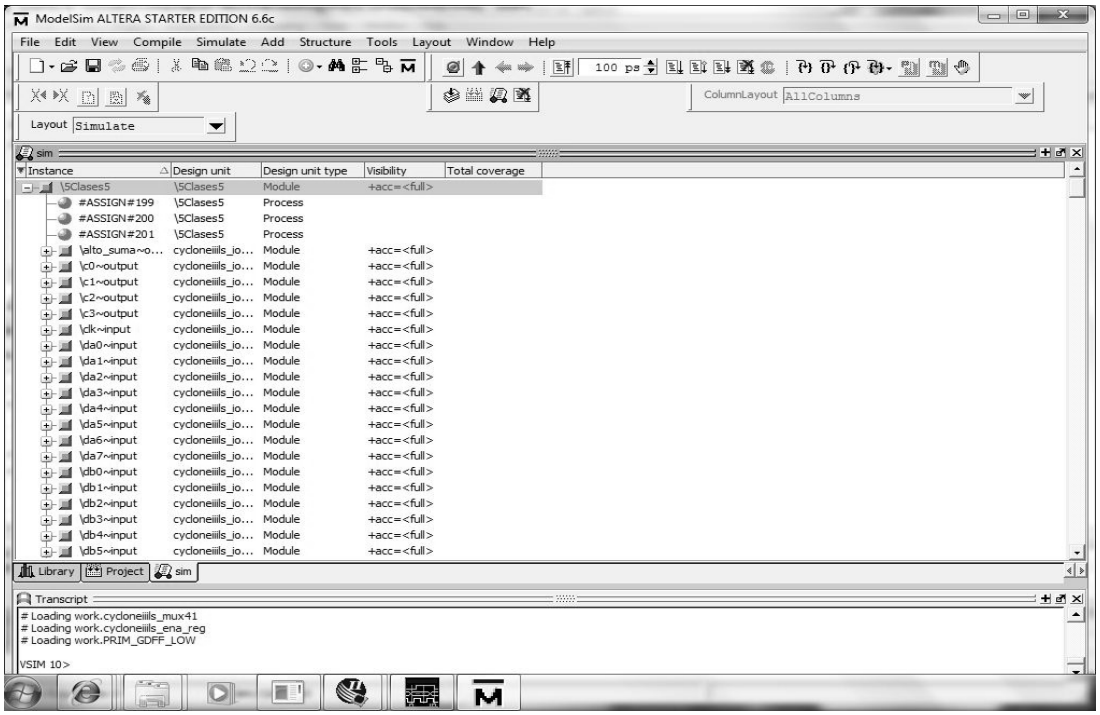

Figura A.21. Llamado de señales a operar

Cuando se tiene el editor de formas de onda es necesario entonces ingresar, o designar, el comportamiento que debieran tener cada una de las entradas para que el sistema las opere y muestre el resultado que se obtiene, con el cual se verifique si el diseño que se ha estructurado cubre totalmente o no el objetivo por el cual fue implementado, esto se logra a través del editor de señales con el que cuenta, Figura A.22. De no ser así, es necesario regresar a hacer las correcciones necesarias. Lo cual puede implicar regresar al programa de aplicación donde fue diseñado tal sistema.

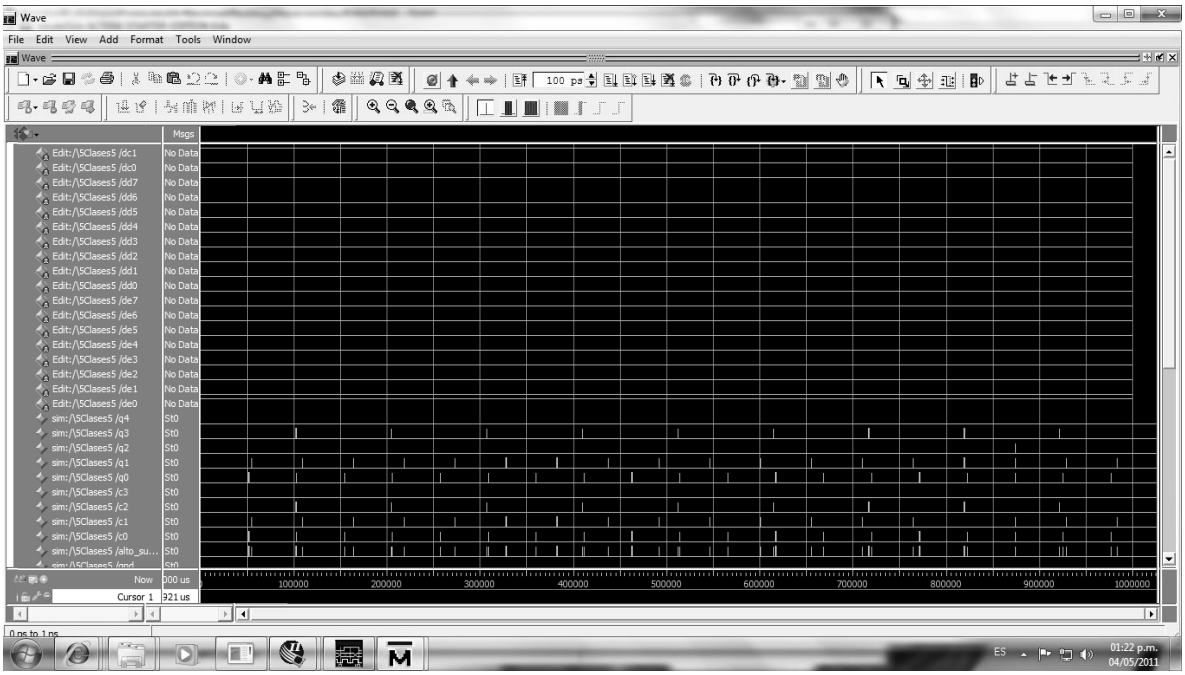

Figura A.22. Editor de formas de onda