Desarrollo de un modelo de regresión con redes neuronales artificiales para estimar la resistencia rotórica de un motor de inducción.

Juan Carlos Jaimes Jauregui
Universidad de Pamplona, Pamplona, Colombia, juankja85@hotmail.com

Oscar Eduardo Gualdrón Guerrero
Universidad de Pamplona, Pamplona, Colombia, oscar.gualdron@unipamplona.edu.co

Jorge Luis Diaz Rodriguez
Universidad de Pamplona, Pamplona, Colombia, jdiazcu@gmail.com

ABSTRACT

This paper presents the implementation of Neural Networks in a programmable logic device such as the FPGA, the aim is to develop a model in Sysgen able to estimate the rotor resistance in induction motors. To create a neuron in Sysgen is from the standard model of an artificial neuron, which is the sum of weights minus the threshold, all multiplied by an activation function, in this case the activation function was used for Tansig Purelin hidden layer to output layer. The neural network uses decimal data input/output, and the FPGA is digital, therefore Sysgen was designed in a fitting capable of handling binary data on the card, for this we used a Digital-Analog Converter, and was created as suppress the number of pins used in the FPGA, designing a stage of registers used to store the desired data and thus send it to one of the inputs of the neural network. The data frame becomes progressively each record is used to store the decimal value present at the output of Digital-Analog converter and each time you change the data logging is activated for the input of the neural network.

Keywords: Identification, induction motor, rotor resistance, FPGA, system generator.

RESUMEN

En este trabajo se presenta la implementación de Redes Neuronales Artificiales en un dispositivo lógico programable especificamente en una FPGA, la finalidad es desarrollar un modelo en *Sysgen* capaz de estimar la resistencia rotórica en motores de inducción, para llevar a cabo el modelo es necesario realizar una serie de pasos iniciando por conocer el modelo simulado del sistema motor de induccion, el cual fue implementado en Simulink de Matlab, siguiendo con el proceso de desarrollo del modelo de regresion neuronal conociendo previamente los datos respectivos de las variables implicadas en el calculo de la resistencia rotorica. Para crear una neurona en *Sysgen* se debe partir del modelo estándar de una neurona artificial. La red neuronal utiliza datos decimales de entrada-salida, y la FPGA es digital, por lo tanto se diseñó en *Sysgen* un acondicionamiento capaz de manejar datos binarios en la tarjeta, para esto se utilizó un conversor Digital-Análogo, y se creó la manera de suprimir el número de pines utilizados en la FPGA, diseñando una etapa de registros que sirve para guardar el dato deseado y por ende enviarlo a una de las entradas de la red neuronal. Una vez finalizado la implementacion es validada comparando los resultados presentados en el display de la FPGA con los obtenidos en la simulacion, generando resultados interesantes con alto grado de aciertos.

Palabras clave: Identificación, motor de inducción, resistencia del rotor, FPGA, *system generator*.

1. INTRODUCCIÓN

En cualquiera de las ramas de la ingeniería existe la necesidad de crear sistemas inteligentes que puedan tener autonomía a la hora de solucionar un problema concreto (Acosta y Zuluaga, 2000; Martín y Sanz, 2002). En la presente investigación se utilizaron redes neuronales artificiales para predecir el valor de la resistencia rotórica teniendo en cuenta un modelo en *simulink* del motor de inducción jaula de ardilla desarrollado con este fin, del cual se extrajeron los resultados, entrenando un modelo neuronal artificial para obtener un valor próximo al real.

Debido a que la implementación es digital y las entradas de la red son números con punto flotante, la red neuronal no se puede ejecutar directamente en la tarjeta, sólo simular, por esto se tiene que diseñar una adaptación del programa en bloque para que se pueda implementar la red neuronal para estimar la resistencia rotórica de un motor de inducción. La inquietud de aplicar una red neuronal en algún dispositivo de *hardware* como lo son los dispositivos lógicos programables sigue siendo un tematica actual de investigación (Moctezuma y Torres, 2006; Medina, 2009), ya que se enfrenta a diferentes retos; como son los tipos de sistemas y sus conversiones; tales como en las redes neuronales con entradas analógicas surge el reto de transformarlas a su versión discreta por ser más práctica una implementación digital, a menos que el Dispositivo Lógico Programable (PLD) tenga entradas analógicas. El estudio de las redes neuronales ha sido un gran aporte al área de la electrónica digital ya que ha permitido el desarrollo de numerosas aplicaciones (Moctezuma y Torres, 2006; Medina, 2009).

Algunos de los estudios sólo se han quedado en proyectos de simulación, varios autores han desarrollado sistemas para simular el comportamiento de algún tipo de red neuronal. La mayoría de trabajos han sido desarrollados por medio de herramientas orientadas a la simulación, como lo es *Matlab,* ya que este software profesional proporciona numerosas y potentes herramientas como módulos de programación y graficación de los resultados.

2. MARCO TEORICO

La implementación de Redes Neuronales Artificiales (RNA) en Dispositivos Lógicos Programables (PLD), como las FPGAs (*Field Programmable Gate Arrays*), ayudan a desarrollar investigaciones ingenieriles de avanzada, motivacion suficiente para incursionar en el desarrollo de nuevas aplicaciones en esta aréa. A continuación se revisan los conceptos y principios básicos que sirven de base a este trabajo.

2.1 MODELO GENERAL DE LA NEURONA ARTIFICIAL

Las neuronas artificiales se modelan mediante unidades de proceso. Como se muestra en la figura 1, se componen de un núcleo (donde se realiza la ponderación de las entradas, función de propagación y función de activación) encargado de los cálculos, una red o vector de conexiones de entrada, y una salida.

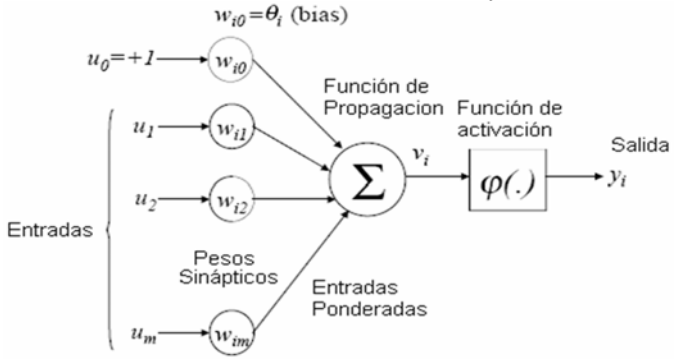


Figura 1: Modelo de una neurona artificial.

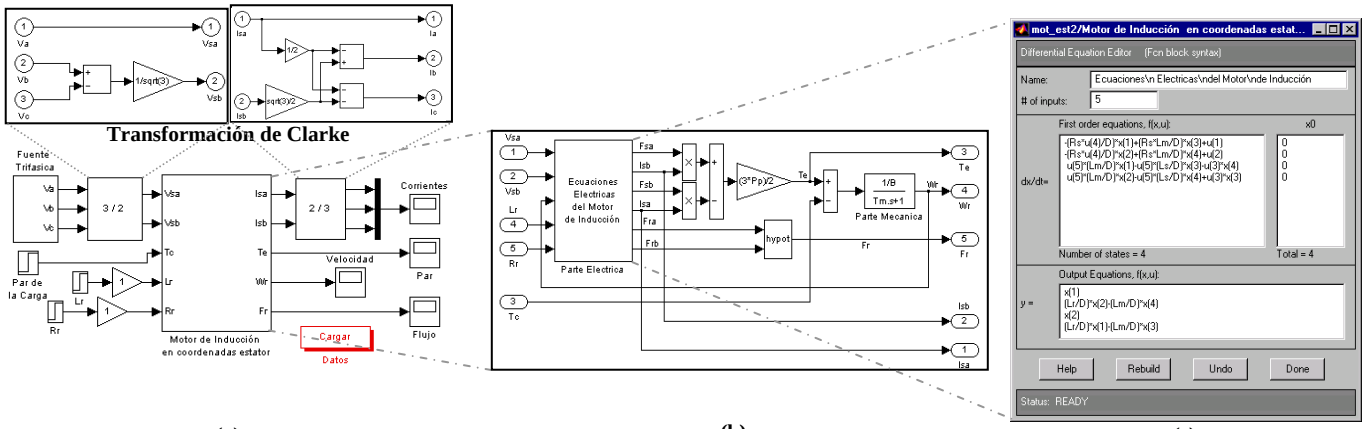
Entradas ponderadas: Realiza las conexiones sinápticas, el peso de la conexión equivale a la fuerza o efectividad de la sinapsis. Las existencias de conexiones determinan si es posible que una unidad influya sobre otra, el valor de los pesos y el signo de los mismos definen el tipo (excitatorio/inhibitorio) y la intensidad de la influencia.

Función de propagación: Calcula el valor de base o entrada total a la unidad, generalmente como simple suma ponderada de todas las entradas recibidas, es decir, de las entradas multiplicadas por el peso o valor de las conexiones. Equivale a la combinación de las señales excitatorias e inhibitorias de las neuronas biológicas.
Función de activación: Es quizá la característica principal o definitoria de las neuronas, la que mejor define el comportamiento de la misma. Se usan diferentes tipos de funciones, desde simples funciones de umbral a funciones no lineales. Se encargan de calcular el nivel o estado de activación de la neurona en función de la entrada total.
Salida: Calcula la salida de la neurona en función de la activación de la misma, aunque normalmente no se aplica más que la función identidad, y se toma como salida el valor de activación. El valor de salida cumplirá la función de la tasa de disparo en las neuronas biológicas.

2.2 MODELO DEL MOTOR DE INDUCCIÓN

Para la elaboración de los programas primeramente se simuló el modelo del motor de inducción (MI), el cual puede expresarse como: un sistema de coordenadas fijo en el estator (coordenadas estacionarias), un sistema de coordenadas en el rotor o de coordenadas sincrónica, obteniéndose resultados idénticos. (Pardo y Díaz, 2005).

Aunque el sistema en coordenadas sincrónica ha sido el más utilizado en el estudio del motor de inducción, por presentar la importante propiedad de representar las variables sinusoidales del sistema de coordenadas trifásico en valores constantes, trayendo consigo mayor estabilidad numérica al solucionar el sistema de ecuaciones diferenciales no lineales propias del modelo del motor de inducción. Con la potencia de cálculo que se cuenta en la actualidad esto ya no es un problema, por lo que el modelo finalmente desarrollado está en coordenadas estacionarias con la ventaja que no necesita de la transformación para un sistema de ejes en movimiento (transformación de Park), ni requiere de la velocidad sincrónica como una variable más de entrada.



(a) **(b)** **(c)**

Figura 2: Modelo del MI. a) Diagrama en Bloques. b) Modelo del motor. c) Ecuaciones del motor.

La resistencia del rotor del motor de inducción puede variar hasta un 100% debido al calentamiento del rotor, y obtener este valor con un modelo de temperatura o un sensor no es una solución práctica. Además la resistencia del rotor puede cambiar de manera significativa con la frecuencia del rotor debido al efecto de proximidad en las máquinas con doble jaula y rotores de barra profunda. Obtener el valor exacto de la resistencia del rotor es requisito indispensable para los esquemas de control vectorial por campo orientado (FOC).

2.3 FPGAS

Las FPGAs contienen bloques lógicos relativamente independientes entre sí, con una complejidad similar a un PLD de tamaño medio. Estos bloques lógicos pueden interconectarse, mediante conexiones programables, para formar circuitos mayores. Existen FPGAs que utilizan pocos bloques grandes (*Pluslogic*, *Altera* y *AMD*) y otras que utilizan muchos bloques pequeños (*Xilinx*, *AT&T*, *Plessey* y *Actel*). A diferencia de los PLDs, no utilizan arquitectura de matriz de puertas AND seguida de la matriz de puertas OR y necesitan un proceso adicional de ruteado del que se encarga un software especializado. La primera FPGA la introdujo *Xilinx* en el año 1985. La

programación de las FPGAs de *Xilinx* basadas en RAM estática es diferente a la programación de los PLDs. Cada vez que se aplica la tensión de alimentación, se reprograma con la información que lee desde una PROM de configuración externa a la FPGA. Una FPGA basada en SRAM (RAM estática) admite un número ilimitado de reprogramaciones sin necesidad de borrados previos. En general la complejidad de una FPGA es muy superior a la de un PLD. Los PLD tienen entre 100 y 2000 puertas, las FPGAs tienen desde 1200 a 20.000 puertas y la tendencia es hacia un rápido incremento en la densidad de puertas. El número de *flip-flops* de las FPGA generalmente supera al de los PLD. Sin embargo, la capacidad de la FPGA para realizar lógica con las entradas suele ser inferior a la de los PLD. Por ello: "*los diseños que precisan lógica realizada con muchas patillas de entrada y con pocos flip-flops, pueden realizarse fácilmente en unos pocos PLDs, mientras que en los diseños en los que intervienen muchos registros y no se necesita generar combinaciones con un elevado número de entradas, las FPGAs pueden ser la solución óptima*".

3. MODELO EN SYSGEN CON REDES NEURONALES PARA ESTIMACIÓN DE LA RESISTENCIA ROTÓRICA

Para crear una red neuronal en *System Generator*, se debe conocer el comportamiento del modelo del motor de inducción, para esto se debe desarrollar una data en *Matlab* que se utiliza para el entrenamiento y desarrollo de la red neuronal, utilizando la función *newff* para la creación de la red y la función *train* para el entrenamiento.

La *data* es obtenida de un modelo desarrollado previamente en el entorno de simulación *Simulink* de *Matlab*, este modelo posee la resistencia del rotor (R_r) como variable de entrada, lo que permite su variación durante la simulación. Para obtener el valor de la resistencia rotórica basta con conocer el valor de algunas variables y simular el modelo que consta de varias ecuaciones (figura 2c), una forma de representar el sistema se muestra en la figura 3, teniendo las cinco entradas y el valor de la resistencia rotórica como salida, la red neuronal se establece de acuerdo al comportamiento de la salida, a mayor picos inconstantes, mayor complejidad.

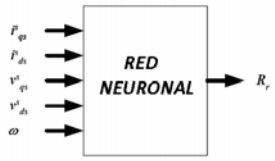


Figura 3: Red neuronal para estimación de resistencia rotórica.

Las entradas y salida de la red neuronal son las siguientes:

- i^s_{qs} Corriente del estator en cuadratura en coordenadas sincrónicas.
- i^s_{ds} Corriente del estator en directa en coordenadas sincrónicas.
- v^s_{qs} Voltaje del estator en cuadratura en coordenadas sincrónicas.
- v^s_{ds} Voltaje del estator en directa en coordenadas sincrónicas.
- ω Velocidad del Rotor.
- R_r Resistencia del Rotor.

Luego de tener la *data* en *Matlab* se crea la red neuronal (seleccionando el número de capas), luego se entrena la red con el método de retropropagación del error, procedimiento por el cual se obtienen los pesos y umbrales; y finalmente se simula la red en el *Matlab*. Existen dos formas de comprobar que tan exacto aproxima la red, la primera es tabular los datos de salida simulados con los datos originales, la segunda es graficando estos mismos valores, se optó por la primera por la accesibilidad a los valores y la facilidad de establecer la comparación.

3.1 RED NEURONAL PARA PREDECIR LA RESISTENCIA ROTÓRICA

El número de neuronas depende del programador y de la complejidad de la señal, este comprueba que tan viable es utilizar más o menos neuronas o capas, a mayor número de neuronas la salida puede ser más parecida a la original, pero si se quiere implementar la red en una FPGA se debe establecer que tanta memoria posee el dispositivo y que otras aplicaciones adicionales se van a desarrollar además de la red para que la FPGA pueda soportar todo el programa, a menor número de neuronas la red pierde exactitud, todo depende del comportamiento de la señal original.

Para el entrenamiento se deben poseer los datos posibles de entrada y salida correspondiente, en este caso del comportamiento del motor de inducción jaula de ardilla (modelado), como se observa en la figura 4 que muestra el comportamiento de las entradas.

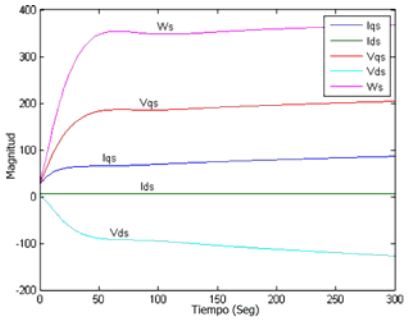


Figura 4: Gráfica del comportamiento de las entradas de la red neuronal.

Se utiliza el comando *newff* del *Matlab* para crear la red. Esta red tiene varias características, una de ellas es que se puede escribir el número de neuronas y la función de activación en la capa oculta y en la capa de salida, las únicas funciones de activación que maneja esta función es *tansig*, *logsig*, *purelin*, estas son las que permite el tipo de entrenamiento neuronal retropropagación (*backpropagation*) que es el adecuado para la investigación puesto deja crear varias neuronas y varias capas. Conociendo el número de neuronas, por consiguiente los pesos y los umbrales, se procede a crear la red neuronal artificial en *Xilinx System Generator* partiendo del modelo de una neurona estándar, como se ilustra en la figura 5.

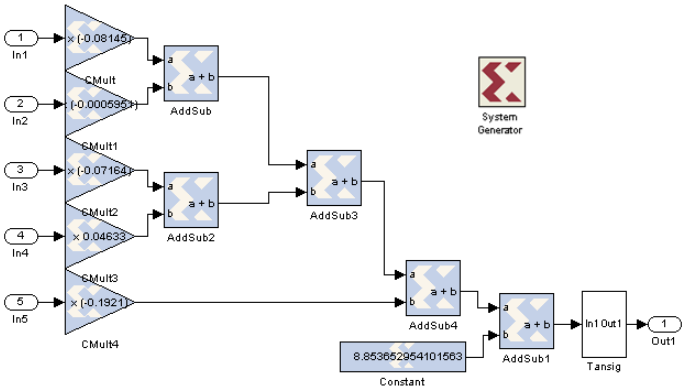


Figura 5: Modelo en *Sysgen* de una neurona estándar.

Luego de tener la neurona se crea la capa oculta que consta de doce neuronas cada una con función de activación *Tansig* y una neurona de salida con función de activación *Purelin*, en la figura 6 muestra el funcionamiento de la red neuronal sin adecuaciones de implementación.

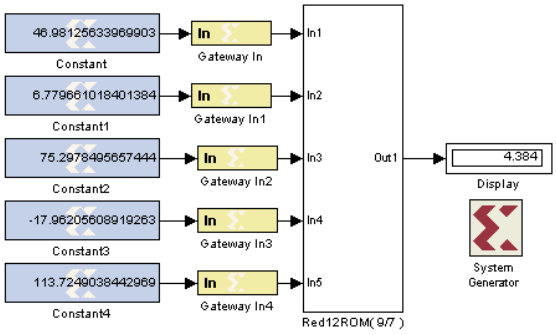


Figura 6: Red neuronal para estimación de resistencia rotórica.

3.2 ACONDICIONAMIENTO DEL MODELO PARA LA IMPLEMENTACIÓN EN LA FPGA

Las entradas y salidas de la *FPGA SPARTAN3 XC3S1000-5FT256* son digitales, y la red neuronal maneja sólo datos decimales, para lo que se diseña un programa en bloques en *Sysgen* para recibir datos externos digitales, y convertirlos a valores decimales permitiendo enviarlos a las entradas de la red neuronal y así visualizar el valor obtenido de la resistencia rotórica del motor de inducción en el display de la FPGA.

3.2.1 CONVERSOR DIGITAL-ANÁLOGO Y ANÁLOGO-DIGITAL

Para la conversión digital-análogo se utiliza dos bloques de *System Generator* uno de ellos es *concat* que realiza una concatenación de los vectores de *n* bits representados por números enteros sin signo, es decir, números sin signo con puntos binarios en la posición cero, el otro bloque es *reinterpret* que sirve para reinterpretar el dato de entrada teniendo en cuenta el número de bits y el punto binario que denota el bloque *reinterpret*. El bloque *concat* es el que realmente hace la conversión digital-análoga (con o sin signo), lo único que no puede hacer es manejar punto binario por lo que entonces la salida siempre va resultar entera, con este fin se utiliza el bloque *reinterpret*, este sirve para expresar la ubicación del punto binario (figura 7).

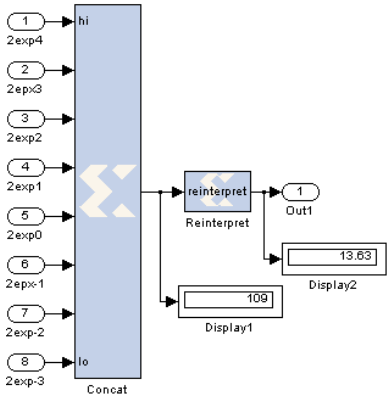


Figura 7: Bloques para la conversión digital-análoga.

Para la conversión análoga-digital se utilizan dos bloques, uno de ellos es *cast* que se utiliza como configurador de número de bits y punto binario, este bloque se conecta a bloques de *Slice* (la cantidad de bloques depende del número de bits digitado en el bloque *cast*). El bloque *Slice* cumple la función de realizar la conversión de análogo-digital, cada bloque tiene un número específico que comienza desde el menos significativo hasta el más significativo, para conformar así el conversor, ya que cada salida del bloque *Slice* es un dato binario que hace parte de la trama total. En la figura 8 se muestran los bloques para la conversión análoga-digital.

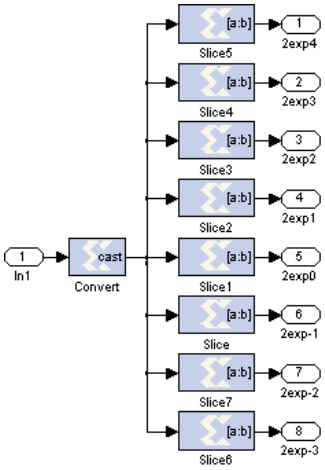


Figura 8: Bloque para la conversión análoga-digital.

3.2.2 REGISTROS

La configuración de registros que se utilizaron sirven para ahorrar entradas físicas en la FPGA, porque si no se utilizaran estos sería necesario configurar 75 entradas y la red de interruptores se haría demasiado extensa. El bloque de registros funciona de la siguiente manera: como la red neuronal necesita de 5 entradas entonces se necesita ese mismo número de bloques *register*, todos los registros tienen una entrada en común que viene del conversor digital-análogo (*Dato*) y que posee la información, cada registro tiene una entrada *enable* (*en*) cuya función es guardar el *Dato* (este se comporta como un pulso ascendente).

La secuencia de envió de información a la red neuronal parte del primer bloque *register*, entonces cuando ya se tiene el dato correcto a la salida del conversor se habilita *enable* (1er bloque *register*) por medio de un 1 lógico para que la información quede disponible a la salida del registro o sea guardada. Si se quiere introducir otro dato, el paso a seguir es colocar un 0 lógico en el *enable* de todos los bloques *register*, para que posteriormente se pueda enviar la información al registro competente habilitándolo con un 1 lógico en la entrada *enable*, y así quede guardado el dato en ese bloque *register*, la red de registros resultante para un sólo dato se ilustra en la figura 9.

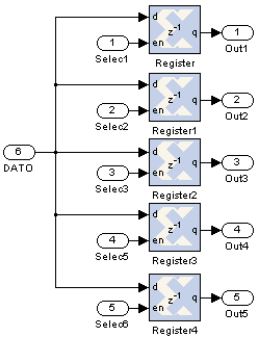


Figura 9: Red de registros para un solo dato.

3.2.3 ETAPA SEPARA DÍGITOS

Como es objetivo visualizar el dato de la resistencia rotórica del motor de inducción en una FPGA, se crea entonces la etapa separa cifras, ya que el *display* que tiene el dispositivo lógico programable maneja la configuración de los *LEDs* a una sola línea para todos, pero cada uno de ellos es activado por un ánodo común propio. La estructura consiste en tomar un número completo decimal y separarlo, sólo basta con separar 4 números puesto que ese es el número de display que tiene la FPGA. Para hacer la separación se utilizaron bloques de multiplicación *CMult* cuya función es ir corriendo el punto decimal y así poder utilizar condicionales para convertir ese número en entero y tener un mejor manejo en la clasificación.

Cuando el número ya es entero y se quiere obtener por lo menos el tercero de ellos de izquierda a derecha, se utilizan bloques de *System Generator* que puedan soportar código concurrente, uno de ellos es *mcode*. La clave para este problema específico es realizar un programa que vaya corriendo los datos de derecha a izquierda hasta llegar al número deseado y así condicionarlo a un sólo número. La figura 10 muestra el interior del subsistema separa cifras y se puede ver claramente la simulación y como el sistema separa los números.

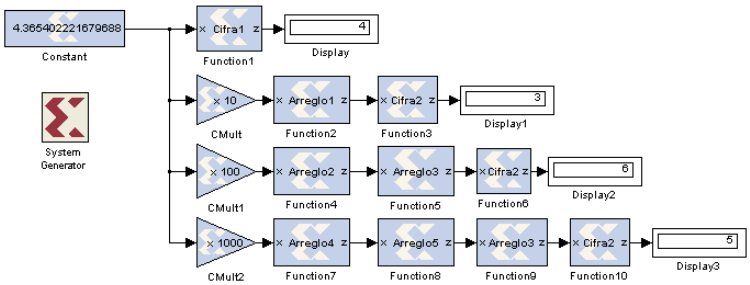


Figura 10: Modelo utilizado para separar números.

3.2.4 SISTEMA DE CONTADORES

La secuencia de envió de información consiste en activar todo el sistema por medio de la entrada habilita. Internamente el subsistema de la *etapa_count1* activa un contador y a su vez envía un 0 lógico al ánodo común para que se encienda el *display* correspondiente de la FPGA, además posee otras salidas como *Actusig* cuya función es habilitar la etapa siguiente *rst* que tiene la tarea de resetear el contador siguiente para que toda la simulación se desarrolle secuencialmente, y *Padcom*, que sirve para habilitar el registro de la segunda etapa cuando el contador (*etapa_count2*) empiece a funcionar.

Todas las etapas tienen comportamientos similares la única diferencia es que las etapas 3 y 4 tienen otro contador alterno que sirve para enviar una ráfaga de pulso al registro correspondiente cuando el contador principal ejecuta su acción. En la figura 11 muestra la configuración de contadores secuencialmente que envían la información progresivamente a la red de multiplexado y a los ánodos comunes de los respectivos *displays*. Uno de los problemas más frecuentes es los ciclos de activación y desactivación en los registros para que se guarde el dato, porque con el primer ciclo se podía activar el registro de la tercera etapa, ya que toda la simulación se realiza de forma paralela, motivación para implementar ráfagas internas con otro contador alterno.

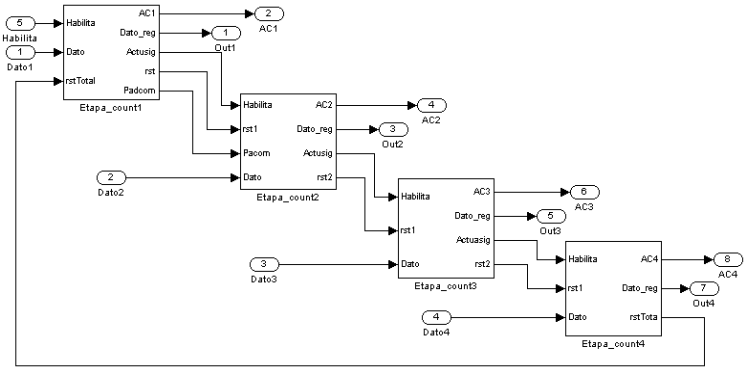


Figura 11: Etapas de contadores para el envío de información secuencial.

3.2.5 MULTIPLEXORES

La red de multiplexores sirve para seleccionar el dato que va hacer visualizado en el *display* teniendo en cuenta la información que provee de la red de contadores para que solamente pase un dato al bloque que genera la activación de cátodos comunes de la FPGA.

El funcionamiento del bloque multiplexores es el siguiente: en la red de contadores se genera un 1 lógico secuencialmente, entonces se crean varios bloque acumuladores para que generen un número cuando algún ánodo este activado, ese número es tomado por un bloque *mcode* para que posteriormente se cree una combinación binaria de acuerdo a ese número y deje pasar el dato correcto, como se ilustra en la figura 12.

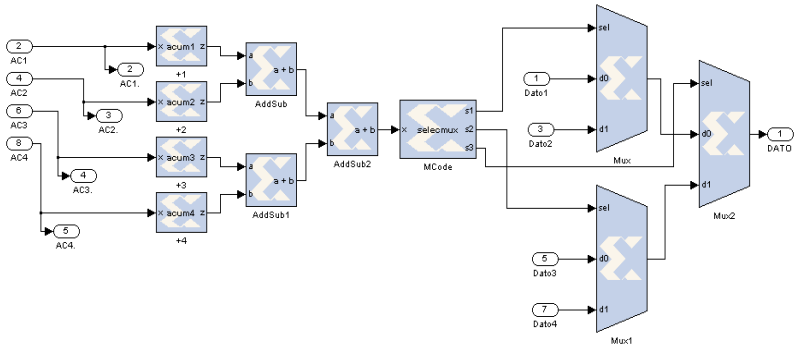


Figura 12: Bloque de selección de dato.

3.2.6 ETAPA VISUALIZACIÓN EN EL DISPLAY DE LA FPGA

Para este subsistema se utilizó un bloque de *mcode*, en este se realiza la debida programación la cual consiste en identificar un número del 1 al 9 y posteriormente enviar una trama de datos binarios para que pueda encender los diferentes cátodos y así formar el número correspondiente (figura 13).

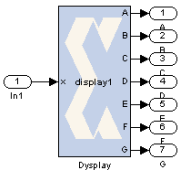


Figura 13: Bloque para visualización de display.

3.3 IMPLEMENTACIÓN DEL MODELO DE REDES NEURONALES EN LA FPGA

Para implementar el programa en bloques de *Sysgen* en una FPGA, lo primero que se debe hacer es tener la certeza de la correcta simulación en *Simulink* del programa. El siguiente paso es generar el archivo *bitstream* utilizando el bloque *System Generator*, la cual debe ser configurado de la siguiente manera (ver figura 14).

A continuación se configura todo el bloque de *System Generator* es darle click en el botón *Generate* para que de inmediato empiece a generar el archivo *Bitstream*. En el momento de generar el tipo archivo en un destino seleccionado, se despliega un enlace para poder terminar, y el trabajo en *Simulink* ha llegado a su final.

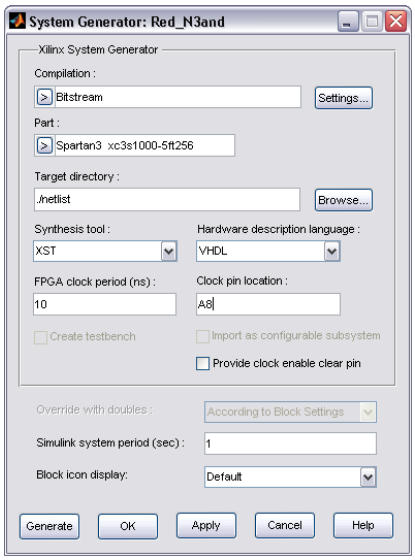


Figura 14: Configuración del *System Generator*.

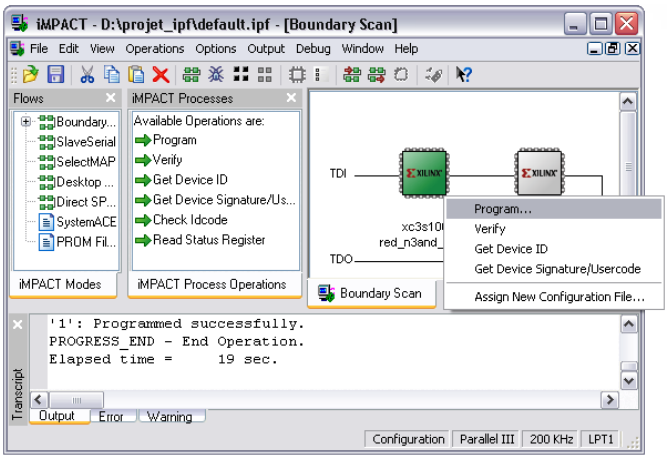


Figura 15: Entorno del Software *iMPACT*.

Para la implementación en hardware se utiliza una herramienta computacional llamada *iMPACT*, que se consigue en *Xilinx ISE 8.2i / Accesories / iMPACT*, al pinchar ahí se despliega una ventana la cual se muestra en la figura 15, esta sirve para crear un nuevo proyecto o para abrir un proyecto guardado, aquí se selecciona *create a new Project* y se selecciona *OK*.

Seguidamente se despliega una ventana de *iMPACT*, antes de oprimir el botón *finish* se debe encender la FPGA para que el programa reconozca el PLD, después que ejecute esta acción se abre una ventana que sirve para buscar el archivo *Bitstream* e introducirlo en la tarjeta, posteriormente aparece un aviso, este sale siempre que se busca el destino del archivo *netlist*, se selecciona *OK*, para que salga otra ventana inutilizable, y después aparezca el entorno de *iMPACT*.

El programa *iMPACT* detecta todos los componente configurables que se encuentran en la tarjeta, por esta razón es que aparece una ventana al inicio para cada *chip*, y no se utiliza el segundo *chip* porque se está seleccionando es la memoria *Flash* de la *Naxis* y para este tipo de programación no la puede ejecutar. Para iniciar la programación ahí que dirigirse al esquema *chip* donde está la información, darle clic derecho sobre ese componente y se selecciona la opción *program*, como se ilustra en la figura 15, después se despliega una ventana que permite cambiar la configuración de programación, se deja la configuración inicial, y se le da clic en *OK* para que empiece la programación y la finalice, y así verificar la funcionalidad del programa realizado en *System Generator* en la FPGA.

3.4 OBTENCIÓN Y COMPARACIÓN DE RESULTADOS

Una vez generado el archivo de programación, el siguiente paso es introducir el programa en la tarjeta de *Nexys* utilizando el software *Impact* para que ejecute la red neuronal en la FPGA. En el momento que el programa se este ejecutando en la tarjeta lo común ha realizar es verificar el correcto funcionamiento de este, una forma es realizar una tabla en donde se compara el valor real (extraído del modelo del motor), con el valor que proporciona la tarjeta en los *displays* (extraído de la ejecución del programa), como se muestra en la tabla 1.

En esta tabla se puede observar la diferencia que existe entre los dos resultados es relativamente pequeña, inferior al 2% en el peor de los casos, por lo que se puede afirmar que los valores obtenidos por la red neuronal se asemejan en gran medida al valor original obtenido por el modelo matemático. Los datos fueron extraídos aleatoriamente para verificar su funcionalidad.

Tabla 1: Comparación del dato real con el dato obtenido.

Posición	Dato real (modelo)	Dato obtenido Display	Diferencia
2	4.1743	4.1794	0.0051
3	4.2124	4.2111	0.0013
4	4.2495	4.2432	0.0063
5	4.2858	4.2749	0.0109
6	4.321	4.3154	0.0056
7	4.3549	4.3471	0.0078
8	4.3877	4.3839	0.0038
9	4.4195	4.4184	0.0011
10	4.4501	4.4535	0.0034
11	4.4799	4.4869	0.0070
12	4.5088	4.5079	0.0009
13	4.540	4.5394	0.0006
14	4.5641	4.5700	0.0059
15	4.5907	4.5870	0.0037
16	4.6167	4.6210	0.0043
17	4.642	4.6429	0.0031

Posición	Dato real (modelo)	Dato obtenido Display	Diferencia
18	4.6669	4.6648	0.0189
19	4.6912	4.6980	0.0068
20	4.7157	4.7094	0.0063
40	5.1323	5.1241	0.0082
80	5.8503	5.8500	0.0003
100	6.1935	6.1935	0.0000
120	6.5282	6.5240	0.0042
140	6.8534	6.8575	0.0041
160	7.1694	7.1748	0.0054
180	7.4773	7.4715	0.0005
200	7.7777	7.7828	0.0051
220	8.0713	8.0577	0.0136
240	8.3588	8.3701	0.0113
260	8.6404	8.6383	0.0021
280	8.9165	8.8918	0.0247
300	9.1874	9.1697	0.0177

4. CONCLUSIONES

El hecho de llevar a un dispositivo hardware modelos neuronales que permitan predecir el comportamiento de variables o clasificar características hacen que sea una temática interesante de estudio en la actualidad, en este caso en particular fue analizado el comportamiento de la resistencia rotórica en motores de jaula de ardilla, a partir de diferentes valores de corrientes del rotor (i_{qs}, i_{ds}), voltajes del rotor (v_{qs}, v_{ds}),y velocidad del rotor ω (5 entradas), destacando que la obtención del valor exacto de la resistencia del rotor es requisito indispensable para los esquemas de control vectorial por campo orientado. La necesidad de implementar en una FPGA dicho proceso de identificación es fundamentada en el hecho de que obtener estos valores con modelo de temperatura o un sensor no sería una solución práctica.

La estrategia se desarrolló partiendo del conocimiento respectivo del modelo matemático de inducción del motor implementado en Simulink de MATLAB, del cual son extraídas las respectivas datas de entrada y salida relacionadas con los parámetros requeridos. Una vez creado el modelo de identificación neuronal utilizando

MATLAB fue tomada la información relevante del mismo (pesos y umbrales respectivos) para ser llevada a la FPGA, explotando en su capacidad máxima los bloques internos de dicho dispositivo debido a la necesidad de generar etapas de multiplexación y demultiplexacion por la limitante del mismo a solo trabajar con una entrada analógica.

La tabla comparativa entre los datos reales del modelo matemático simulados y los obtenidos en el display del dispositivo es prometedora ya que se verifica que los márgenes de error entre una y otra son mínimos, (inferiores al 2 %), siendo de esta manera una estrategia novedosa de estimación no solo de las resistencias rotóricas sino para cualquier aplicación donde sea necesario predecir los comportamientos de las variables.

El siguiente paso en el proceso de investigación será realizar pruebas experimentales del dispositivo implementado acoplado a un motor tomando la información en tiempo real de las variables de corriente, voltaje, etc.

REFERENCIAS

Acosta, M.I. y Zuluaga, C.A. (2000). "Tutorial sobre redes neuronales aplicadas en Ingeniería Eléctrica y su implementación en un sitio Web". Trabajo de grado, Universidad Tecnológica de Pereira, Colombia.

Martín, B. y Sanz, A. (2002). *Redes Neuronales y Sistemas Difusos*. 2ª Edición, Ed. Alfaomega, Madrid, España.

Moctezuma, J.C. y Torres, C. (2006). "Estudio sobre la implementación de redes neuronales artificiales usando Xilinx System Generator". Trabajo de grado, Benemérita Universidad Autónoma de Puebla, Puebla, México.

Medina, J.O. (2009). "Metodología para la implementación de redes neuronales en dispositivos lógicos programables aplicadas al control de desplazamiento de un robot móvil". Trabajo de grado Maestría en Controles Industriales, Universidad de Pamplona, Colombia.

Pardo, A. y Díaz, J.L. (2005). *Fundamentos en sistemas de control automático*. Universidad de Pamplona, Colombia.

Authorization and Disclaimer