SINTESIS HARDWARE DE REDES ALN PARA APLICACIONES EN CONTROL

J. P. OLIVER* , A. FONSECA DE OLIVEIRA, J. PEREZ, R. CANETTI

Instituto de Ingeniería Eléctrica,

Facultad de Ingeniería. Universidad de la República,

Herrera y Reissig 565, Montevideo - Uruguay

e-mail: jpo@iie.edu.uy

Palabras Clave: FPGAs, hardware reconfigurable, redes neuronales artificiales, Redes Lógicas Adaptivas (ALN), Control en Tiempo Real.

Resumen

Este trabajo es parte de un proyecto que estudia la realización de algoritmos de redes neuronales en hardware reconfigurable como manera de obtener un procesador neuronal de alta performance. El objetivo de este procesador es poder ejecutar procesamiento neuronal de alta velocidad, para aplicaciones de tiempo real en control y tratamiento de señales.

Se presentan los resultados para redes binarias del tipo Redes Lógicas Adaptivas (Adaptive Logic Network: ALN), sintetizadas sobre lógica programable, aplicadas en control. El procesador desarrollado se probó en el control realimentado de una planta inestable y no lineal.

Los diseños se realizaron sobre una plataforma hardware consistente en un computador personal como host y una placa reconfigurable ALTERA RIPP10 para el cálculo en hardware. La placa contiene nueve FPGAs FLEX8K y 512KB de RAM. El computador personal se comunica con la placa RIPP10 a través del bus ISA.

Para el trabajo con redes ALN se utiliza habitualmente el paquete de software ATREE, que consiste en una biblioteca en lenguaje C con funciones que permiten a un diseñador definir la arquitectura de la red, realizar su entrenamiento con un conjunto de patrones y posteriormente evaluar la salida de la red correspondiente a una entrada dada.

En el presente trabajo se desarrolló una herramienta software para convertir automáticamente la descripción de una red ALN que resulta del proceso de entrenamiento con ATREE en la especificación de un circuito en un lenguaje de descripción hardware (AHDL). Esta metodología permite generar fácilmente el hardware necesario para evaluar las funciones combinatorias de gran tamaño que resultan de una red ALN entrenada. Se escribieron rutinas que utilizan los circuitos generados para evaluar la salida de la red frente a un patrón de entrada dado. Estas rutinas presentan a los usuarios de la biblioteca ATREE una interfaz idéntica a la presentada por las rutinas originales de dicho paquete.

La planta a controlar consiste de un péndulo invertido con fuertes no-linealidades introducidas por las características de la fricción (valores altos y dependencia espacial).

Una red ALN previamente entrenada se incluyó en el lazo de control del péndulo invertido, con buenos resultados. La velocidad de cálculo obtenida es importante para la utilización de este tipo de redes en aplicaciones de control en tiempo real.

Para este ejemplo se presentan los resultados de la comparación entre los tiempos de ejecución de la evaluación de la red utilizando el procesador neuronal y los de la realización puramente en software.

1. Introducción

La utilización de lógica programable o reconfigurable para acelerar algoritmos de diverso tipo viene siendo aplicada con éxito desde hace ya algunos años (Guccione, 1995, Cloutier *et al.*, 1996). En este trabajo se utiliza el esquema de lógica reconfigurable trabajando en paralelo con la CPU de un computador personal, la lógica reconfigurable realiza la parte más pesada de un determinado algoritmo, aumentando así su velocidad de ejecución.

Para poder ampliar la utilización de esta tecnología a aplicaciones de potenciales usuarios es necesario salvar la brecha que existe entre las herramientas de desarrollo diseñadas para especialistas en el área de diseño digital y las aplicaciones finales.

En este sentido aparecen dos grandes tendencias: la creación de lenguajes y compiladores de alto nivel que sean capaces de traducir y trasladar automáticamente determinados algoritmos a hardware reconfigurable, y el desarrollo de bibliotecas o módulos reutilizables que

* Author to whom correspondence should be addresed

puedan ser invocados por las aplicaciones. Esta última solución es la que se plantea en este trabajo.

El enfoque de este trabajo es la realización de bibliotecas hardware de redes tipo Adaptive Logic Networks (ALN) desde el punto de vista del usuario final. En una biblioteca de funciones C se sustituyen algunas por su versión hardware y se comparan los resultados obtenidos.

Este trabajo es continuación de uno anterior presentado por los mismos autores (Oliver *et al.*, 1998).

En la sección 2 de este artículo se realizan algunos comentarios acerca de las redes neuronales artificiales y su implementación hardware. La sección 3 es una breve introducción sobre Redes Lógicas Adaptivas (ALN). La plataforma hardware utilizada se describe en la sección 4. La sección 5 describe un ejemplo de una red ALN previamente entrenada para controlar un péndulo invertido. En la sección 6 se describe el experimento realizado durante el control de una planta física real. Finalmente, en la sección 7, se presentan las conclusiones.

2. Redes Neuronales Artificiales

Las redes neuronales artificiales (ANNs) presentan múltiples aplicaciones en el campo del control y el reconocimiento de patrones (autores varios en *Artificial Neural Networks: Paradigms, Aplications, and Hardware Implementations, 1992*, Hecht-Nielsen, 1991, Haykin, 1994, Supynuk y Armstong, 1992, Armstrong y Gecsei, 1978). En este ámbito han cobrado un creciente interés como una herramienta alternativa y factible para resolver una enorme cantidad de problemas de ingeniería, en especial aquellos en los que los procedimientos convencionales presentan marcadas limitaciones. Tal es el caso, por ejemplo, cuando se requiere de la toma de decisiones frente a situaciones difíciles de cuantificar y por lo tanto no muy bien definidas analíticamente. Este hecho es debido fundamentalmente a la capacidad que las redes neuronales poseen de procesar un conjunto de hipótesis representativo del problema a resolver, emitiendo una respuesta acorde a la experiencia previa adquirida aún cuando tales hipótesis no fueron anteriormente presentadas (generalización).

La realización software de ANN implica una carga importante de procesamiento, ya que cuando es implementada de esta forma el procesamiento se da en forma secuencial (inherente de la arquitectura de la computadora de propósito general sobre la cual el programa esta funcionando). Esto limita sus aplicaciones a cálculos fuera de línea o a aplicaciones en tiempo real sobre procesos lentos. Otra limitación proviene de que determinados algoritmos de adaptación de las redes (o de entrenamiento), requieren de un poder de cómputo extremadamente alto cuando la dimensión de la red a ser adaptada es relativamente grande. En estos casos, aun cuando la red entrenada tenga una velocidad de procesamiento adecuada, resulta poco práctico el tiempo necesario para la adaptación de esta.

Si bien la utilización de redes neuronales como herramienta para la solución de problemas reales de ingeniería (y otras áreas) resulta una alternativa interesante, las razones expuestas anteriormente muestran una clara limitación en su empleo si esta es implementada mediante programas secuenciales. Esto hace atractivo contar con una herramienta que permita implementar redes neuronales en hardware, permitiendo de este modo explotar la potencialidad del cálculo paralelo.

2.1 Implementación hardware de redes neuronales

Los primeros intentos de realizar redes neuronales en hardware son tan antiguos como la propia área. En 1951 Marvin Minsky construyó la primera neuro-computadora denominada Snark. Aunque esta máquina operaba bien desde el punto de vista técnico (realizaba el ajuste automático de los pesos), nunca llegó a resolver ninguna aplicación con resultados importantes desde el punto de vista del procesamiento de la información suministrada. El primer prototipo exitoso fue el Mark I Perceptron desarrollado por Frank Rosenblatt, Charles Wightman y otros. El Mark I fue utilizado con éxito en el reconocimiento de caracteres (Hecht-Nielsen, 1991).

En los últimos años tanto la academia como la industria han trabado incesantemente sobre como implementar redes en hardware. Buenas referencias sobre los trabajos existentes pueden ser encontradas en los artículos de Lindsey y Lindblad (1994), Heemserk (1995), Hegt (1993) y Kuhn (1995).

Este trabajo es parte de un proyecto que estudia la implementación de algoritmos de redes neuronales en hardware reconfigurable como forma de obtener un procesador neuronal de alta performance. Uno de los tipos de redes estudiado fue la Adaptive Logic Network que se describe en la sección siguiente. Es una red digital que después de entrenada puede ser representada como un conjunto de compuertas lógicas.

3. Redes Lógicas Adaptivas (Adaptive Logic Network ALN)

Una ALN puede describirse en general como un conjunto de árboles binarios balanceados con codificación y decodificación de entradas y salidas respectivamente y con capacidad de adaptación. Los bits de los patrones de entrada se invierten de modo de formar un nuevo patrón de tamaño doble. Cada bit de entrada tiene conexiones al azar con entradas del árbol, tantas veces como sea posible, hasta utilizar todas las entradas de la capa inferior del árbol (hojas). La probabilidad de obtener éxito en la adaptación mejora con la relación entre cantidad de hojas y cantidad de bits del patrón de entradas.

Este tipo de árboles es capaz de realizar cualquier función booleana con la propiedad de ser robusto ante pequeñas variaciones de las entradas (Armstrong y Godbout, 1975, Armstrong *et al.*, 1991, Bochmann y Armstrong, 1974).

La Fig. 1 muestra un esquema básico de una red de este tipo.

Cuando el problema estudiado contiene señales reales, estas deben ser codificadas a valores binarios, que constituyen las entradas a la base de los árboles binarios que forman la ALN. Cada nodo de cada árbol recibe dos entradas lógicas y emite una salida lógica que puede ser una de las siguientes funciones: AND, OR, LEFT o RIGHT. Cada árbol da una salida que corresponde a la salida de su nodo raíz. El conjunto de salidas de todos los árboles decodificadas constituye la salida de la red neuronal. La capacidad de adaptación de estas redes se halla en los nodos de cada árbol, es decir en encontrar las funciones lógicas correspondientes a cada nodo que verifiquen la relación entrada-salida deseada. Es, por lo tanto y en esencia, una red con aprendizaje supervisado.

Cada nodo puede verse como un autómata que mantiene en su estado interno cual de las cuatro funciones (AND, OR, LEFT, RIGHT) está implementando. Dicho estado interno reside en el valor de dos contadores binarios con saturación asociados a cada nodo.

El aprendizaje se basa en la noción de responsabilidad, es decir en cómo cuantificar la dependencia que tiene el valor de salida de la red con la función elegida en cada nodo de los árboles (Armstrong y Gecsei, 1979). Cada nodo, además de las entradas ski y skd recibe desde el nodo padre (o desde la capa inmediata superior) en la estructura del árbol una entrada *r* de responsabilidad.

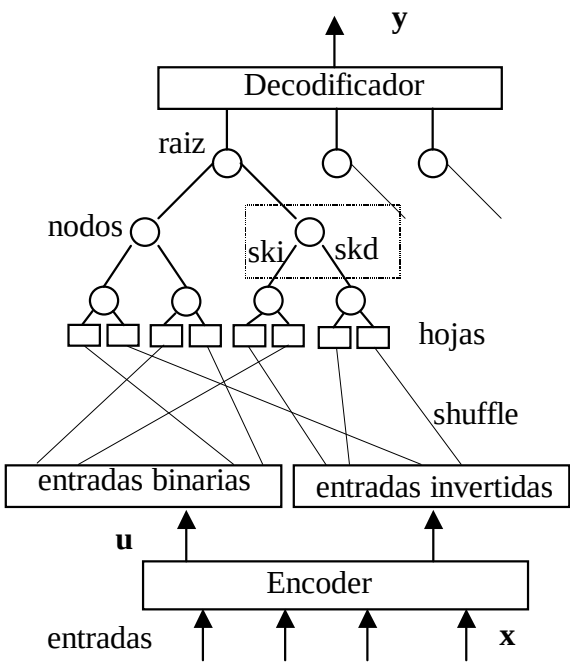


Fig. 1 Esquema de una ALN

El aprendizaje se realiza presentando a la entrada de la red el patrón de entradas y comparando la salida obtenida con la salida deseada. Cada nodo determina la responsabilidad de sus nodos hijos en función de sus entradas ski y skd, de la salida deseada, de su estado interno actual y de su propia entrada de responsabilidad.

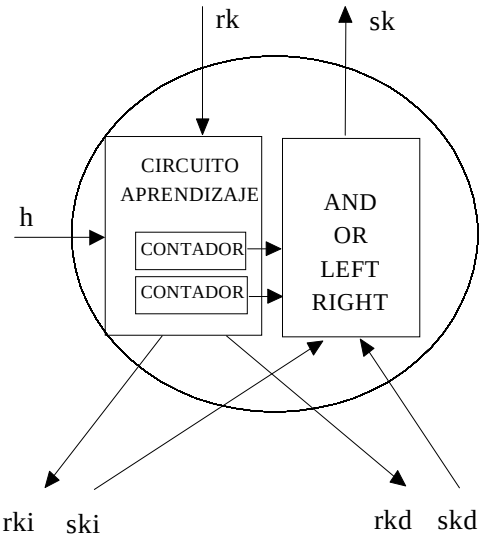


Fig. 2: Un nodo ALN

El nodo raíz siempre se considera responsable. Si un nodo es responsable, éste modifica su estado interno teniendo en cuenta también el estado actual, la salida deseada y las entradas sk desde los nodos hijos procurando tender a la relación entrada-salida necesaria para resolver el problema.

El aprendizaje comienza una vez que se ha completado la evaluación de la salida correspondiente al patrón de entradas. En el primer paso el nodo raíz de cada árbol actualiza sus contadores y evalúa las entradas de responsabilidad de los dos nodos de la segunda capa. El proceso continúa de a una capa por vez hasta llegar a la base del árbol. En algunos casos se vuelve a comparar la salida del árbol con la salida deseada y en caso de ser diferentes se realiza una segunda pasada de entrenamiento.

3.1 Algoritmo de aprendizaje

Existen varios algoritmos para la determinación de las responsabilidades hacia abajo y la actualización de los contadores (Armstrong y Gecsei, 1979), siendo los más conocidos el de "responsabilidad real" y el de "responsabilidad heurística".

El método de aprendizaje de estas redes se basa en la noción de responsabilidad real, es decir en cómo cuantificar la dependencia que tiene el valor de salida de la red con la función elegida en cada nodo de los árboles. Cada nodo tiene asociados una variable booleana r que indica la responsabilidad del nodo y 2 contadores que definen la función actual f para el nodo. El algoritmo de aprendizaje es el siguiente: se anulan los valores de los contadores de cada nodo y luego para cada patrón, es decir par de valores entrada-salida deseada,

a.- Se determinan primero los valores de entrada **u** y salida deseada **h** de cada árbol a través de la codificación de sus valores elegida.

b.- Si el valor de salida de cada árbol no es correcto se asignan las responsabilidades de acuerdo al siguiente algoritmo recursivo:

b1.- El nodo raíz del árbol es responsable.

Para un nodo k genérico, la responsabilidad de sus hijos derecho e izquierdo, rkd y rki respectivamente se fija de acuerdo a:

b2.- rki = true sii rk = true y se cumple alguna de estas condiciones:

i. - fk = Left.
ii. - fk = And y skd = 1.
iii. - fk = Or y skd = 0.

b3.- rkd = true sii rk = true y se cumple alguna de estas condiciones:

i. - fk = Right.
ii. - fk = And y ski = 1.
iii. - fk = Or y ski = 0.

Donde skd y ski son los valores actuales que dan los nodos hijos derecho e izquierdo del nodo k mencionado para el patrón presentado a la entrada de la red.

c.- Se adaptan los valores de los contadores Ca y Cb asignados a cada nodo k responsable de acuerdo al siguiente esquema:

s_{ki}	s_{kd}	**y**	Cambios
0	1	0	dec Ca
0	1	1	inc Ca
1	0	0	dec Cb
1	0	1	inc Cb

d.- Se asignan los valores de las funciones de los nodos según:

F_k	Condiciones	
	Ca	Cb
And	< 0	< 0
Left	< 0	> 0
Right	> 0	< 0
Or	> 0	> 0

En este trabajo se utilizaron las variaciones de éste algoritmo utilizadas en atree 2.0 para Unix y atree2.7 para Windows. Atree es una biblioteca pública en C desarrollada en la Universidad de Alberta, Canadá, que simula este tipo de redes (ver W. Armstrong Home Page).

Desde el punto de vista de su implementación es importante considerar que si el aprendizaje se realiza off-line, el tamaño final de los árboles puede reducirse mucho, ya que los nodos cuyas funciones sean Left o Right permiten eliminar subramas de cada árbol con el consiguiente ahorro en el número de puertas lógicas necesarias.

4. Plataforma hardware

La plataforma hardware utilizada está formada por un computador PC y una placa reconfigurable RIPP10 de ALTERA conectada en el bus ISA (ver Altera's Programmable Hardware Development Program Web Page).

La placa RIPP10 tiene un grupo ("array") de ocho FPGAs FLEX81188 interconectados con un total de 8064 celdas lógicas. Cada uno de estos chips está conectado con sus cuatro vecinos a través de buses de 32 bits. Un bus global de 36 bits provee acceso a todos los chips del array desde un FLEX8452 que es el que realiza la interfaz con el bus ISA.

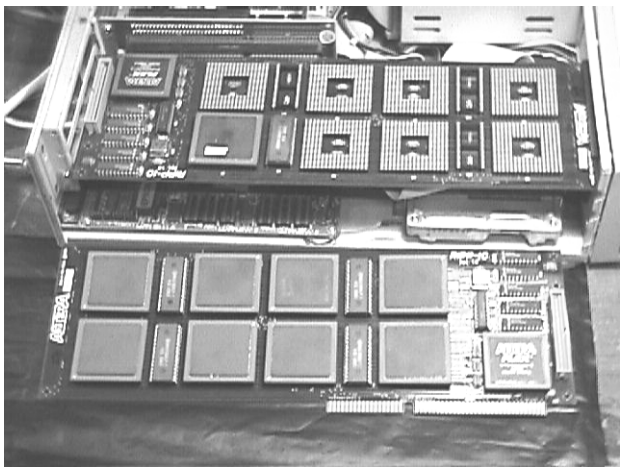


Fig. 3: Dos placas RIPP10

La placa posee además cuatro zócalos de RAM estática con un chip de 128KB cada uno. Cada chip de RAM es accesible desde un par de 81188. El bus global está directamente conectado a una interfaz externa mediante buffers. La RAM y la interfaz externa no fueron utilizadas en este trabajo.

Tanto los chips del array (81188) así como el chip de interfaz (8452) son programados a través del bus ISA. El proceso consiste en programar primero el chip de interfaz con un circuito que permita la programación de los chips del array. Luego se programan los chips del array uno por vez a través del bus global. Finalmente, si es necesario, se puede cargar un nuevo diseño en el chip de interfaz de acuerdo a la aplicación.

La RIPP10 presenta varias ventajas y desventajas desde el punto de vista de este proyecto. Es una plataforma flexible que permite evaluar fácilmente diferentes soluciones para un determinado problema. Tiene una gran cantidad de lógica disponible y buena intrconexión entre chips. Además la simplicidad del bus ISA permite prototipar rápidamente diferentes soluciones.

Por otra parte esta plataforma tiene algunas desventajas si se la compara con placas reconfigurables más modernas. No existe manera de reconfigurar en forma independiente cada uno de los chips del array, la única manera es borrar de una vez la configuración en todo el array y luego cargar los nuevos circuitos en cada uno de los ocho chips. Este es un proceso lento que limita severamente la posibilidad de multiplexar en el tiempo diferentes circuitos sobre la RIPP10. La velocidad y ancho de banda del bus ISA hoy en día ha sido ampliamente superada por una interfaz PCI.

5. Diseño del hardware desde una ALN previamente entrenada

Como ya fue visto anteriormente una red ALN ya entrenada es un árbol binario en donde cada nodo realiza una de las siguientes funciones lógicas: AND, OR, LEFT o RIGHT.

Es decir que el circuito necesario para implementar un árbol ALN ya entrenado es una función combinatoria. Para problemas reales dicha función combinatoria posee una gran cantidad de nodos. Es habitual en problemas resueltos con ALN la utilización de árboles de varias decenas de miles de nodos, aunque como se mencionó anteriormente siempre la cantidad de entradas al sistema es mucho menor.

En la aplicación elegida como ejemplo la cantidad de hojas es 2048, lo que implica un total de 2047 nodos organizados en 11 capas, y la cantidad de entradas del sistema es 60.

La red producto del entrenamiento se encuentra almacenada en archivos de texto con el formato utilizado por Atree que consiste en las funciones de cada árbol representadas en notación polaca inversa, utilizando los caracteres `&', `--', `L', `R' representando a las funciones de los nodos AND, OR, LEFT, RIGHT respectivamente. Las hojas se almacenan como un número representando el índice del bit conectado a cada una opcionalmente precedido por un `!' para indicar negación.

Se desarrolló una herramienta llamada AtreeToHardware (A2H) que convierte automáticamente los archivos de los árboles generados por Atree a un lenguaje de descripción hardware. A2H además de realizar el proceso de traducción realiza una simplificación de la red ya que las funciones lógicas que

genera no contienen nodos tipo LEFT o RIGHT, es decir que el árbol sólo queda compuesto de ANDs y ORs.

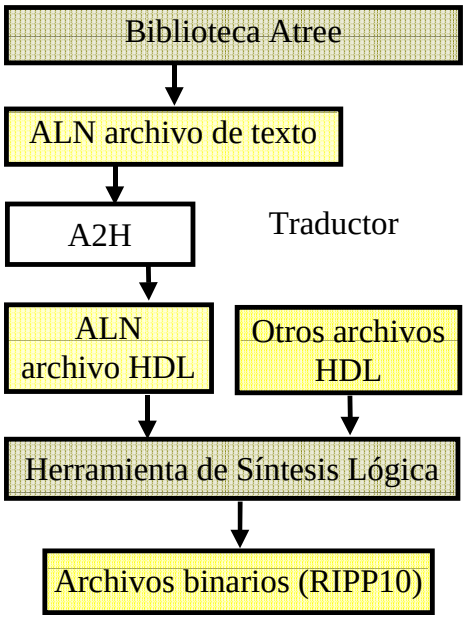


Fig. 4: Procedimiento de diseño

El lenguaje usado fue Altera Hardware Description Language (AHDL) pero A2H puede ser migrado fácilmente a otros lenguajes de descripción hardware. Los fuentes AHDL generados con A2H se compilan posteriormente con Max+Plus II. Esto genera una función lógica para cada árbol de la red. El Max+Plus II realiza una minimización de estas funciones lógicas, haciendo posible el mapeo de redes de gran tamaño en los chips de la RIPP10.

La herramienta desarrollada es genérica, y permite automatizar la traducción a hardware de este tipo de redes ya entrenadas. Esto facilita el diseño final de aplicaciones, reduciendo el tiempo de desarrollo de sistemas en hardware para aplicaciones específicas.

En esta fase del proyecto el diseño no está totalmente automatizado. La parte manual del proyecto es relativamente sencilla y se trabaja adaptando tamaños de funciones paramétricas ya realizadas, antes de realizar las compilaciones finales de los chips para cada chip del array.

Del lado del usuario se ha realizado una función C para brindar una interfaz con el hardware. Esta función realiza la evaluación de un grupo de árboles, siendo su interfaz idéntica a una de las incluídas en la biblioteca Atree (atree_eval).

6. Un ejemplo de aplicación en control : control en tiempo real del equilibrio de un péndulo invertido.

Sistema Físico.

Como ejemplo de aplicación experimental, se eligió implementar una red ALN ya entrenada para controlar un péndulo invertido resultado de un trabajo ya realizado en el IIE (Ferreira *et al.*, 1992, Ferreira y Fonseca de Oliveira, 1994).

Las razones de esta elección: porque se trata de un sistema complejo en el que se justifica usar un control no clásico, por ser un banco de pruebas típico con el cual es posible comparar performances y por el hecho de disponer de un sistema de estas características para el cual se ha desarrollado un simulador y varios controladores Chavarría *et al.*(1991), Ferreira *et al.* (1992). En particular, esta realización física del péndulo invertido posee una fricción compleja con fuertes rugosidades, y una dependencia periódica en la posición.

Este mismo problema ha sido encarado anteriormente con redes neuronales (p.ej: Tolat y Widrow (1988), Anderson (1989)).

Un esquema del péndulo invertido puede verse en la Fig. 5.

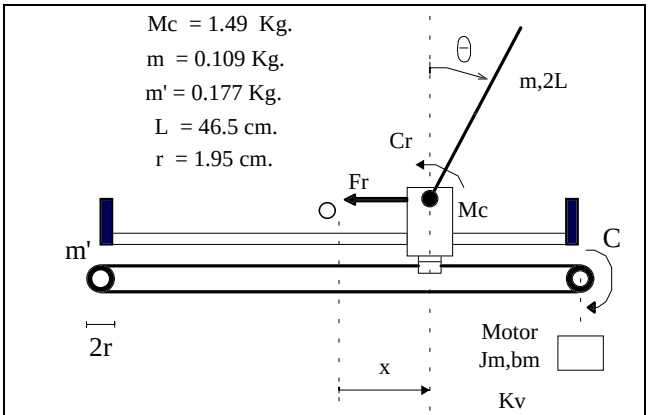


Fig. 5: Esquema mecánico

El carro que se desplaza sobre una guía horizontal es accionado por un motor de DC a través de una correa dentada. La posición lineal x y la posición angular θ son medidas por medio de un sensor resistivo y un transformador diferencial respectivamente. La acción de control se ejerce a través del par motor estableciendo la corriente de armadura. Las señales son sensadas y la

acción de control es impuesta desde un PC vía un conversor A/D-D/A Data Translation 2801A. El PC está equipado con la placa reconfigurable RIPP10 que computa la red ALN.

Modelo

Las expresiones que siguen describen el modelo matemático del sistema.

$$\ddot{x} = \frac{1}{\mathrm{J}}(\mathrm{K_v} r V + \mathrm{f}(\theta, \dot{\theta}) - \mathrm{g}(x, \dot{x}))$$

$$\ddot{\theta} = \frac{3}{4L}(g \sin\theta - b'\dot{\theta} - \ddot{x}\cos\theta)$$

donde

$$\mathrm{J} = \mathrm{J_m} + (M_c + m + \frac{m'}{2}) r^2 - \frac{3}{4} m r^2 \cos^2\theta$$

$$f(\theta, \dot{\theta}) = m r^2 (L\dot{\theta}^2 \sin\theta - \frac{3}{4}\cos\theta (g \sin\theta - b'\dot{\theta}))$$

$$g(\theta, \dot{\theta}) = r^2 F_r(x, \dot{x}) + b_m \dot{x}$$

$\mathbf{F_r}$ es la fuerza de fricción del sistema. Las constantes del modelo valen:

k_V = 0.0326 N m / V.

Jm = 0.000243N m s.

bm = 0.00056 N m s.

b` = 0.202

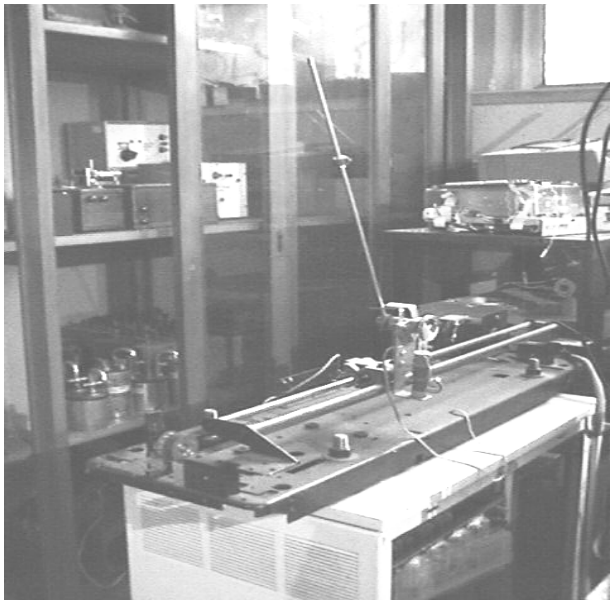
El esquema de control del péndulo es el de la Fig. 7.

Las entradas de control del sistema son *x, θ, dx/dt, dθ/dt*x, y la salida es la corriente del motor que mueve el carro del péndulo.

Las 4 variables de entrada son reales codificadas en 15 bits cada una, conformando un total de 60 bits de entrada.

La salida de control está compuesta de una variable real codificada en 6 bits.

Cada uno de estos 6 bits de salida es calculado con un factor de redundancia igual a tres (3 votos, decisión por mayoría), siendo necesarios entonces 18 árboles binarios.



Cada uno de estos árboles posee 2048 hojas, siendo sus conexiones con los bits de entrada y sus negados al azar.

Fig. 6: El péndulo invertido empleado.

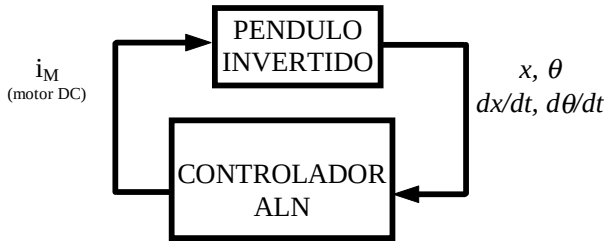


Fig. 7: Esquema de control del péndulo

El código utilizado fue termómetro dando 16 intervalos de cuantificación en las entradas y 7 en la salida.

Esta red fue entrenada previamente utilizando la library Atree 2.7 for Windows, tomando como patrones las acciones de control de un controlador por realimentación de estados diseñado previamente para el sistema dado, con una zona lineal en una región próxima al origen y saturada fuera de ella.

Los archivos producto de este entrenamiento fueron procesados con A2H y posteriormente compilados con Max+Plus II

Cada chip utilizado en el array aloja 3 árboles que corresponden a cada uno de los votos de una misma salida. De esta forma se utilizaron 6 chips del array para realizar los 18 árboles.

Para completar el diseño se agregan algunos módulos más en los chips del array:

- Se construyó una función lógica que toma la decisión por mayoría entre los 3 votos para generar un bit de la salida.

- Shift registers para memorizar las 60 entradas de la red

Finalmente para la aplicación elegida, en la cual se programan 18 árboles con 2048 hojas cada uno, la utilización de los chips es la siguiente:

CHIP	L CELLS	% utilizado
u1	541	53 %
u2	576	57 %
u3	539	53 %
u4	542	53 %
u5	479	47 %
u6	509	50 %
u7	no usado	0 %
u8	no usado	0 %

Como puede apreciarse de la tabla, los chips del array (EPF81188GC232) tienen aproximadamente un 45% de las celdas libres.

Los 60 bits de entrada se escriben en cuatro transferencias consecutivas a un puerto en el bus ISA, posteriormente se leen los 6 bits de salida generados por la red en el mismo puerto. En ambos casos los datos son transferidos a través del bus global y del chip de interfaz en la RIPP10.

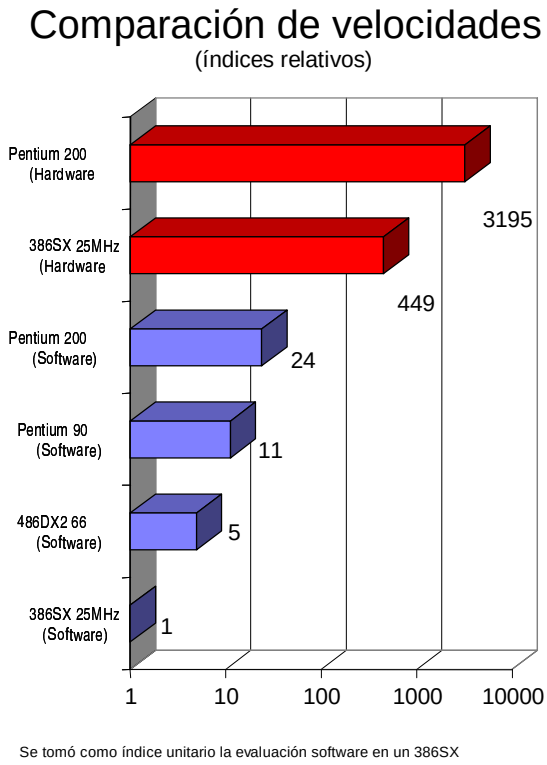
6.1 Resultados.

Se desarrolló una función C (atree_ripp_eval()) que realiza la evaluación de la red utilizando el hardware con una interfaz idéntica a la función original de la biblioteca Atree (atree_fast_eval()).

Para realizar la comparación de velocidades un mismo programa que evalúa 4096 patrones se compiló con cada una de las dos funciones de evaluación.

Los tiempos de ejecución listados en la tabla reflejan el punto de vista de una aplicación final. Incluyen todo el loop del programa que evalúa los 4096 patrones. No incluyen los tiempos de inicialización, tales como la configuración de los chips en la versión hardware o la creación e inicialización de las estructuras de datos en la versión software.

Comparación de velocidades



Se tomó como índice unitario la evaluación software en un 386SX

Computador	Evaluación Software atree_fast_eval() Tiempo[ms]	Evaluación Hardware Atree_ripp_eval() Tiempo[ms]
Pentium 200 MHz	3626	27
Pentium 90 MHz	7692	
486DX2 66 MHz	17473	
386SX 25MHz	86264	192

atree_fast_eval()- función que evalúa un árbol ya definido en software y preprocesado (RAM del PC)

atree_ripp_eval()- función que evalúa un árbol ya definido en hardware (RIPP10)

Como puede observarse en la tabla el tiempo de la evaluación hardware se reduce significativamente entre un 386sx y un Pentium a pesar que la frecuencia de reloj en la RIPP10 es la misma en ambos casos (reloj del ISA). Esto indica que la parte principal del tiempo de evaluación se consume en la función C que realiza la interfaz software, que se ve acelerada en un PC más rápido.

Puede calcularse una cota inferior para el tiempo de ejecución de la evaluación en hardware dado por las limitaciones de entrada-salida en la interfaz ISA. Para la evaluación de cada patrón son necesarias un total de cinco ciclos de bus ISA de 16 bits (4 ciclos WR y 1 ciclo RD). Cada uno de ellos dura tres períodos de reloj dando un total de 1800ns por patrón. Esto daría una cota inferior de aproximadamente 7.4ms para la evaluación de los 4096 patrones. Esto indica que la limitación de este sistema con un Pentium 200 está en el software que no es capaz de transferir los datos a velocidad suficiente por el bus ISA.

Simulando el circuito con la herramienta "Timing Analysis" de Max+Plus se deduce un tiempo de ciclo de mínimo de 300ns para la evaluación de un patrón. Esto muestra que el troughput estaría acotado por el bus ISA en una máquina lo suficientemente rápida.

Perfomance del controlador

En la Fig. 8 se observa la evolución de las variables de estado (x, θ, dx/dt, $d\theta/dt$x) y la salida vs. el tiempo durante los ensayos.

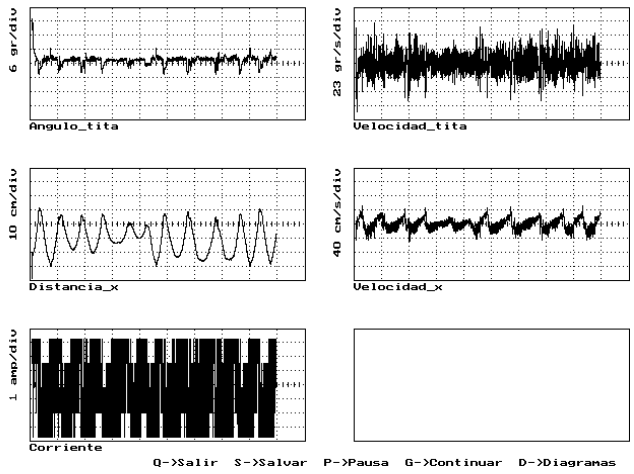


Fig. 8: Evolución del estado y la salida.

La Fig. 9 muestra el mismo experimento, a través de las proyecciones sobre los planos son dx/dt *vs.* x, y $d\theta/dt$x vs. θ, del espacio de fase.

Se constató una gran congruencia en la amplitud y la frecuencia de las oscilaciones entre las previstas durante el entrenamiento con el simulador (Ferreira y Fonseca de Oliveira, 1994), y las observadas en el experimento.

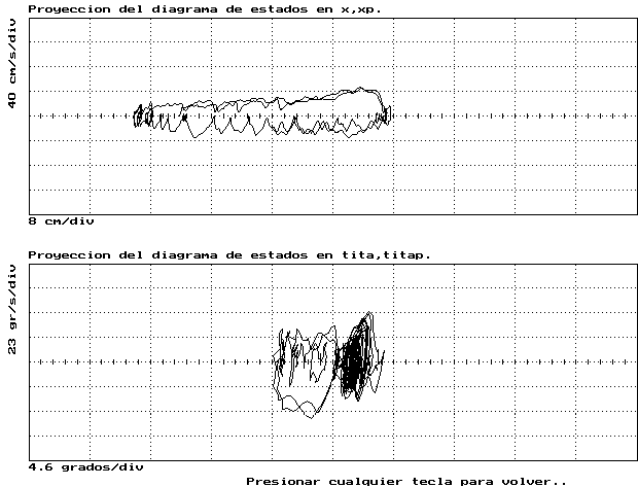


Fig. 9: Trayectoria del estado observada experimentalmente.

7. Conclusiones

Se implementó en hardware una red lógica adaptiva sobre una placa lógica reconfigurable para PC. La red fue ensayada con una aplicación de control en tiempo real.

Desde el punto de vista de un usuario final, los algoritmos implementados en hardware sustituyen exactamente a funciones en C. De este modo para un usuario habituado a trabajar con la biblioteca Atree la utilización de la versión en hardware de algunas funciones es casi transparente. Este era uno de los principales objetivos de este trabajo.

Al analizar los tiempos de ejecución se concluye que la utilización de la versión implementada en hardware es altamente conveniente, obteniéndose órdenes de magnitud de mejora.

La placa RIPP10 se mostró adecuada para realizar redes ALN de gran tamaño, las cuales son aplicables a control en tiempo real.

La plataforma obtenida en este trabajo permitió desarrollar en tiempos muy breves un sistema de control en tiempo real.

El desempeño del controlador fue adecuado y de acuerdo a lo previsto en las simulaciones realizadas durante el entrenamiento de la red.

Agradecimientos

Este trabajo fue realizado con financiación del Consejo Nacional de Investigaciones Científicas y Técnicas del Uruguay, proyecto BID-CONICYT Nº 355 "Procesador neuronal basado en lógica programable".

La placa RIPP10 fue donada por el "Programmable Hardware Development Program" de Altera (un agradecimiento especial a Stephen Smith).

Queremos agradecer al Ing. Gabriel Eirea su colaboración en la fase experimental.

Referencias

Altera's Programmable Hardware Development Program. Página web. http://www.altera.com/html/programs/phd.html

Anderson, C.W.. "Learning to Control an Inverted Pendulum Using Neural Networks", *IEEE Control Systems Magazine*, Vol. 9, no. 3: 31-37. (1989)

Armstrong W. W. and Godbout G., "Properties of Binary Trees of Flexible Elements Useful in Pattern Recognition", *IEEE 1975 International Conf. on Cybernetics and Society*, 447-449, San Francisco, (1975).

Armstrong W. and Gecsei J., "Adaptation Algorithms for Binary Tree Networks", IEEE Trans. on Systems, Man and Cybernetics, Vol. 9, 276-285, (1979).

Armstrong W. and Gecsei J., "Architecture of a Tree-based Image Processor", *12th Asilomar Conf. on Circuits, Systems and Computers*, 345-349, Pacific Grove, (1978).

Armstrong W. "William Armstrong Home Page". http://www.cs.ualberta.ca/~arms

Armstrong W. W., Dwelly A., Liang J., Lin D. and Scott Reynolds, "Learning and Generalization in Adaptive Logic Networks", *Artificial Neural Networks, Proc. of the 1991 Int. Conf. on Artificial Neural Networks (ICANN-91)*, T. Kohonen, K. Makisara, O. Simula, J. Kangas, eds, pp.1173-1176, North Holland,.Espoo, Finland, (1991).

Autores varios, *Artificial Neural Networks: Paradigms, Aplications, and Hardware Implementations*, editado por Edgar Sánchez-Sinencio y Clifford Lau, IEEE Press, New York, (1992).

Bochmann G. V. and Armstrong W., "Properties of Boolean Functions with a Tree Decomposition", BIT, Vol. 13, 1-13, (1974).

Chavarrría, P., Belzarena, P., Bonino, E.. "Control del Péndulo Invertido", *INGELECTRA*, VII Seminario Estudiantil, Universidad de Tarapaca, Arica, Chile. (1991)

Cloutier J., Cosatto E., Pigeon S., Boyer F. R. and Simard P. Y., "VIP: An FPGA-based Processor for Image Processing and Neural Networks", MicroNeuro '96, Lausanne, Switzerland, (1996).

Ferreira E., Planchón E. and Trochón C.,"Neural Network Controller for the Inverted Pendulum".*XIII Simposio Nacional de Control Automático*, AADECA'92, Buenos Aires, (1992).

Ferreira E., and Fonseca de Oliveira A.,"Una Aplicación de Redes Binarias Adaptivas (ALN) en Control", *XIV Simposio Nacional de Control Automático*, AADECA'94, Buenos Aires, 1994. (1992).

Guccione S., *Programming Fine-grained Reconfigurable Architectures*, Ph. D. Thesis, University of Texas at Austin, (1995).

Haykin S., *Neural Networks*, Macmillan, Englewood Cliffs, NJ, (1994).

Hecht-Nielsen R., *Neurocomputing*, Addison-Wesley, Reading, Massachusetts, (1991).

Heemskerk J. N. H., "Overview of Neural Hardware", Chapter 3, Ph. D. Thesis *Neurocomputers for Brain-Style Processing. Design, Implementation and Application*, Leiden University, The Netherlands, (1995). (ftp://ftp.mrc-apu.cam.ac.uk/pub/nn).

Hegt J.A., "Hardware Implementations of Neural Networks", *Measurement and Artificial Neural Networks*, Proceedings 'Themadag van de Werkgemeenschap Meten', Utrecht, (1993),(ftp://ftp.tue.nl/pub/neural/hardware_general.ps.gz)

Lindsey C. S. and Lindblad T., "Review of Hardware Neural Networks: A User's Perspective", Third Workshop on Neural Networks: From Biology to High Energy Physics, Isola d'Elba, Italy, (1994).

Oliver J.P., Fonseca de Oliveira A., Pérez Acle J., de la Vega R.J., Canetti R., "Implementation of Adaptive Logic Networks on an FPGA board", en *Configurable Computing: Technology and Applications*, John Schewel, Editor, Proceedings of SPIE Vol. 3526, 264-273 (1998).

Paolo I.. and Kuhn G., "Digital Systems for Neural Networks", *Digital Signal Processing Technology*, P. Papamichalis and R. Kerwin, editors, Vol. CR57 of *Critical Reviews Series*, 314-345, SPIE Optical Engineering, (1995).

Supynuk A. G. and Armstrong W. W., "Adaptive Logic Networks and Robot Control", Proc. Vision Interface Conference '92, also called AI/VI/GI '92, 181 - 186, Vancouver B. C., (1992).

Tolat, V.V. & Widrow, B.. "An Adaptive room Balancer with Visual Inputs", *Proc. IEEE Int. Conf. on Neural Networks*, II641-II647. (1988)