



**Universidad de Las Palmas de Gran Canaria**

**METODOLOGÍAS DE DISEÑO DE REDES NEURONALES  
SOBRE DISPOSITIVOS DIGITALES PROGRAMABLES PARA  
PROCESADO DE SEÑALES EN TIEMPO REAL**

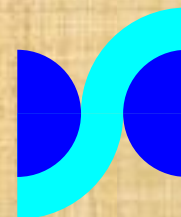
*Santiago T. Pérez Suárez*

Directores:

*Dr. Carlos M. Travieso González*

*Dr. Jesús B. Alonso Hernández*

**Departamento de Señales y Comunicaciones**



**IdETIC® Instituto para el Desarrollo Tecnológico y la Innovación en Comunicaciones**

# ÍNDICE

**1. INTRODUCCIÓN**

**2. IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL**

**3. METODOLOGÍA EXPERIMENTAL**

**4. EXPERIMENTOS Y RESULTADOS**

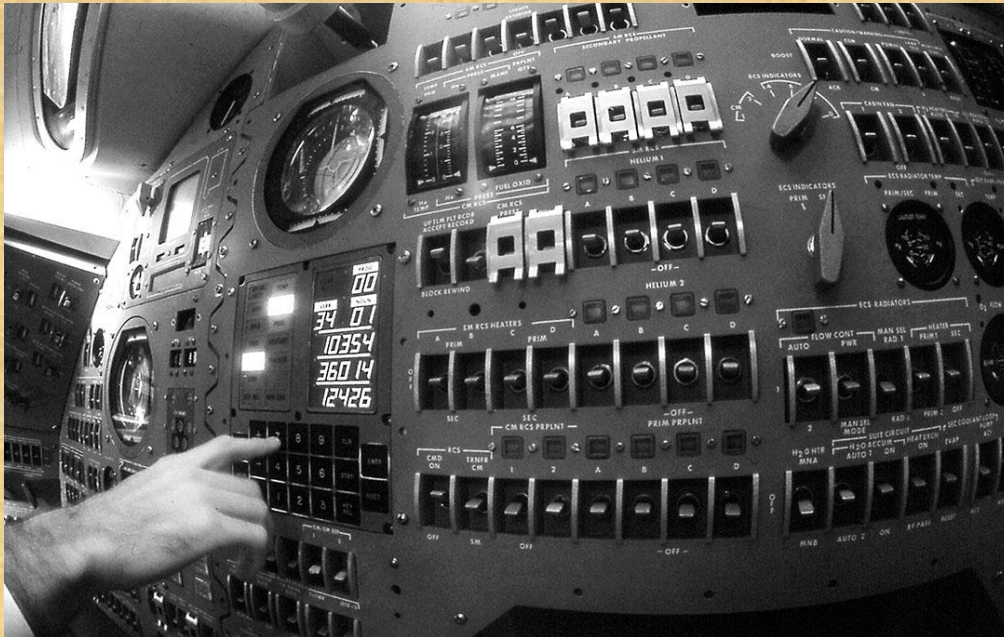
**5. CONCLUSIONES Y LÍNEAS FUTURAS**

**6. DEMOSTRACIÓN**

# INTRODUCCIÓN

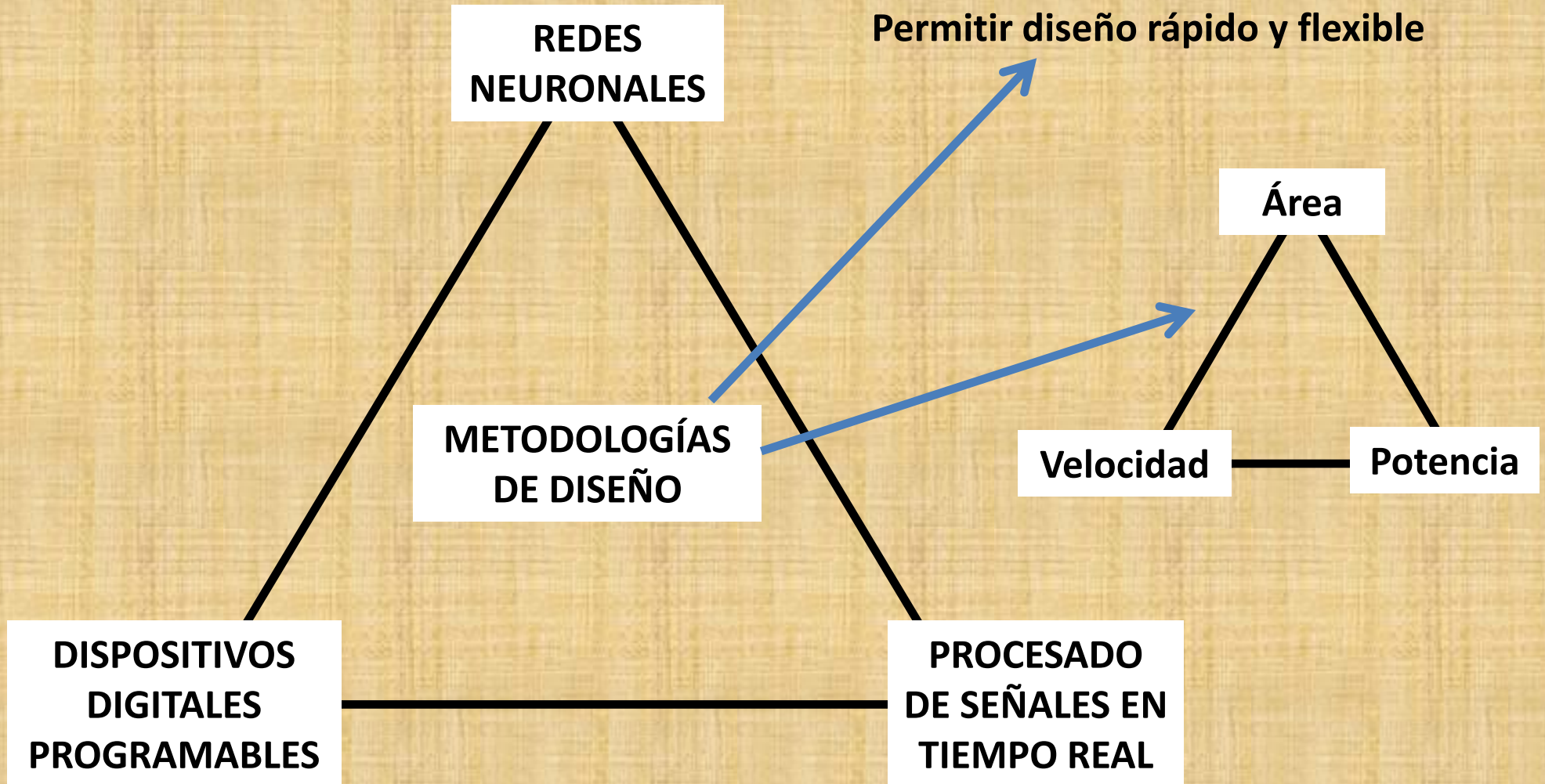
## Motivación y justificación

### *Apollo Guidance Computer*



# INTRODUCCIÓN

## Motivación y justificación





# INTRODUCCIÓN

## Motivación y justificación

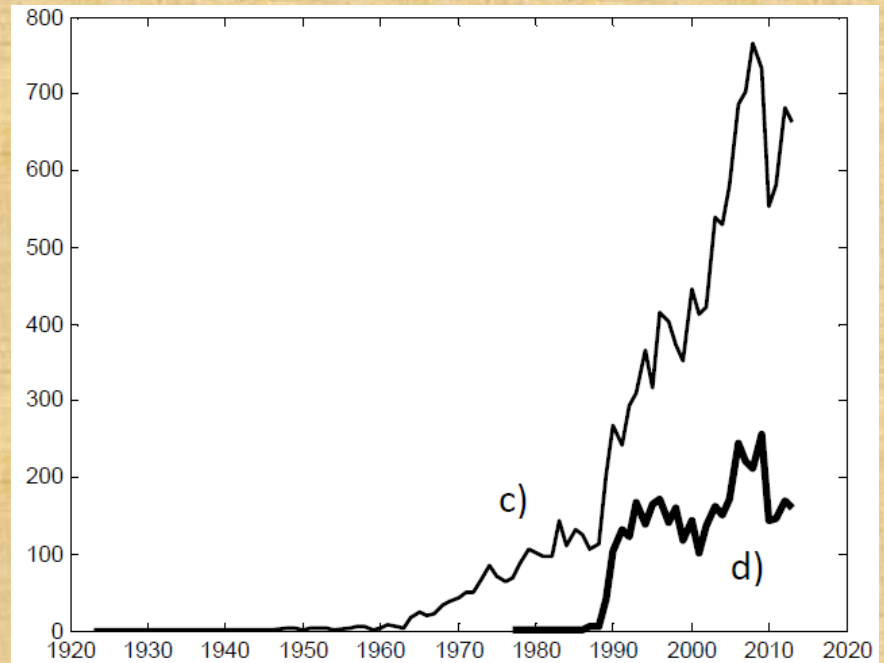
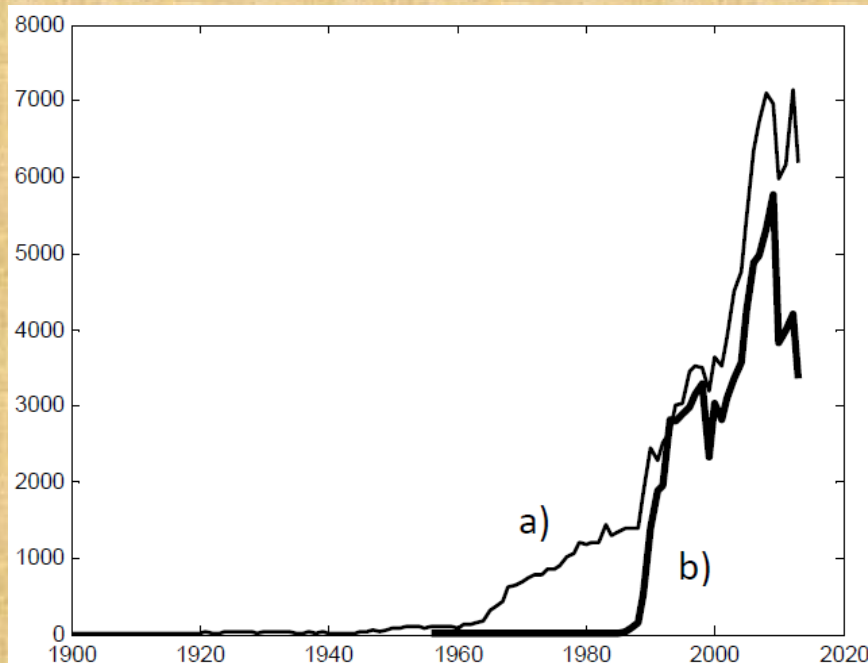
Apariciones por año en el *Web of Science* de:

a) “filter”,

b) “neural network” (NN),

c) “filter” and (“design” or “implementation” or “hardware”),

d) “neural network” and (“design” or “implementation” or “hardware”).



# INTRODUCCIÓN

## Antecedentes y estado del arte

- ▶ Los diseños pueden no incluir el entrenamiento: **offline**.
- ▶ Cuando los circuitos incluyen las estructuras de entrenamiento: **online**.
- ▶ Normalmente diseñados en **FPGA**.
- ▶ Normalmente en aritmética de **punto fijo**.
- ▶ Normalmente la **función** de transferencia usada es de **tipo sigmoidea**.
- ▶ Muchas aportaciones **se centran en la función de transferencia**.

# INTRODUCCIÓN

## Antecedentes y estado del arte

[Tommiska, 2003] Tommiska, M. T.; “Efficient digital implementation of the sigmoid function for reprogrammable logic”, *IEE Proceedings-Computers and Digital Techniques*, vol. 150, nº 6, pp. 403-411, 2003.

- ➔ Aproximación **por tramos**: lineales y polinomios de 2º orden.
- ➔ Los diseños se describieron con **VHDL**.
- ➔ Se estima el área y la velocidad, pero **no la potencia**.
- ➔ La **funcionalidad** se analizó con el **error**.
- ➔ Se estudió el efecto del **número de bits** (algunos valores).
- ➔ El autor propuso una fórmula como **factor de calidad**:

$$Q = \frac{f_{\max}}{\text{Área} \cdot \text{Error}_{\text{máximo}} \cdot \text{Error}_{\text{medio}}}$$

# INTRODUCCIÓN

## Antecedentes y estado del arte

[Gomperts et al., 2011] Gomperts, A.; Ukil, A.; Zurfluh, F.; “**Development and Implementation of Parameterized FPGA-Based General Purpose Neural Networks for Online Applications**”, *IEEE Transactions on Industrial Informatics*, vol. 7, nº 1, pp. 78-89, 2011.

- Herramienta para **sintetizar una NN** en VHDL con número de capas variable.
- Las funciones de transferencia se generaron con **LUT** (*Lookup-Table*).
- Los autores estudiaron el efecto del **número de bits** (algunos valores).
- Se da estimación de área y velocidad, pero **no de potencia**.
- El sistema permite entrenamiento **online**.



# INTRODUCCIÓN

## Antecedentes y estado del arte

[Reis et al., 2011] Reis, L.; Aguiar, L.; Baptista, D.; Morgado, F.; “**ANGE – Automatic Neural Generator**”, *Lecture Notes in Computer Science*, vol. 6792, nº 1, pp. 446-453, 2011.

- ➡ Herramienta de **generación automática de NN** con una capa oculta.
- ➡ Las **funciones de transferencia**: identidad y la tangente hiperbólica (LUT).
- ➡ Genera el diagrama de bloques (**paralelizado**) para *System Generator* de Xilinx.
- ➡ Analizan la funcionalidad y el área; **no la velocidad ni la potencia**.
- ➡ Todas las **entradas** tienen el mismo formato binario.
- ➡ En cada capa los **coeficientes** usan el mismo formato binario.

# INTRODUCCIÓN

## Antecedentes y estado del arte

- La aritmética de **punto flotante** es poco usada.
- Debe buscarse el **número idóneo de bits**.
- A veces el formato se reduce a un **número entero de octetos**.
- La mayor parte usa **FPGA**.
- Todavía se presentan diseños usando un **lenguaje de descripción hardware**, o incluso **edición de esquemáticos**.
- Las herramientas avanzadas que operan sobre **Simulink** se usan con moderación.
- Apenas existen aportaciones que usan las **utilidades propias de Matlab**.
- Casi todos los autores dan estimación de área y de velocidad; pero pocos dan estimaciones de la **potencia**.
- En algunos diseños se incluyen las estructuras para el entrenamiento (**online**).

# INTRODUCCIÓN

## Objetivos de la tesis

Se pretende demostrar la **hipótesis**:

*“Es posible encontrar **métodos de diseño** sobre **dispositivos digitales programables** para implementar **redes neuronales** que operen en **tiempo real** en procesamiento digital de la señal.*

*Los métodos deben ser **rápidos y flexibles**; y permitir evaluar el efecto del **número de bits** sobre la arquitectura.*

*Además, deben posibilitar la comprobación de la total **funcionalidad** del sistema y las **prestaciones físicas** de área, potencia y velocidad”.*

# INTRODUCCIÓN

## Metodología

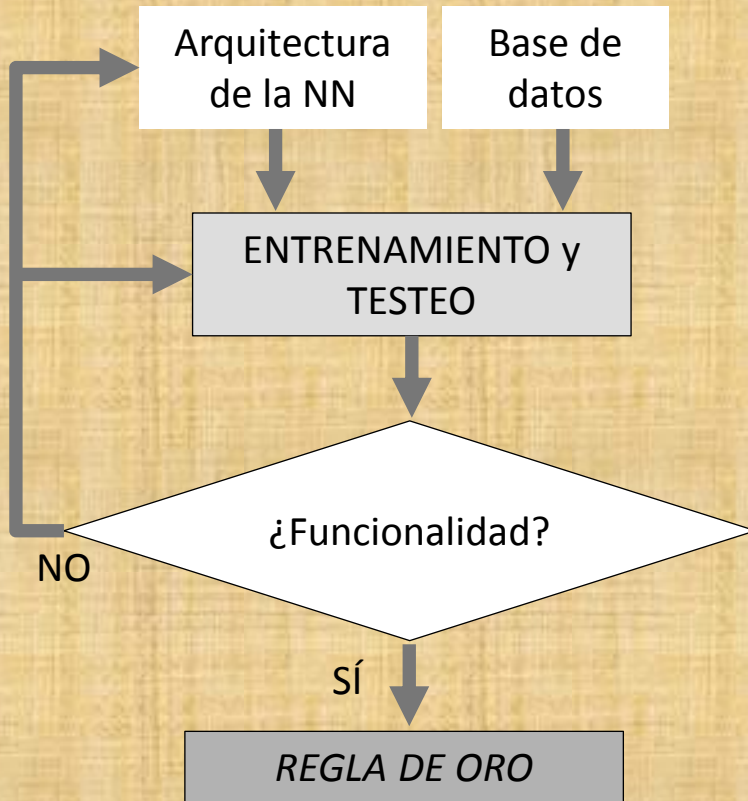
- ➔ Esta tesis se centra en la NN tipo perceptrón multicapa con conexión hacia adelante (***Feedforward Multilayer Perceptron***). Totalmente **paralelizada**.
- ➔ Este tipo de despliegue puede ser tolerado por las **FPGA**.
- ➔ La **parte crítica** lo constituye la función de transferencia.
- ➔ Almacenar muestras de la función en una memoria (**LUT**): menor retardo.
- ➔ En ocasiones se recurrirá a funciones de transferencia **lineales por tramos**.
- ➔ Se optó por un método sobre **Simulink** de Matlab.
  - Descripción rápida y flexible.
  - Simulación del sistema: comprobar su total funcionalidad.
  - Prestaciones físicas: área, velocidad y potencia.
- ➔ Se diseñarán NN que funcionarán sobre **diferentes escenarios**.



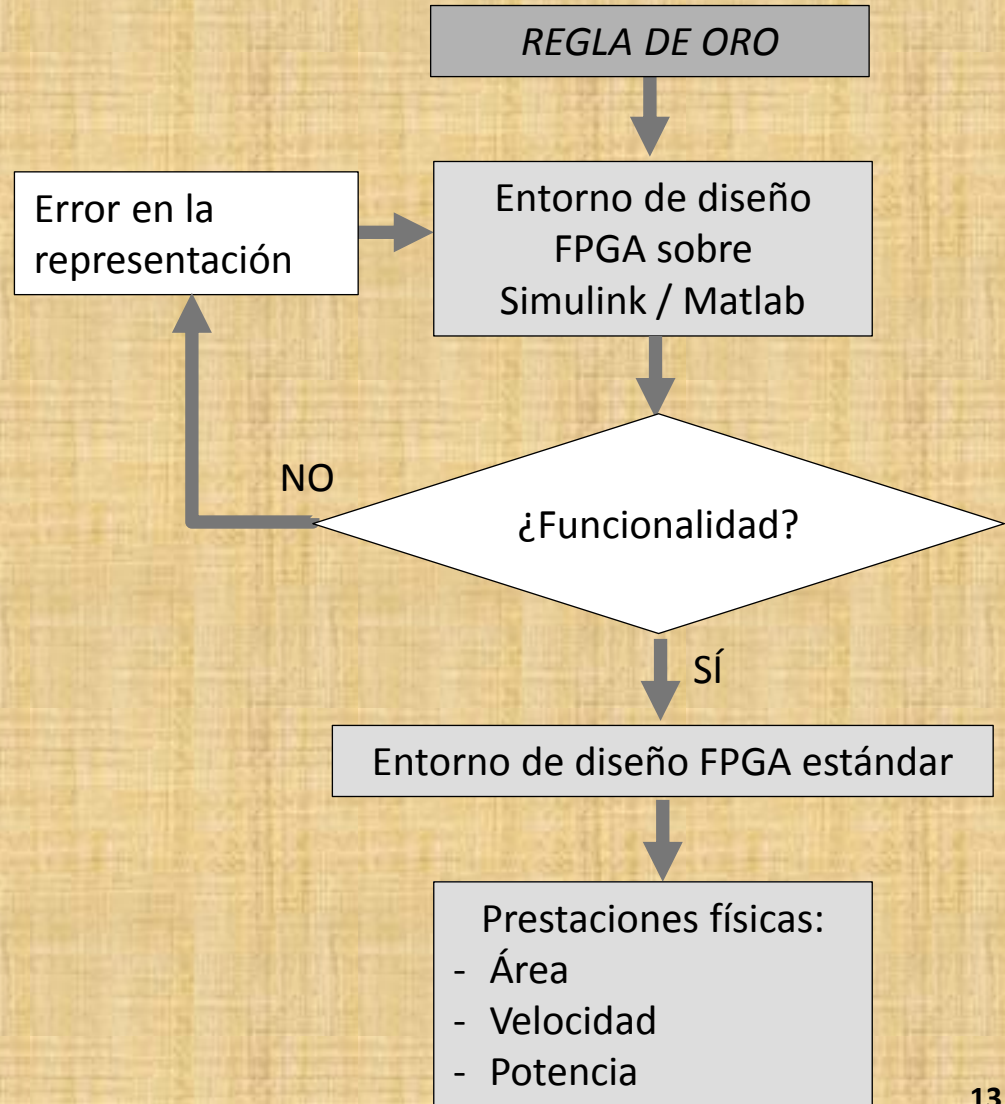
# INTRODUCCIÓN

## Metodología

### MODELADO EN PUNTO FLOTANTE



### MODELADO EN PUNTO FIJO

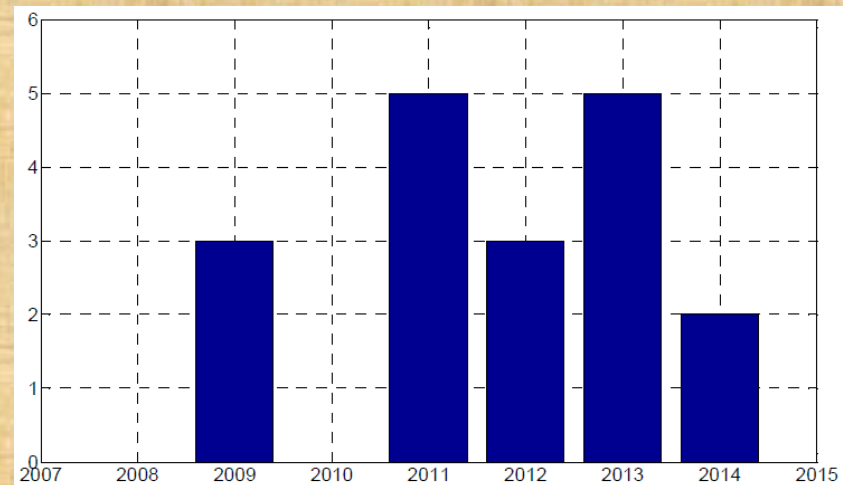


# INTRODUCCIÓN

## Contribuciones y resultados

Publicaciones	Cantidad	Referencias
<i>Con JCR</i>	2	[Pérez et al., 2013] [Vásquez et al., 2013]
<i>Sin JCR</i>	2	[Pérez et al., 2009a] [Travieso et al., 2013a]
<i>Capítulos de libros</i>	2	[Pérez et al., 2011b] [del Pozo et al., 2012]
<i>Congresos internacionales</i>	9	[Pérez et al., 2009b] [Pérez et al., 2011c] [Pérez et al., 2011d] [Ticay et al., 2011] [Vásquez et al., 2012] [Vásquez et al., 2012] [Travieso et al., 2013b] [Pérez et al., 2014a] [Pérez et al., 2014b]
<i>Congresos nacionales</i>	3	[Pérez et al., 2009c] [Pérez et al., 2011a] [Alonso et al., 2013]

### Aportaciones por año:



- Concesión de un sexenio de investigación para los años evaluados: 2004, 2005, 2006, 2011, 2012 y 2013. Concedido por la Comisión Nacional Evaluadora de la Actividad Investigadora.

# ÍNDICE

1. INTRODUCCIÓN

**2. IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL**

3. METODOLOGÍA EXPERIMENTAL

4. EXPERIMENTOS Y RESULTADOS

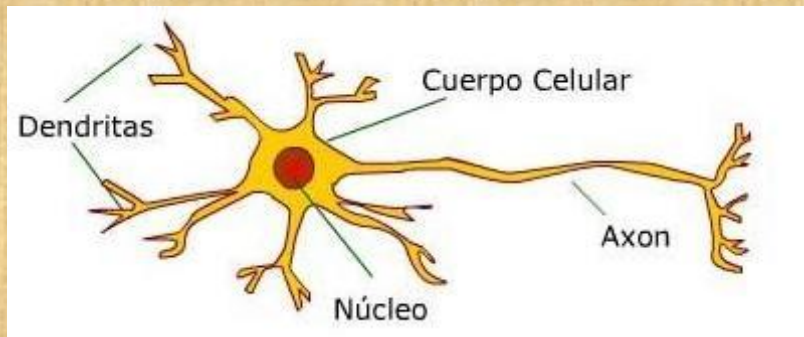
5. CONCLUSIONES Y LÍNEAS FUTURAS

6. DEMOSTRACIÓN

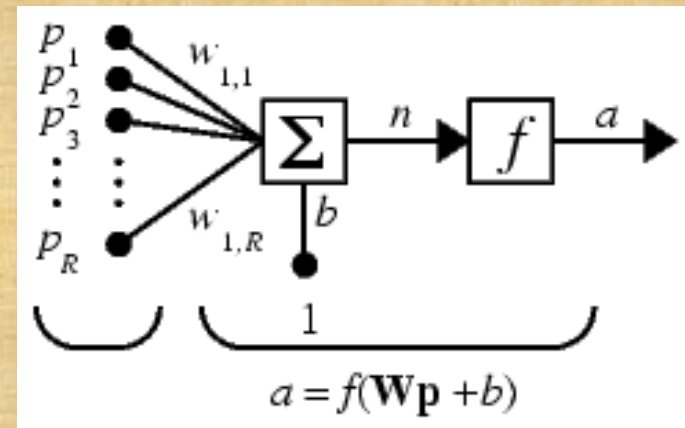
# IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

## Las redes neuronales

Estructura de una neurona nerviosa.



Estructura de una neurona artificial.

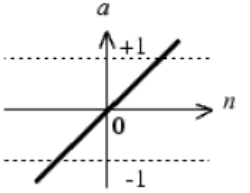
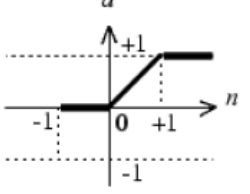
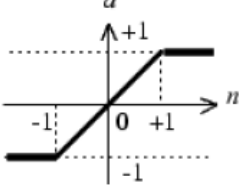
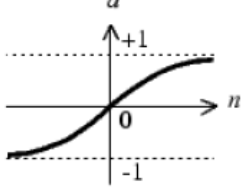




# IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

## Las redes neuronales

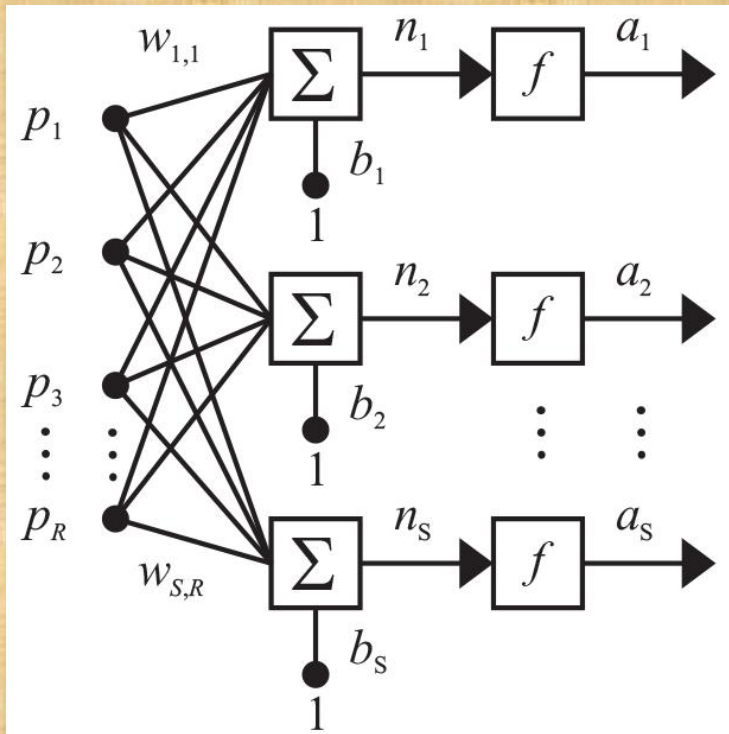
### Funciones de transferencia típicas.

<p><i>Linear transfer function</i></p>	 <p><math>a = \text{purelin}(n)</math></p>	$a = n$
<p><i>Saturating linear transfer function</i></p>	 <p><math>a = \text{satlin}(n)</math></p>	$a = \begin{cases} 0 & n \leq 0 \\ n & 0 \leq n \leq 1 \\ 1 & n \geq 1 \end{cases}$
<p><i>Symmetric saturating linear transfer function</i></p>	 <p><math>a = \text{satlins}(n)</math></p>	$a = \begin{cases} -1 & n \leq -1 \\ n & -1 \leq n \leq 1 \\ 1 & n \geq 1 \end{cases}$
<p><i>Hyperbolic tangent sigmoid transfer function</i></p>	 <p><math>a = \text{tansig}(n)</math></p>	$a = \frac{1 - e^{-2n}}{1 + e^{-2n}}$

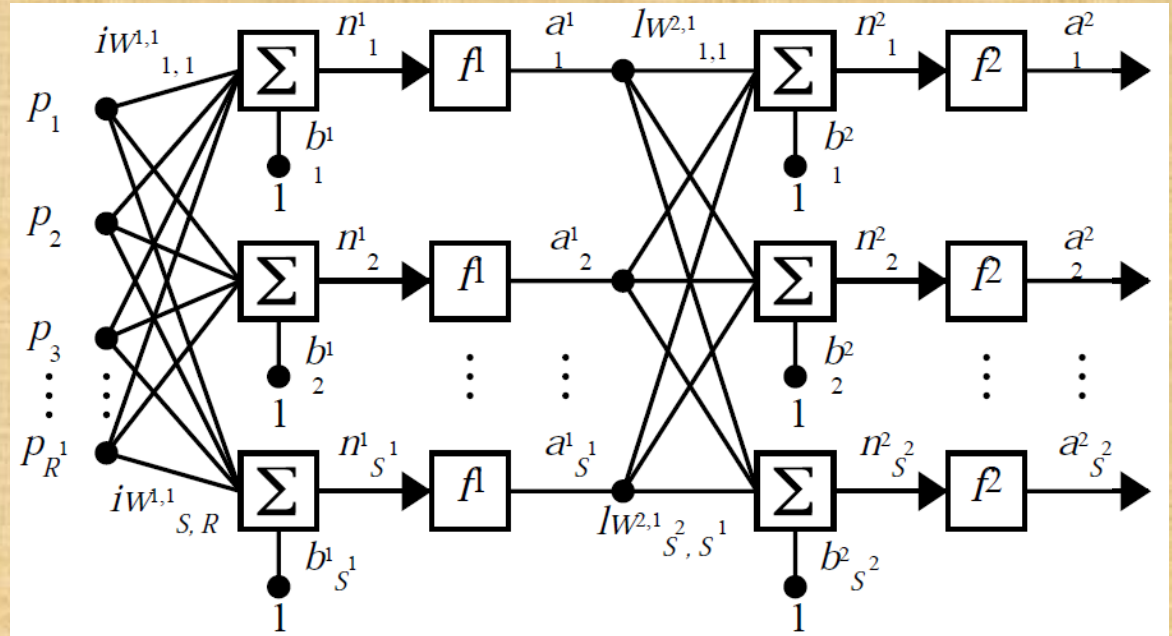
# IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

## Las redes neuronales

Red neuronal con R entradas conectadas a una capa de S neuronas.












Red neuronal con  $R^1$  entradas, una capa intermedia de  $S^1$  neuronas y una capa de salida de  $S^2$  neuronas.



# IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

## La variabilidad en el diseño de las redes neuronales

- 
- Decisiones** del diseñador:
- en función de su experiencia
  - o “prueba y error”.

- 
- Supóngase el tipo **perceptrón multicapa con conexión hacia adelante**:
- 
- El número de **capas**.
- 
- El número de **neuronas**.
- 
- El número de **entradas**.
- 
- El número de **salidas**.
- 
- El tipo de **función de transferencia** para cada capa.
- 
- Además, debe elegirse el **algoritmo de entrenamiento** y sus **parámetros**.
- 
- Base de datos: dividirla para el **entrenamiento** y el **testeo**.

# IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

## Los tipos de aritmética binaria

### Punto flotante:



- Muchos recursos hardware.
- Necesita muchos ciclos de reloj.
- Gran cantidad de potencia.
  
- IEEE define un estándar con representaciones de 4, 8 y 16 octetos.

### Punto fijo:

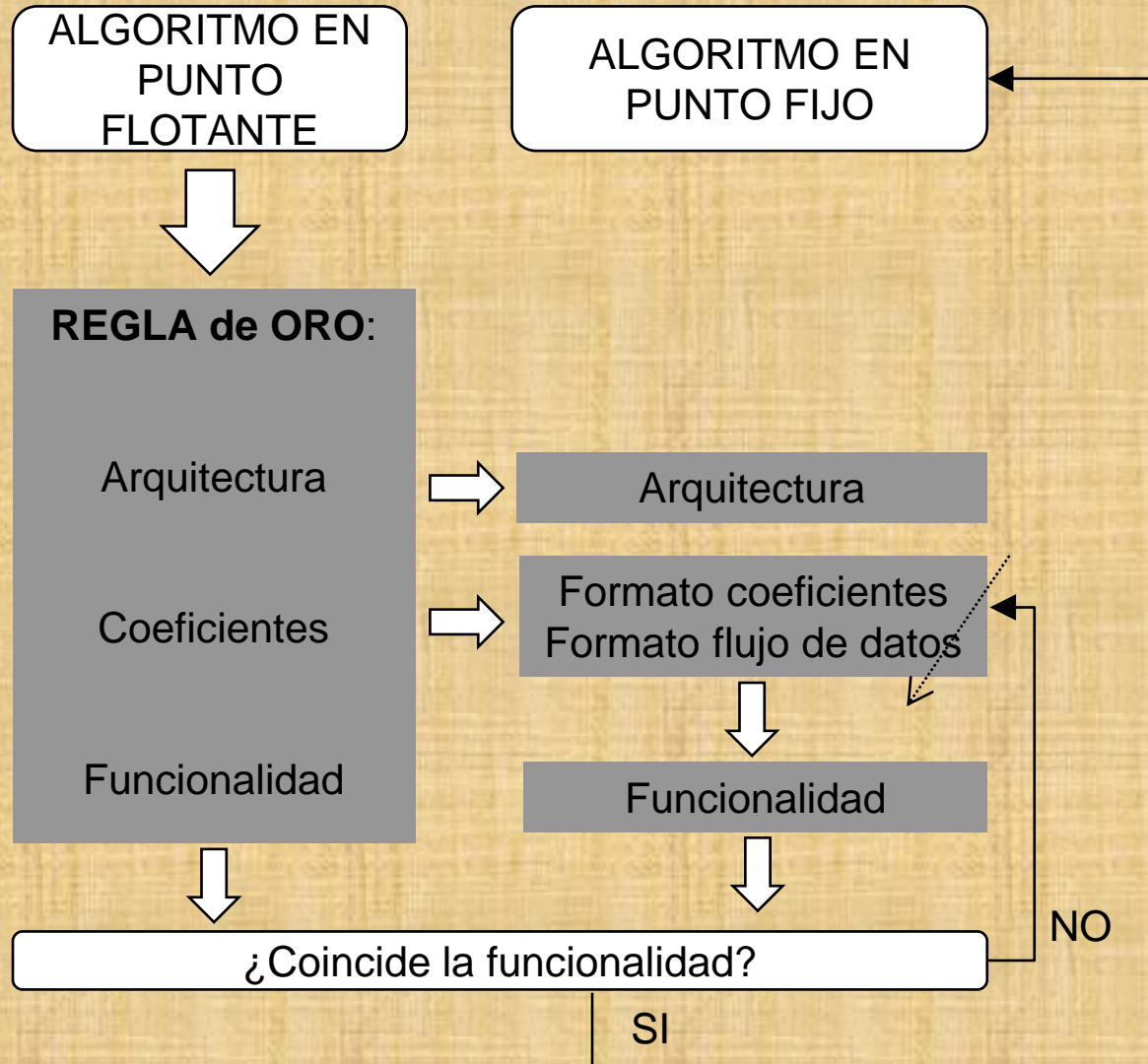


- Menos recursos hardware.
- A partir de un ciclo de reloj.
- Menos potencia.
  
- Vigilar el rango y la precisión.
- Es posible usar cuando se conoce a priori el rango en amplitud de las señales.
- Complemento a dos.



# IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

## Paso del modelo de punto flotante a punto fijo: la regla de oro



# IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

## Las tecnologías disponibles

**ASIC**, *Application Specific Integrated Circuit*.

**FPGA**, *Field Programmable Gate Array*.

**FPAA**, *Field Programmable Analog Array*.

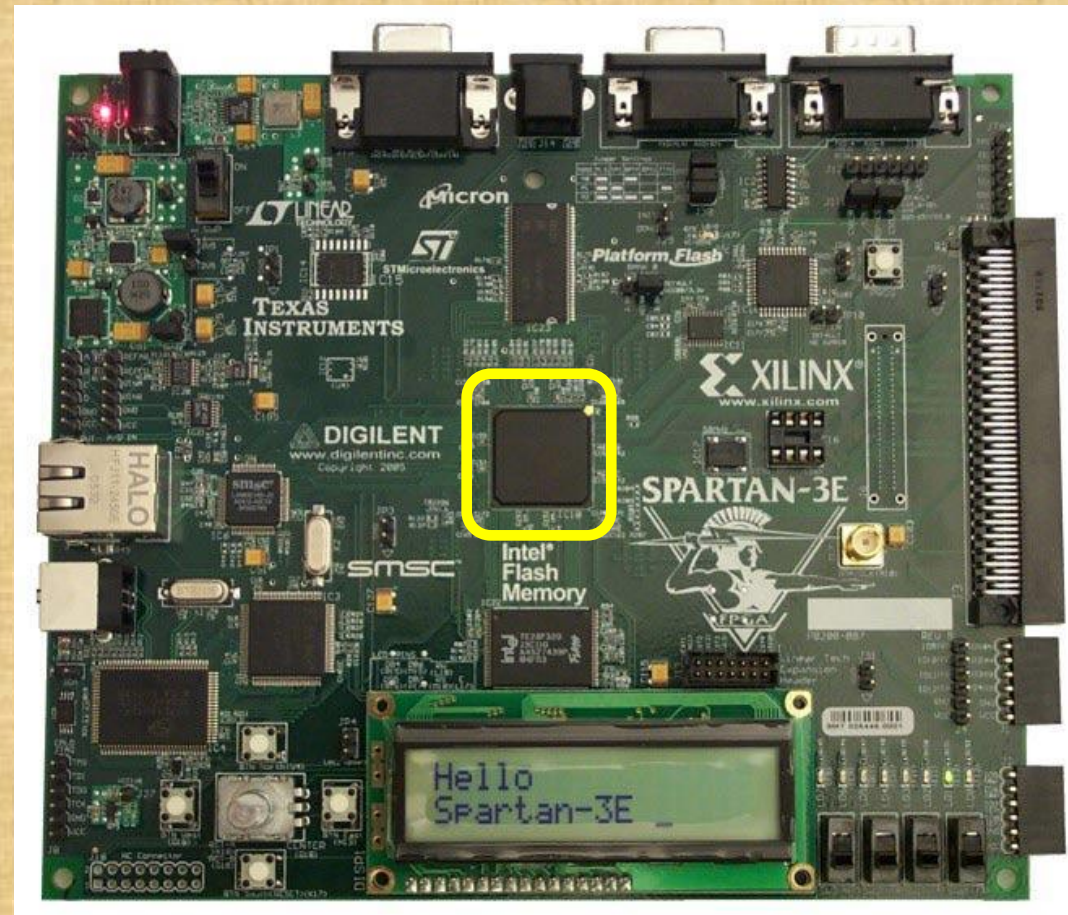
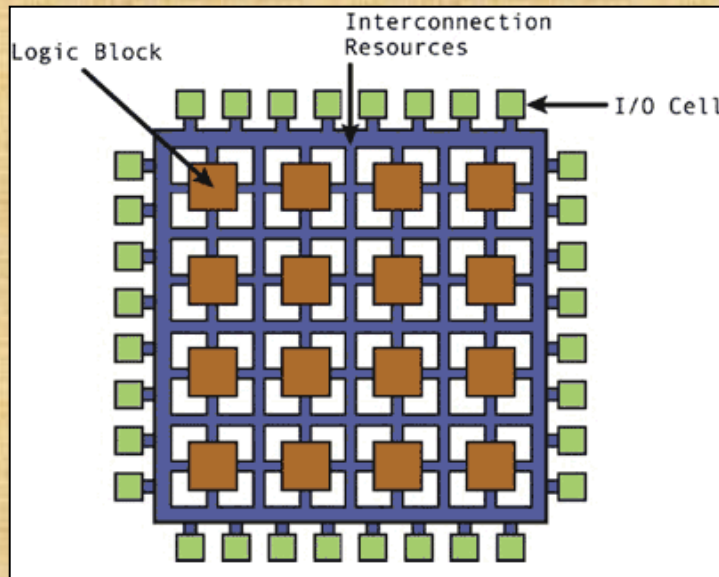
**DSP**, *Digital Signal Processor*.

# IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

## Las tecnologías disponibles

Matrices de puertas digitales programables por el diseñador.

*FPGA, Field Programmable Gate Array.*





# IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

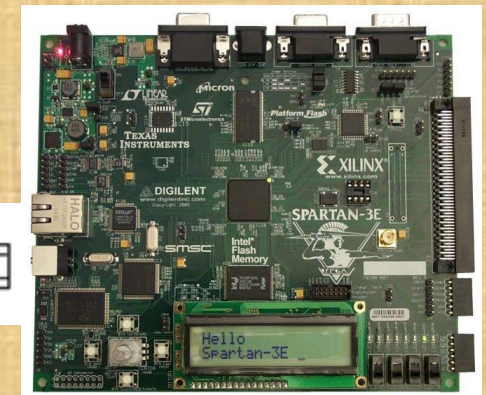
## Las tecnologías disponibles

Matrices de puertas digitales programables por el diseñador.

**FPGA**, *Field Programmable Gate Array*.



*FPGA download  
FPGA program  
FPGA configuration*







# IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

## Los métodos de diseño para FPGA

### Los lenguajes de descripción hardware

- ➔ Aparecen en los **años ochenta**.
- ➔ **HDL**, *Hardware Description Language*.
- ➔ Permiten describir sistemas digitales usando **texto**.
- ➔ Existen dos HDL estándar:
  - **Verilog**
  - **VHDL**, (*Very High Speed Integrated Circuit Hardware Description Language*, **VHSIC-HDL**)
- ➔ **Portabilidad** a cualquier dispositivo FPGA.

```
library ieee;
use ieee.math_real.all,ieee.std_logic_1164.all;
use work.all;

entity phyto is
    port (ir_light,red_light : in bit;
          ProtC : out bit := '1');
end;

architecture behaviour of phyto is

Begin

    encode : process (ir_light,red_light)
    Begin
        if red_light = '1' then
            ProtC <= '0' after 10 sec;
        elsif ir_light = '1' then
            ProtC <= '1' after 10 sec;
        end if;
    end process;

end behaviour;
```

# IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

## Los métodos de diseño para FPGA

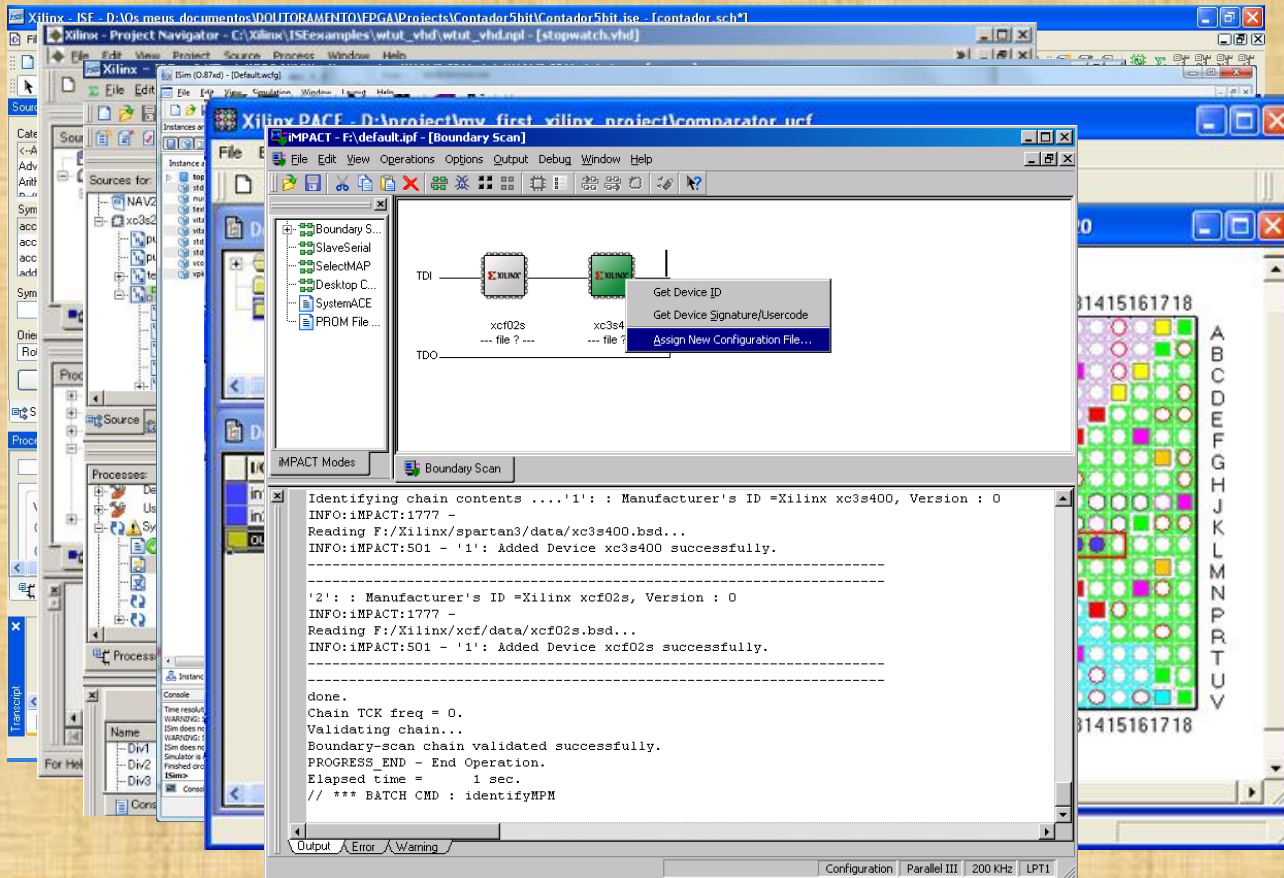
### Los entornos de los fabricantes

Entornos de diseño **estándar**.

No son gratuitas (salvo donación o demostración).

Prestaciones físicas:

- Área
- Velocidad
- Potencia





# IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

## Los métodos de diseño para FPGA

### Los entornos de los fabricantes



Entornos de diseño sobre **Simulink** de Matlab.

Tras la instalación aparecen varios **Blocksets**.

Con **Simulink** se diseña de forma **rápida y flexible**.

Entorno de diseño gráfico que usa **diagrama de bloques**.

Los bloques son configurables mediante sus **ventanas de diálogo**.

Ofrece facilidades para la **simulación**.

Acceso directo al **espacio de variables de Matlab**.

Las simulaciones son **muy rápidas** porque tienen un nivel pobre de detalle.

Es posible la comprobación de la total **funcionalidad** del sistema.

Se facilita la comprobación de **diferentes arquitecturas**.

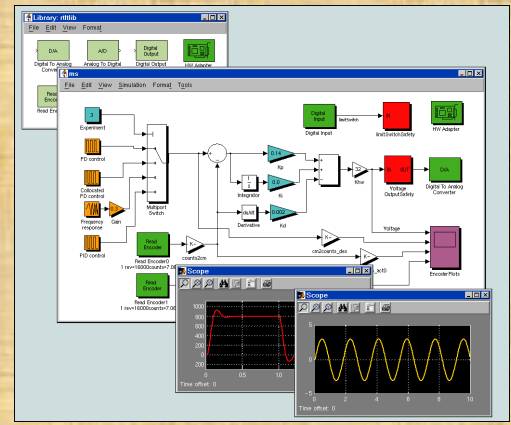
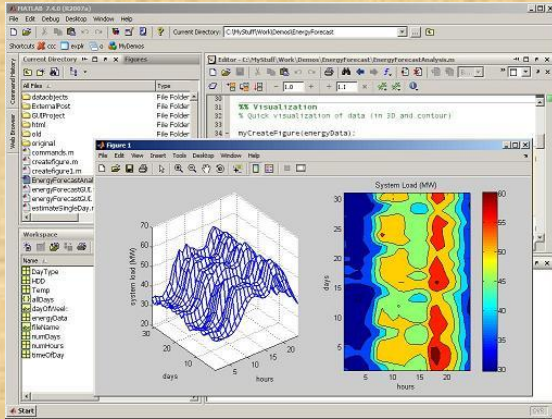
Una compilación genera el proyecto para el entorno **estándar**.



# IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

## Los métodos de diseño para FPGA

## Los entornos de los fabricantes



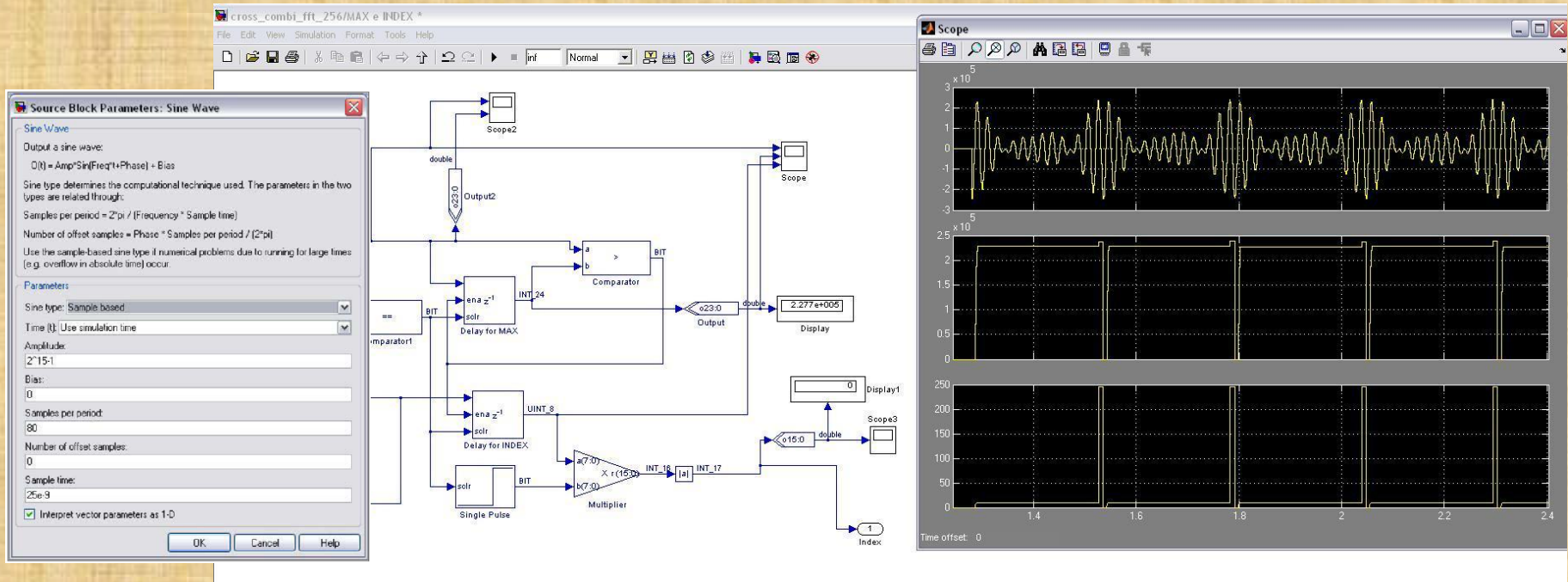
The Simulink Library Browser window displays the 'Xilinx Blockset/Basic Elements' library. The search results include:

- System Generator
- Assert
- Black Box
- Concat
- cast
- Delay
- Expression
- Out
- LFSR
- Mux
- Addressable Shift Register
- BitBasher
- Clock Enable Probe
- Constant
- Counter
- Down Sample
- Gateway In
- Inverter
- Logical
- Parallel to Serial

# IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

## Los métodos de diseño para FPGA

### Los entornos de los fabricantes



No portable a otros fabricantes.

Prestaciones físicas:

- Pobre estimación de área.
- No estima velocidad.
- No estima potencia.



Simulaciones muy rápidas.

Permite comprobar la total funcionalidad.



Compilación: HDL para entorno estándar.

# IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

## Los métodos de diseño para FPGA

### Xilinx versus Altera



Estándar



Simulink





# IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

## Los métodos de diseño para FPGA

### Parámetros de los métodos de diseño

*“Otra fuente de confusión es la plétora de herramientas disponibles hoy en día”*

Oldfield, J.; Dorf, R.; *Field-Programmable Gate Array*,  
John Wiley and Sons, 1995.

- El coste económico.
- El periodo de aprendizaje del diseñador.
- El soporte de las herramientas.
- La actualización del sistema.
- Los sistemas operativos sobre los que funciona.
- La ayuda ante errores.
- La portabilidad entre fabricantes y dispositivos.
- La flexibilidad del diseño.
- El tiempo de diseño.
- El tiempo de compilación.
- El tiempo de simulación.
- El tipo de reconfiguración del sistema.
- La seguridad y privacidad del diseño.
- Interactuación con otras herramientas.



# ÍNDICE

1. INTRODUCCIÓN

2. IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

**3. METODOLOGÍA EXPERIMENTAL**

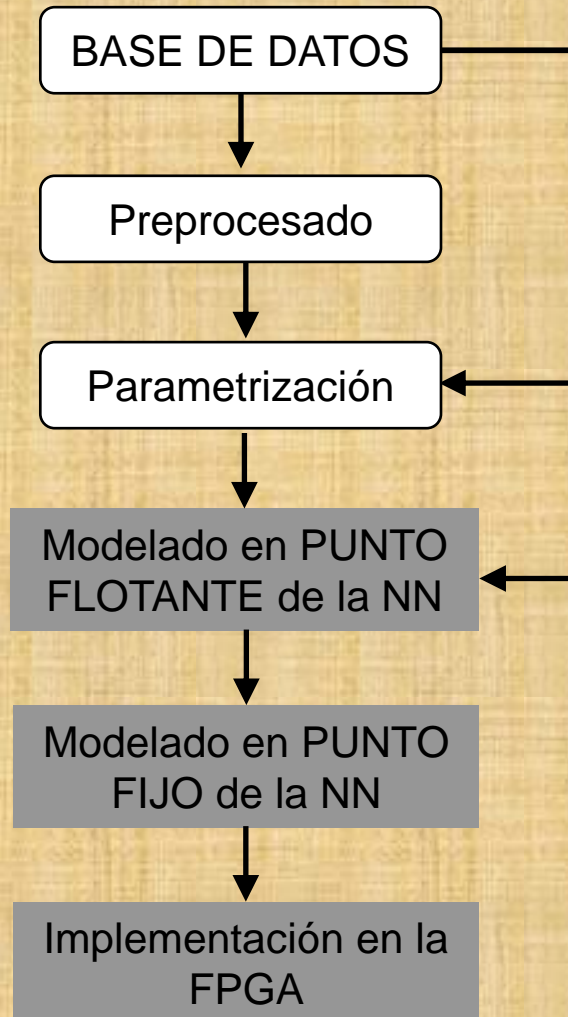
4. EXPERIMENTOS Y RESULTADOS

5. CONCLUSIONES Y LÍNEAS FUTURAS

6. DEMOSTRACIÓN

# METODOLOGÍA EXPERIMENTAL

## Esquema general de la metodología experimental



# METODOLOGÍA EXPERIMENTAL

## Las bases de datos

Los cuatro escenarios usados son:

- ➔ clasificación de la **palmera pejibaye**,
- ➔ clasificación de **pulsos de electrocardiograma**,
- ➔ predicción de **temperatura**,
- ➔ y ecualizador para **señal binaria con ruido**.



# METODOLOGÍA EXPERIMENTAL



## Las bases de datos

### La palmera pejibaye (*Bactris gasipaes* Kunth)

- ➔ Especia nativa de la **América tropical**.
- ➔ Sus frutos y brotes tiernos se usan como **alimento**.
- ➔ El tronco de la palmera adulta se usa como **madera**.
- ➔ La base de datos usada procede del **Banco de Germoplasma de la Universidad de Costa Rica**:
  - ➔ marcadores moleculares de amplificación aleatoria del ácido desoxirribonucleico polimórfico (RAPD, *Random Amplification of Polymorphic Deoxyribonucleic acid*).
- ➔ Cada elemento dispone de **10 parámetros**.



Clase	Denominación	Código	Número de elemento
Clase 1	<i>Costa rica - utilitis</i>	c	01-13
Clase 2	<i>Tuira</i>	t	14-26
Clase 3	<i>Pará</i>	p	27-39
Clase 4	<i>Yurimagua</i>	y	40-52
Clase 5	<i>Putumayo</i>	u	53-65
Clase 6	<i>Bolivia – tembé</i>	b	66-78

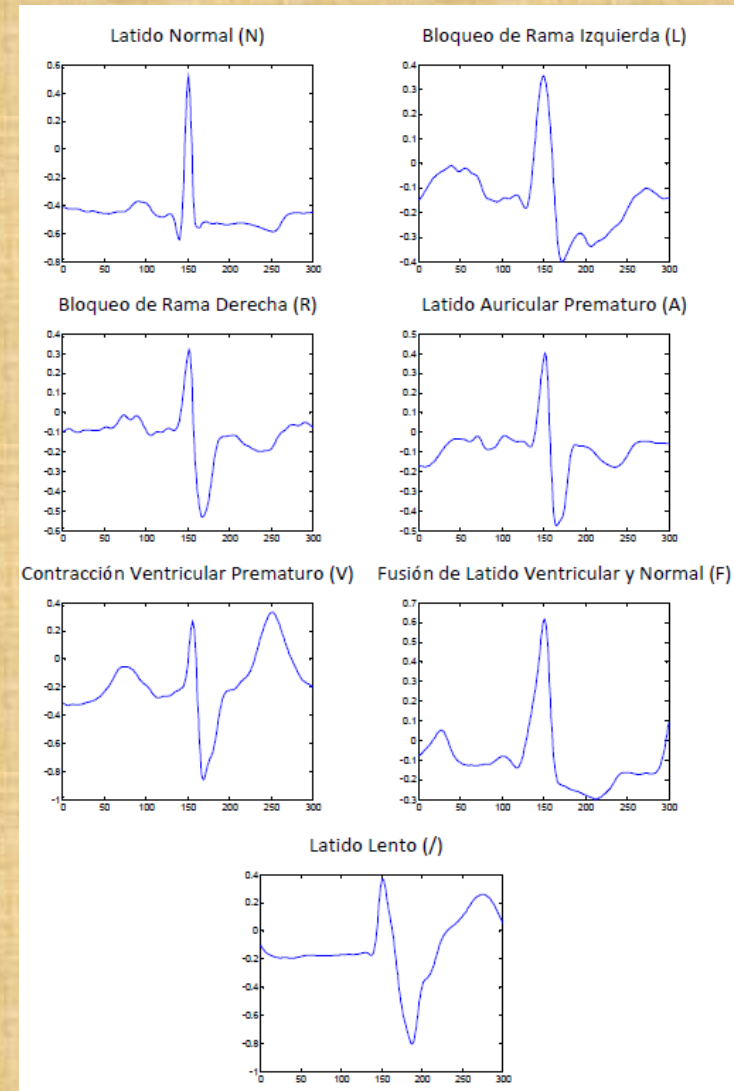


# METODOLOGÍA EXPERIMENTAL

## Las bases de datos

### Pulsos electrocardiográficos

- ➔ **MIT-BIH** (*Massachusetts Institute of Technology-Beth Israel Hospital*) **Arrhythmia Database**.
- ➔ **48 registros** de electrocardiograma.
- ➔ Cada registro dura **30 minutos**.
- ➔ **19 tipos de cardiopatías** diferentes.
- ➔ Tomados a **360 muestras por segundo**.
- ➔ Conversión analógica a digital con **11 bits**.
- ➔ Se detectó los **7 tipos de pulsos** más frecuentes.



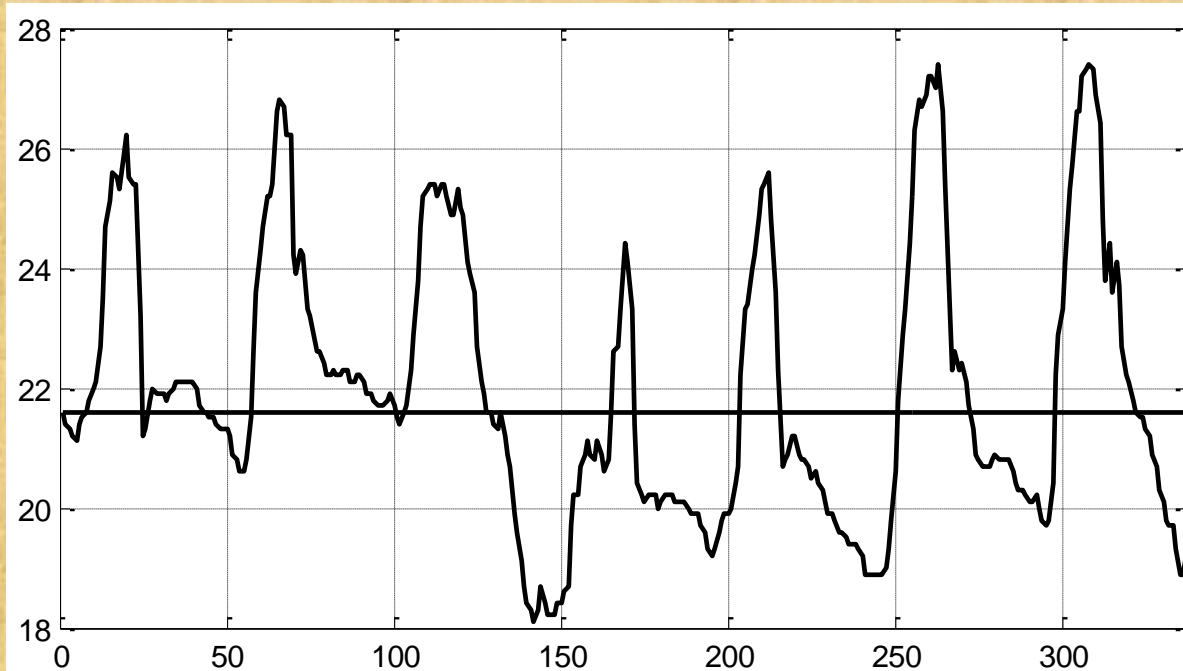
# METODOLOGÍA EXPERIMENTAL



## Las bases de datos

### Temperatura

- Estación meteorológica de la **Universidad de Costa Rica**.
- Ciudad de **Turrialba**.
- Desde mediados del año **2007** hasta mediados de **2010**.
- Tomada a intervalos de **media hora**.
- Fluctuó entre **11,9 y 31,8 °C**.

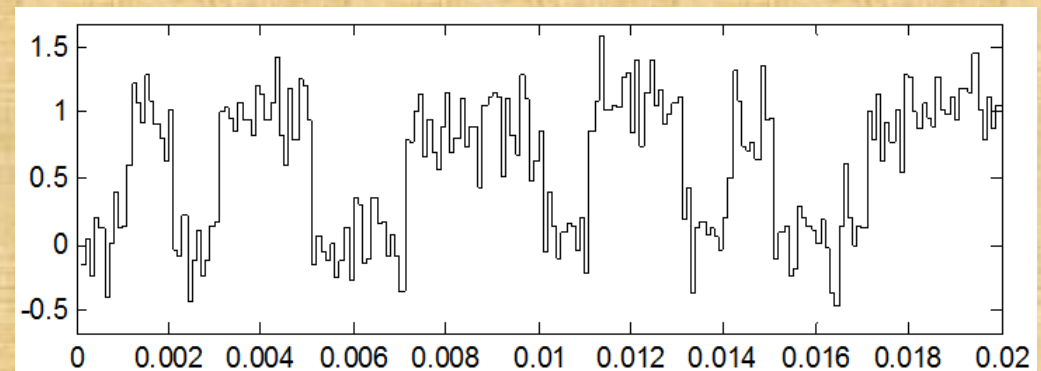
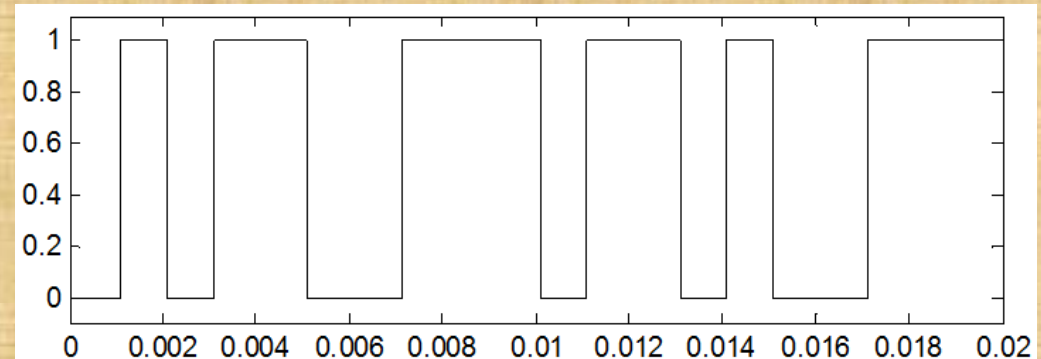


# METODOLOGÍA EXPERIMENTAL

## Las bases de datos

### Señal binaria con ruido

- ➔ Base de datos **sintética**.
- ➔ Símbolos **equiprobables**.
- ➔ Señal binaria **unipolar** tipo **NRZ**.
- ➔ Inicialmente de **1 kilobit por segundo**.
- ➔ **Ruido blanco gaussiano aditivo**.
- ➔ **Muestreada** a 10 kHz.
- ➔ Se usó **2.000 bits** generados de forma aleatoria.
- ➔ La **relación señal a ruido** varió entre +20 dB y -5 dB.
- ➔ El objetivo fue diseñar un **ecualizador**.





# METODOLOGÍA EXPERIMENTAL

## Las bases de datos

### Características de las bases de datos

	<b>Pejibaye</b>	<b>ECG</b>	<b>Temperatura</b>	<b>Señal binaria con ruido</b>
<b><i>Tipo numérico</i></b>	Real	Real	Real	Real
<b><i>Dimensionalidad</i></b>	Multidimensional	Unidimensional	Unidimensional	Unidimensional
<b><i>Origen</i></b>	Natural	Natural	Natural	Sintética
<b><i>Aleatoriedad</i></b>	Aletoria	Pseudoperiódica	Pseudoperiódica	Aletoria
<b><i>Nivel de ruido</i></b>	Fijo	Variable	Fijo	Variable
<b><i>Etiquetado</i></b>	Especialista externo	Especialista externo	No etiquetada	Etiquetado propio
<b><i>Coste</i></b>	Gratuito	Gratuito	Gratuito	Gratuito

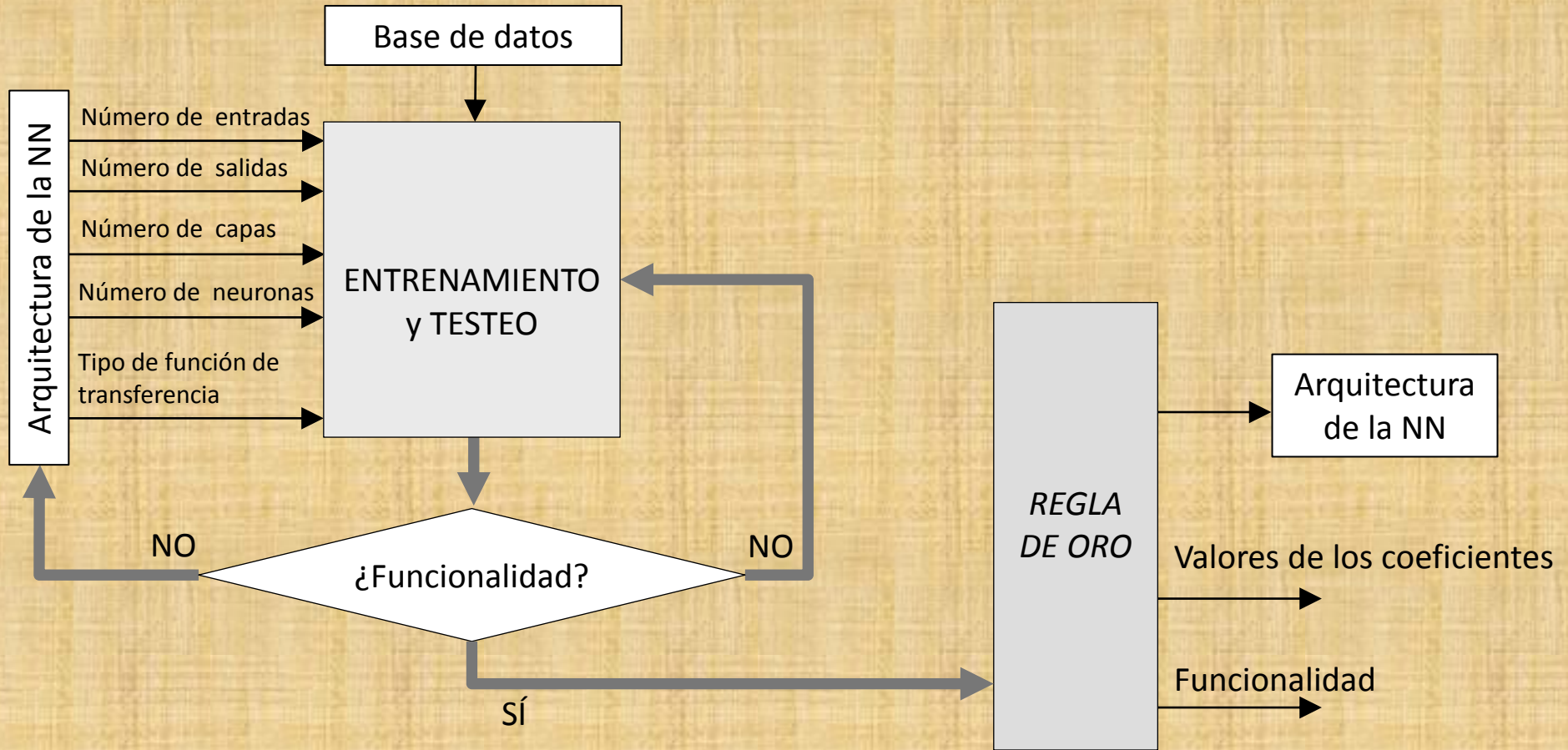
### Preprocesado y parametrización

	<b>Pejibaye</b>	<b>ECG</b>	<b>Temperatura</b>	<b>Señal binaria con ruido</b>
<b><i>Preprocesado</i></b>	No	Sí	No	No
<b><i>Parametrización</i></b>	No	Sí	No	No



# METODOLOGÍA EXPERIMENTAL

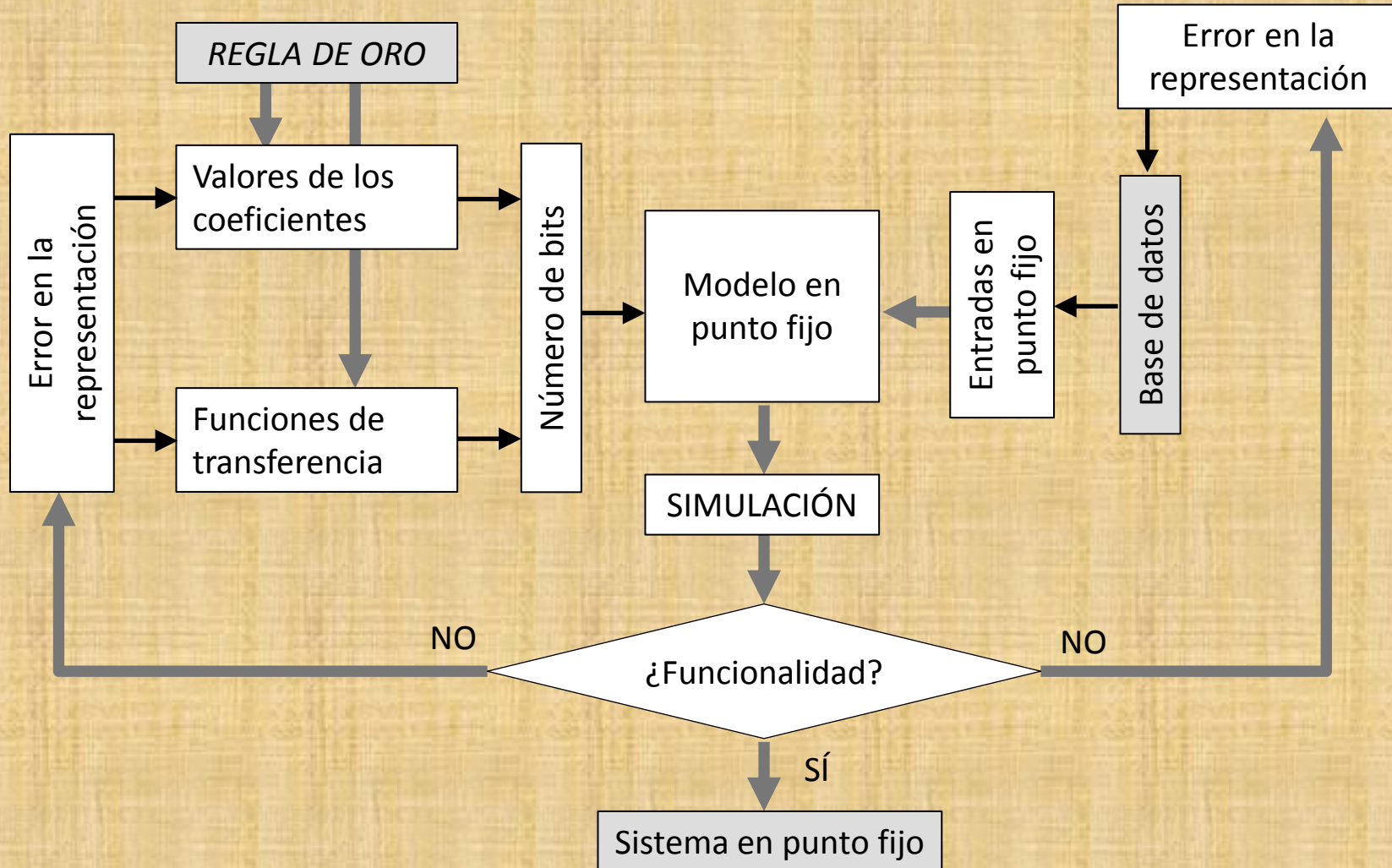
## Modelado en punto flotante



	Pejibaye	ECG	Temperatura	Señal binaria con ruido
<b>Operación de la NN</b>	Clasificador	Clasificador	Predictor	Predictor
<b>Funcionalidad</b>	Tasa de acierto	Tasa de acierto	Valor medio del error absoluto	Relación señal a ruido

# METODOLOGÍA EXPERIMENTAL

## Modelado en punto fijo



# METODOLOGÍA EXPERIMENTAL

## Herramienta y flujo de diseño

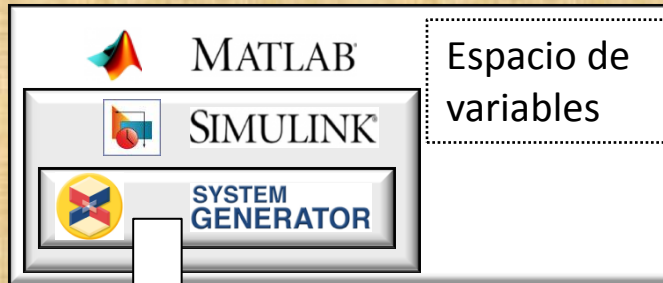
Debe garantizar algunos de los parámetros descritos anteriormente:

- El coste económico.
- El periodo de aprendizaje.
- El soporte de las herramientas.
- La actualización del sistema.
- Los sistemas operativos sobre los que funciona.
- **La ayuda ante errores.**
- La portabilidad entre fabricantes y dispositivos.
- **La flexibilidad del diseño.**
- **El tiempo de diseño y compilación.**
- **El tiempo de simulación.**
- El tipo de reconfiguración del sistema.
- La seguridad y privacidad del diseño.
- Interactuación con otras herramientas.

Entornos de los  
suministradores de FPGA  
sobre **Simulink** de Matlab.

# METODOLOGÍA EXPERIMENTAL

## El entorno de diseño de Xilinx *System Generator*

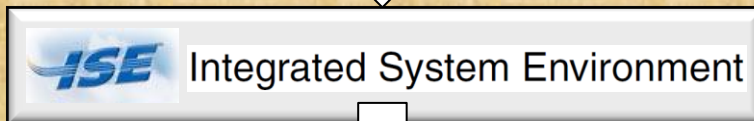


Total funcionalidad

Descripción en HDL

Generación del *Testbench*

Estimación aproximada de **área**  
No estimación de **velocidad**  
No estimación de **potencia**



Estimación de **área**  
Estimación de **velocidad**  
Estimación de **potencia**

### Versiones usadas:

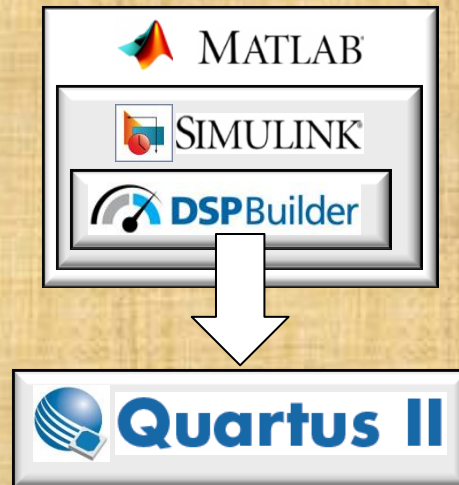
- *System Generator for DSP and AccelDSP 10.1*
  - o *Integrated System Environment 10.1*
  - o Windows XP (32 bits)
  - o Matlab R2007a
- *System Generator for DSP 13.1*
  - o *Integrated System Environment 13.1*
  - o Windows XP (32 bits) y Windows 7 (64 bits)
  - o Matlab R2010a y R2010b



# METODOLOGÍA EXPERIMENTAL

El entorno de diseño de Altera

*DSP Builder*



# ÍNDICE

1. INTRODUCCIÓN

2. IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

3. METODOLOGÍA EXPERIMENTAL

**4. EXPERIMENTOS Y RESULTADOS**

5. CONCLUSIONES Y LÍNEAS FUTURAS

6. DEMOSTRACIÓN

# EXPERIMENTOS Y RESULTADOS

## La clasificación de la palmera pejibaye

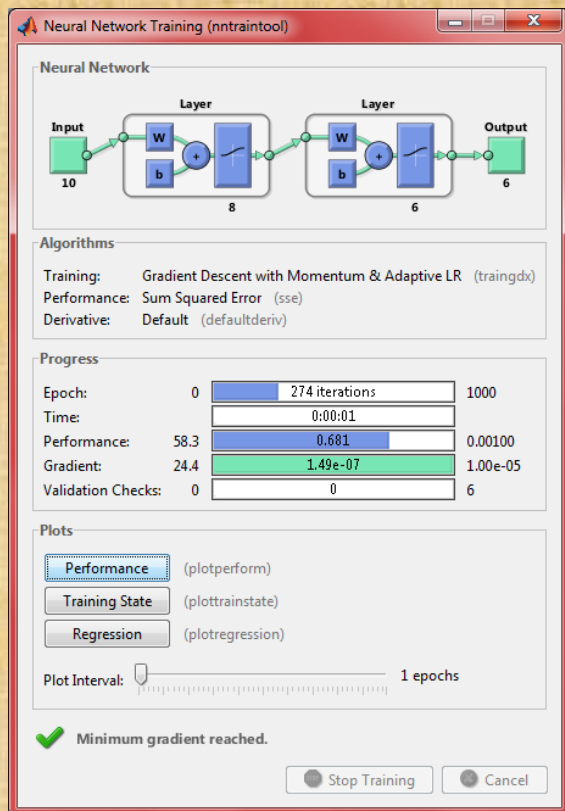
### Modelado en punto flotante

- Clasificación con **diez parámetros**.
- **Seis razas**.
- Entrenamiento se realizó con el *Neural Network Toolbox* de Matlab.
- El entrenamiento es **supervisado**.
- Se varió el **número de capas**, y el **número de neuronas**.
- Se alcanza el **100%** con una sola capa intermedia de 8 neuronas.
- La NN es del tipo **10-8-6**.
- Se optó por la función tipo *logsig* para todas las neuronas
- Algoritmo de entrenamiento: *traingdx* . Algoritmo de entrenamiento adaptativo con retroalimentación del error.
- Función de error: **sse** (*Sum squared error*).

# EXPERIMENTOS Y RESULTADOS

## La clasificación de la palmera pejibaye Modelado en punto flotante

Ventana obtenida al finalizar el entrenamiento.



Matriz de confusión obtenida en la etapa de testeo.

The screenshot shows the 'Array Editor - Matriz\_confusion\_porcentaje' window. The title bar is blue with the text 'Array Editor - Matriz\_confusion\_porcentaje'. Below the title bar is a menu bar with 'File', 'Edit', 'View', 'Graphics', 'Debug', 'Desktop', 'Window', and 'Help'. Below the menu bar is a toolbar with icons for file operations and a 'Stack' dropdown menu set to 'Base'. The main area contains a 6x6 confusion matrix table.

	1	2	3	4	5	6
1	100	0	0	0	0	0
2	0	100	0	0	0	0
3	0	0	100	0	0	0
4	0	0	0	100	0	0
5	0	0	0	0	100	0
6	0	0	0	0	0	100



# EXPERIMENTOS Y RESULTADOS

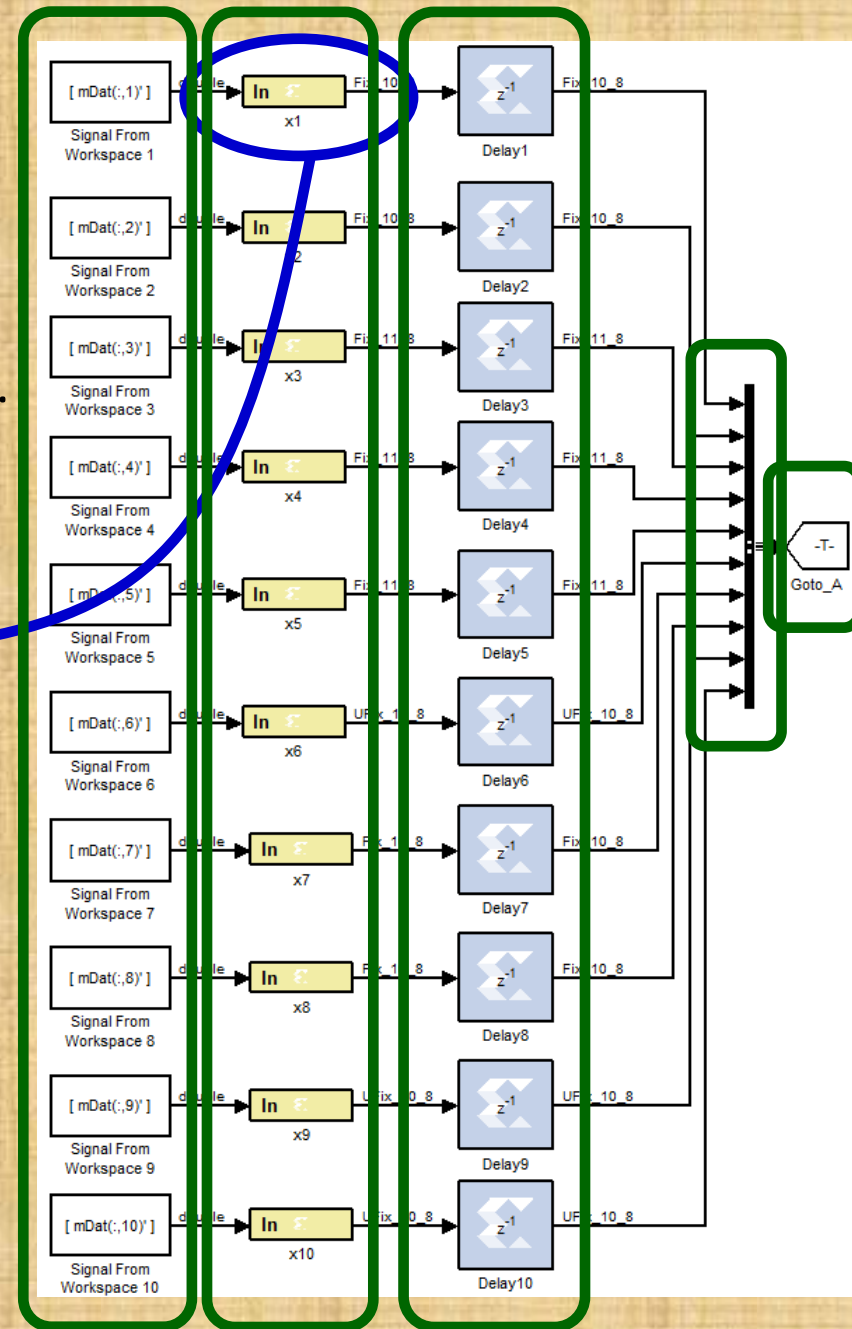
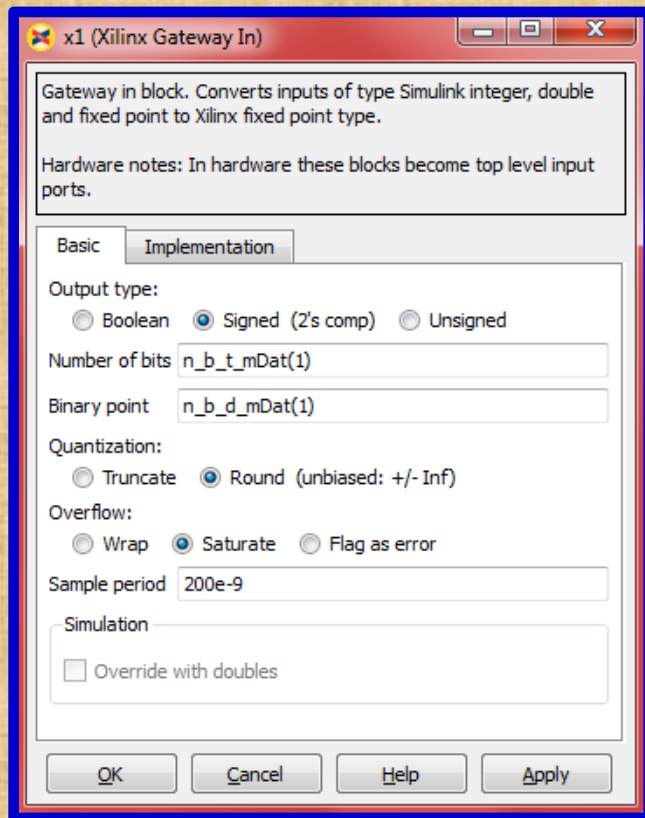
## La clasificación de la palmera pejibaye

### Diseño en punto fijo

### Diseño con *System Generator*

Etapa de entrada de la NN.

Ventana de configuración de un *Gateway In*.



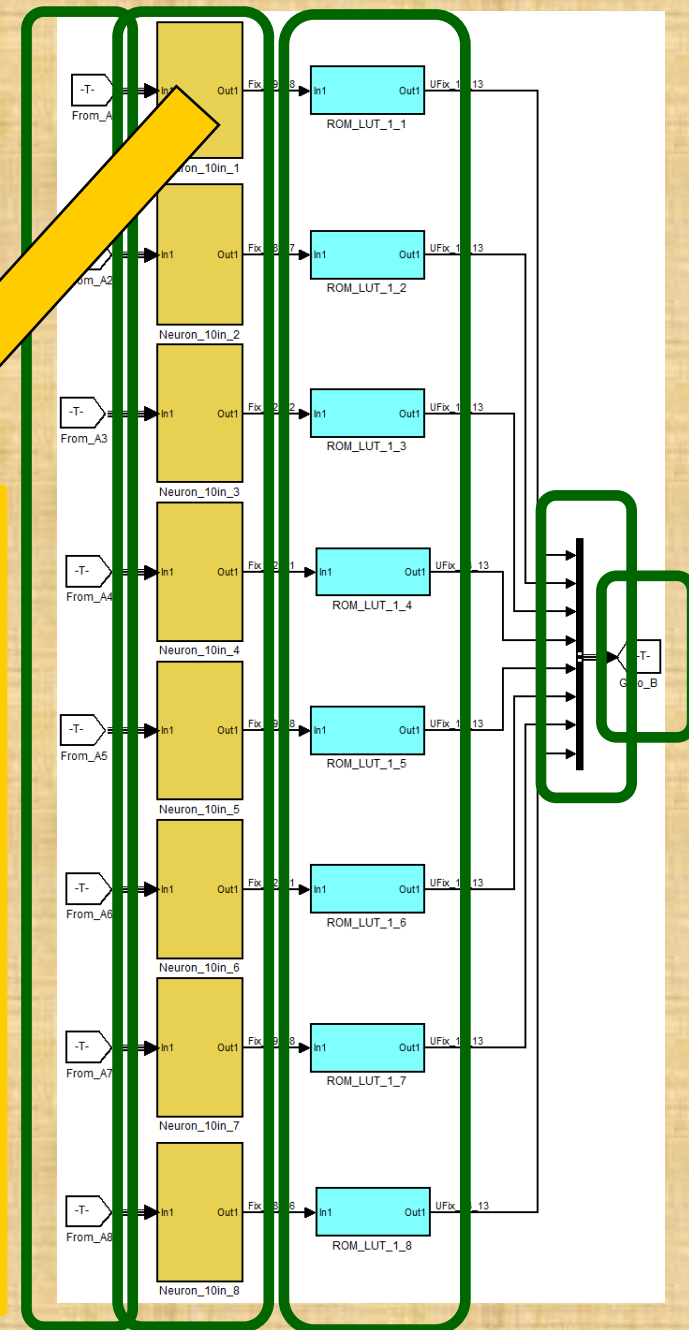
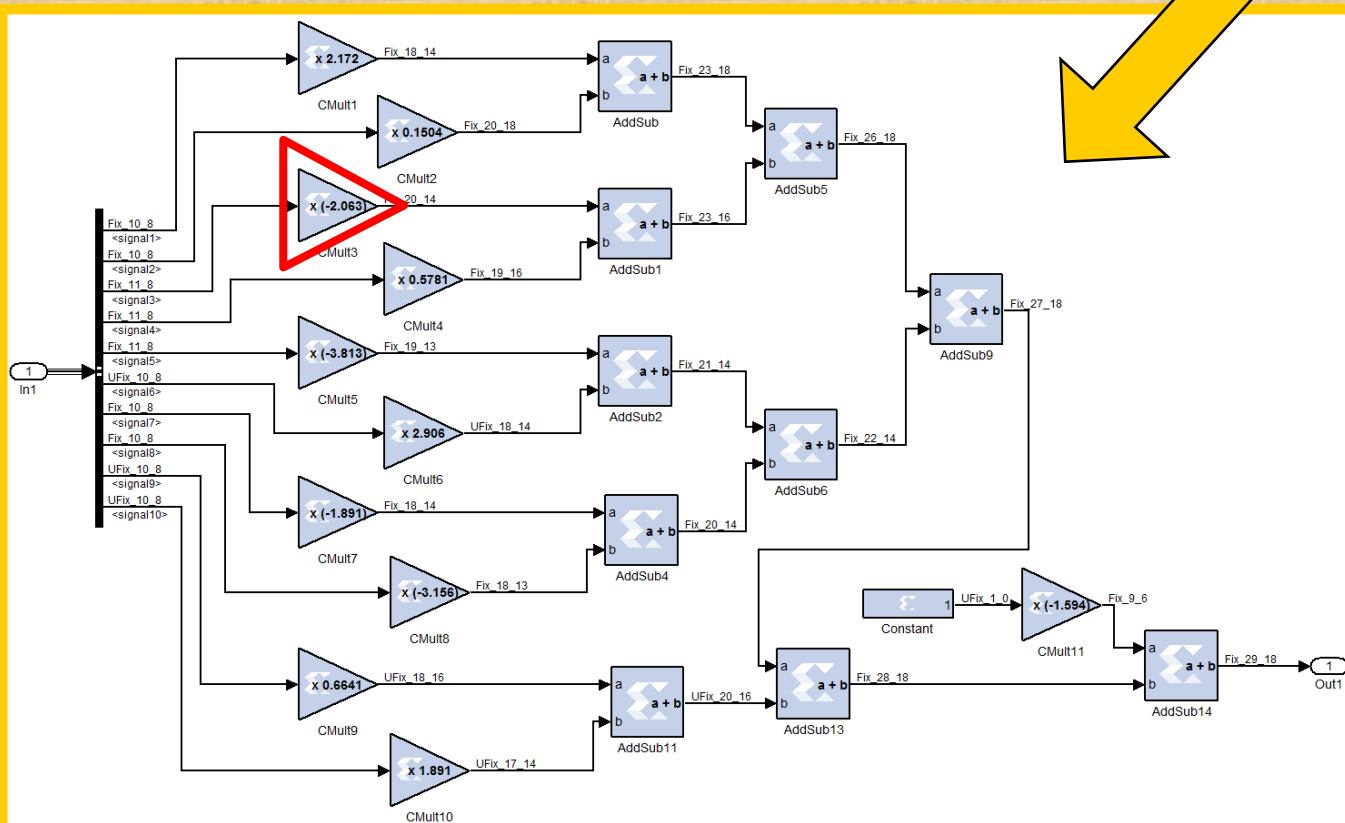
# EXPERIMENTOS Y RESULTADOS

## La clasificación de la palmera pejibaye

Diseño en punto fijo

Diseño con *System Generator*

Capa intermedia de la NN.

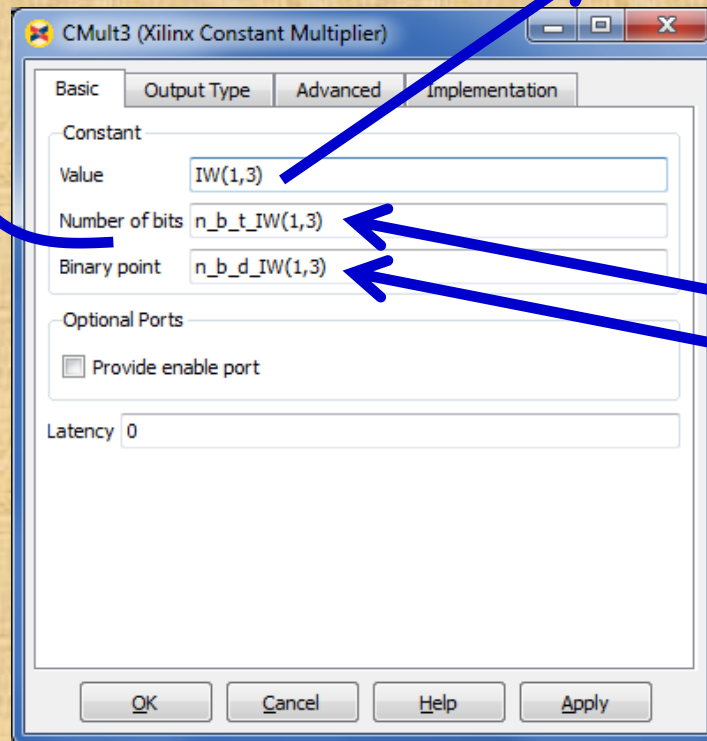
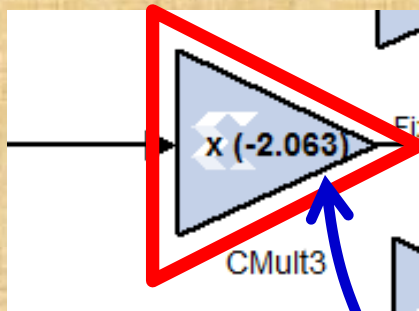


# EXPERIMENTOS Y RESULTADOS

## La clasificación de la palmera pejibaye

Diseño en punto fijo

Diseño con *System Generator*



-2.060084996074594      1 %  
Error

Programa MATLAB

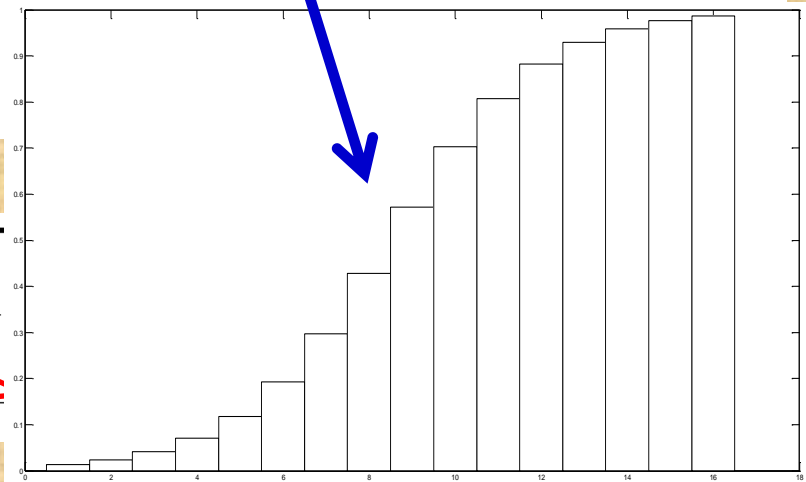
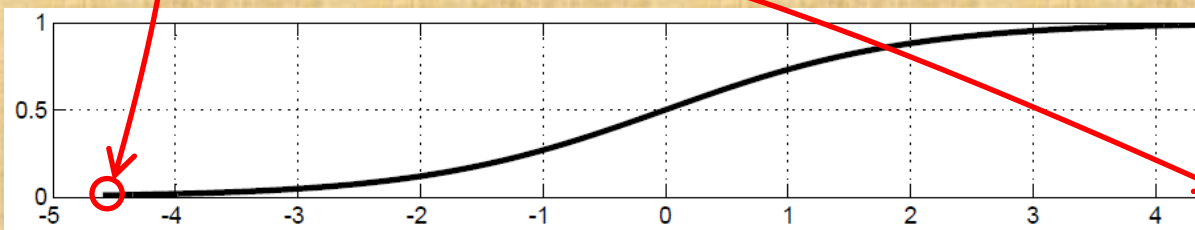
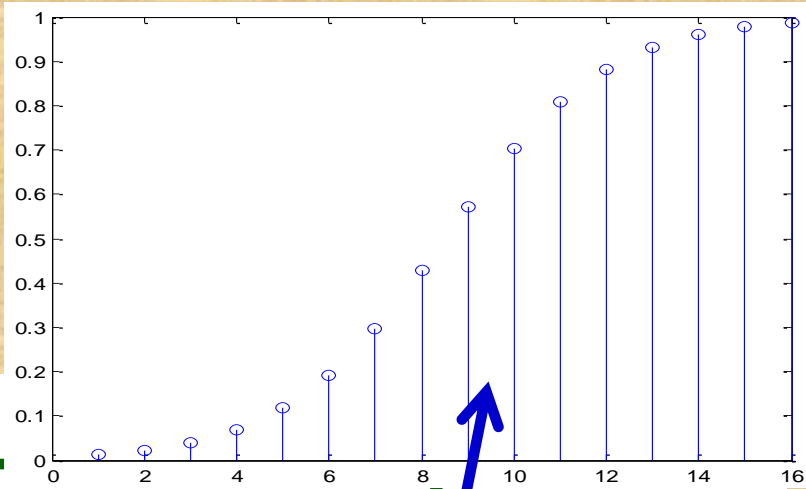
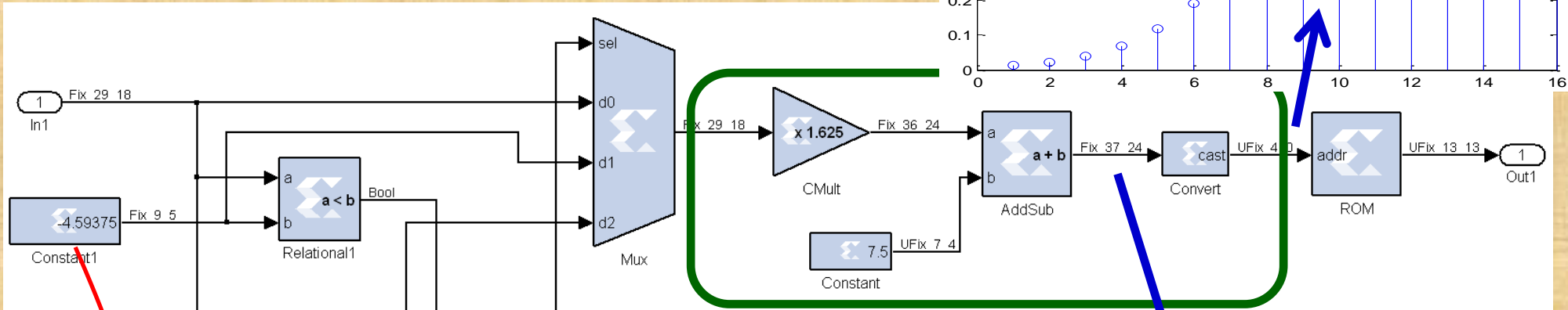
Number of bits = 9  
Binary point = 6

# EXPERIMENTOS Y RESULTADOS

## La clasificación de la palmera pejibaye

Diseño en punto fijo

Diseño con *System Generator*





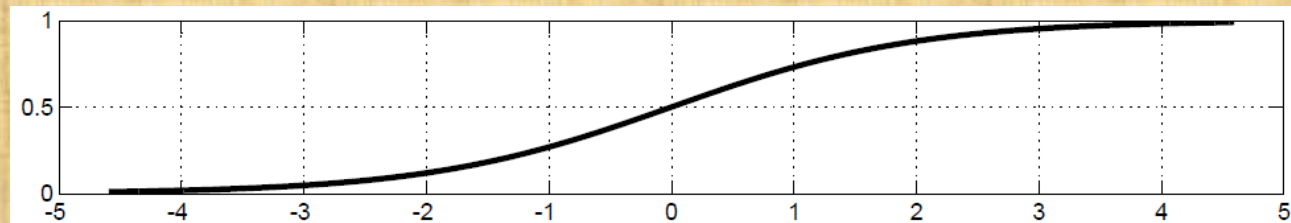
# EXPERIMENTOS Y RESULTADOS

## La clasificación de la palmera pejibaye

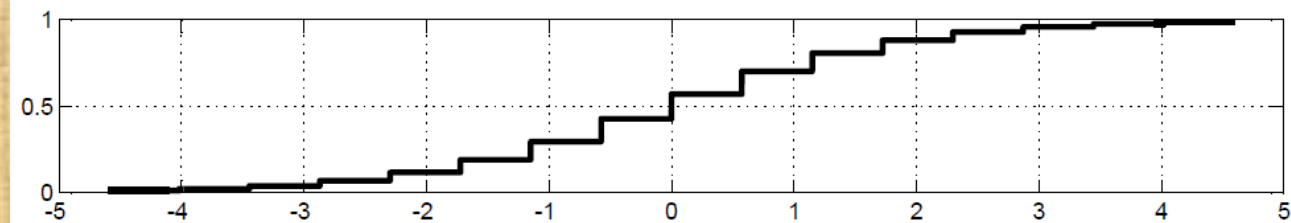
Diseño en punto fijo

Diseño con *System Generator*

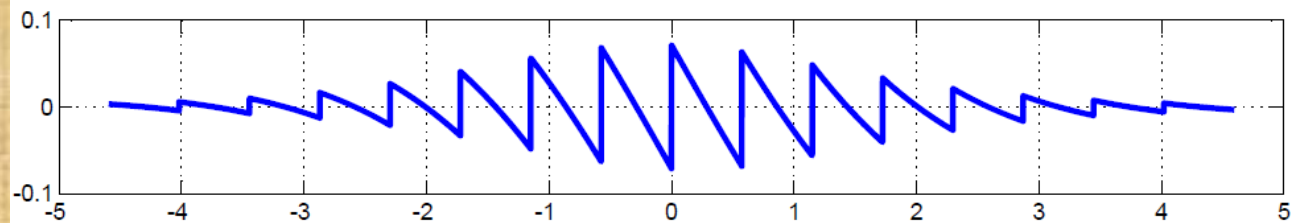
Función *logsig*



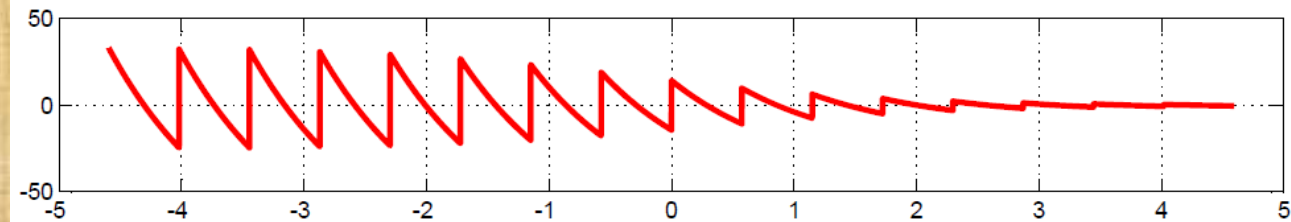
Salida de la función implementada



Error



Error relativo



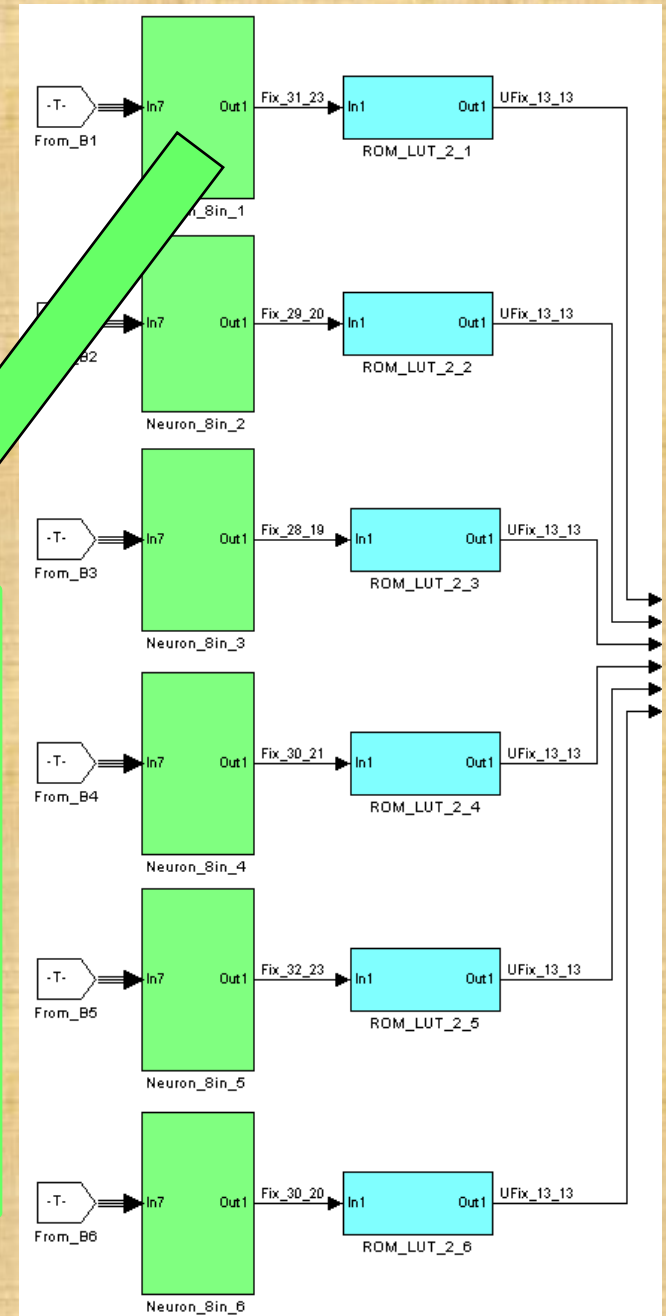
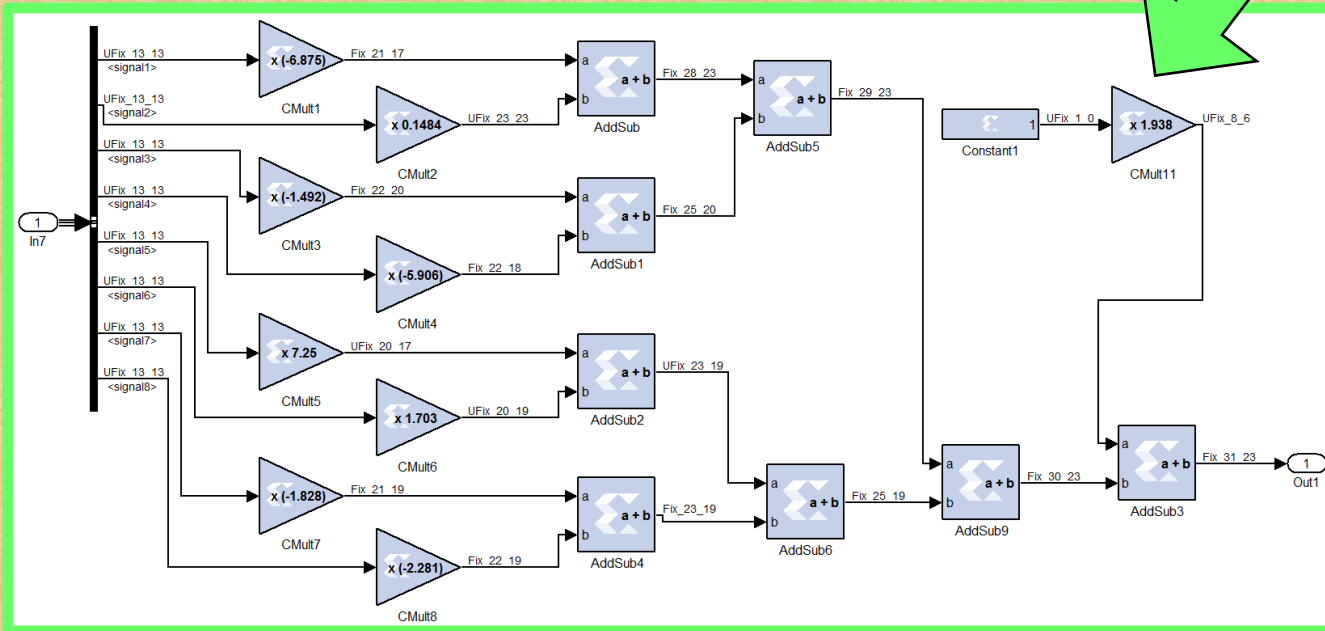
# EXPERIMENTOS Y RESULTADOS

## La clasificación de la palmera pejibaye

Diseño en punto fijo

Diseño con *System Generator*

Capa de salida de la NN.

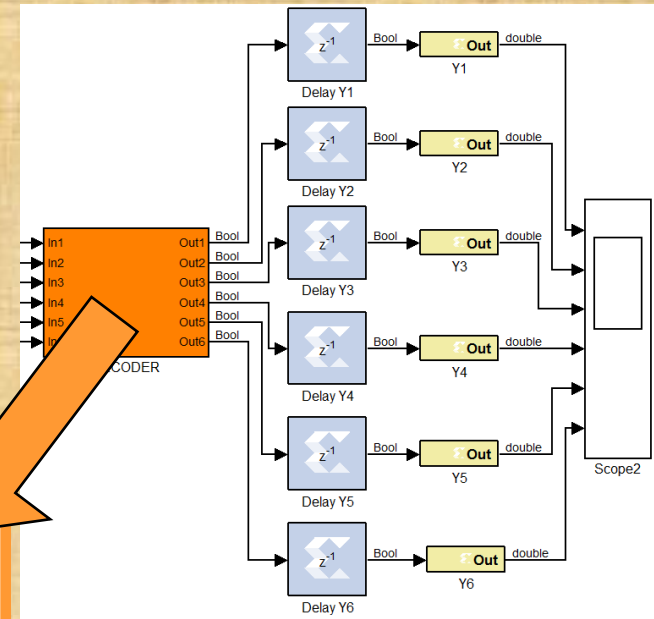
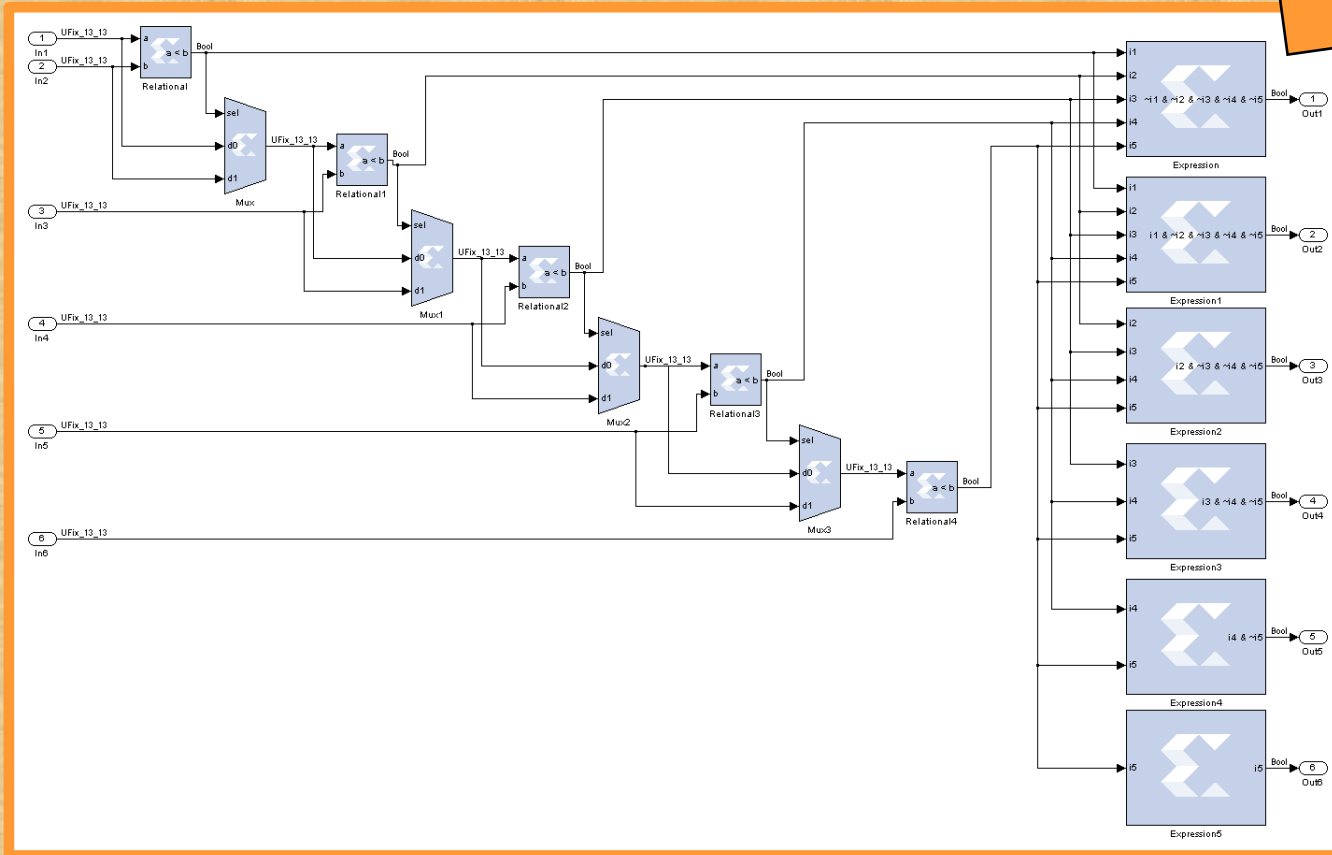


# EXPERIMENTOS Y RESULTADOS

## La clasificación de la palmera pejibaye

### Diseño en punto fijo

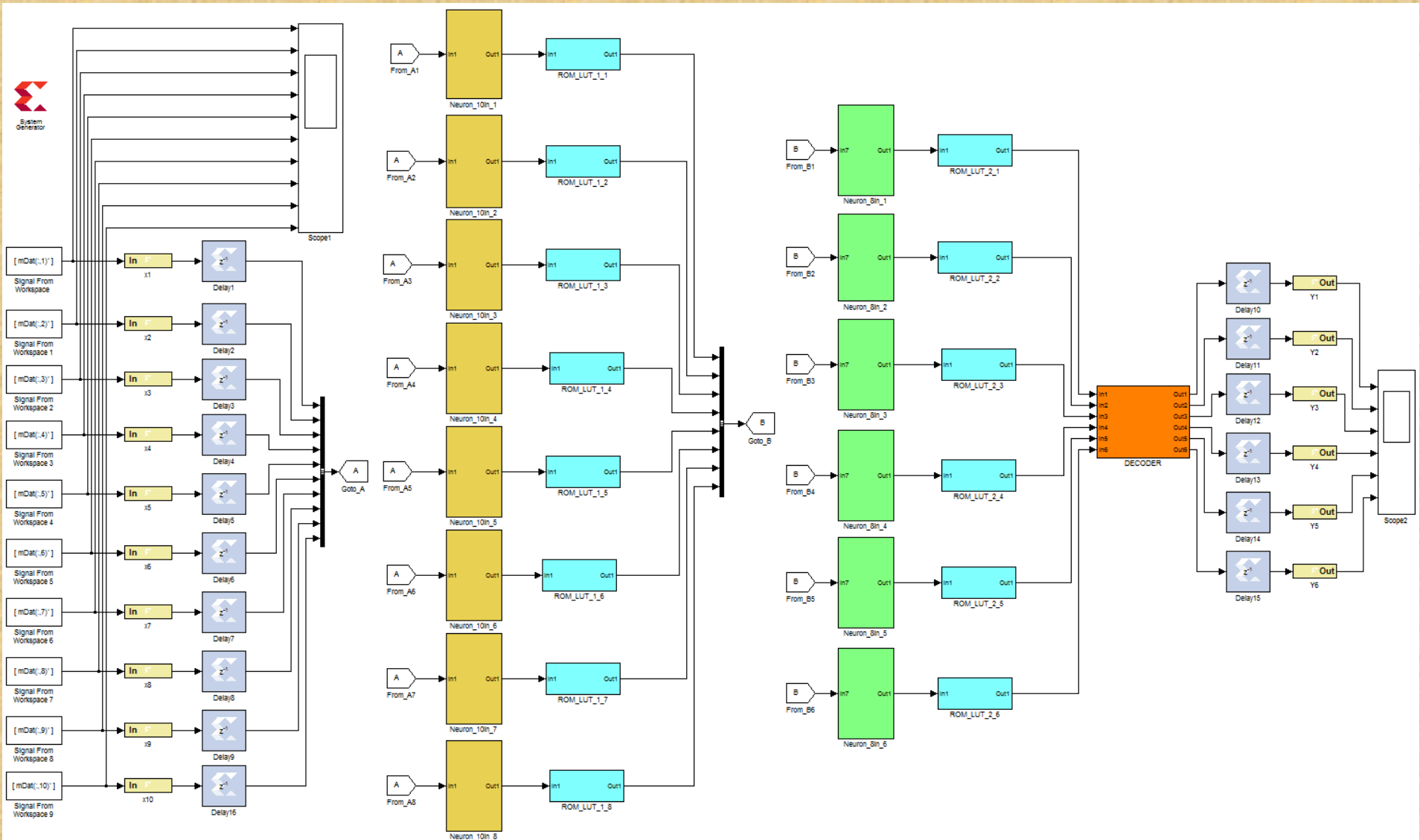
### Diseño con System Generator



# EXPERIMENTOS Y RESULTADOS

## La clasificación de la palmera pejibaye

Diseño en punto fijo  
Diseño con *System Generator*





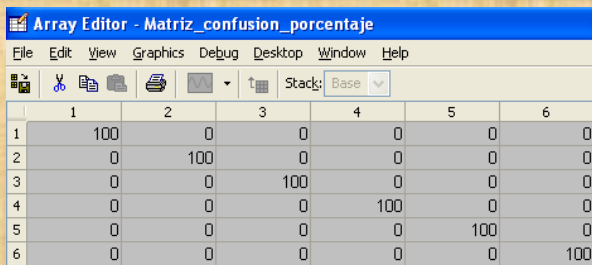
# EXPERIMENTOS Y RESULTADOS

## La clasificación de la palmera pejibaye

### Diseño en punto fijo

### Diseño con *System Generator*

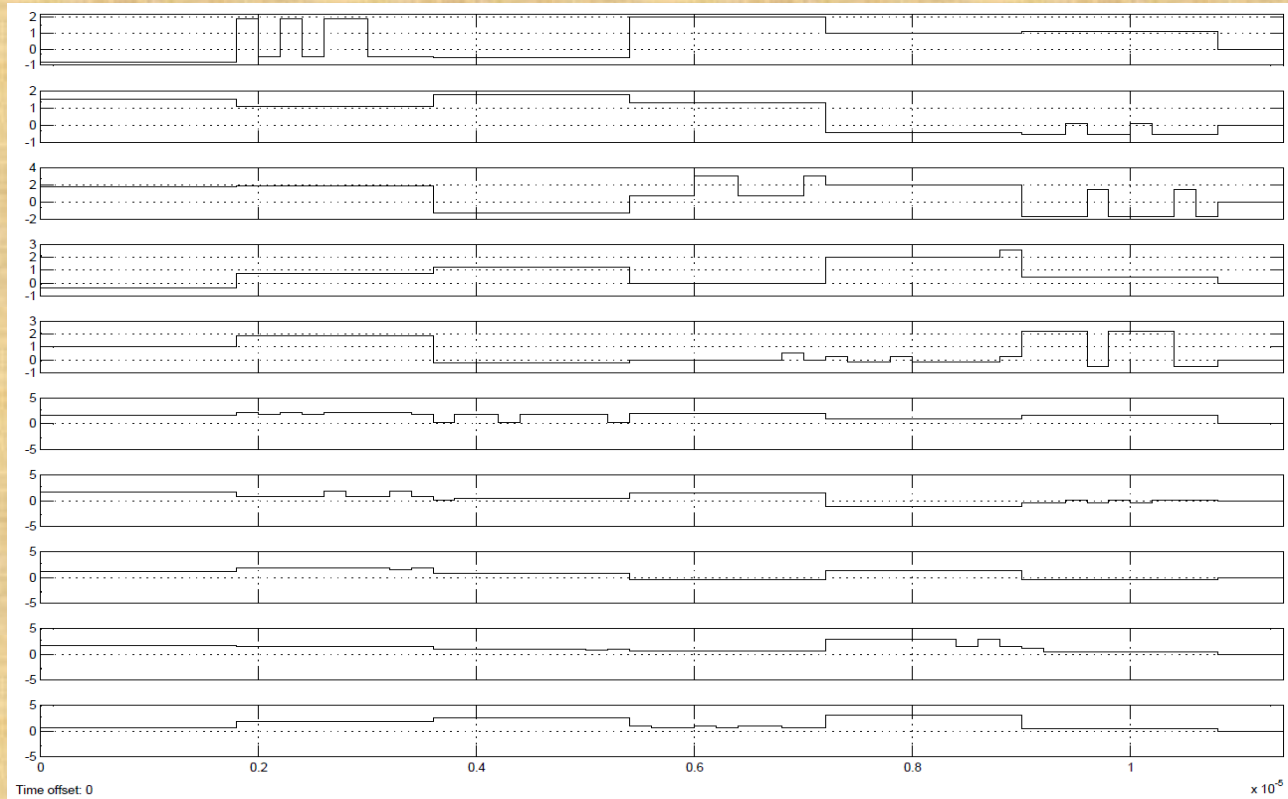
Matriz de confusión.



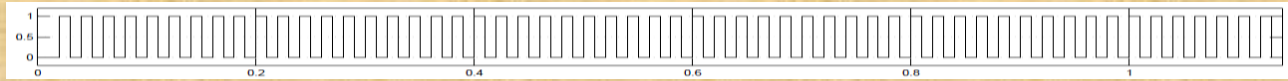
Array Editor - Matriz\_confusion\_porcentaje

	1	2	3	4	5	6
1	100	0	0	0	0	0
2	0	100	0	0	0	0
3	0	0	100	0	0	0
4	0	0	0	100	0	0
5	0	0	0	0	100	0
6	0	0	0	0	0	100

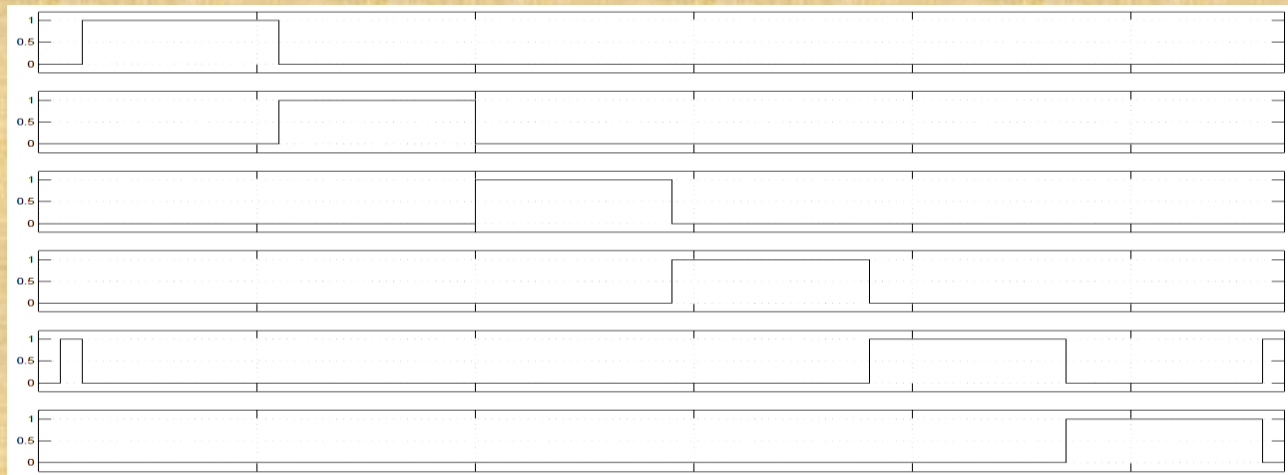
ENTRADAS



Reloj



SALIDAS

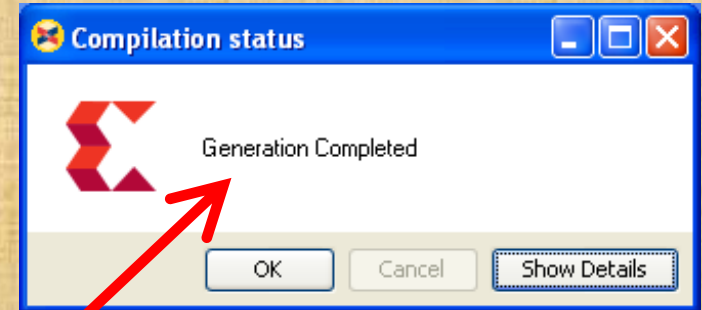
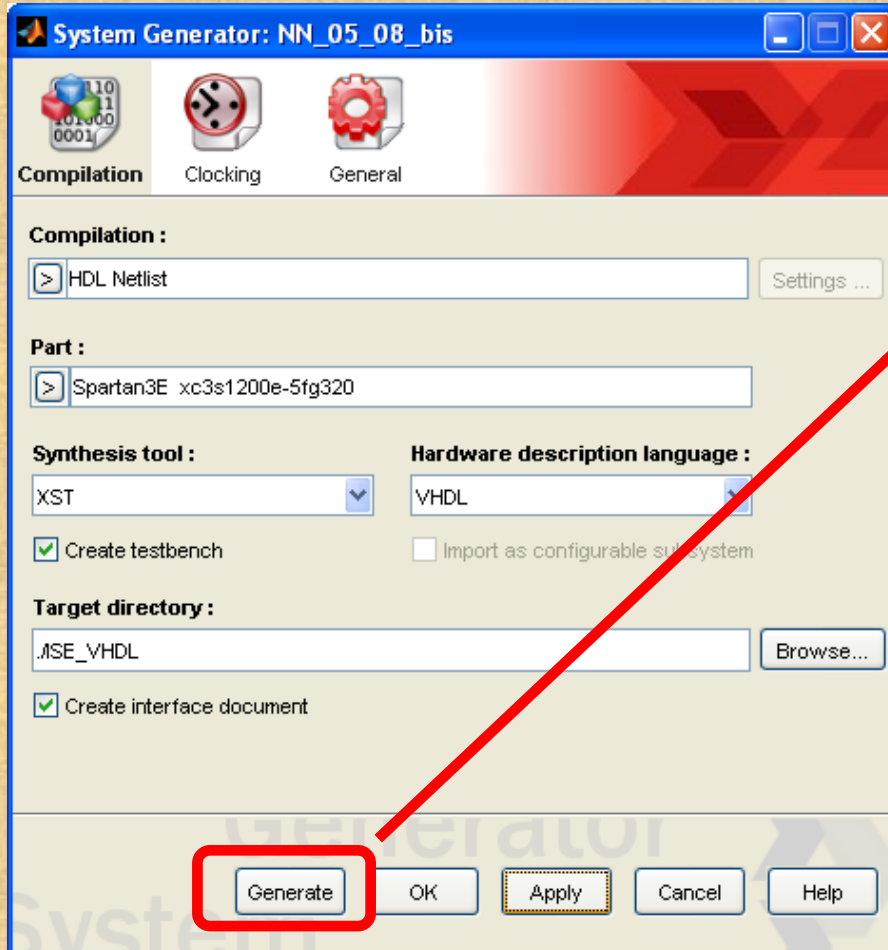
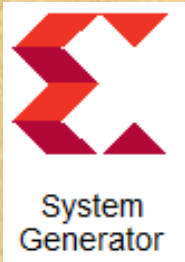


# EXPERIMENTOS Y RESULTADOS

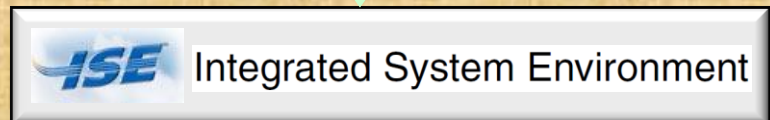
## La clasificación de la palmera pejibaye

### Diseño en punto fijo

### Diseño con *System Generator*



Los ficheros de la descripción estructural en VHDL o Verilog



# EXPERIMENTOS Y RESULTADOS

## La clasificación de la palmera pejibaye

### Diseño en punto fijo

### Diseño con *System Generator*

Funcionalidad del sistema frente al error en la representación y el número de palabras en las ROM.

		Error en la representación (%)													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Número de palabras en las ROM	1024	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	NO
	512	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	NO
	256	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	NO
	128	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	NO
	64	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	NO	NO
	32	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	NO	NO
	16	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	NO	NO
	8	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	NO	NO
	4	SI	SI	SI	SI	SI	SI	SI	SI	NO	NO	NO	NO	NO	NO
	2	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO

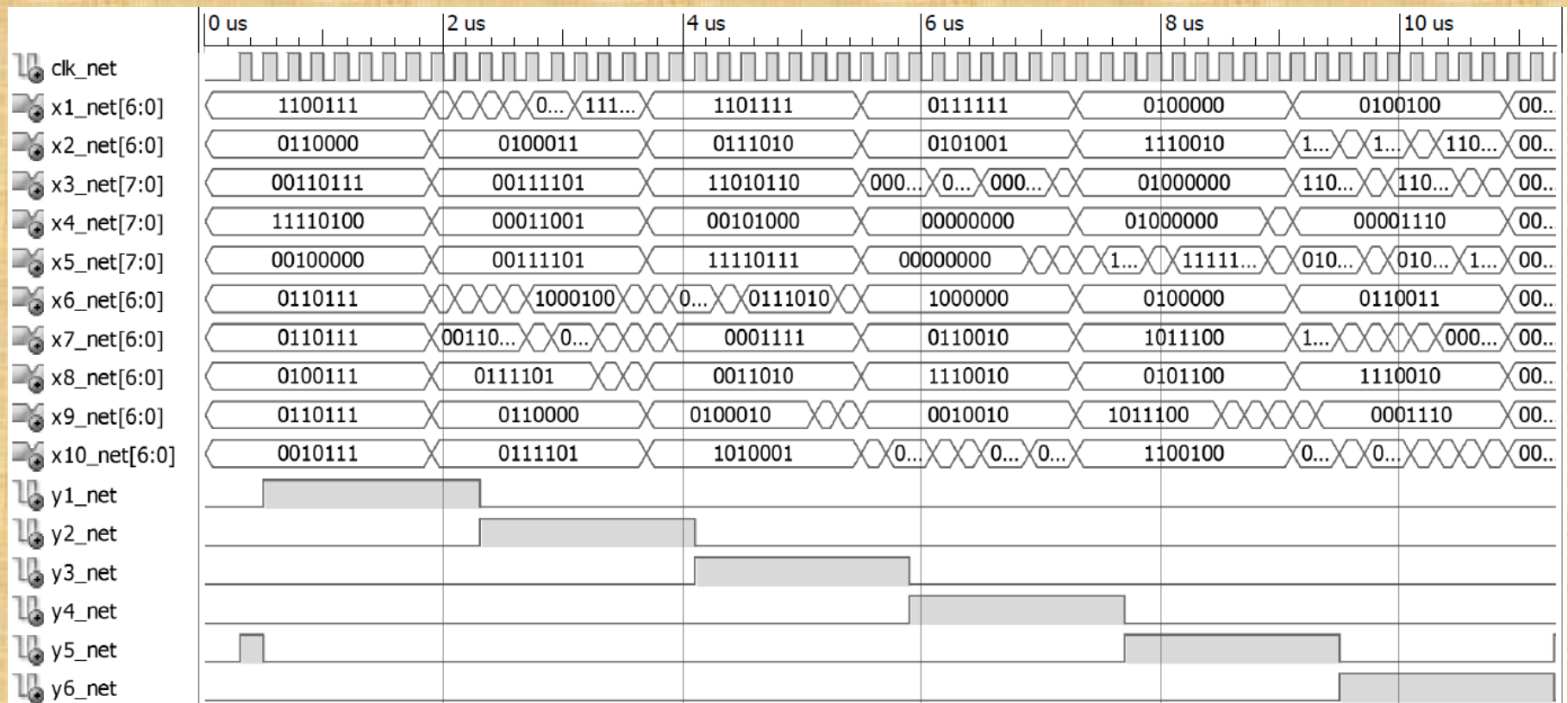
# EXPERIMENTOS Y RESULTADOS

## La clasificación de la palmera pejibaye

### Diseño en punto fijo

### Implementación con *Integrated System Environment*

Simulación de la implementación para pejibaye en la FPGA, 8% de error y 4 palabras en las ROM.





# EXPERIMENTOS Y RESULTADOS

## La clasificación de la palmera pejibaye

### Diseño en punto fijo

### Implementación con *Integrated System Environment*

Prestaciones de las implementaciones para pejibaye en la FPGA.

	VHDL		Verilog	
8% de error 4 palabras en la ROM	Área	2.750 SLICES	Área	2.780 SLICES
	Frecuencia máxima	11,034 MHz	Frecuencia máxima	10,2 MHz
	Potencia (5 MHz)	167,02 mW	Potencia (5 MHz)	166,86 mW
12% de error 8 palabras en la ROM	Área	2.574 SLICES	Área	2.640 SLICES
	Frecuencia máxima	11,561 MHz	Frecuencia máxima	10,8 MHz
	Potencia (5 MHz)	166,66 mW	Potencia (5 MHz)	166,60 mW
13% de error 128 palabras en la ROM	Área	2.802 SLICES	Área	2.800 SLICES
	Frecuencia máxima	11,014 MHz	Frecuencia máxima	10,686 MHz
	Potencia (5 MHz)	167,13 mW	Potencia (5 MHz)	166,99 mW

# EXPERIMENTOS Y RESULTADOS

## La clasificación de los pulsos electrocardiográficos

### Modelado en punto flotante

- ➔ **Preprocesado.** Filtrado clásico y filtrado usando la transformada Wavelet discreta.
- ➔ De cada **tipo de pulso** se tomaron 800 de la base de datos.
- ➔ Se dispuso la **mitad para entrenamiento** y la **mitad para el testeo**.
- ➔ Tras el preprocesado se tienen **77 parámetros**.
- ➔ Reducción con el **Análisis de Componentes Independientes: 15 parámetros**.

# EXPERIMENTOS Y RESULTADOS

## La clasificación de los pulsos electrocardiográficos

### Modelado en punto flotante

➔ Se **normalizaron** los parámetros:  $[-1,+1]$ .

➔ La NN final es tipo **15-30-7**.

➔ Las funciones usadas fueron tipo **tansig** en la capa intermedia y **purelin** en la salida.

**Matriz de confusión obtenida en el testeo de punto flotante:**

	N	L	R	A	V	F	/	Éxito(%)
N	400	0	0	0	0	0	0	100
L	0	400	0	0	0	0	0	100
R	1	0	394	3	2	0	0	98,50
A	4	0	6	375	3	12	0	93,75
V	0	1	0	10	380	7	2	95
F	0	0	0	9	15	374	2	93,50
/	0	0	0	0	0	1	399	99,75
Total								<b>97,21</b>

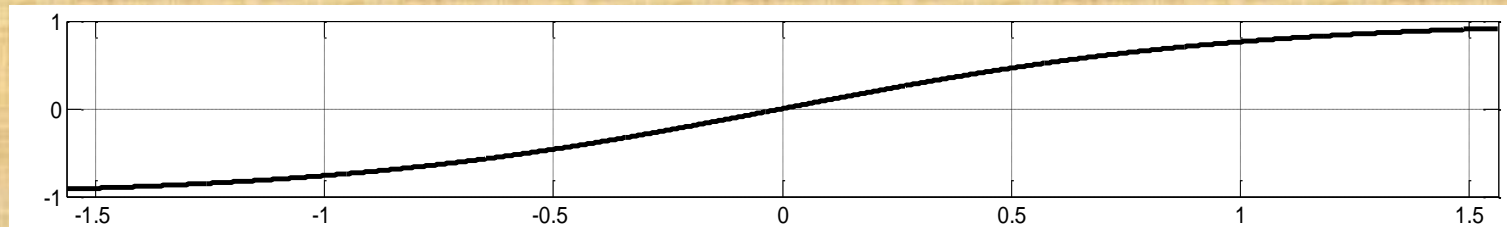
# EXPERIMENTOS Y RESULTADOS

## La clasificación de los pulsos electrocardiográficos

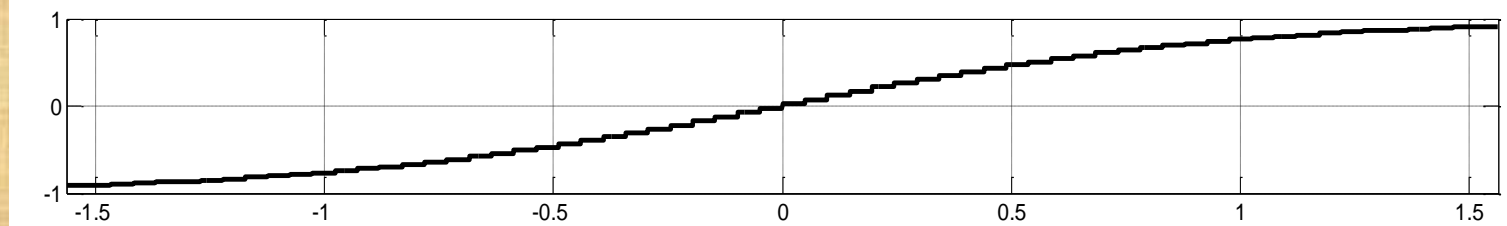
Diseño en punto fijo

Diseño con *System Generator*

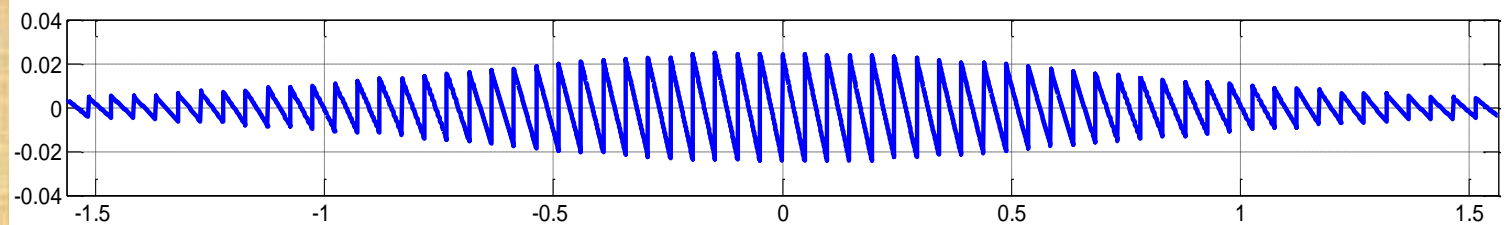
**Función *tansig***



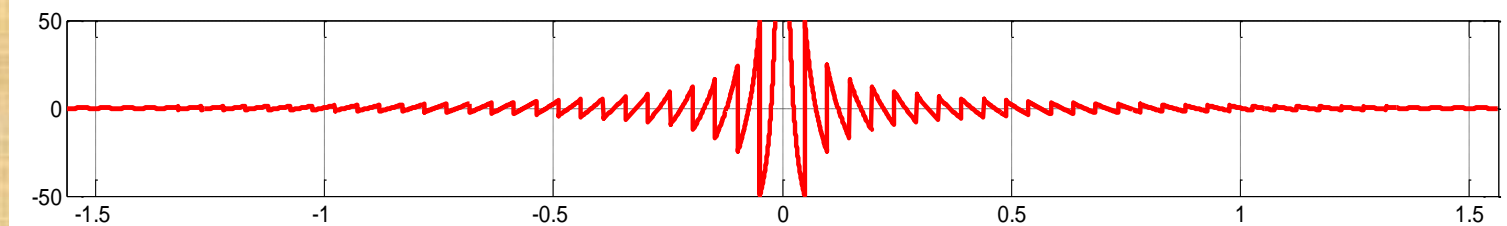
**Salida de la función implementada**



**Error**



**Error relativo**



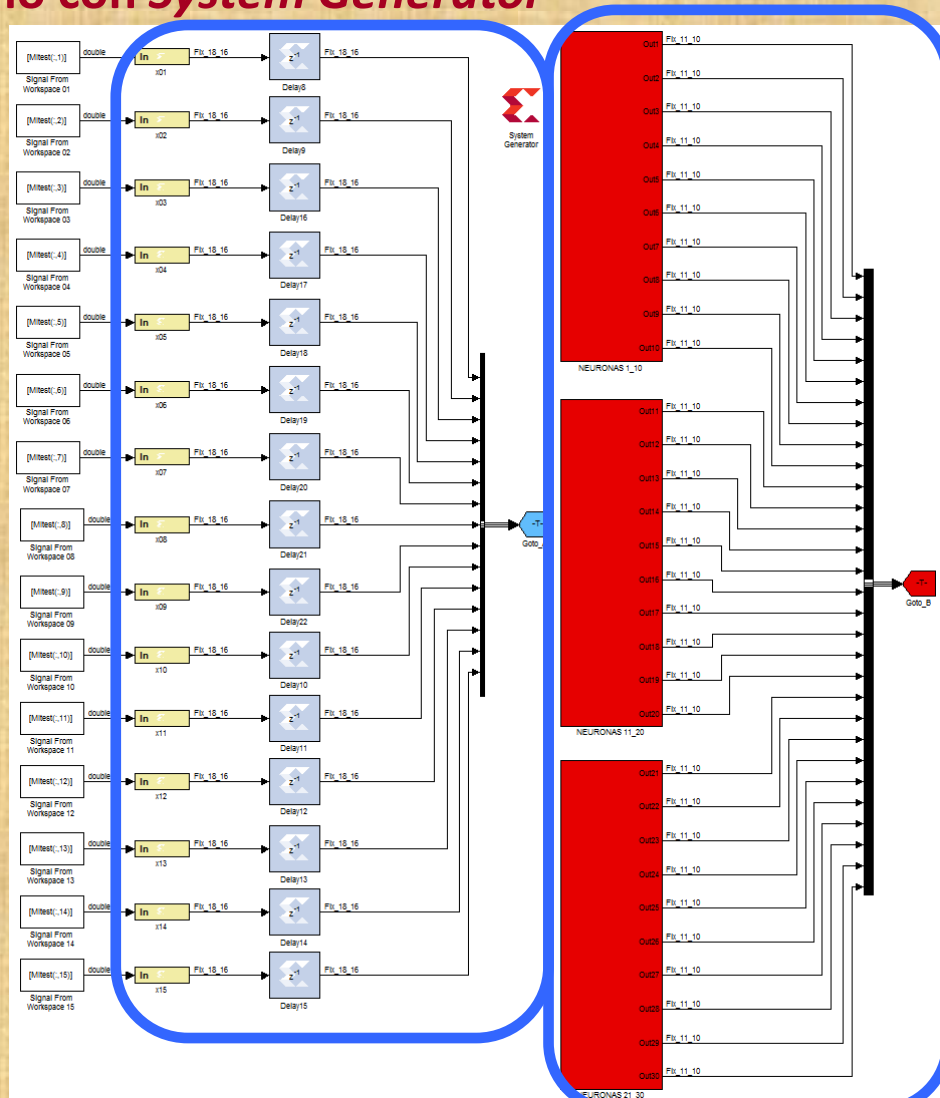


# EXPERIMENTOS Y RESULTADOS

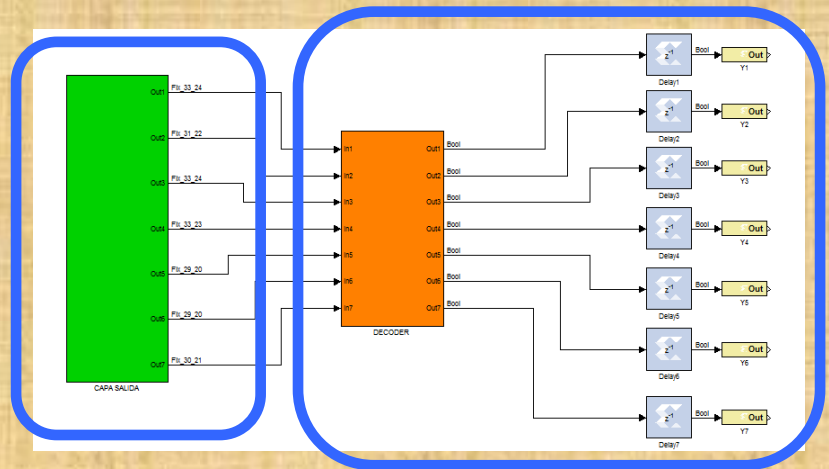
## La clasificación de los pulsos electrocardiográficos

### Diseño en punto fijo

### Diseño con *System Generator*



Diseño en *Simulink* del sistema final.





# EXPERIMENTOS Y RESULTADOS

## La clasificación de los pulsos electrocardiográficos

### Diseño en punto fijo

### Diseño con *System Generator*

Funcionalidad del sistema frente al error en la representación y el número de palabras en las ROM.

		Error en la representación (%)								
		4,00	4,10	4,20	4,30	4,40	4,50	4,60	4,70	4,80
Número de palabras en las ROM	65.536	SI	SI	SI	SI	SI	SI	SI	SI	NO
	32.768	SI	SI	SI	SI	SI	SI	SI	SI	NO
	16.384	SI	SI	SI	SI	SI	SI	SI	SI	NO
	8.192	SI	SI	SI	SI	SI	SI	SI	SI	NO
	4.096	SI	SI	SI	SI	SI	SI	SI	SI	NO
	2.048	SI	SI	SI	SI	SI	SI	SI	SI	NO
	1.024	SI	SI	SI	SI	SI	SI	SI	SI	NO
	512	SI	SI	SI	SI	SI	SI	SI	SI	NO
	256	SI	SI	SI	NO	NO	NO	NO	NO	NO
	128	SI	SI	SI	NO	NO	NO	NO	NO	NO
	64	SI	SI	SI	NO	NO	NO	NO	NO	NO
32	NO	NO	NO	NO	NO	NO	NO	NO	NO	



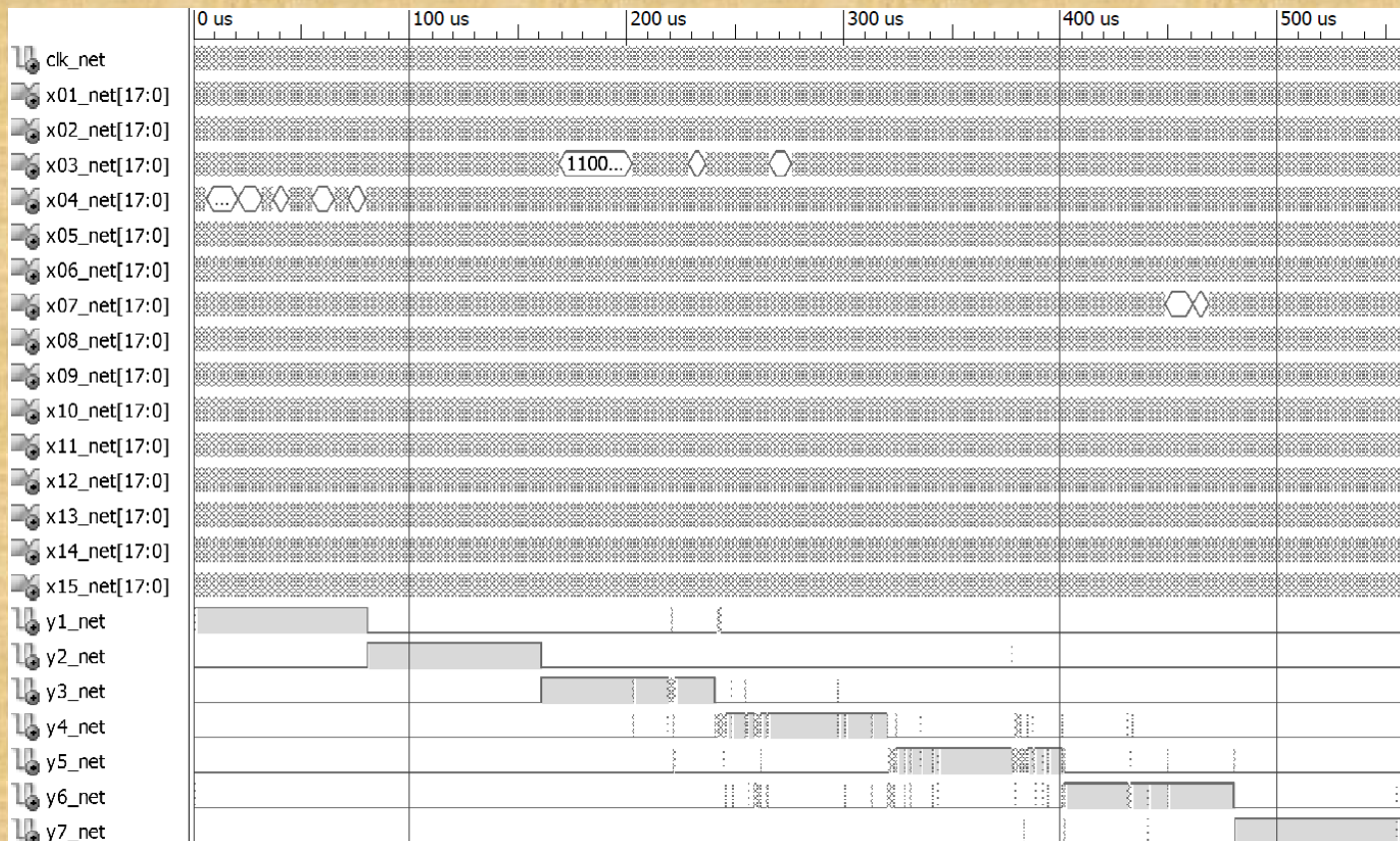
# EXPERIMENTOS Y RESULTADOS

## La clasificación de los pulsos electrocardiográficos

### Diseño en punto fijo

### Implementación con *Integrated System Environment*

Simulación para la fase final de la implementación en la FPGA para 4,2% de error y 64 palabras en la ROM.





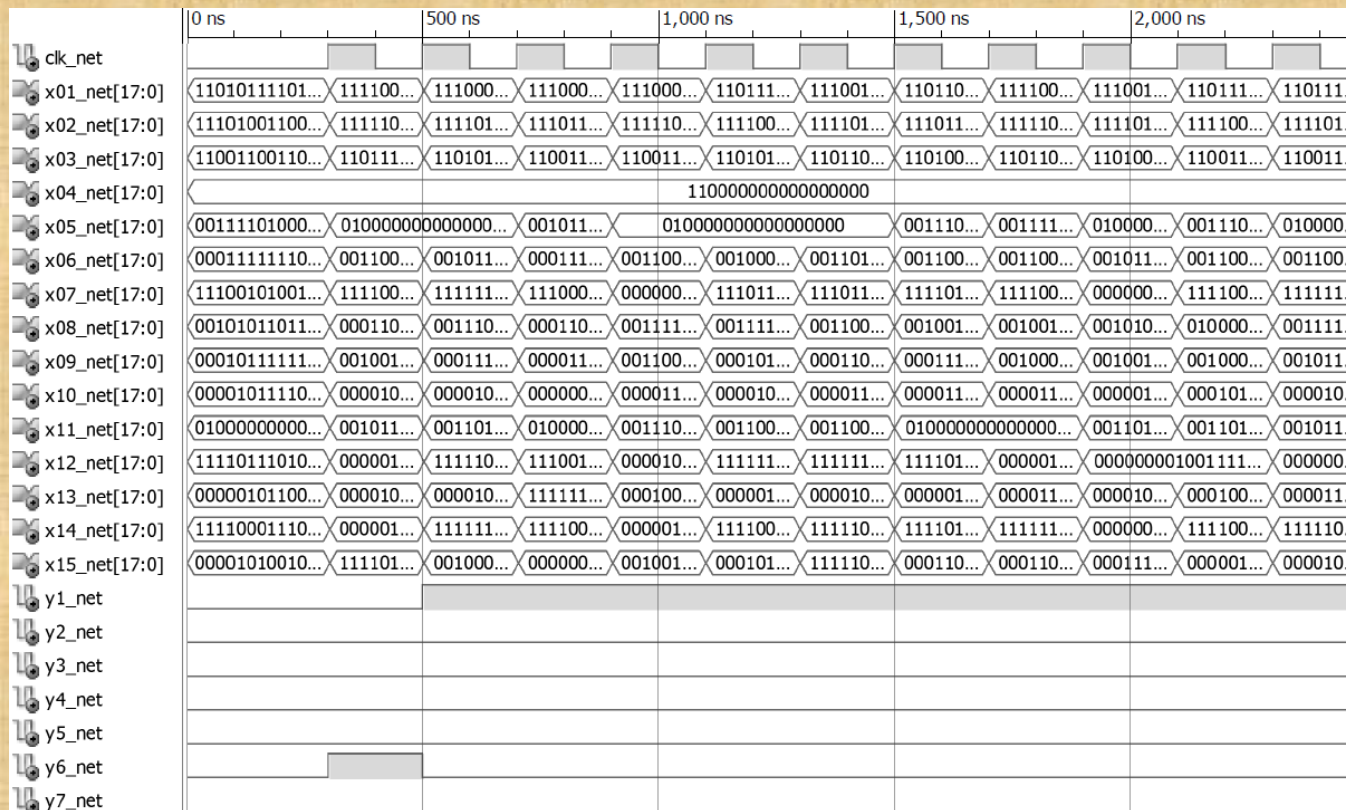
# EXPERIMENTOS Y RESULTADOS

## La clasificación de los pulsos electrocardiográficos

### Diseño en punto fijo

### Implementación con *Integrated System Environment*

Detalle de los primeros 2.500 ns de la simulación en la FPGA para 4,2% de error y 64 palabras en la ROM.



# EXPERIMENTOS Y RESULTADOS

## La clasificación de los pulsos electrocardiográficos

### Diseño en punto fijo

### Implementación con *Integrated System Environment*

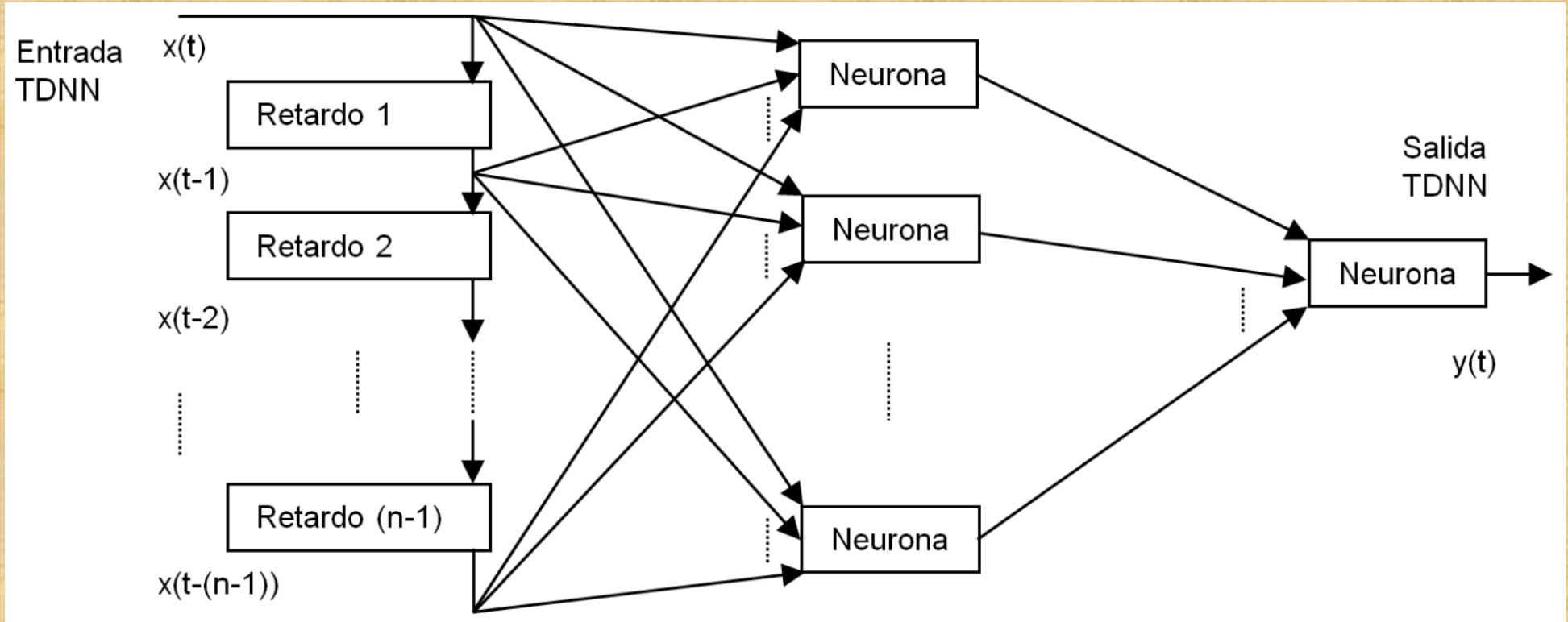
Resumen de los casos estudiados.

	VHDL		Verilog	
<b>4,2% de error</b> <b>64 palabras en la ROM</b>	<i>Área</i>	29.401 SLICES	<i>Área</i>	29.429 SLICES
	<i>Frecuencia máxima</i>	12,086 MHz	<i>Frecuencia máxima</i>	12,057 MHz
	<i>Potencia (5 MHz)</i>	2.886,64 mW	<i>Potencia (5 MHz)</i>	2.860,67 mW
<b>4,7% de error</b> <b>512 palabras en la ROM</b>	<i>Área</i>	37.840 SLICES	<i>Área</i>	37.393 SLICES
	<i>Frecuencia máxima</i>	10,852 MHz	<i>Frecuencia máxima</i>	11,560 MHz
	<i>Potencia (5 MHz)</i>	2.937,03 mW	<i>Potencia (5 MHz)</i>	2.922,70 mW

# EXPERIMENTOS Y RESULTADOS

## La predicción de temperatura Modelado en punto flotante

Red neuronal con línea de retardo.  
TDNN, *Time Delay Neural Network*.



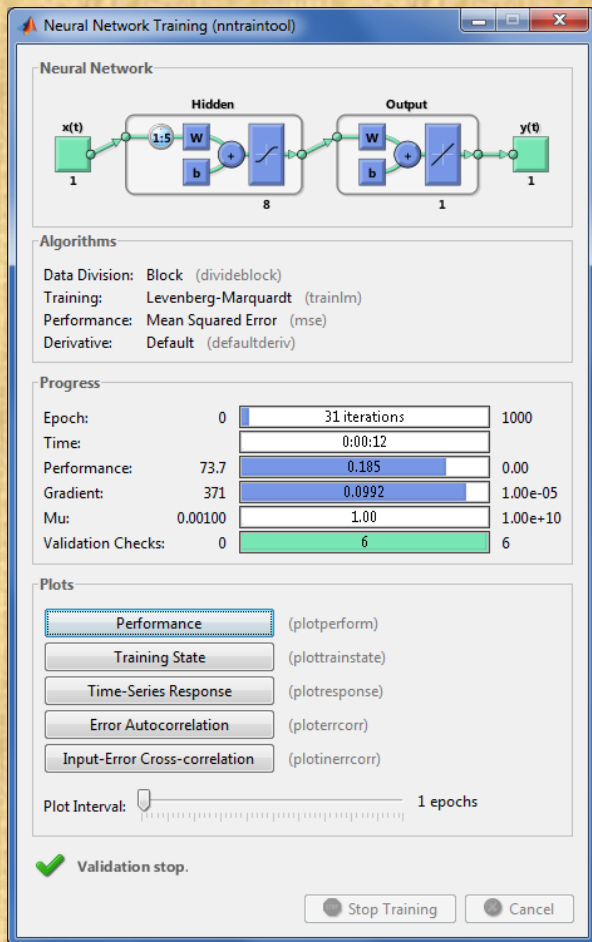
$$y(t) = f(x(t), x(t - Tm), x(t - 2Tm), \dots, x(t - (n - 1)Tm))$$

**Predictor:**  $y(t) = x(t+Tm)$

# EXPERIMENTOS Y RESULTADOS

## La predicción de temperatura Modelado en punto flotante

Ventana obtenida en el entrenamiento con el *Neural Network Time Series Tool*.



Entrenamiento con el año **2008**, testeo con el **2009**.

Elementos de **retardo**: 4.

**Capa intermedia**: 8 neuronas con funciones *tansig*.

Una neurona en la **capa de salida** con la función identidad (*purelin*).

**Error** medio:  $-0,032$  °C.

**Valor medio del error absoluto**: 0,317 °C.

**Valor medio del error al cuadrado**:  $0,210$  °C.

**Funcionalidad**: valor medio del error absoluto.

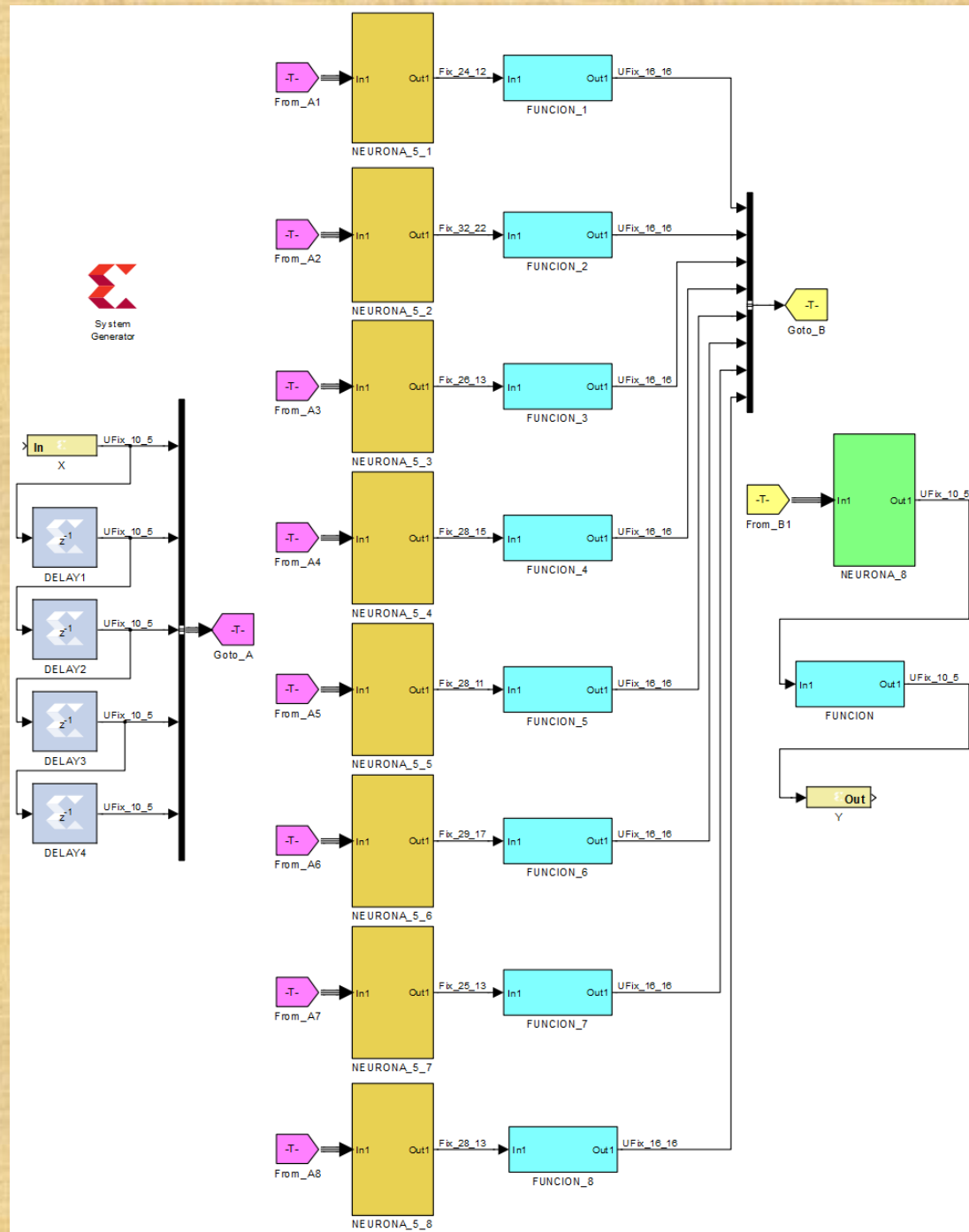


# EXPERIMENTOS Y RESULTADOS

## La predicción de temperatura

### Diseño en punto fijo

### Diseño con *System Generator*



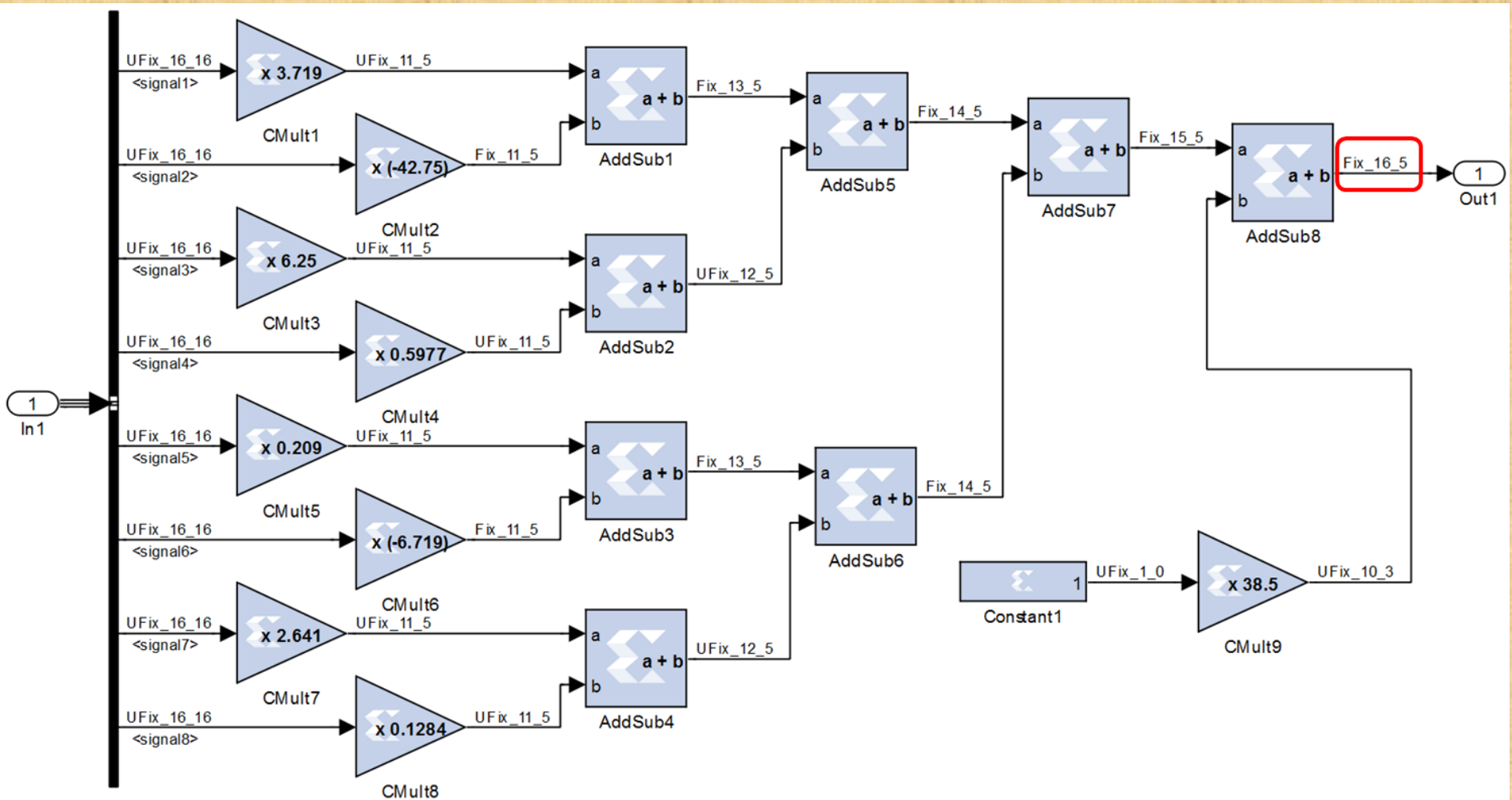
# EXPERIMENTOS Y RESULTADOS

## La predicción de temperatura

### Diseño en punto fijo

### Diseño con *System Generator*

Neurona de salida en el predictor de temperatura con **resolución completa**:



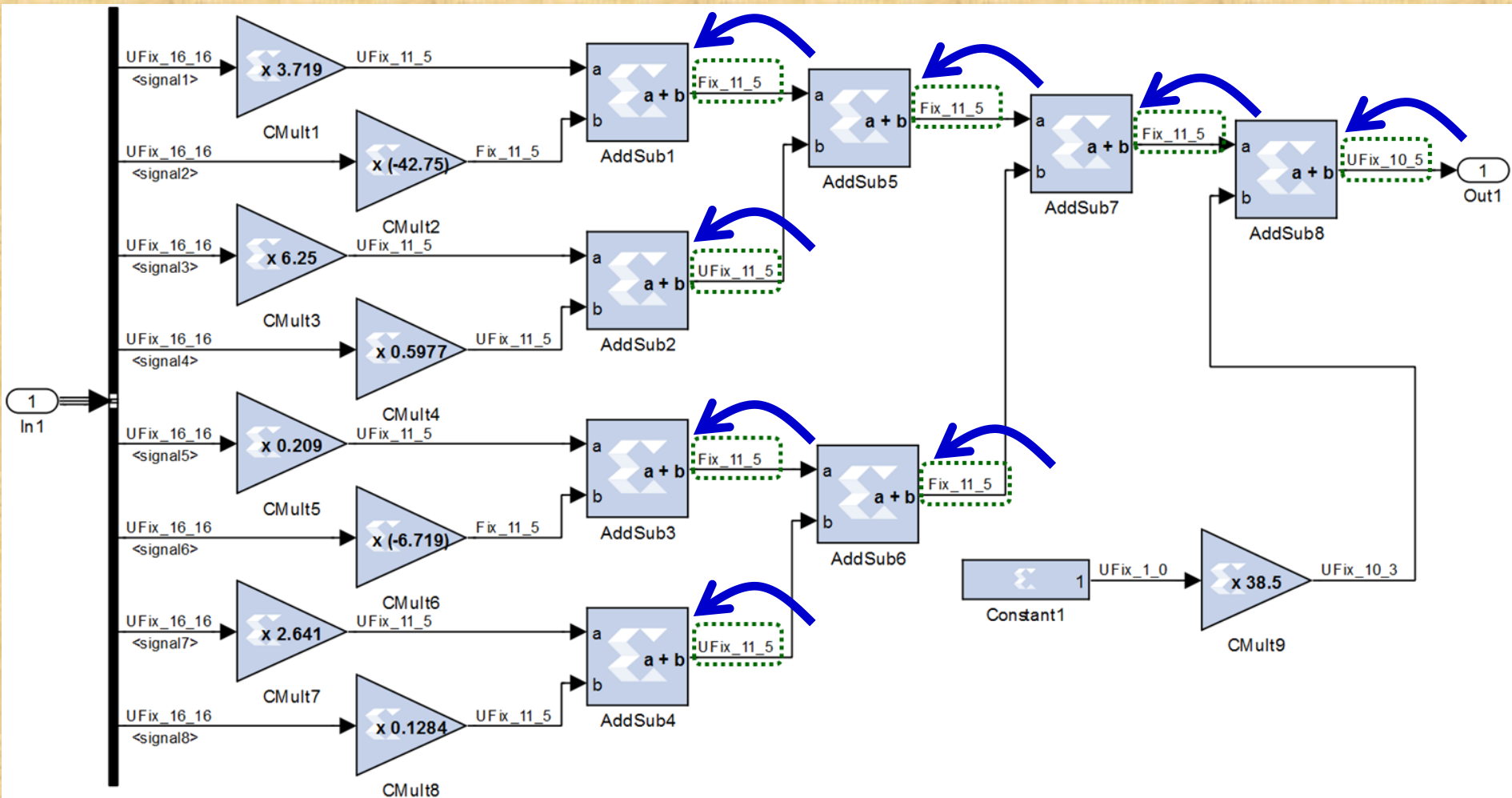
# EXPERIMENTOS Y RESULTADOS

## La predicción de temperatura

### Diseño en punto fijo

### Diseño con *System Generator*

Neurona de salida en el predictor de temperatura con **resolución ajustada**:



# EXPERIMENTOS Y RESULTADOS

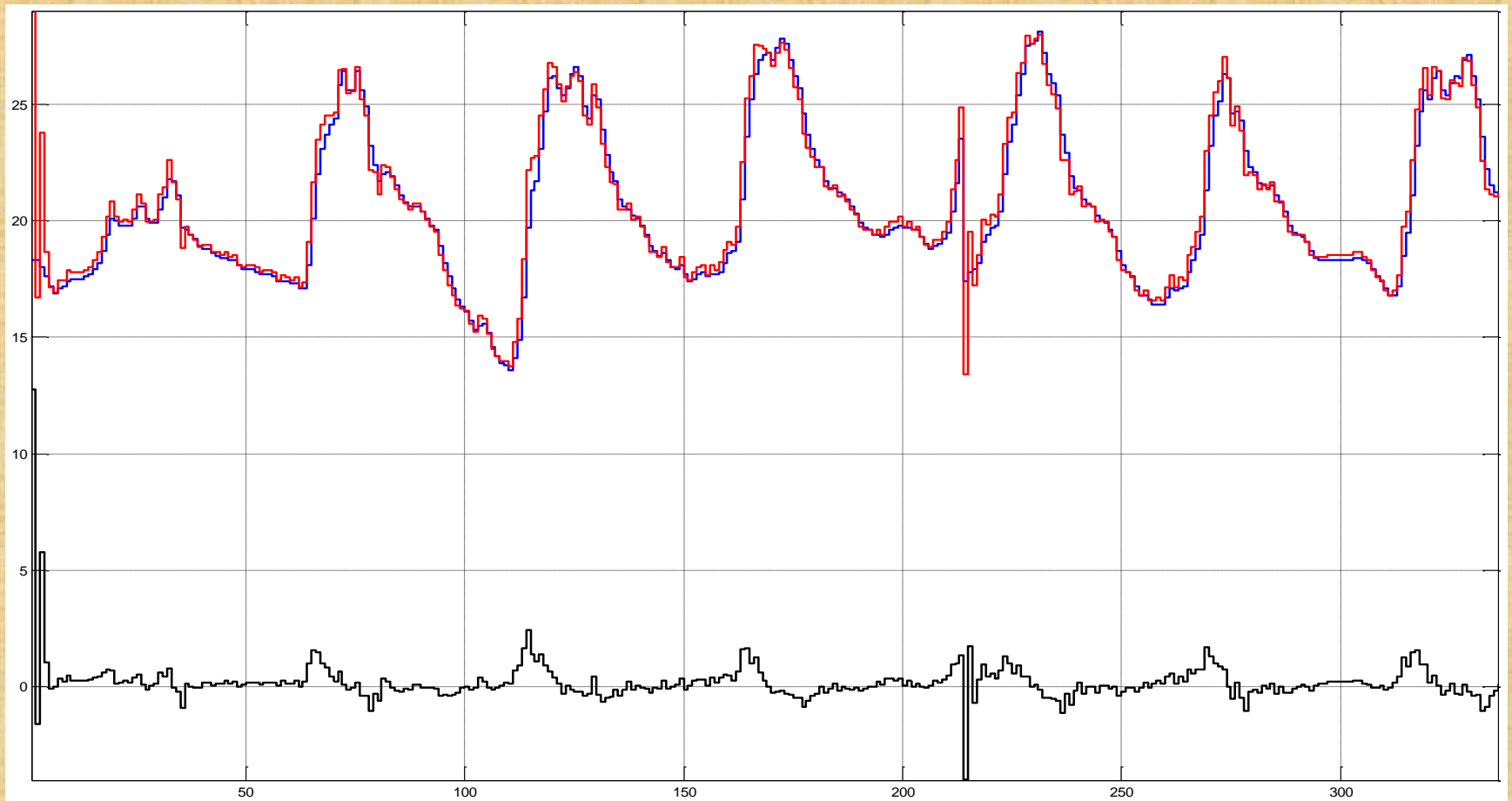
## La predicción de temperatura

Diseño en punto fijo

Diseño con *System Generator*

Simulación de 2009 con *System Generator*.

Entrada.  
Estimación  
en la salida.



Señal de error.



# EXPERIMENTOS Y RESULTADOS

## La predicción de temperatura

### Diseño en punto fijo

### Diseño con *System Generator*

**Funcionalidad del predictor** de temperatura en función del error permitido en la representación y el número de palabras en las memorias ROM.

		Error en la representación (%)												
		0,001	...	0,40	0,42	0,44	0,46	0,48	0,50	0,52	0,54	0,56	0,58	0,60
Número de palabras en las ROM	$2^{20}$	SI	...	SI	SI	SI	SI	SI	SI	SI	SI	SI	NO	NO
	...	...	...	...	...	...	...	...	...	...	...	...	...	...
	$2^{14}$	SI	...	SI	SI	SI	SI	SI	SI	SI	SI	SI	NO	NO
	$2^{13}$	SI	...	SI	SI	SI	SI	SI	SI	SI	SI	SI	NO	NO
	$2^{12}$	SI	...	SI	SI	SI	SI	SI	SI	SI	SI	SI	NO	NO
	$2^{11}$	SI	...	SI	SI	SI	SI	SI	SI	SI	SI	SI	NO	NO
	$2^{10}$	SI	...	SI	SI	SI	SI	SI	SI	SI	NO	NO	NO	NO
	$2^9$	NO	...	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO

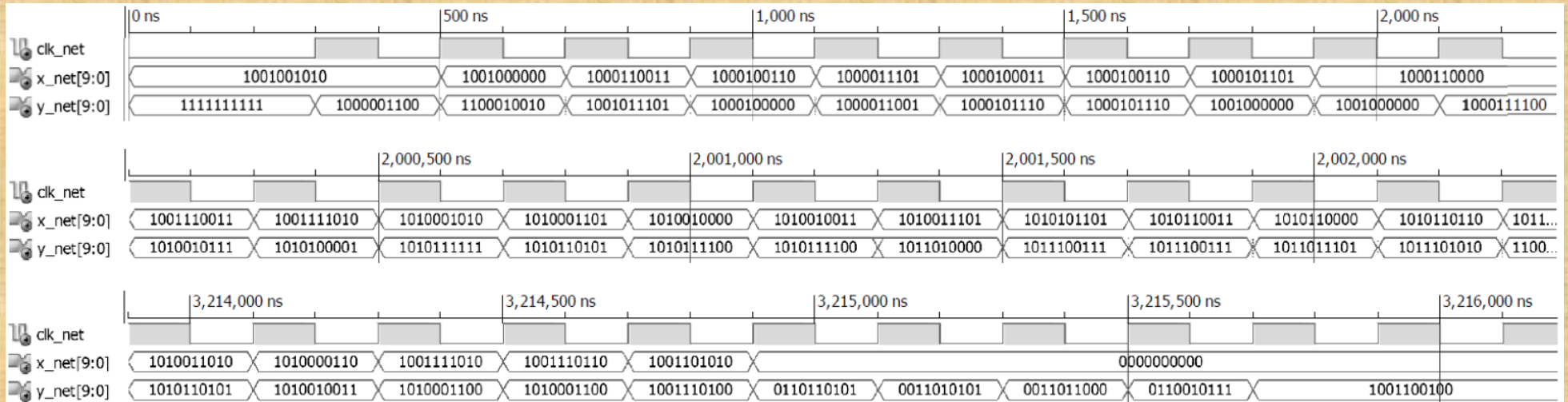
# EXPERIMENTOS Y RESULTADOS

## La predicción de temperatura

### Diseño en punto fijo

### Diseño con *Integrated System Environment*

Simulación del predictor de temperatura después del colocado y conexionado de los componentes en la FPGA para 0,52% de error y 1.024 palabras en la ROM.



# EXPERIMENTOS Y RESULTADOS

## La predicción de temperatura

### Diseño en punto fijo

### Diseño con *Integrated System Environment*

Resumen de los casos estudiados para el predictor de temperatura.

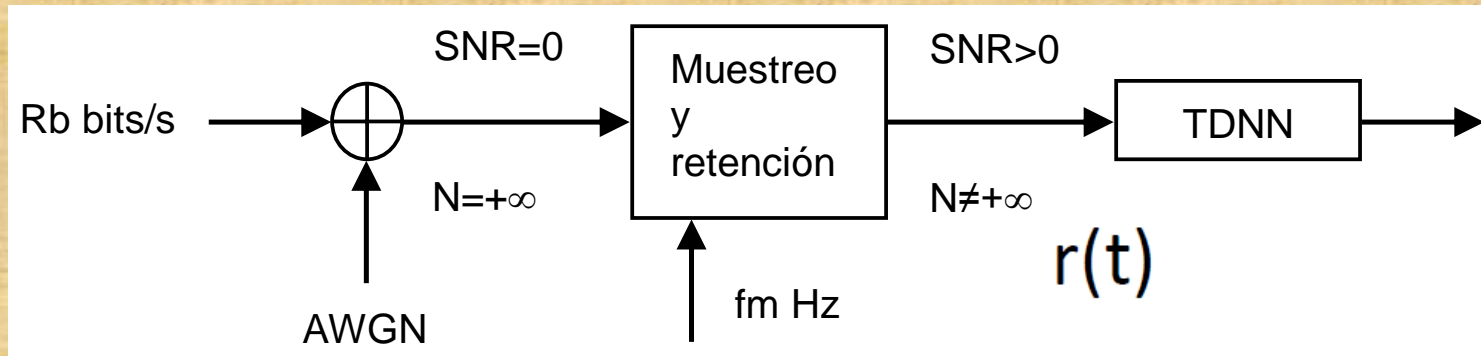
	VHDL		Verilog	
<b>0,52% de error</b> <b>1024 palabras en la ROM</b>	Área	6.228 SLICES	Área	6.239 SLICES
	Número de pines	21	Número de pines	21
	Frecuencia máxima	275,482 MHz	Frecuencia máxima	301,023 MHz
	Potencia (5 MHz)	174,45 mW	Potencia (5 MHz)	174,41 mW
<b>0,56% de error</b> <b>2048 palabras en la ROM</b>	Área	7.672 SLICES	Área	7.621 SLICES
	Número de pines	19	Número de pines	19
	Frecuencia máxima	382,409 MHz	Frecuencia máxima	358,038 MHz
	Potencia (5 MHz)	178,37 mW	Potencia (5 MHz)	177,86 mW

# EXPERIMENTOS Y RESULTADOS

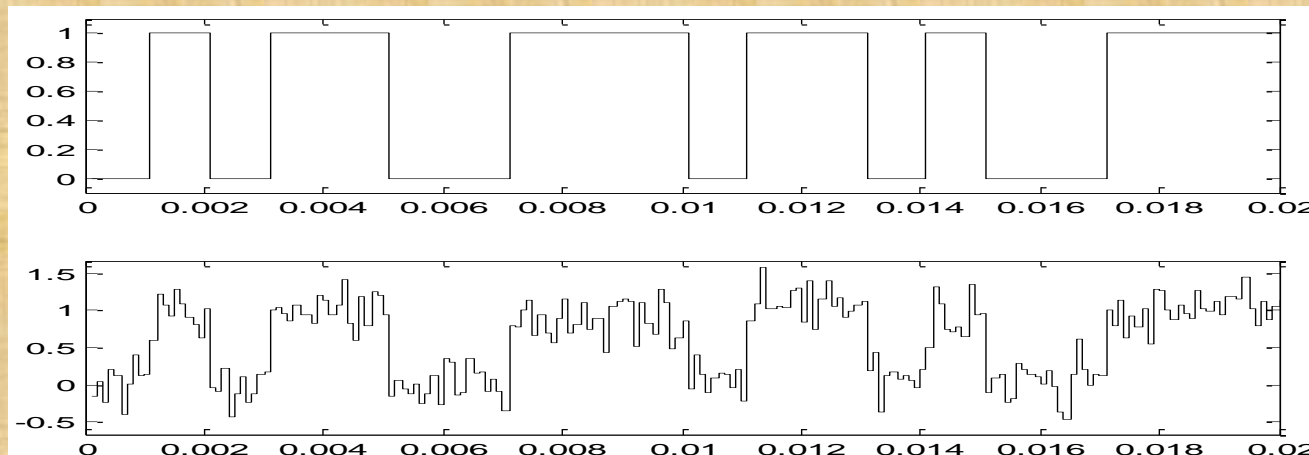
## El ecualizador para señal binaria

### Modelado en punto flotante

Modelo propuesto para el ecualizador.



Señal de datos original y señal con ruido muestreada en la entrada de la TDNN.



$d(t)$

$$r(t) = d(t) + n(t)$$



# EXPERIMENTOS Y RESULTADOS

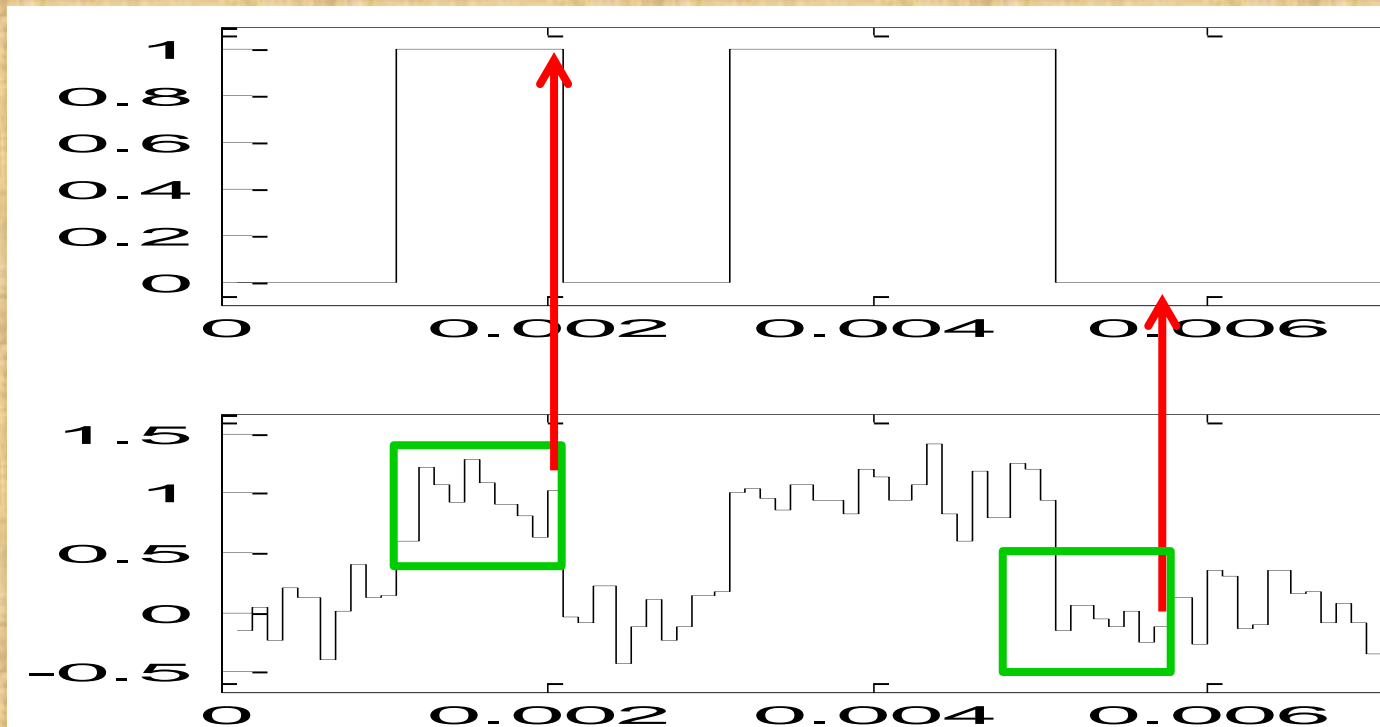
## El ecualizador para señal binaria

### Modelado en punto flotante

En un instante dado, llamado  $t_o$ , en los elementos de retardo de la red neuronal se tienen las **muestras de valor**  $r(t_o - kTm)$ , donde  $k=0, 1, 2, 3, \dots, 9$ .

El **objetivo** de la red neuronal en la salida es el valor del dato original en  $t_o$ :  $d(t_o)$

La **ventana de observación** vale  $Tb$  segundos.



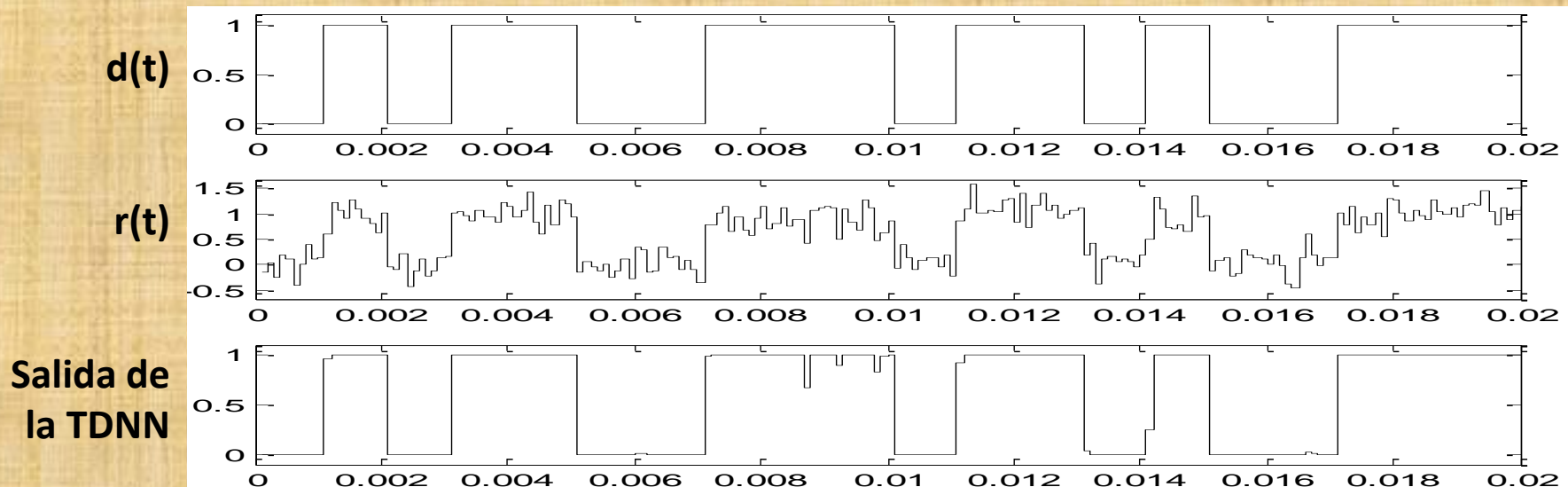
# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

### Modelado en punto flotante

Inicialmente se usó:

- **SNR entrenamiento y testeo de +10 dB;**
- se usó una capa intermedia, con cinco neuronas (tipo **10-5-1**);
- como función de transferencia se usó la función **logsig** en todas las neuronas;
- el algoritmo de entrenamiento fue el de **Levenberg-Marquardt**;
- como función de error se usó el **error cuadrático medio (Mean squared error)**.



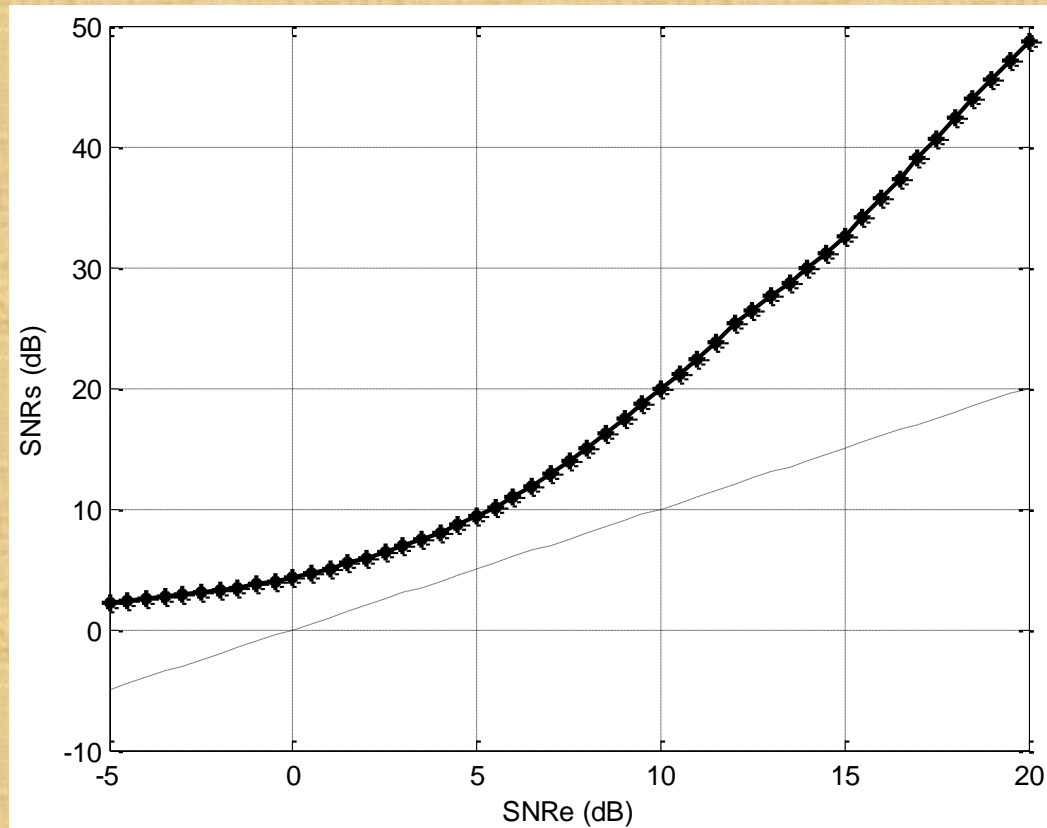
# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

### Modelado en punto flotante

Posteriormente se varió la **SNR de testeo** en la entrada de la TDNN:

- desde **-5 dB hasta +20 dB**, en incrementos de 0,5 dB.



# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

### Modelado en punto flotante

Conviene resaltar el elevado número de parámetros que intervienen en este estudio.

En el sistema puede ajustarse:

- el **número de capas intermedias**;
- el **número de neuronas** en cada capa;
- la **funciones de transferencias** usadas en cada capa;
- el **algoritmo de entrenamiento**;
- la **función de error** usada en el entrenamiento;
- la **ventana de observación**;
- la **SNR** usada en el **entrenamiento**;
- las **SNR** usadas en el **testeo**.

**¡ Una sola capa intermedia !**

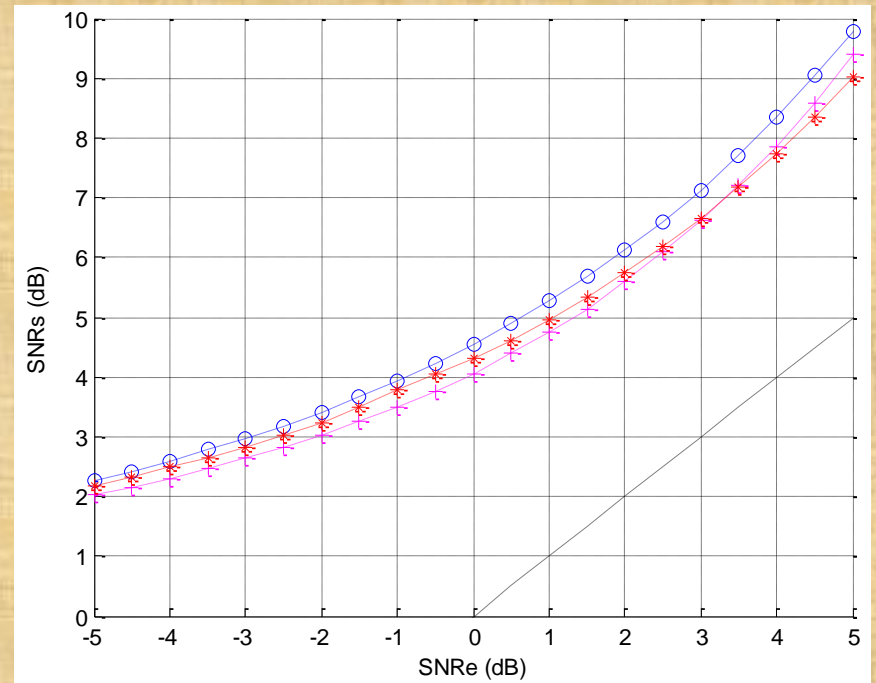
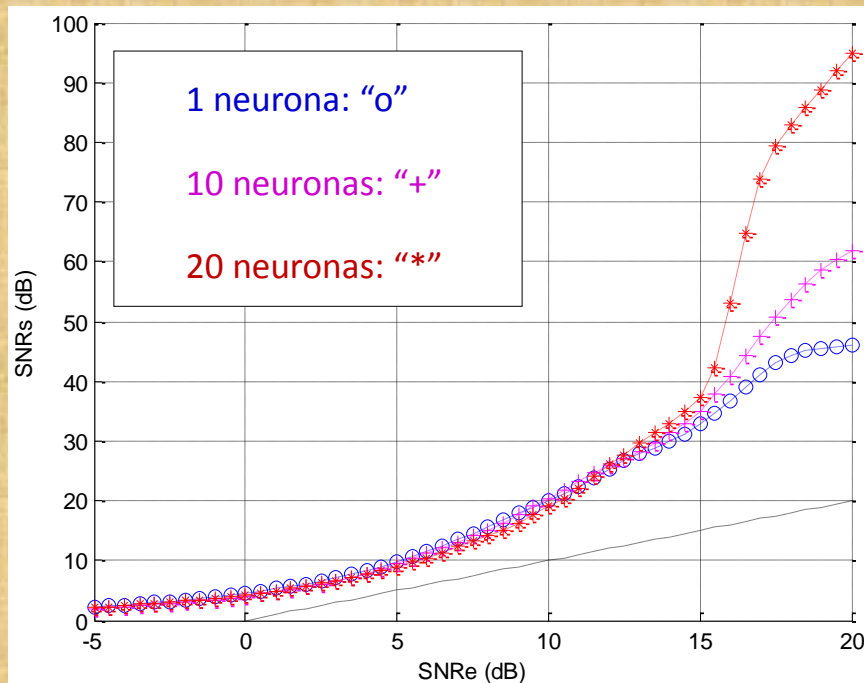


# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

### Modelado en punto flotante

Se varió el número de neuronas en la capa intermedia, entre 1 y 20.

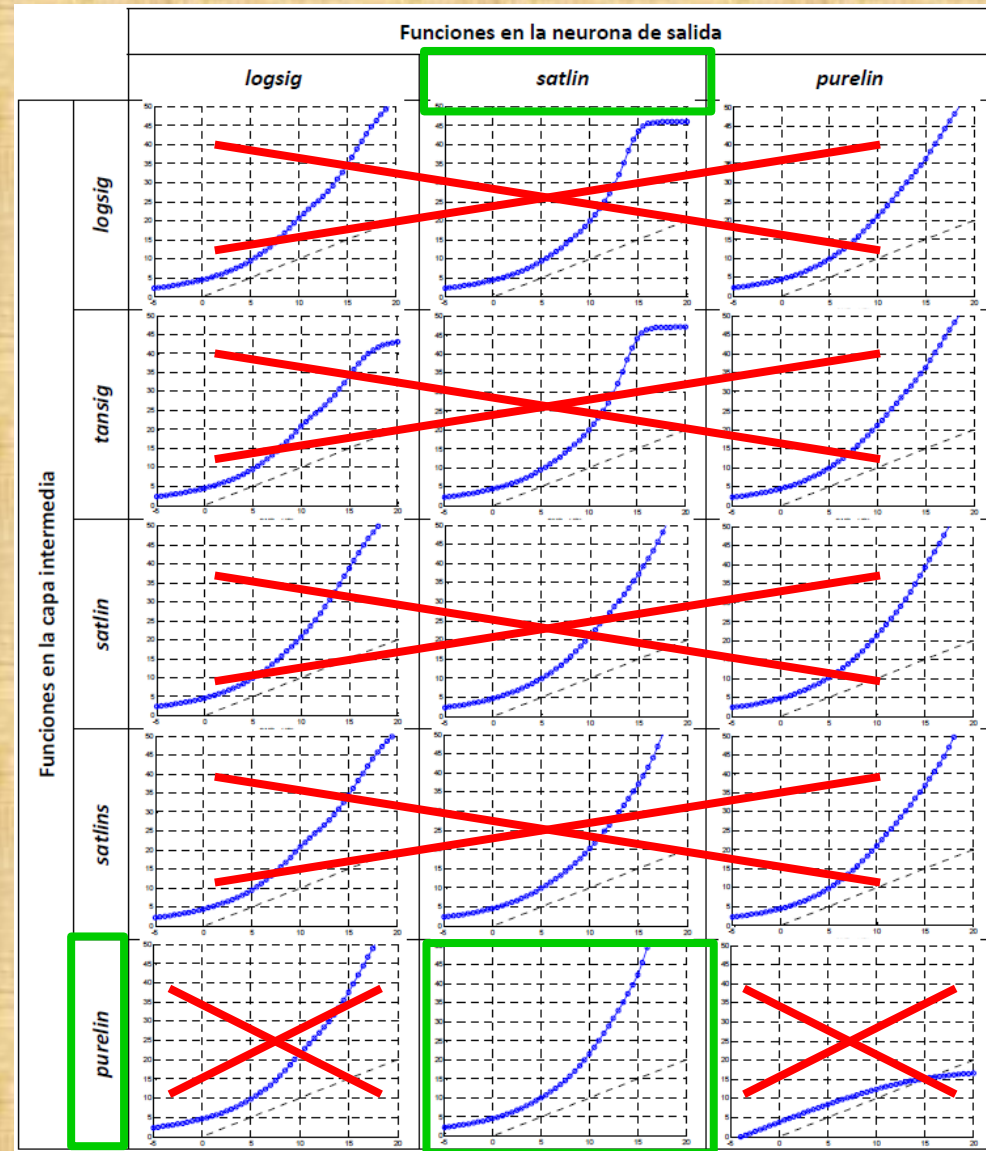
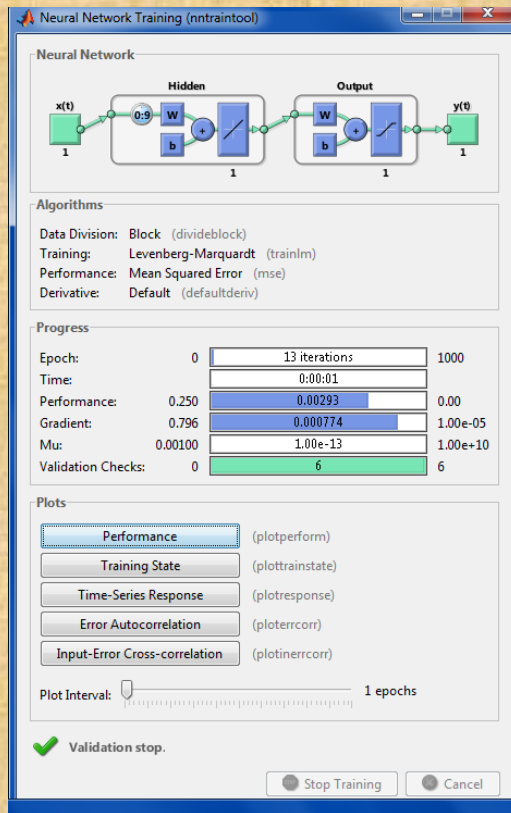


# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

### Modelado en punto flotante

Dependencia de la curva SNRs-SNRe frente a las funciones de transferencia usadas.



# EXPERIMENTOS Y RESULTADOS

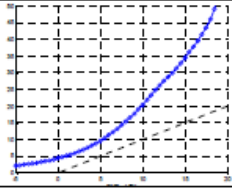
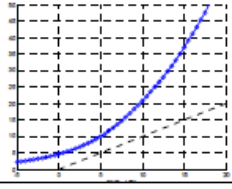
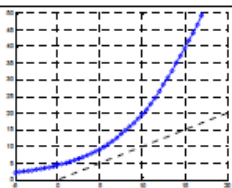
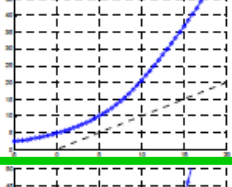
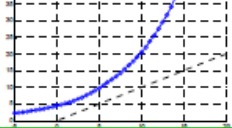
## El ecualizador para señal binaria

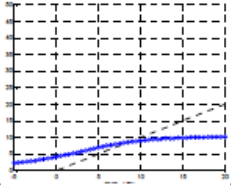
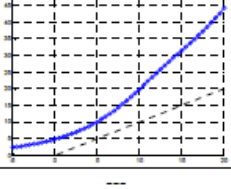
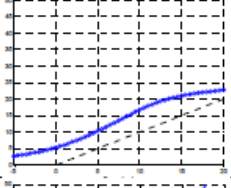
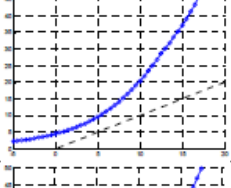
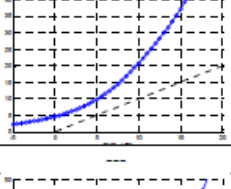
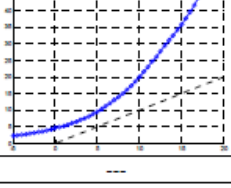
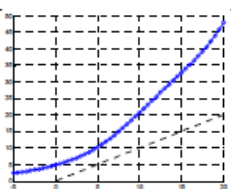
### Modelado en punto flotante

Curvas obtenidas en función del algoritmo de entrenamiento para el ecualizador.

En el futuro se fijará el algoritmo *traincgp* (Conjugate gradient backpropagation with Polak-Ribière updates):

- la **SNR obtenida** en la salida,
- el limitado **número de iteraciones**,
- y la **rapidez** de la convergencia.

ALGORITMO	Resultado	Comentarios
<i>trainb</i>	---	Entrena, no converge
<i>trainbfg</i>		Número de iteraciones: 25
<i>trainbfgc</i>	---	No entrena, produce error
<i>trainbr</i>		Número de iteraciones: 1000
<i>trainbu</i>	---	No entrena, produce error
<i>trainc</i>	---	No entrena, produce error
<i>traincgb</i>		Número de iteraciones: 22
<i>traincgf</i>		Tarda 1 segundo Número de iteraciones: 17
<i>traincgp</i>		Tarda 1 segundo Número de iteraciones: 23

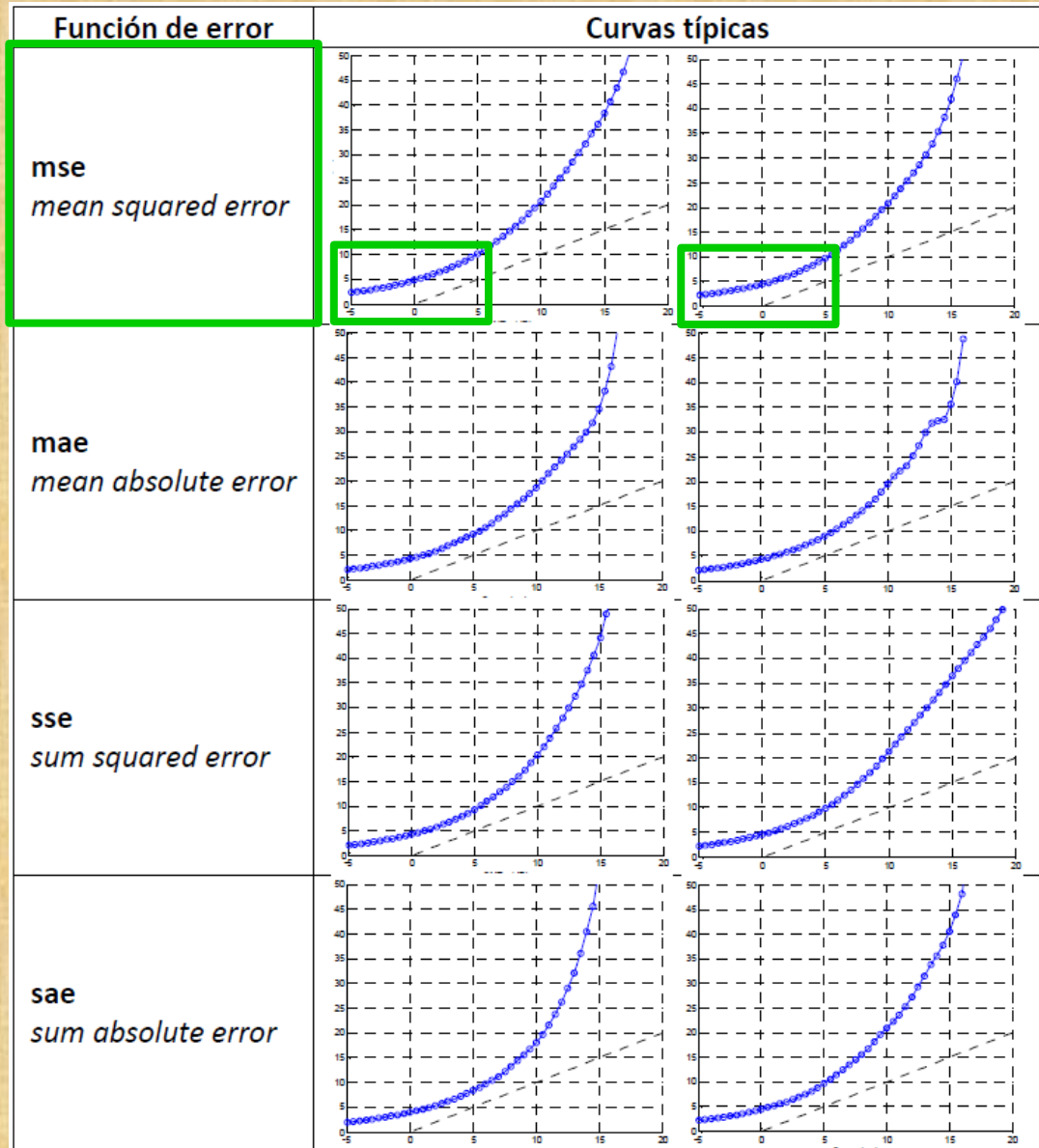
<i>traingd</i>		Número de iteraciones: 1000
<i>traingda</i>		Número de iteraciones: 167
<i>traingdm</i>	---	Entrena, no converge
<i>traingdx</i>		Número de iteraciones: 110
<i>trainlm</i>		Número de iteraciones: 14
<i>trainoss</i>		Número de iteraciones: 26
<i>trainr</i>	---	No entrena, produce error
<i>trainrp</i>		Número de iteraciones: 162
<i>trainru</i>	---	Entrena, no converge
<i>trains</i>	---	Entrena, no converge
<i>trainscg</i>		Número de iteraciones: 28

# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

### Modelado en punto flotante

Curvas obtenidas dependiendo de la función de error.





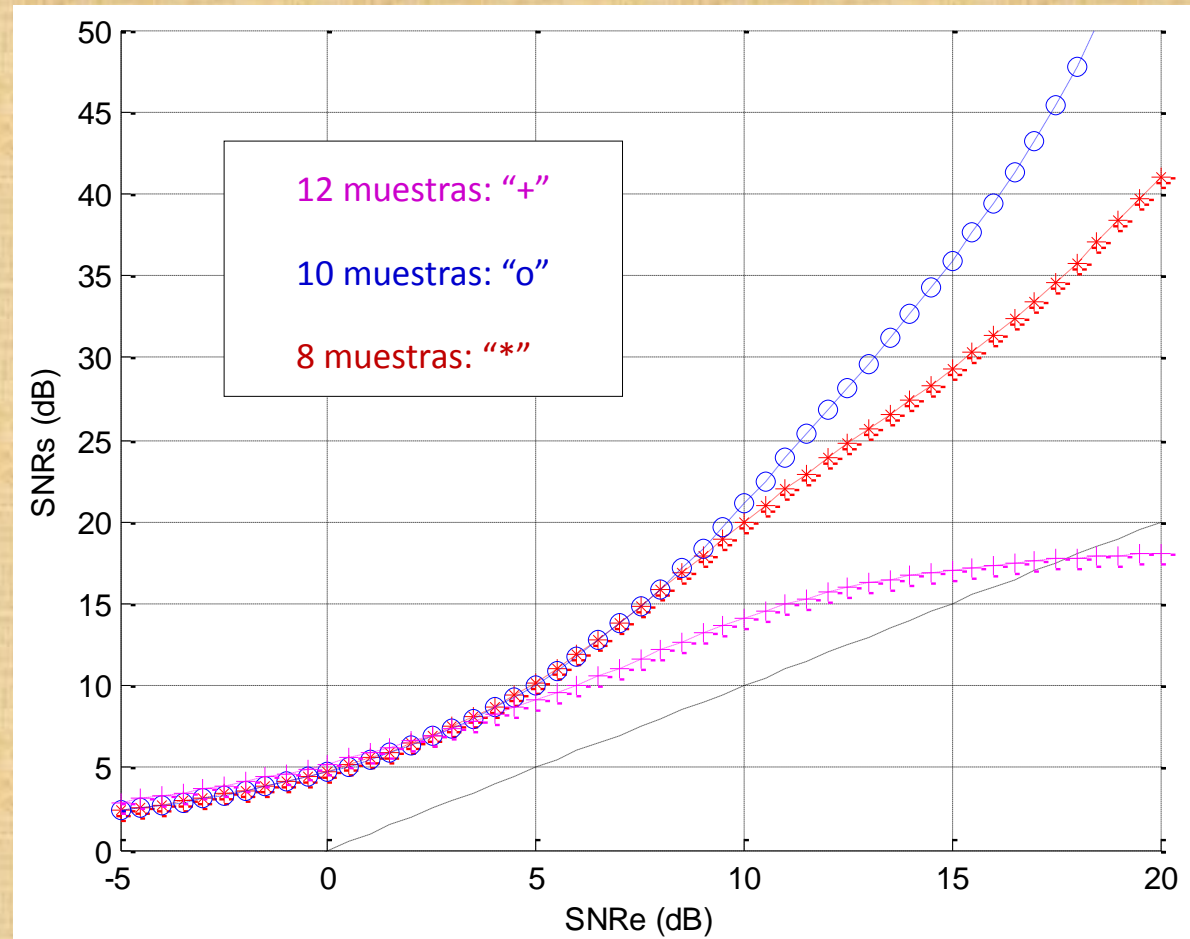
# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

### Modelado en punto flotante

Efecto del tamaño de la **ventana de observación**:

- 10 muestras por bit.
- SNR de entrenamiento: +10 dB.
- Se varía el número de muestras entre 1 y 20.



# EXPERIMENTOS Y RESULTADOS

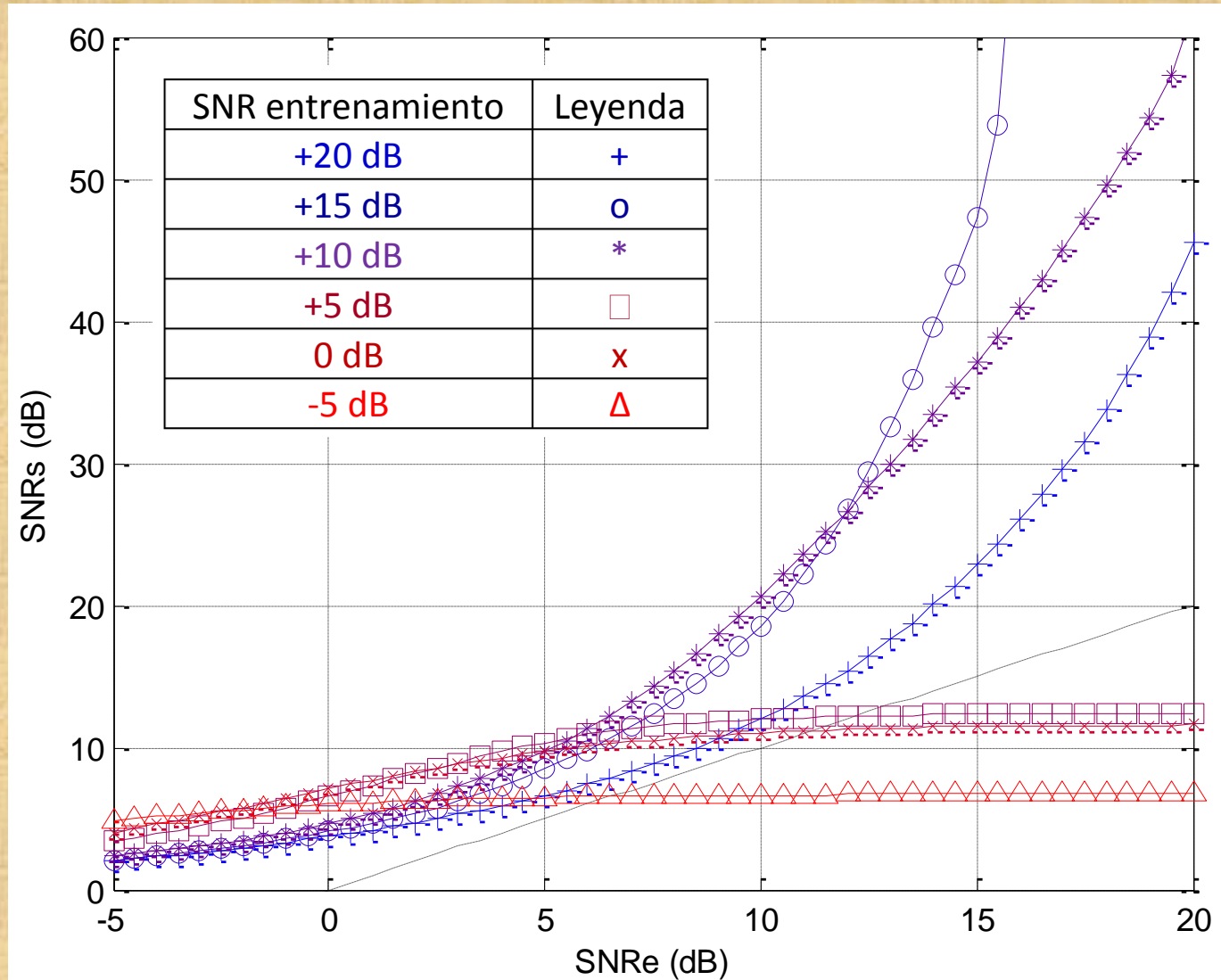
## El ecualizador para señal binaria

### Modelado en punto flotante

#### Efecto de la SNR en el entrenamiento.

SNR de entrenamiento:  
-5 hasta +20 dB,  
en pasos de 1 dB.

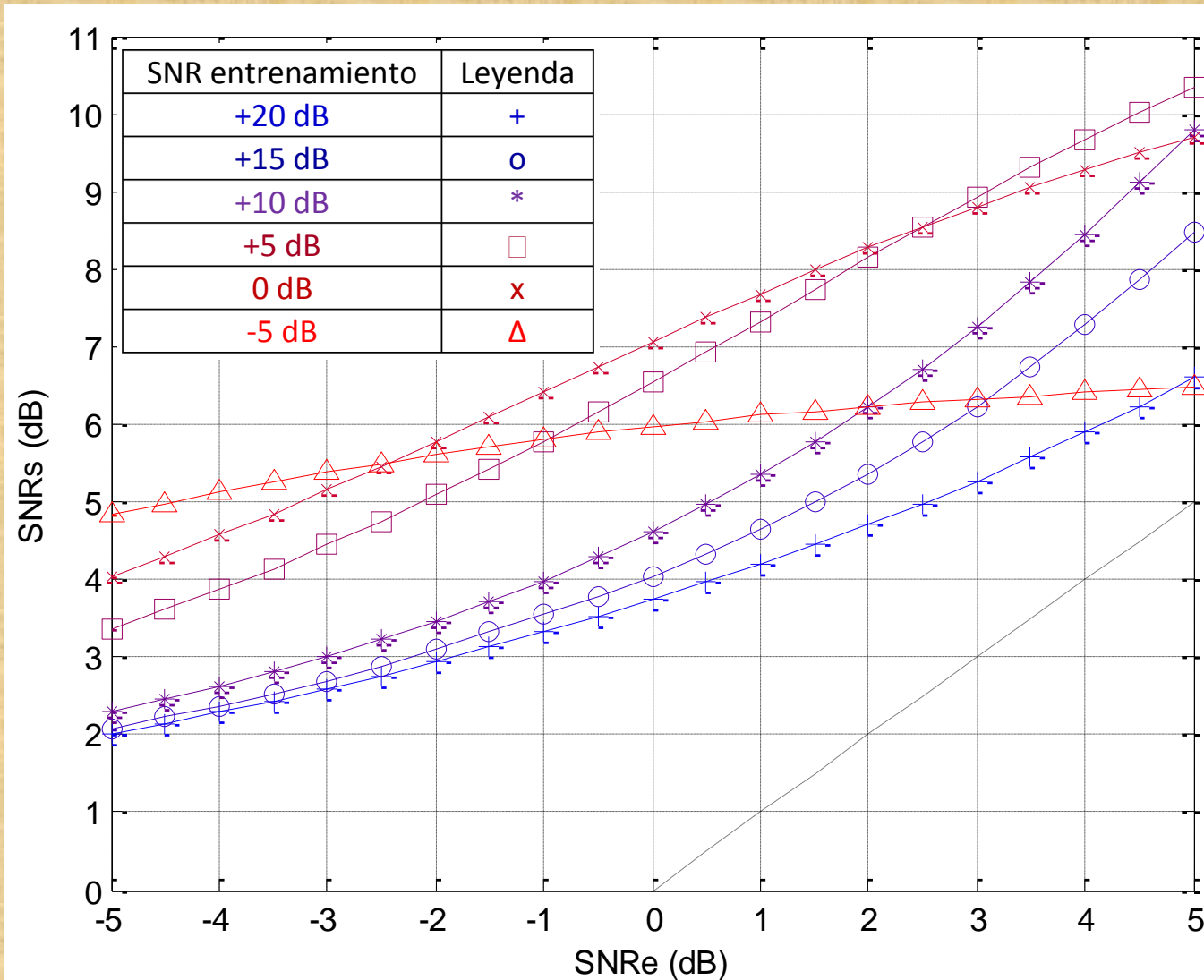
SNR de testeo:  
-5 hasta +20 dB,  
en pasos de 0,5 dB.



# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

### Modelado en punto flotante

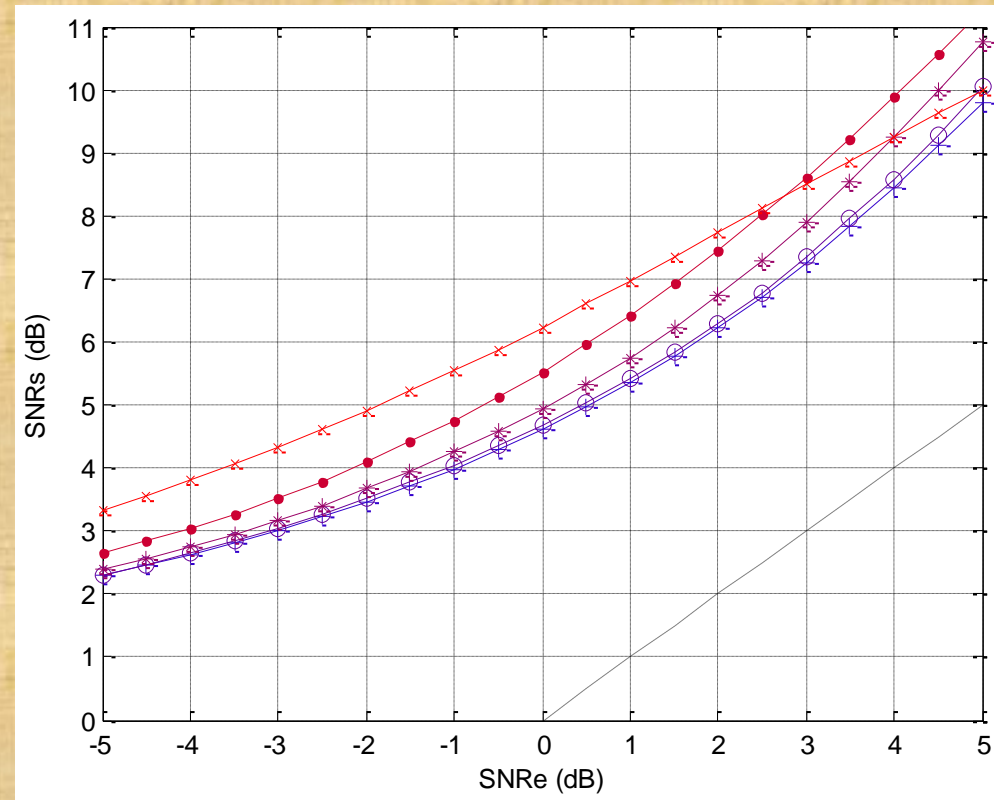
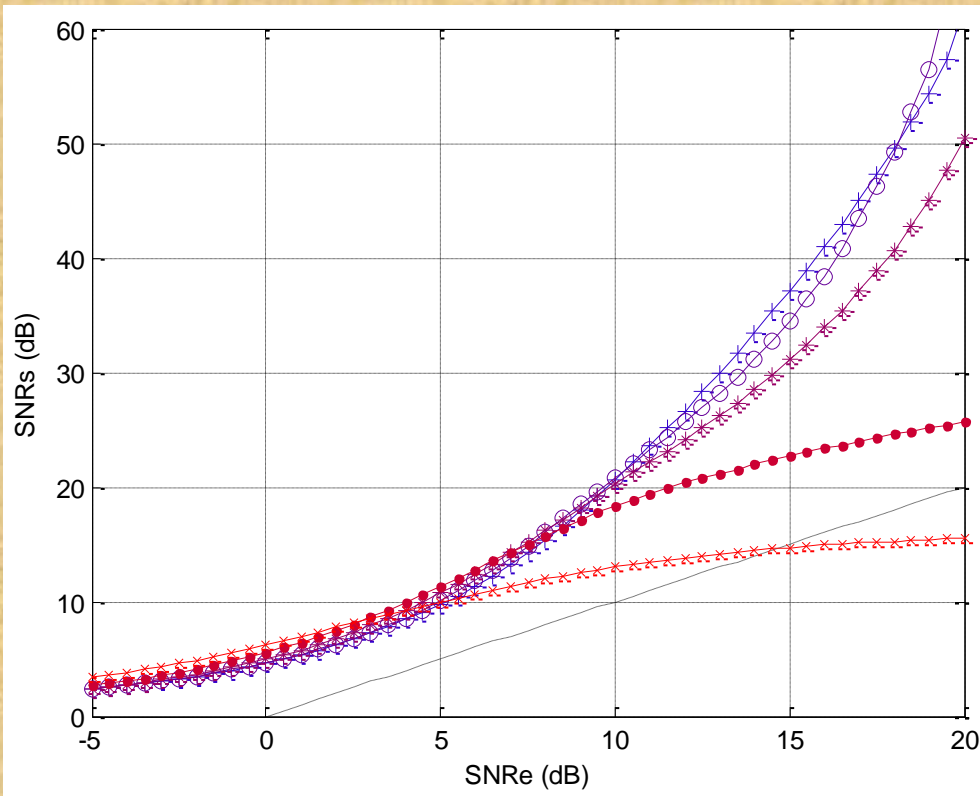


# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

### Modelado en punto flotante

SNR entrenamiento	Legenda
+10 dB	+
+9 dB	o
+8 dB	*
+7 dB	·
+6 dB	x



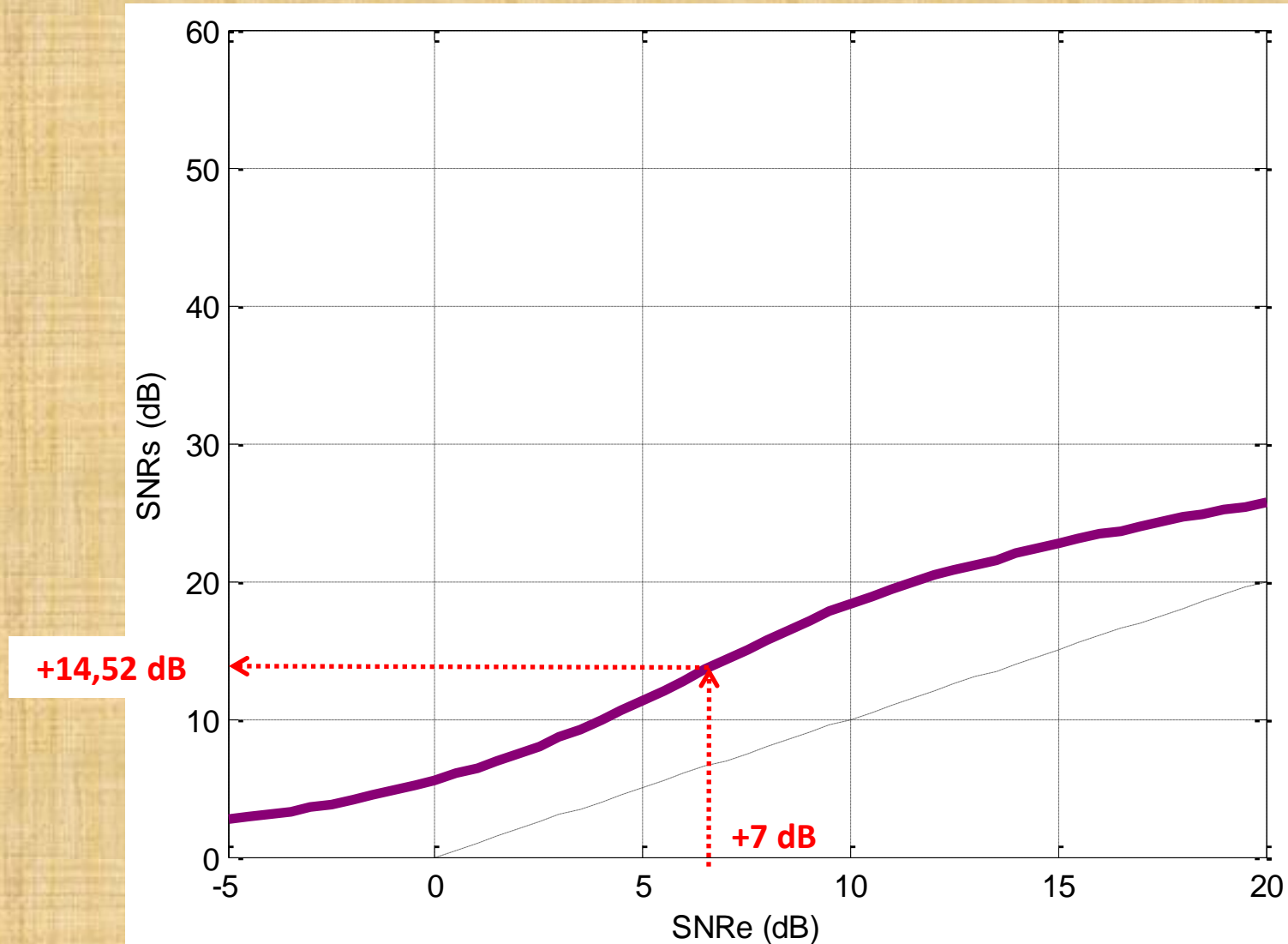


# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

### Modelado en punto flotante

Curva para una SNR óptima de entrenamiento de +7 dB.



# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

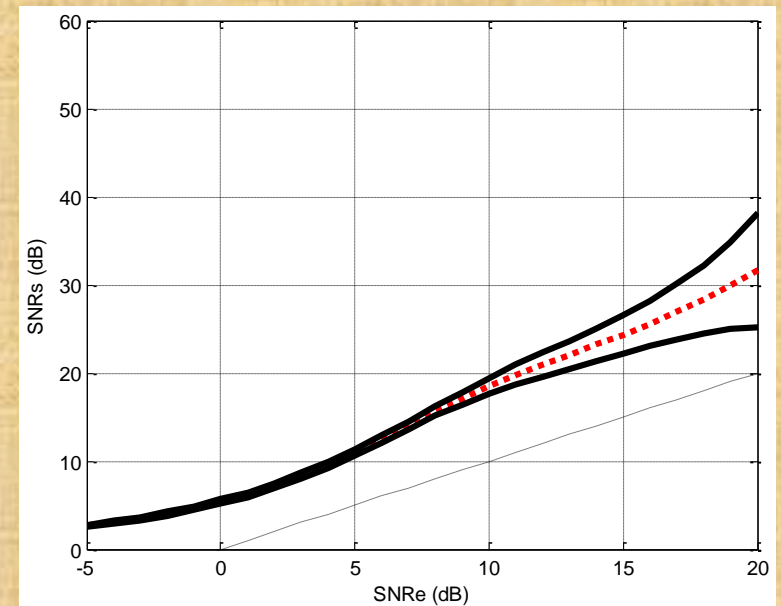
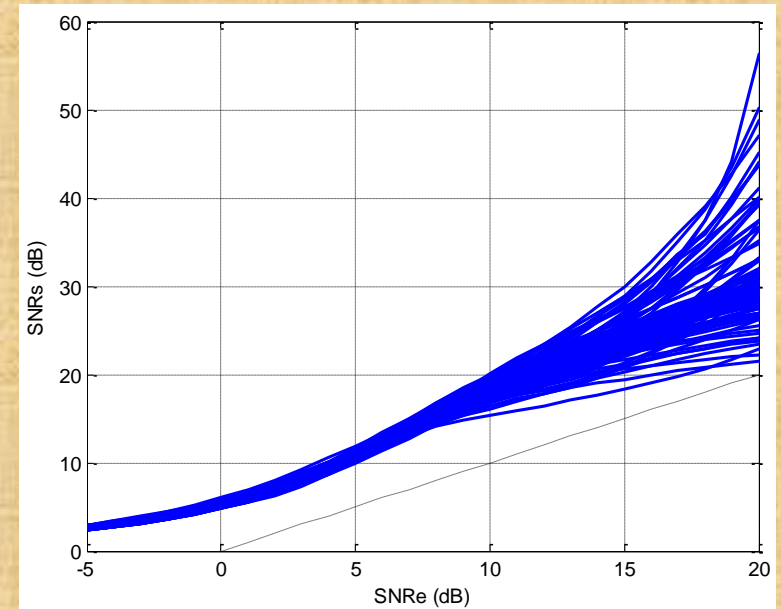
### Modelado en punto flotante

Se ha llegado a la arquitectura de la TDNN.

Cada vez que se entrena se obtiene una TDNN con distintos coeficientes y curvas.

**Banda de estabilidad:** zona donde que encajan las curvas.

Dentro de la banda de estabilidad cayó el 76% de los entrenamientos válidos.



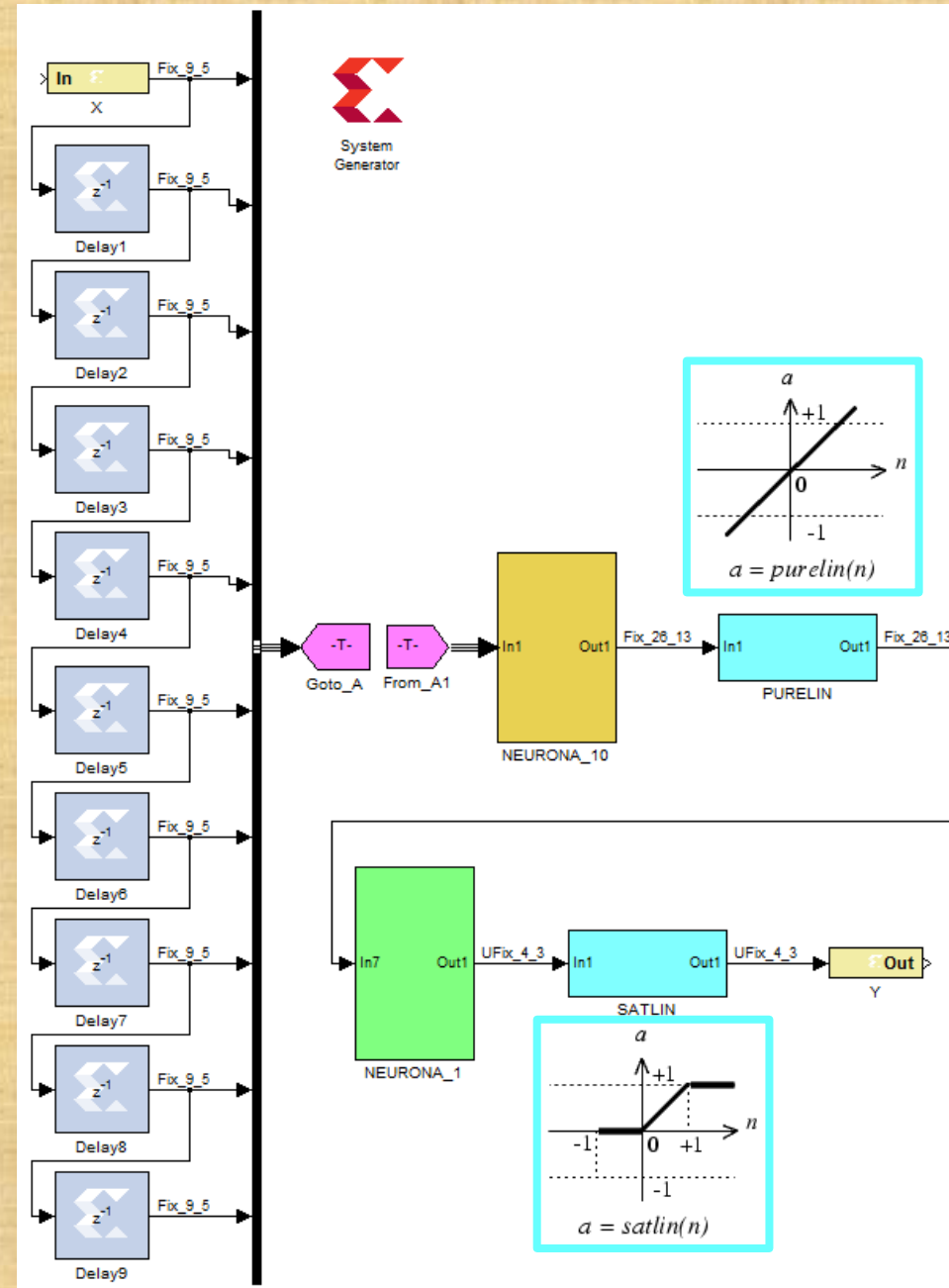
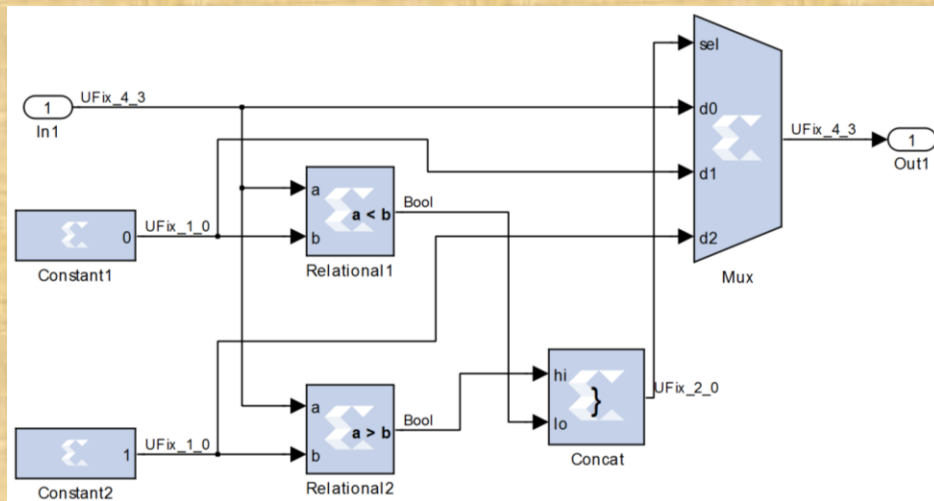
# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

### Diseño en punto fijo

### Diseño con *System Generator*

Diseño de la función *satlin* con el uso de un multiplexor.



# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

### Diseño en punto fijo

#### Diseño con *System Generator*

Los valores de la variable  $n$ , con función de densidad de probabilidad gaussiana, se encuentran en torno a la media:

$$\Pr ( \mu - 3\sigma \leq n \leq \mu + 3\sigma ) = 0,997$$

En el caso peor:

- la SNR es de -5 dB
- la varianza y potencia de ruido es 1,58
- la desviación típica del ruido es 1,26

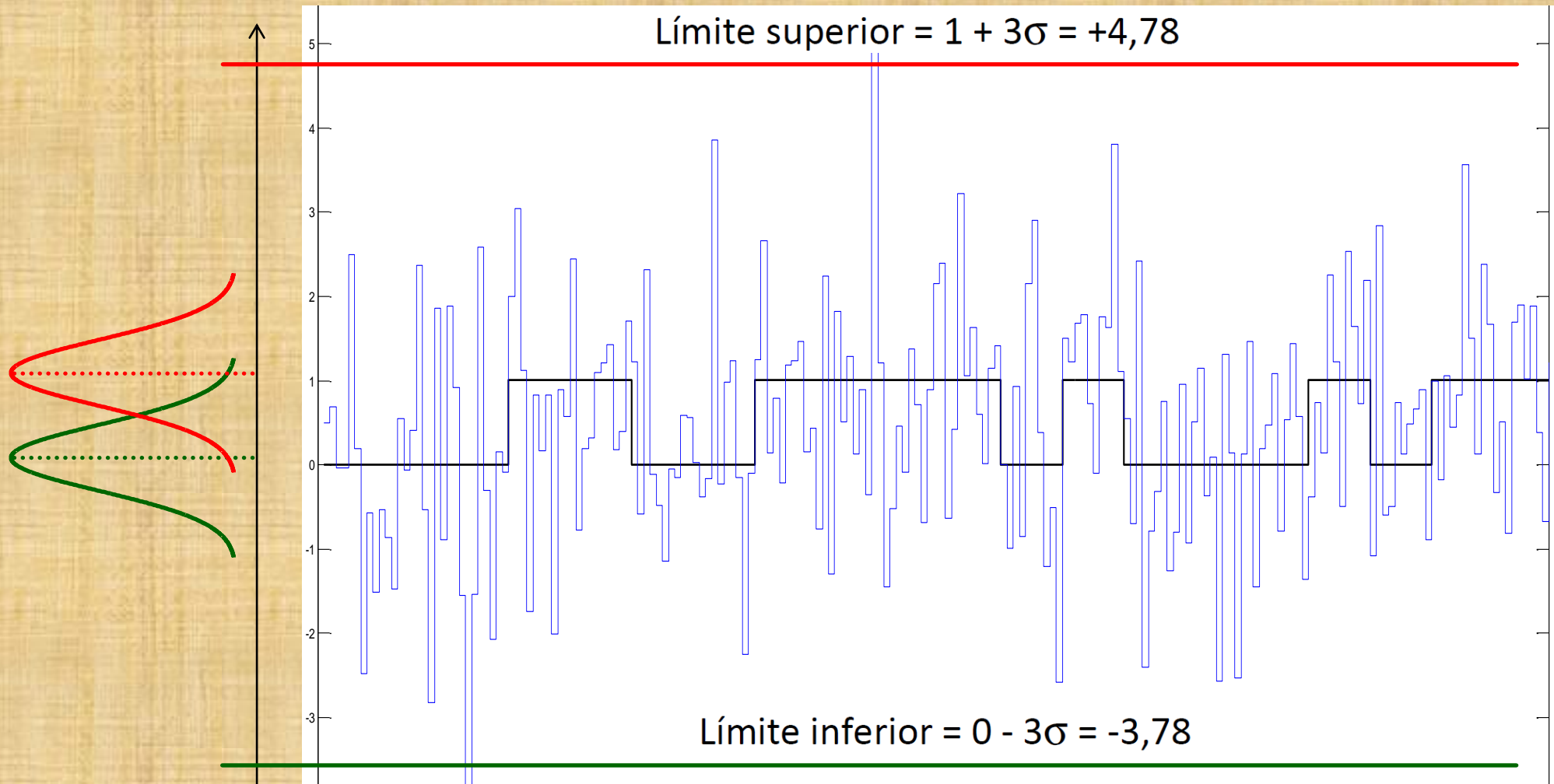


# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

Diseño en punto fijo

Diseño con *System Generator*



# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

### Diseño en punto fijo

#### Diseño con *System Generator*

Para establecer el **número de bits fraccionarios** (*nbf*) de la señal de **entrada**, el intervalo de cuantificación ( $\Delta$ ) se fijó como una fracción (*p*) del valor de pico a pico de la señal de entrada.

$$\Delta = 2^{-nbf} \leq p(A-0)$$

A es igual a 1, y el error máximo de cuantificación es  $nq = \Delta/2$ .

$$nbf \geq -\log_2[p(A-0)]$$

# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

Diseño en punto fijo

Diseño con *System Generator*

El error máximo debe fijarse con algún **criterio**.

Por ejemplo, que la curva obtenida en punto fijo se desvíe **menos de 1 dB** respecto a la de punto flotante.

<b>Error</b>	<b>4,0%</b>	SNRs punto fijo: * SNRs punto flotante: o	SNRs punto fijo - SNRs punto flotante
SNRs (SNRe=+7 dB)	14,47		
Caso peor	-1,01		
Formato de entrada	Fix_9_5		
Formato de salida	Fix_34_19		
Resolución en la salida	Completa		
<b>Error</b>	<b>3,9%</b>	SNRs punto fijo: * SNRs punto flotante: o	SNRs punto fijo - SNRs punto flotante
SNRs (SNRe=+7 dB)	14,46		
Caso peor	-0,71 dB		
Formato de entrada	Fix_9_5		
Formato de salida	Fix_35_20		
Resolución en la salida	Completa		
<b>Error</b>	<b>3,9%</b>	SNRs punto fijo: * SNRs punto flotante: o	SNRs punto fijo - SNRs punto flotante
SNRs (SNRe=+7 dB)	14,47		
Caso peor	-0,91		
Formato de entrada	Fix_9_5		
Formato de salida	UFix_4_3		
Resolución en la salida	Ajustada		

# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

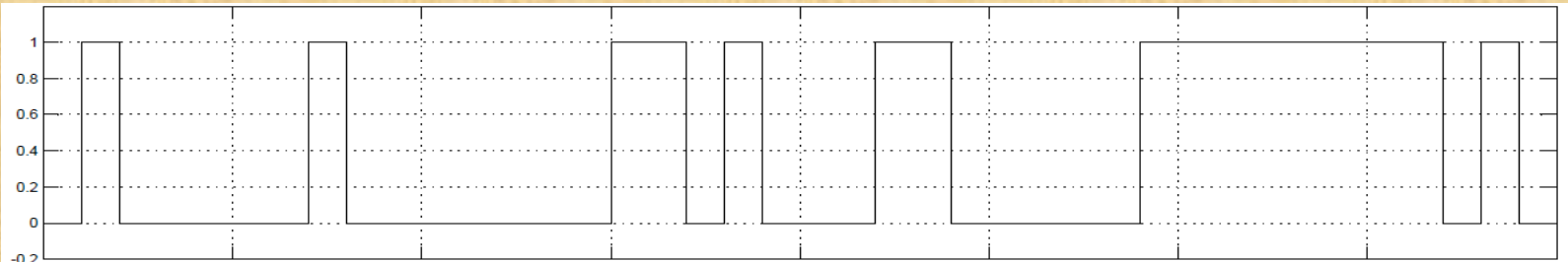
Diseño en punto fijo

Diseño con *System Generator*

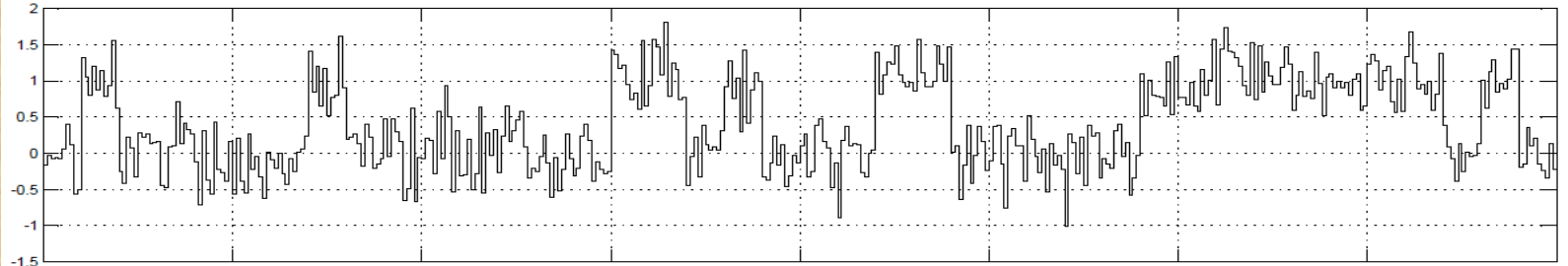
Formas de ondas obtenidas con *Simulink*:

- error del **3,9%**,
- **resolución ajustada** en la salida,
- y **SNR en la entrada de +7 dB**.

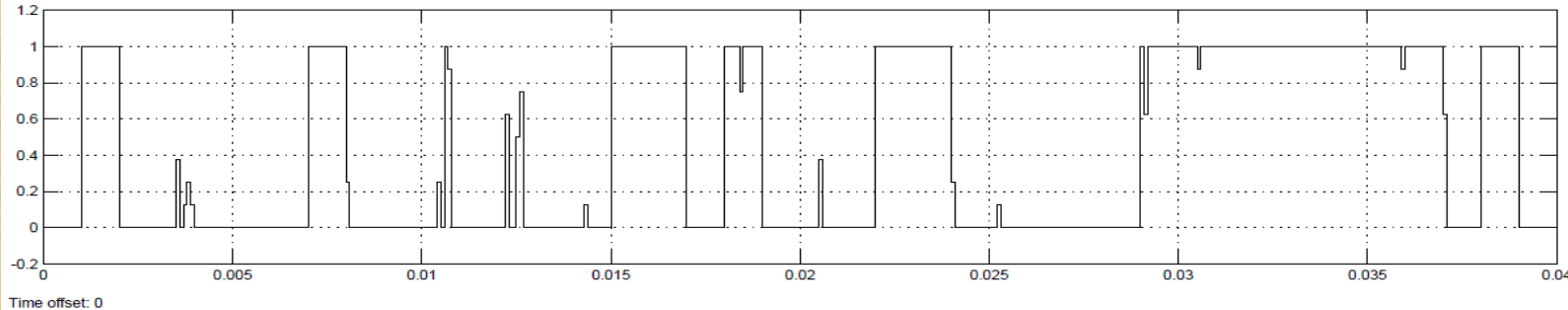
$d(t)$



$r(t)$



Salida de  
la TDNN



Time offset: 0



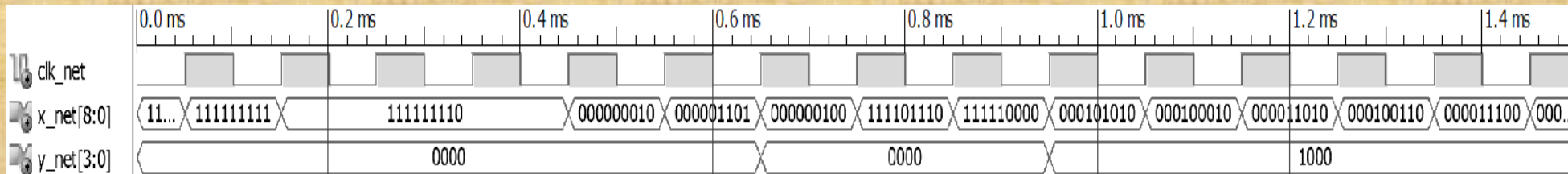
# EXPERIMENTOS Y RESULTADOS

## El ecualizador para señal binaria

### Diseño en punto fijo

### Implementación con *Integrated System Environment*

Formas de onda de las primeras quince muestras después del colocado y conexionado en la FPGA.



Resumen de los casos estudiados para el ecualizador.

	VHDL		Verilog	
<b>3,9% de error</b>	Área	401 SLICES	Área	387 SLICES
	Frecuencia máxima	312,695 MHz	Frecuencia máxima	357,654 MHz
	Potencia (5 MHz)	34,69 mW	Potencia (5 MHz)	34,65 mW

Frecuencia máxima = 357,654 MHz  $\Rightarrow$  Rb = 35,765 Mbits/s

# EXPERIMENTOS Y RESULTADOS

## Comparación con el estado del arte

Es difícil hacer una **comparación cuantitativa** con el estado del arte: los diferentes trabajos tienen enfoques distintos.

Las diferencias con la tesis establecen que esta constituye un **trabajo original**.

El **tipo de NN estudiada** (*Feedforward Multilayer Perceptron*), otros se centran en otro tipo de NN [Nambiar et al., 2014].

La mayoría de los trabajos están implementados en **aritmética de punto fijo**, aunque algunos se han desarrollado para **aritmética de punto flotante** [Cavuslu et al., 2011].

Las funciones de transferencia se implementan mediante **LUT**, otros trabajos usan **otras formas de aproximación** [Nascimento et al., 2013].

El método de diseño planteado es **rápido y flexible**; algunos autores usan métodos menos versátiles, como puede ser la descripción usando un HDL [Ogrenci, 2008].

# EXPERIMENTOS Y RESULTADOS

## Comparación con el estado del arte

Se insiste en la extracción de las prestaciones de **área, velocidad y potencia**; la mayoría de los autores no extraen el consumo de potencia [Orlowska y Kaminski, 2011].

Es objetivo de esta tesis realizar el entrenamiento **offline**, muchos trabajos implementan NN capaces de realizar un entrenamiento **online** [Gomperts et al., 2011].

Se **paralelizó el cálculo** en las neuronas, igual que en [Bahoura y Park, 2011]. En la mayoría de las referencias se presenta el **cálculo secuencializado**, mediante uso de elementos MAC (*Multiplier-Accumulator*) [Oniga et al., 2009].

Los autores que **paralelizan el cálculo** en las neuronas **asignan el mismo número de bits** a los coeficientes de la misma capa [Bahoura y Park, 2011].

Además, los autores usan un número de bits **fijado de forma arbitraria** [Mishra et al., 2007]; como mucho, las posibles arquitecturas son probadas de forma discreta, sin barrer todas las posibles soluciones [Gomperts et al., 2011].

# EXPERIMENTOS Y RESULTADOS

## Comparación con el estado del arte

En particular, en la NN totalmente paralelizada, se asigna una representación binaria **diferente para cada coeficiente**.

La cantidad de **bits usados** no se varían directamente por el diseñador; sino a través del valor del **error**.

La principal aportación de esta tesis es que el método planteado permite comprobar rápidamente **diferentes arquitecturas**, hasta llegar a su implementación física.

Se define la **calidad** de una implementación atendiendo a las **prestaciones físicas** (área, velocidad y potencia), para una cierta **funcionalidad**; algunos autores definen la calidad de un diseño mezclando la funcionalidad con las prestaciones físicas, pero excluyendo la potencia [Tommiska, 2003].



# ÍNDICE

1. INTRODUCCIÓN

2. IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

3. METODOLOGÍA EXPERIMENTAL

4. EXPERIMENTOS Y RESULTADOS

5. CONCLUSIONES Y LÍNEAS FUTURAS

6. DEMOSTRACIÓN

# CONCLUSIONES Y LÍNEAS FUTURAS

## Conclusiones

**Hipótesis** que se expuso en el capítulo primero:

*“Es posible encontrar métodos de diseño sobre dispositivos digitales programables para implementar redes neuronales que operen en tiempo real en procesamiento digital de la señal. Los métodos deben ser rápidos y flexibles; y permitir evaluar el efecto del número de bits sobre la arquitectura. Además, deben posibilitar la comprobación de la total funcionalidad del sistema y las prestaciones físicas de área, potencia y velocidad”.*

- **Matlab** se ha convertido en un entorno estándar para el estudio y diseño de las NN.
- Un factor importante de los métodos de diseño es el **soporte ante los errores**.
- La compilación con el ***Integrated System Environment*** optimizando en **área o velocidad** dio peores resultados.
- Sí se observó **dependencia** en la compilación de *System Generator* frente al **HDL** elegido.

# CONCLUSIONES Y LÍNEAS FUTURAS

## Conclusiones

En **pruebas finales**, realizadas en la **compilación**, en diseños con un solo multiplicador, se observó que no responde como fuera de esperar antes **casos triviales**.

Al multiplicar una señal por uno o cero **mantiene el circuito multiplicador**, cuando es eliminable. También se mantiene el **sumador**, aunque una de sus entradas sea nula.



# CONCLUSIONES Y LÍNEAS FUTURAS

## Líneas futuras

Se puede operar en cada capa con un **error distinto** para los coeficientes, diferente error para las funciones de transferencia en cada capa; e incluso, un error diferente para las entradas.

En los dos últimos escenarios, **en la salida** de la TDNN, se realiza un **ajuste** en el número de bits. Este proceso podría automatizarse mediante el uso de parámetros y el conveniente código ejecutable.

Ha quedado de manifiesto, en el diseño del ecualizador, que a veces es posible usar las **funciones** de Matlab que son **lineales por tramos**. Esto supone grandes ventajas en la implementación de punto fijo.

De todas formas, pueden usarse **funciones no incluidas en Matlab**, como las formadas por dos tramos parabólicos, lo que simplifica la implementación.

Finalmente, aún usando **funciones sigmoideas**, pueden implementarse **otras aproximaciones**.



# CONCLUSIONES Y LÍNEAS FUTURAS

## Líneas futuras

Otra línea de trabajo puede ser no implementar la NN totalmente paralelizada; es decir, **cambiar el secuenciamiento** del cálculo de diferentes formas.

En el futuro pueden implementarse **otro tipo de clasificadores o NN**. Cabe destacar la NN realimentadas.

Debe insistirse en la conveniencia del *Neural Network Time Series Tool* de Matlab.

Mediante el uso de las TDNN, realimentadas o no, puede procederse a la **demodulación y detección de señales**.

Debe destacarse que en el ecualizador **no es posible normalizar la señal**. La etapa que procedería añadir sería un **control automático de ganancia**.

Con el método planteado con *System Generator*, cada vez que se implementa una NN de **distinto tamaño**, es preciso su rediseño.

Se podría crear una **NN suficientemente grande**.

# ÍNDICE

1. INTRODUCCIÓN

2. IMPLEMENTACIÓN DE REDES NEURONALES EN TIEMPO REAL

3. METODOLOGÍA EXPERIMENTAL

4. EXPERIMENTOS Y RESULTADOS

5. CONCLUSIONES Y LÍNEAS FUTURAS

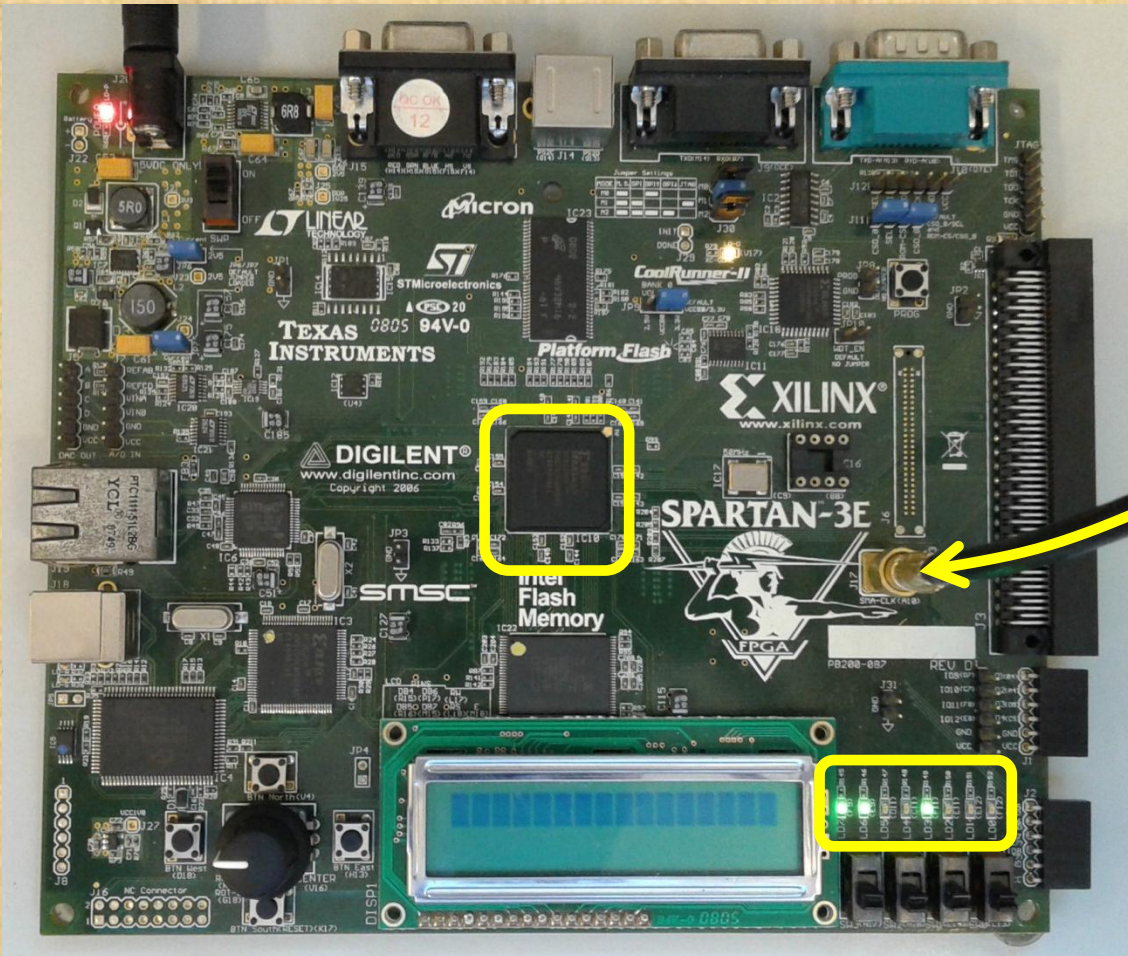
6. DEMOSTRACIÓN

# DEMOSTRACIÓN

## El ecualizador para señal binaria

PLACA: Spartan-3E Starter Kit Board

FPGA: Spartan3E xc3s500e -4fg320



Prueba a baja frecuencia.

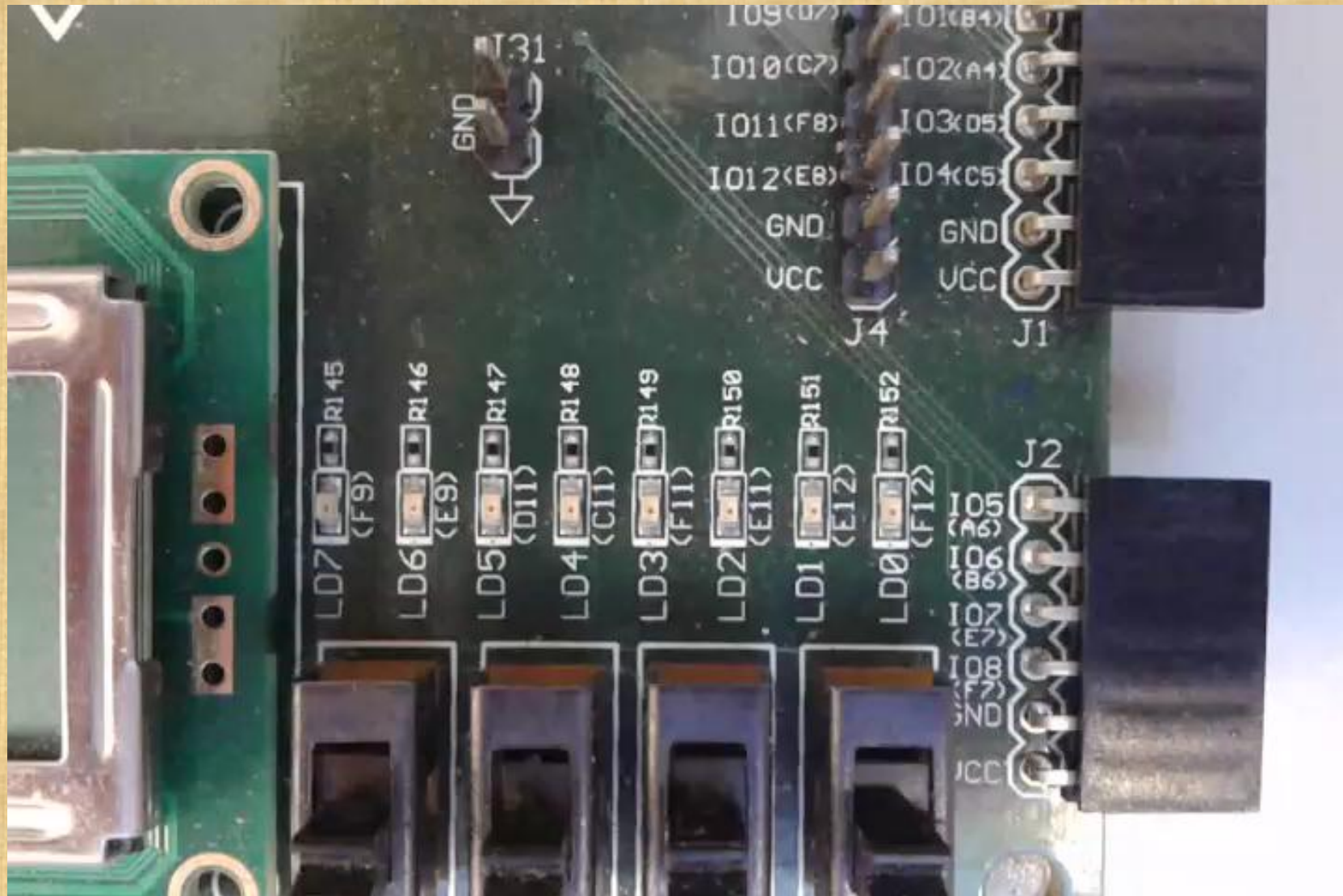
$$f_{\text{máxima}} = 61,588 \text{ MHz}$$





# DEMOSTRACIÓN

## El ecualizador para señal binaria



Windows Media Player



VLC media player





**METODOLOGÍAS DE DISEÑO DE REDES NEURONALES  
SOBRE DISPOSITIVOS DIGITALES PROGRAMABLES PARA  
PROCESADO DE SEÑALES EN TIEMPO REAL**

