eman ta zabal zazu

Universidad del País Vasco   Euskal Herriko Unibertsitatea

Efficient Electronic Implementations of Adaptive Systems for Ambient Intelligence Environments

DISSERTATION
to obtain the degree of doctor at the University of the Basque Country, by

Raúl Finker de la Iglesia

Thesis advisors:
Javier Echanobe Arias
Inés del Campo Hagelström

Leioa, June 2015

Agradecimientos

El haber llegado a ser capaz de realizar esta tesis ha sido posible gracias a la ayuda directa e indirecta de un gran número de personas. Mencionarlas a todas es imposible y aunque no sean citadas en las siguientes líneas han de saber que no me he olvidado de ellos.

En primer lugar, quiero agradecer a mis directores, Javier Echanobe e Inés del Campo por haberme sabido guiar por este tortuoso camino que empecé con ellos aquel ya lejano Septiembre del 2011.

También quiero dar las gracias a los compañeros del Grupo de Diseño en Electrónica Digital: José Tarela, Mariví Martinez, Koldo Basterretxea, Guillermo Bosque, Estibaliz Asua, Unai Martinez y Sandra Seijo por haberme ayudado con las dudas surgidas tanto de la tesis como en otros asuntos surgidos durante mi etapa como investigador. También quiero agradecer a nuestro antiguo compañero Pablo Echevarria la ayuda que me ha dado pese a estar en Berlín.

Tengo que agradecer también a los profesores del Master en Sistemas Electronicos Avanzados de la Escuela Superior de Ingeniería de la Universidad del País Vasco/Euskal Herriko Unibertsitatea por haberme introducido en el campo de los dispositivos de Logica Reconfigurable, que fue la primera piedra en la se cimentó esta tesis. En particular gracias a Josu Jugo por presentarme, cuando era su alumno en el máster, a los que ahora son mis directores de tesis.

Un saludo especial a la gente que componen el Departamento de Electricidad y Electrónica por la paciencia que muchas veces han tenido que tener conmigo, sobre todo a la gente del Becadromo por ayudarme con las duda que les he presentado con respecto al papeleo relacionado con la mención internacional.

Also, I want to thank the people I met during my internship in the University of Coventry, in particular to my advisor there, Faiyaz Doctor. Thanks to all of you my first experience studying in a foreign country was really delightful.

Aunque no me ha ayudado directamente con este proyecto, mis más sinceros agradecimientos a Pedro Luis Arias Ergueta. Amigo de mi familia, siempre ha sido un referente y modelo que he intentado seguir desde mi infancia y que sin él no creo que me hubiera decantado por el campo de la investigación científica.

Fuera del ámbito académico, también agradecer la ayuda de Fernando García, pues sus conocimientos del campo de la informática me ayudaron a completar parte de esta tesis. También mencionar a las asociaciones culturales Motsukora y Tarasu. Gracias a los eventos y actividades realizados por ellos he conseguido liberar mi mente por un tiempo, y así he conseguido solucionar problemas en los que estaba completamente obcecado sin poder ver la salida.

Por último, naturalmente agradecer a mi familia el esfuerzo realizado durante toda mi vida para que yo pudiera estudiar y llegar hasta este punto. A todos, ¡gracias!

Nire amamari eskainia.
Mila esker egin duzunagatik.

Introduction

In recent years much effort has been devoted to research in the field of Ambient Intelligence (AmI). This recent paradigm, proposes environments (e.g. public or private halls, rooms or spaces), also known as Intelligent Environments, endowed with a set of electronic systems, which are able to adapt to the preferences and necessities of the people existing in them in order to make their daily activities more easy and comfortable. To cope with these special skills, these systems can take advantage of intelligent paradigms such as Soft Computing (SC) techniques.

One of the requirements for achieving this goal in its broadest sense, is certainly, the availability of small-size, low-cost and low-power electronic devices with also high processing speed as they act in a scenario that requires real-time response. However, such features are hardly compatible with the high computational requirements of the above-mentioned intelligent paradigms. As a result, most existing solutions are basically Personal Computer (PC) based, because they focus more on the feasibility of the models, rather than on their physical implementations.

The aim of this thesis is to develop a Soft Computing-based intelligent system for AmI environments. In particular Artificial Neural Networks (ANN) are used in this project. To solve the problem of computational, size, cost and power requirements, the system is implemented using a Field Programmable Gate Array (FPGA) which is an integrated circuit designed to be configured by the user or the designer after manufacturing. These devices have another advantage, the possibility of implementing customized microprocessors within them, thus, allowing the inclusion of software programs to obtain simpler designs. As a result, a hybrid hardware/software (HW/SW) architecture can be implemented in a single FPGA (i.e. a System on Programmable Chip: SoPC).

Modern FPGAs provides also another property that can be very useful to save power and size, the Dynamical Partial Reconfiguration (DPR) technology. This feature allows modifying dynamically a part of the logic of an FPGA while the rest continues operating without interruption, dramatically enhancing the flexibility that FPGAs offer. In this work DPR is applied in the proposed architecture to obtain more efficient designs.

To validate the feasibility of the proposed solution, two applications in real-world environments are presented: the first one intended for an intelligent inhabited environment (the iDorm dormitory developed by the University of

Essex), and the second one intended for a smart car scenario.

This thesis is divided in the following chapters.

Chapter 1 provides an introduction to AmI and the SC techniques, with a main focus on ANNs. Finally, the chapter also makes an introduction to Programmable Logic Devices, explaining their evolution from the initial devices to the last generation FPGAs and their different types.

In Chapter 2 the HW/SW architecture implemented in the FPGA is presented. Firstly, the architecture from a global perspective is described. Next, the kernel of this thesis (i.e. the hardware implementation of the Artificial Neural Network) is carefully analyzed. Finally, an alternative methodology to implement the activation function of the neurons of the ANN is proposed, followed by its integration in the ANN.

Chapter 3 is devoted to explain how DPR technology can be applied to the system designed in Chapter 2, with the aim of adding new features to the system and of achieving a size and power consumption reduction.

In Chapter 4 two real-world applications are presented to demonstrate the viability of the architecture presented in Chapter 2 and to show how some of the new features described in Chapter 3 can be also applied.

Finally, in Chapter 5 the main conclusions of this thesis and the future work are presented. Also, the different publications written during the realization of this project are listed.

The platforms used in this thesis for the implementations are the last generation of FPGAs of Xilinx manufacturer. Also, Matlab numerical computing environment has been used to perform simulations of the system before its implementation in hardware. The Matlab, VHDL and C source code developed by the author is available in the CD attached to this thesis. The source code has been commented with Doxygen documentation generator available at www.doxygen.org.

Nomenclature

ACC	Accuracy
AI	Artificial Intelligence
ALU	Arithmetic Logic Unit
AmI	Ambient Intelligence
ANN	Artificial Neural Networks
ART	Adaptive Resonance Theory
ASIC	Application-specific Integrated Circuit
AXI-Stream	Advanced eXtensible Interface v4 - Stream
BP	Gradient Descend Backpropagation
CAN	Controller Area Network
CPLD	Complex Programmable Logic Devices
CPU	Central Processing Unit
CRI	Centred Recursive Interpolation
CS	Computer Science
DEV	Standard Deviation
DPR	Dynamic Partial Reconfiguration
ELM	Extreme Learning Machine
EMC	External Memory Controller
ET	Electronic Technologies
FFD	Type D Flip-Flop
FIFO	First-Input First-Output

FIS	Fuzzy Inference System
FPGA	Field-Programmable Gate Array
FPU	Floating Point Unit
FSL	Fast Simplex Link
FSM	Finite State Machine
GPGPU	General-purpose Computing on Graphics Processing Units
HMI	Human-Machine Interface
HW	Hardware
IC	Integrated Circuit
ICAP	Internal Configuration Access Port
ICT	Information and Communications Technologies
IE	Intelligent Environment
IMU	Inertial Measurement Unit
IP	Intellectual Property
ISTAG	European Community's Information Society Technology
JTAG	Joint Test Action Group
LAPU	Live-At-Power-Up
LMB	Local Memory Bus
LUT	Look-up Table
MAC	Multiply-Accumulator
MLP	Multilayer Perceptron Neural Network
MSB	Most Significant Bit
OTP	One-Time Programmable
PAL	Programmable Array Logic
PC	Personal Computer
PCA	Principal Component Analysis
PLA	Programmable Logic Array
PLC	Power-Line Communication

PLD	Programmable Logic Device
PR	Partial Reconfiguration
PWL	Piecewise Linear approximation
RALUT	Range Addressable Look-up Table
RAM	Random Access Memory
RFID	Radio Frequency IDentification
ROM	Read Only Memory
SC	Soft Computing
SEU	Single Event Upset
SLFN	Single-hidden-layer Feedforward Neural Networks
SoPC	System on Programmable Chip
SPLD	Simple Programmable Logic Device
SVD	Singular Value Decomposition
SVM	Support Vector Machine
SW	Software
UART	Universal Asynchronous Receiver/Transmitter
VHDL	Very-High-Speed Integrated Circuits Hardware Description Language

Contents

List of Figures

List of Tables

Resumen

En los últimos años se ha dedicado mucho esfuerzo a la investigación en el campo de la Inteligencia Ambiental (AmI en sus siglas en inglés). Este reciente paradigma propone ambientes (e.g. salas públicas o privadas, habitaciones o espacios), también conocidos como Entornos Inteligentes, dotados con una serie de sistemas electrónicos, que hacen que el entorno se adapte a las necesidades y preferencias de las personas que viven en él, de forma que sus actividades diarias sean más fáciles y confortables.

Uno de los requisitos para conseguir este objetivo es que los dispositivos electrónicos sean de pequeño tamaño, bajo coste y bajo consumo, pero al mismo tiempo que tengan una gran velocidad de procesamiento debido a que deben ser capaces de dar una respuesta en tiempo real. No obstante, estas características son poco compatibles con los requisitos de capacidad de computación mencionados anteriormente. Como resultado, las soluciones que se plantean para estos entornos están basadas en ordenadores personales (PC), dado que se centran en la verificación de los modelos, más que en su implementación física.

El objetivo de esta tesis es desarrollar sistemas inteligentes para entornos AmI basados en el uso de Redes Neuronales Artificiales. El sistema se implementará en un dispositivo tipo Field Programmable Gate Array (FPGA). Las FPGAs, son circuitos integrados que puede ser reconfigurados por el usuario o el diseñador tras su producción. Estos dispositivos tienen otra ventaja, la posibilidad de implementar en ellos microprocesadores a medida, permitiendo así la inclusión de programas software para obtener diseños más sencillos. Como resultado, una arquitectura híbrida hardware/software puede ser implementada en una única FPGA.

Las FPGAs modernas ofrecen otra propiedad que puede ser muy útil para la reducción de consumo y tamaño, la tecnología denominada Reconfiguración Dinámica Parcial (DPR en sus siglas en inglés). Esta tecnología permite modificar dinámicamente una parte de la lógica de la FPGA mientras el resto continúa operando sin interrupción. En este trabajo, se introduce DPR en la arquitectura propuesta para obtener mejores diseños.

Para validar la viabilidad de los diseños, se presentan dos aplicaciones del mundo real: la primera es una propuesta para un entorno inteligente habitado (el entorno iDorm de la Universidad de Essex), y en la segunda el escenario es un automóvil inteligente.

Estado del Arte

Inteligencia Ambiental

En primer lugar hay que presentar el concepto de AmI. Este concepto fue introducido por primera vez por la Comisión Europea en 2001 cuando la European Community's Information Society Technology (ISTAG) lanzó el desafío AmI. Pese a que inicialmente AmI fue un concepto surgido en Europa, rápidamente se extendió por el resto del mundo y surgieron un gran número de proyectos y programas de investigación.

El concepto de AmI puede ser descrito como un modelo interacción donde las personas están rodeadas por un entorno digital y éste es consciente de su presencia, es sensible al contexto y es capaz de responder en una forma adaptativa y transparente a las necesidades y/o hábitos de los usuarios para hacerles la vida más fácil [1]. Para hacer esto, el sistema debe ser capaz de obtener información en tiempo real y procesarla usando el conocimiento adquirido y cierto grado de inteligencia. En este caso, el término *Inteligencia* hace referencia a la habilidad del entorno para analizar y entender el contexto, identificar los usuarios y las tareas que están haciendo para aprender sus hábitos y preferencia, para adaptarse a cambios y, eventualmente, para ser capaces de reconocer emociones [2].

Ejemplos de Entornos de Inteligencia Ambiental

Para comprender mejor el concepto de AmI, a continuación se muestran una serie de ejemplos de entornos inteligentes:

1. Casas inteligentes: Una casa inteligente es un buen ejemplo para comprender la idea de AmI, dado que es un entorno donde la gente pasa bastante tiempo en él y donde se pueden incluir fácilmente sensores y actuadores. En la actualidad hay muchos proyectos de casas inteligentes, entre los cuales podemos mencionar los siguientes:

   (a) MavHome [3]: En este proyecto se busca automatizar las actividades que el usuario hace manualmente.

   (b) Proyecto Gator Tech [4]: Este proyecto busca crear un entorno capaz de asistir a personas de avanzada edad o con discapacidades.

   (c) iSpace [5]: Se trata de un piso de 2 habitaciones desarrollado por la Universidad de Essex. Es una evolución del proyecto iDorm, el cual era inicialmente una habitación de un dormitorio.

2. Entornos sanitarios: Estos entornos tienen un gran potencial para la implementación de AmI en ellos. Una forma de aplicar AmI es en el campo de la monitorización de pacientes en el hogar [6], [7]; reduciendo así el estrés de los pacientes y de sus cuidadores. La segunda opción consiste en la creación de hospitales inteligentes. En este caso se podría usar AmI para ayudar a los trabajadores sanitarios, por ejemplo, avisándoles que

los resultados que esperaban ya están disponibles y mostrándoselos en la terminal más próxima, ahorrando tiempo al facultativo [8].

3. Industria del transporte: En la industria automovilística se puede aplicar AmI de diferentes formas. Por ejemplo, en el campo del confort del conductor y de los pasajeros, no obstante, hay otras posibilidades:

   (a) Proyecto NEDO [9]: Su objetivo es la identificación del conductor, así como la detección de distracciones y el reconocimiento de maniobras.

   (b) Detección de fatiga: Los principales fabricantes de automóviles están incluyendo estos detectores en sus modelos. Algunos ejemplos son Mercedes-Benz (Attention Assist) [10], Ford (Driver Alert) [11], Volkswagen (Fatigue detection system) [12] or Volvo (Driver Alert Control) [13] entre otros.

   (c) Detección de infartos: Ford junto con la Universidad Técnica de Aquisgrán están desarrollando un asiento con una serie de sensores capaz de detectar si el conductor está sufriendo un infarto de miocardio [14]. En ese caso el coche se detendrá y avisará a los servicios de emergencia indicando su localización [15].

Redes Neuronales Artificiales

Para dotar de inteligencia a un entorno existen diferentes técnicas que se pueden usar, entre ellas las Redes Neuronales Artificiales (ANN en sus siglas en inglés). Las ANN son técnicas pertenecientes al campo denominado Soft Computing (SC) que, de acuerdo con el profesor Lofti Zadeh, *"definen la computación que emula la habilidad de la mente humana para razonar y aprender en un entorno de incertidumbre e imprecisión"* [16]. Esta definición hace dos declaraciones, la primera es que la información del entorno es imprecisa, y la segunda es que las técnicas de SC surgen de los procesos naturales.

Las ANNs están basadas, al igual que las redes biológicas, en neuronas y sus conexione. Las neuronas artificiales, cuyo modelo matemático fue propuesto por McCulloch y Pitts en 1943 [17], están basadas en la morfología de las neuronas biológicas:

- Dendritas (x_i): Son las entradas de la neurona proceden del exterior de la red o de la salida de otras neuronas.

  – Pesos (w_i): Valores asociados a las entradas, responsables de darle mayor o menor fuerza a éstas.

- Soma: cuerpo de la neurona donde se suman todos los valores ponderados de las entradas. A continuación se aplica una función de transferencia conocida como función de activación.

  – Offset (θ): Este valor es el responsable de añadir un umbral a la suma de entradas ponderadas.

- Axon: Es la salida de la neurona. Su única función es la de dar un camino para los datos de salida.

La salida de la neurona está dada por la siguiente ecuación:

$$y = f(\sum_{i=1}^{n} x_i w_i + \theta)$$

Cuando un cierto número de neuronas se interconectan, se forma una estructura capaz de realizar ciertas funciones. Normalmente la red, se suele organizar en capas:

- Capa de entrada: Actúa como *buffer* de la red, es decir, solo tiene una dendrita sin pesos y la salida tiene el mismo valor de la entrada.
- Capas ocultas: Puede haber una, varias o ninguna. Se caracterizan porque las dendritas de las neuronas están conectadas a los axones de todas las neuronas de la capa anterior, a su vez los axones se conectan a las dendritas de todas las neuronas de la capa siguiente.
- Capa de salida: Es la salida de la red. Las dendritas de las neuronas están conectadas a todos los axones de la capa anterior. Los axones de la capa de salida no se ramifican dado que sólo están conectados a la salida de la red.

Las redes pueden aprender y adaptarse mediante algoritmos de aprendizaje. Estos algoritmos pueden ser de dos tipos: aprendizaje de parámetros, donde se modifican los pesos y los offsets de la red; y aprendizaje de estructura, donde se modifica el número de neuronas, el número de capas o las funciones de activación. Estos algoritmos se pueden usar por separado o conjuntamente.

En la presente tesis, se han considerado ambos tipos de algoritmos. En el caso del aprendizaje de parámetros, se han usado dos algoritmos diferentes. En primer lugar, el algoritmo de Backpropagation (BP) desarrollado por Parker en 1982 [18] y Rumelhart et al. en 1986 [19], uno de los más usados y que se basa en variar el valor de los pesos proporcionalmente al gradiente del error. Por otro lado se han usado la llamadas Extreme Learning Machines (ELM) desarrolladas por Huang [20] en 2006. Estas redes tienen la peculiaridad de que los pesos de la capa oculta son aleatorios y las neuronas de la capa de salida no tienen ni offsets ni función de activación lo que permite que se puedan entrenar usando un sólo ciclo de aprendizaje. En el caso del aprendizaje de estructura, se ha usado un algoritmo de crecimiento/poda que varía el número de neuronas en la capa oculta [21]. Este algoritmo se usa junto con el algoritmo de BP.

En relación con la implementación de la red neuronal existen varias posibilidades:

- Redes software: Se ejecutan en microprocesadores o microcontroladores con el problema de que no existe ningún paralelismo en la ejecución salvo en el caso de procesadores multinúcleo. Esta implementación se usa cuando no hay muchas restricciones en cuanto a tamaño o potencia consumida.

- Redes hardware: Éstas pueden ser digitales o analógicas. Tienen la ventaja de poder implementar diversos tipos de paralelismo y tener menor consumo. La principal desventaja de las redes analógicas es su sensibilidad a los cambios de temperatura y voltaje, mientras que en el caso de las redes digitales su principal desventaja es el gran número de recursos necesarios para la implementación de los algoritmos de aprendizaje.

- Redes hardware/software: en este caso se realiza una partición de los algoritmos computacionales en dos bloques, uno hardware y otro software, de forma que se optimice el rendimiento del sistema. Esta es la mejor solución cuando es necesario un término medio entre versatilidad y rendimiento.

En el presente proyecto se ha optado por el diseño de una arquitectura hardware/software, en donde la red es implementada en la partición hardware y los algoritmos de aprendizaje están implementados en la partición software.

La arquitectura hardware/software (HW/SW) es implementada en una FPGA, dado que es posible implementar en dichos dispositivos procesadores, ya sea haciendo uso de los propios recursos de la FPGA, como los procesadores NIOS de Altera [22] o el MicroBlaze de Xilinx [23], o bien porque vienen integrados en el silicio del dispositivo. Otra ventaja es que también se pueden implementar dentro del propio dispositivo los buses de comunicación y periféricos necesarios por el sistema sin necesidad de usar ningún dispositivo externo, creando así los llamados System on Programmable Chip (SoPC).

Arquitectura Hardware/Software para una Red Neuronal Artificial

Para la implementación de la Red Neuronal Artificial, se ha diseñado una arquitectura HW/SW en la cual todo el sistema está integrada en una única FPGA.

En la partición hardware se ha desarrollado una red multicapa completamente escalable mediante el uso de Very-High-Speed Integrated Circuits Hardware Description Language (VHDL) estándar, de forma que la red puede ser implementada en diferentes dispositivos independientemente del fabricante. Las neuronas de la red se han implementado de forma que hacen uso de los Procesadores Digitales de Señales (DSP en sus siglas en inglés) incluidos en las FPGAs, obteniendo así un ahorro de recursos y un aumento del rendimiento, dado que estos recursos son más rápidos que el uso de la lógica interna para realizar operaciones de multiplicación/acumulación. A su vez, las neuronas de las diferentes capas se han implementado en paralelo para así obtener un diseño más rápido. En caso de que la red se quiera implementar en FPGAs sin DSPs o multiplicadores dedicados, las neuronas se pueden implementar usando la lógica de la FPGA.

Se han diseñado dos arquitecturas de redes distintas dependiendo del método usado para el almacenamiento de los pesos: una usando memorias de sólo lectura (ROMs) y otra usando memorias de acceso aleatorio (RAMs). En el caso de la implementación de la función de activación, ésta se ha implementado mediante ROMs. Pese a que el diseño inicial de la red es para una sola capa oculta, el esquema de diseño usado es generalizable a redes de varias capas ocultas.

La partición software se ha realizado usando un procesador MicroBlaze y la memoria interna de la FPGA para hacer que la ejecución del software sea lo más rápida posible, evitando así el uso de elementos externos. El procesador es responsable de realizar el control del sistema y la ejecución de los diferentes algoritmos de aprendizaje (i.e. Backpropagation, Extreme Learning Machines y crecimiento/poda). El código de estos algoritmos se ha implementado en lenguaje C estándar independiente de la plataforma en lo máximo posible, por lo tanto, puede ser usado en diferentes dispositivos realizando sólo cambios mínimos. En relación con el rendimiento de estos algoritmos, ELM tiene un tiempo de ejecución muy bajo mientras que Backpropagation puede ser excesivamente alto, dependiendo de la estructura de la red y el número de ciclos de aprendizaje que se realicen. Para acelerarlo, se ha modificado la ANN de forma que pueda ser usada como coprocesador del algoritmo consiguiendo aceleraciones de hasta un 61 %.

Junto a lo anterior, se ha obtenido un nuevo método para la obtención de la función de activación basado en el teorema de Taylor. Usando un polinomio de Taylor de 2º orden y el término complementario de Lagrange, es posible aproximar las funciones sigmoide y tangente hiperbólica con una precisión controlada [24].

En suma, se ha diseñado una arquitectura HW/SW flexible que puede ser adaptada a un gran número de situaciones. Es capaz de cubrir un gran rango de especificaciones relacionadas con el tamaño, velocidad y precisión. El sistema se ha implementado correctamente en una gran variedad de dispositivos; FPGAs de bajo, medio y alto rendimiento, demostrando que sus requisitos de tamaño no son muy exigentes. Por lo tanto, el sistema es apto para ser implementado en los entornos AmI donde el tamaño y/o el rendimiento son una de las más importantes restricciones del diseño.

Reconfiguración Dinámica Parcial

Para la aplicación de la tecnología Reconfiguración Dinámica Parcial en la arquitectura diseñada se han propuesto diferentes estrategias.

Reconfiguración de una red neuronal completa

En situaciones donde se usan redes con los pesos en memorias ROM, se puede reconfigurar la red para diferentes contextos (periodo del día o la estación del año), en vez de implementar todos los módulos en el dispositivo. Un ejemplo sería utilizar en una casa inteligente una red para controlar el sistema de aire acondicionado y las persianas por el día, mientras que por la noche, se reconfigura la red con otra preparada para controlar la iluminación.

También se puede aplicar la reconfiguración de la red en situaciones en donde se necesitan altas precisiones en las funciones de activación durante el periodo de aprendizaje. La solución propuesta consiste en utilizar durante el aprendizaje una red que usa para la función de activación el módulo de Taylor, que usa menos recursos pero que es mucho más lento, mientras que durante el periodo de funcionamiento en tiempo real se reconfiguraría la red implementando memorias ROM con una precisión menor, pero una por neurona haciendo que la red sea más rápida.

Reconfiguración de los módulos internos de la red

Esta estrategia consiste en reconfigurar los módulos de la función de activación y las neuronas de la capa oculta. Debido a limitaciones de la herramienta usada para realizar la reconfiguración (la herramienta PlanAhead [25] de Xilinx) los módulos internos de la red deben ser implementados como módulos independientes para poder ser reconfigurados:

- Reconfiguración de la función de activación: Permite que en el proceso de aprendizaje se usen diferentes funciones de forma que se pueda seleccionar la mejor combinación, para obtener el menor error en el aprendizaje, sin necesidad de tener que implementar simultáneamente todas las funciones en el dispositivo. Este método es muy útil cuando el sistema dispone de un mecanismo de adaptación on-line que decide cambiar las funciones de activación cuando se producen cambios en el entorno.

- Reconfiguración de las neuronas de la capa oculta: Esta estrategia se basa en deshabilitar las neuronas que no son usadas tras el proceso de aprendizaje de estructura, de forma que éstas dejen de consumir potencia. Aunque la metodología propuesta permite reconfigurar las neuronas individualmente, las limitaciones de los dispositivos (especialmente los más pequeños) y las herramientas de diseño, complican en exceso las implementaciones. En vez de eso, las neuronas se han tenido que reconfigurar por pares o en módulos más grandes, especialmente cuando el número de neuronas a reconfigurar es muy alto y la estación de trabajo con la que se lleva a cabo el diseño no es lo suficientemente potente [26].

Como conclusión final se puede comentar que DPR dota a la red con nuevas capacidades y funcionalidades, sin la necesidad de tener que implementar simultáneamente todas las redes o módulos, y obteniendo así un ahorro de espacio, y por lo tanto de consumo de potencia. No obstante, se tiene que tener en cuenta que la inclusión de los dispositivos para el almacenaje de los archivos de reconfiguración y el resto de elementos necesarios para la llevar a cabo ésta pueden, en algunas situaciones, hacer que el consumo de potencia sea mayor que el que se pueda ahorrar o que dicho ahorro sea insignificante. Por tanto, el uso de la alternativa basada en las señales de habilitación de los relojes, presentes en las nuevas FPGAs, y el uso de nuevas técnicas de diseño (e.g. la *Estrategia de Divide y Vencerás* presentada en [27]) pueden ser una mejor solución que DPR para reducir el consumo de potencia, consiguiendo además que el diseño sea más simple.

Aplicaciones de la Arquitectura Hardware/Software en Entornos Inteligentes

Para demostrar que la arquitectura HW/SW propuesta puede ser usada en los entornos AmI para los cuales se ha diseñado, se han desarrollado dos aplicaciones en entornos inteligentes.

Desarrollo de un Agente Inteligente para un Entorno Inteligente Habitado

Como aplicación del sistema, se presenta el desarrollo de un agente inteligente para su implementación para el control en tiempo real de en un entorno inteligente habitado. El agente es implementado en el entorno iDorm, un dormitorio de una habitación, el cual ha sido desarrollado por el Grupo de Entornos Inteligentes de la Universidad de Essex [28], [29], [30].

Los investigadores han recogido cientos de datos de varios usuarios del entorno interaccionando con él durante varias estaciones del año. Aunque este entorno no es computacionalmente exigente, sí lo es en el caso del tamaño y consumo del sistema, lo que lo hace perfecto para la implementación de la arquitectura desarrollada en este proyecto. El entorno contiene 7 entradas obtenidas de diferentes sensores y 8 salidas correspondientes a diferentes actuadores. Las entradas y las salidas están en la Tabla 1.

Se usa una Red Neuronal Artificial adaptativa, implementada en una FPGA con capacidad de DPR. En el caso del SW, se han implementado los algoritmos de Backpropagation y de crecimiento/poda para obtener una red neuronal con un tamaño de la capa oculta óptimo. Gracias a DPR, es posible eliminar las neuronas que no vamos a usar consiguiendo así un ahorro en el consumo de potencia.

El agente adaptativo se comporta de forma optima para un máximo error permitido, mediante el uso de los algoritmos de BP y crecimiento/poda. Estos algoritmos permiten al agente adaptarse a cambios en el comportamiento del usuario del entorno, no sólo adaptando los parámetros de la red sino también adaptando la arquitectura de la misma. Para realizar el modelado del comportamiento del usuario, se necesitan varios segundos para completar 100 ciclos de aprendizaje, por tanto, el aprendizaje/adaptación se puede llevar a cabo en periodos de inactividad.

Otro punto importante es el uso de DPR para la implementación de sólo las neuronas necesarias en la red neuronal. En el peor de los casos, la reconfiguración de las 32 neuronas de la capa oculta, necesitaría 83.6 ms, a

Entrada	Variable	Slida	Actuator
In1	Nivel de luz interna	Out1	Luz de intesidad variable 1
In2	Nivel de luz externa	Out2	Luz de intesidad variable 2
In3	Temperatura ambiental interna	Out3	Luz de intesidad variable 3
In4	Temperatura ambiental externa	Out4	Luz de intesidad variable 4
In5	Presión en la silla (binario)	Out5	Estado de la persiana
In6	Presión en la cama (binario)	Out6	Luz de la cama
In7	Hora	Out7	Luz del escritorio
		Out8	Estado de la calefacción

Tabla 1: Descripción de las entradas y las salidas del entorno idorm

una frecuencia de reloj de 83.3 MHz. Este tiempo es más que suficiente para el entorno iDorm en el cual el tiempo de operación, definido por la interacción de usuario y la máquina, es del orden de segundos.

Desarrollo de un Identificador de Conductores en Tiempo Real para Inteligencia Ambiental Aplicada al Entorno de un Automóvil

La segunda aplicación propuesta es un identificador de conductores basado en el uso de Extreme Learning Machines y variables estadísticas (i.e. media, desviación típica, máximo valor absoluto y suma de valores absolutos) obtenidos de una Unidad de Medidas Inerciales (IMU en sus siglas en inglés) e implementado en una FPGA. Este trabajo contribuye al desarrollo de Sistemas Avanzados de Asistencia al Conductor con una perspectiva centrada en el conductor con el objetivo de mejorar su comportamiento en la conducción de forma personalizada.

Los datos usados han sido suministrados por el “Drive-Safe Consortium”, el cual ha usado un coche tipo sedan equipado con diferentes sensores [31], [32]. La base de datos completa (84 hombres y 17 mujeres) incluye grabaciones de audio y vídeo, señales del bus CAN, registros de la presión en los pedales, un goniómetro laser de 180º y registros de acelerómetros en los ejes XYZ (i.e. IMU).

El sistema se ha implementado en una FPGA modelo Kintex 7 para varios tamaños de la capa oculta, para la identificación de 3 conductores usando 16 entradas (4 variables estadísticas de 4 entradas). Como se puede ver en la Tabla 2, el sistema es capaz de realizar la computación del algoritmo de aprendizaje en muy poco tiempo, casi 40 segundos para 100 neuronas en la capa oculta, y con tasas de reconocimiento superiores al 90 % a partir de 25 neuronas. En la Tabla 3, se puede observar que salvo para un caso de 500 neuronas en la capa oculta, en el resto de los casos la red es capaz de trabajar a frecuencias superiores a 200 MHz.

En [33] se propuso un identificador de conductores usando una red con 2 capas ocultas y entrenándola con BP. Si los resultados se comparan, se puede

Neuronas en la capa oculta (L)	Tiempo de aprendizaje (s) (MB a 100MHz)	Tiempo computacion de la ANN (μs) a 100 MHz	Tasa de reconocimiento medio (%)
10	1.22	0.33	68.7
25	4.16	0.48	93.0
50	12.10	0.73	95.3
100	39.44	1.20	96.8

Tabla 2: Rendimiento del algoritmo ELM. Dispositivo: Kintex 7 XC7K325T-2

Neuronas en la capa oculta (L)	Frecuencia máxima (MHz)	Tiempo de computación (μs)
10	221	0.15
25	220	0.22
50	219	0.33
100	213	0.58
500	119	4.39

Tabla 3: Rendimiento temporal del coprocesador ELM. Dispositivo: Kintex 7 XC7K325T-2

observar que con sólo 25 neuronas ocultas, ELM supera ampliamente al clasificador multicapa. En cuanto a los recursos necesarios, ELM también es mejor dado que se necesitan menos neuronas que usando la red multicapa.

Conclusiones Finales y Trabajos Futuros

Conclusiones

En el presente trabajo de tesis se ha propuesto una solución al problema del diseño de un sistema embebido para entornos de Inteligencia Ambiental. Estos sistemas deben ser de pequeño tamaño, bajo coste y bajo consumo, pero al mismo tiempo deben tener suficiente velocidad de procesamiento para ejecutar los algoritmos inteligentes en tiempo real. En la presente tesis se ha implementado una arquitectura híbrida hardware/software para cumplir con dichos requisitos. En los siguientes párrafos se presentan las principales conclusiones y contribuciones de esta tesis.

Para afrontar el objetivo mencionado, se ha desarrollado una Red Neuronal Artificial HW/SW para Entornos Inteligentes basada en un dispositivo FPGA. Las particiones hardware y software han sido implementadas en el mismo chip, por tanto, no se necesitan otros dispositivos externos.

La Red Neuronal Artificial (la partición HW) ha sido diseñada usando lenguaje VHDL estándar, independiente del dispositivo en el que se implemente. Otra ventaja es su escalabilidad, es decir, el código ha sido diseñado de forma que se adapte a los cambios en el número de entradas, número de salidas, número de neuronas en la capa oculta o longitud de palabra. La mayor ventaja de la red HW es su paralelismo que incrementa el rendimiento computacional.

Diferentes tipos de algoritmos de aprendizaje han sido implementados en la partición software. El sistema es capaz de aprender durante la etapa off-line y adaptarse a cambios durante la etapa on-line. Por un lado, se han implementado dos algoritmos de aprendizaje de parámetros (i.e. Backpropagation y Extreme Learning Machines), y por otro lado, se ha implementado un algoritmo de aprendizaje de estructura (i.e. crecimiento/poda) para obtener el mejor tamaño de la capa oculta. Estos algoritmos se han implementado usando lenguaje C estándar de forma que son en alto grado independientes de la plataforma. Dado que el tiempo de ejecución del algoritmo de Backpropagation puede ser muy elevado, dependiendo de la estructura de la red y el número de ciclos de aprendizaje que se realicen, la red se ha modificado para que se pueda usar como coprocesador del algoritmo acelerándolo hasta un *61.83 %*.

El uso de la tecnología de Reconfiguración Dinámica Parcial ha sido también estudiado con el objetivo de conseguir una reducción en el tamaño y/o consumo de potencia. En lo referente a la reducción de tamaño, DPR permite implementar sólo los módulos que son necesarios en cada momento, obteniendo así una reducción en el tamaño del dispositivo. Por otro lado, en lo referente a la reducción del consumo de potencia, DPR presenta de pocos beneficios, especialmente en las FPGAs de última generación como la Familia 7 de Xilinx. En estos dispositivos, la reducción de potencia se puede obtener también mediante el uso de otras técnicas o recursos (e.g. las señales de habilitación/inhabilitación de los relojes).

Para demostrar la idoneidad del sistema en entornos de Inteligencia Ambiental en el mundo real, se han desarrollado dos aplicaciones diferentes. La primera es un agente adaptativo para un entorno inteligente habitado. Este agente es capaz de modelar el comportamiento del usuario del entorno seleccionando los mejores parámetros de la red, y el mejor tamaño de la capa oculta mediante los algoritmos de Backpropagation y crecimiento/poda. A su vez también es capaz de adaptarse a los cambios durante la etapa on-line. La segunda aplicación desarrollada es un identificador de conductores basado en Extreme Learning Machines. Este sistema contribuye al desarrollo de Sistemas Avanzados de Asistencia al Conductor centrados en el conductor, los cuales se están haciendo cada vez más importantes en la industria de la automoción como una forma de reducir el número de accidentes. El sistema está diseñado para usar las señales ya disponibles en el propio coche de forma que no es necesario instalar nuevos equipos en los coches.

Finalmente, cabe mencionar que durante la realización de esta tesis se han establecido relaciones con instituciones nacionales, como el Automotive Intelligence Center, e internacionales como la Universidad de Coventry en el Reino Unido y el Drive-Safe Consortium en Estambul, Turquía.

Trabajo futuro

Este proyecto ha abierto nuevos temas de investigación que serán desarrollados en el futuro. En primer lugar se comentarán los temas relacionados con la Inteligencia Ambiental seguidos por los temas relacionados con la tecnología de las FPGAs:

1. La percepción de emociones y la inteligencia emocional son cada vez más importantes [2]. Se investigará su implementación tanto en entornos habitados como en coches inteligentes.

2. Se han estudiado las capacidades del algoritmo Extreme Learning Machine para clasificación. En el futuro el estudio de sus propiedades de regresión serán también investigadas.

3. Este proyecto se ha centrado en Redes Neuronales Multicapa. Se estudiará el interés de rediseñar la red HW para implementar otro tipo de redes como las redes recurrentes.

4. Una gran parte de los algoritmos de aprendizaje se han implementado en SW, lo que resulta lento para algunas aplicaciones. En el futuro se estudiará el impacto de realizar el aprendizaje en un coprocesador HW, ya sea usando la propia red u otro módulo.

5. En respuesta a este trabajo sólo se ha tenido en cuenta DPR para reducir el consumo de potencia. En el futuro se estudiarán las nuevas señales de habilitación de relojes como técnica de reducción de consumo de potencia.

6. Durante la última parte de este proyecto, los Dispositivos de Lógica Programable que incluyen procesadores ARM han ganado mucha notoriedad. Con el fin de adaptar los diseños realizados a las nuevas familias de dispositivos, se realizará una migración del bus de comunicaciones de la red a AXI-Stream (parte del estándar abierto AMBA de ARM).

7. Finalmente, en los últimos años nuevas generaciones de FPGAs han aparecido en el mercado. Será importante mantenerse informado de las nuevas capacidades de dichas FPGAs, así como de las nuevas herramientas de diseño desarrolladas por los fabricantes.

Chapter 1

State of the Art

This chapter focuses on the description of the different methods available to develop Ambient Intelligence (AmI) systems and their implementation in different environments, such as homes, cars or hospitals. The development of AmI environments involves a series of multidisciplinary techniques that includes Information and Communications Technologies (ICT), and Electronic Technologies (ET).

First of all, the concept of AmI is presented, including the algorithms and the different implementation devices that can be used to create the intelligent environment. Following that description, Soft Computing (SC) is presented as the field of Computer Science (CS) for working with uncertainty and imprecision, focusing on the techniques with the capability to learn and adapt, and therefore, with the ability to react to changes in the environment. In particular, Artificial Neural Networks (ANN) are introduced.

Finally the implementation of the techniques and algorithms are described. Due to the nature of systems that demand small embedded devices, able to be hidden from sight, this final part of the chapter describes a suitable implementation technology: Programmable Logic Devices (PLD), or more precisely, their most recent evolution, Field Programmable Gate Arrays (FPGA).

1.1 Ambient Intelligence

1.1.1 Introduction to Ambient Intelligence

The AmI concept was introduced for the first time by the European Commission in 2001 when the European Community's Information Society Technology (ISTAG) group launched the AmI challenge. Initially its source was in Europe but it spread fast through the rest of the world and as result, a large number of projects and research programs appeared [34], [35], [36].

The AmI concept [37], [38], [28] can be defined as an interaction model where people are surrounded by a digital environment aware of their presence, sensitive to the context, and that answers in an adaptive and transparent way to the needs and/or habits of the users to make their daily lives easier. Environments with these characteristics can be houses, cars, work places or even public spaces. The main idea behind AmI is that by adding to an environment some embedded electronic devices (i.e. sensors, microprocessors or actuators, among others [39]), all of them interconnected by using a network [40], a system can be obtained which can help, assist and/or take decisions to the benefit of the people present in that environment [1]. To do that, the system obtains the information in real-time and processes it using the knowledge acquired and a certain grade of intelligence. *Intelligence* in this situation refers to the ability of the environment to analyze and understand the context, to identify the users and the duties they are doing, to know and learn the habits and preferences of the users, to adapt to changes and, eventually, to be able to recognize emotions [2].

From a more practical or feasible point of view, the AmI concept can be defined as a way to answer the *5 Ws* (Who, Where, When, What and Why) [41].

- Who: the system must identify the user or the users in the environment and also the role they represent in it and with other users.
- Where: the system must be able to know its geographical location (think about a car as a possible scenario) and record its previous locations. Using this information the system can assist the user/s with more information and even anticipate their needs.
- When: if a more realistic knowledge of the dynamic of the system is required, the knowledge of the duration of the activities and the precise hour of the day they are done are necessary.
- What: the activities the user does must be known by the system so it can provide him/her help in those concrete tasks.
- Why: the ability of the system to infer and understand the intentions and objectives of the user is one of the most important objectives in these environments. It will allow the system to anticipate and serve the user in a more sensitive way.

1.1.2 Description of an Ambient Intelligence Environment

An AmI or Intelligent Environment (IE) is a space similar to that shown in Figure 1.1, [42]. It is a place which contains the following elements [1]:

1. Sensors: thanks to these, the environment is sensitive to the changes within it. These changes can be produced by the inhabitants or any kind of environmental variations. The sensors can be organized in two types:
    - Environmental: these sensors are located in the environment.
        (a) Ambient sensors: temperature, humidity or pressure sensors.

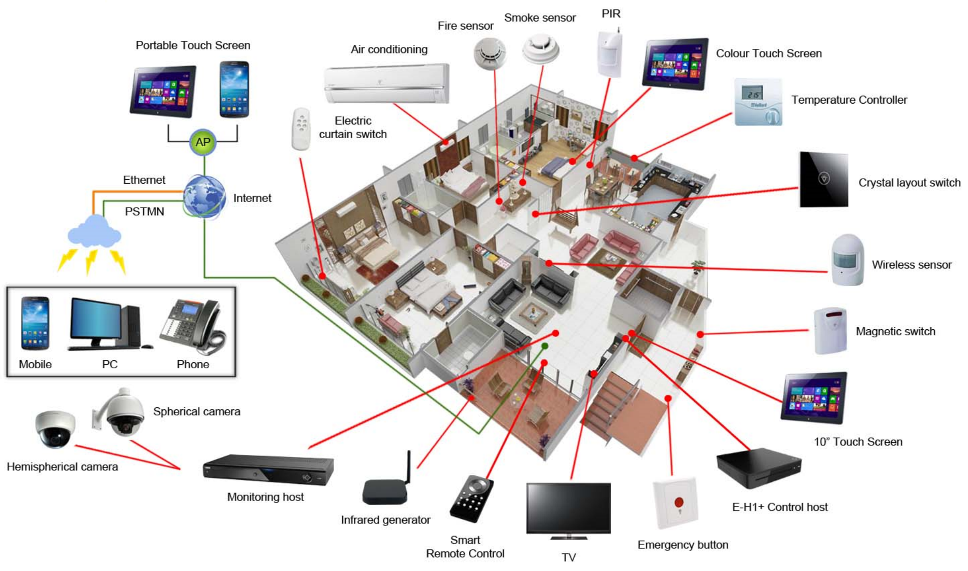


Figure 1.1: Image of an AmI environment with sensors, actuators, processing units and communication systems

   (b) Presence sensors
   (c) Sound sensors
   (d) Image sensors

- Wearable: these sensors are worn by the inhabitant of the environment.
   (a) Radio Frequency IDentification (RFID) tags
   (b) Magnetic tags
   (c) Physiological sensors: pulse, blood pressure, etc.

The main characteristic of the sensors of an AmI environment is that they must be as small and invisible as possible to the user. These characteristics are much more important if we are talking about wearable sensors like RFID tags or the iButtons [43].

2. Actuators: these elements are responsible for acting on the devices present in the environment, making it responsive to the user. Some examples of actuators are:

   - Controlled switches
   - Relays
   - Dimmers

    - Motorised valves

    Like the sensors, the actuators must be small and with low power consumption so they can be attached to the devices they must control in the easiest way. The elements they act on can be devices already present in the environment such as TVs, the air conditioning of a house, the radio, the power window lifts in a car, or the lights of an elevator in a smart building.

3. Communications: this is a key point in the implementation of an IE. There are two kinds of technologies able to perform the communications between the devices:

    - Wired technology: cheap and robust, but the installation of the wires can be complicated.
    - Wireless technology: the installation is very easy, but they are more expensive and not so robust.

    Wireless communications technologies allow for the placing of sensors and actuators without the need of large-scale remodelling. Depending on the transfer rate, there are different technologies available such as Zigbee [44] for low data rate, Bluetooth [45] for higher data rates or WiFi [46] for very high data rates but with higher power consumption.
    Also there are some technologies, such as Power-Line Communications (PLC) that allow the use of the existing infrastructure, power line cables, to send and receive data.
    The use of one technology does not exclude the other, so a combination of both of them can be used in the same environment. In fact, there are gateways in the market [47] to connect PLC devices to Zigbee sensors.

4. Processing: the actuators act on the environment, but this can be only achieved if processing is done with the data obtained from the sensors by applying algorithms to them. The outputs of the algorithms can be actions the actuators must perform, or information for the environment that, in some cases, can even be used by other algorithms. Typical algorithms applied in an IE are, among others, modelling [48], recognition/classification [49] and control and/or decision making [50].
    The processing can be centralized or distributed. In the case of distribute systems, instabilities can happened, for example, due to delays in the communication network [51] or their information-processing speed [52]. Hence, these systems must include algorithms, apart from the above mentioned, to avoid those instabilities [53].

5. Interfaces: the communication between the environment and the user must be done in the easiest and simplest way. As can be seen in Figure 1.1, the environment can be controlled by a remote control or even using a Smartphone or a tablet. However, there are people unable to use those

elements, such as elderly people or persons with a disability. Bearing this in mind, alternative Human-Machine Interfaces (HMI) can be used so the inhabitant can communicate with the environment using, for example, 1) voice commands, e.g. the Maior-Vocce system [54] designed by Fagor to control its Maior-Domo home automation system [55]; 2) motion tracking, 3) gesture recognition [56], 4) facial expression recognition [57], emotion recognition [58], [59], or even whistle processing [60].

1.1.3 Algorithms

An IE must be able to respond adequately to the data received from the sensors. The data must be intelligently processed to give a suitable answer. The most widely used algorithms are described below:

1. Modelling algorithms:
   - These algorithms model the behaviour of the users to give an adequate answer and must also be able to adapt correctly if there are changes in that behaviour. In AmI, user modelling approaches can be based on three characteristics: the data used to build the model, the type of model that is built and the nature of the algorithm (supervised, unsupervised) [61].
   - Modelling technique examples:
     - Soft Computing: Fuzzy Logic [28], Neural Networks [26] or Neuro-Fuzzy Systems [48].
     - Machine Learning: Support Vector Machines [62] and Extreme Learning Machines [20].
2. Recognition and prediction:
   - The environment becomes more effective and can improve the experience of the residents living in it, if it is able to recognize the users, their activities, or even, predict those activities.
   - Some of the techniques used for recognition and prediction are listed below:
     - Naive Bayesian classifiers [63], [64].
     - Markov models [65], [66].
     - Decision trees [67].
     - Dynamic Bayesian networks [68], [69].
     - Conditional Random Fields [70].
3. Decision making:
   - The main idea of this type of algorithm is to make the environment able to select the best suited action from different possible options. An example could be an environment that detects hazardous situations and chooses the best measures to return to a safe state.

- Examples of some decision making algorithms are:
  - Soft Computing: Fuzzy Logic [48] or Neural Networks [71] among others.
  - Hierarchical Task Networks [50].
  - Reinforced Learner techniques [3].

4. Spatial and temporal reasoning:

   - The previously mentioned algorithms are focused on the user and his/her activities, but they do not take into account his/her location and the time context in which the activities are done. The spatial and temporal reasoning does take into consideration these two factors, separately or together [72].
     An example is a cook who leaves the kitchen with the heater on; the environment recognizes that the user has left the kitchen and if he spends too much time in another room, it will warn the user that the activity he started is not finished and/or it will take actions to avoid damage [61].
   - Examples of this reasoning are:
     - Allen's temporal logic [73] and its generalizations [74].
     - The Point Algebra [75].

The algorithms listed above can be used separately or in combination. For example, a combination of decision making and spatial-temporal reasoning algorithms can be used to detect anomalies in the daily activities of a user and then take the proper actions to react to them, if necessary.

It could be also possible that the amount of data given by the sensors could be too large. In that case, in combination with the algorithm, a series of feature selection techniques should be used to select the most relevant inputs or use feature extraction techniques (e.g. Principal Component Analysis) to obtain new input variables [76], [77], [78].

Finally, in order to choose an algorithm, not only does it purpose have to be taken into account, but also its computational load. Depending on the computational load of the algorithm, a certain device might not be powerful enough (e.g. a 16-bit microcontroller), therefore a more powerful device should be used instead.

1.1.4 Implementations of the Algorithms

The implementation of the algorithms can be done using different solutions; for example, one powerful unit can be used, such as a Personal Computer (PC); or instead a distributed system can be used with smaller but less powerful units. Both solutions have their advantages and disadvantages. In the first case, the unit is powerful as already mentioned, but the power consumption and the space needed are big, and in AmI environments the system must be as small as possible. In the case of a distributed system, the size is smaller and the power consumption lower. However, if the systems must be interconnected, the design becomes more complicated.

To implement the processing unit the following devices can be used:

1. Microprocessors: these devices are Integrated Circuits (IC) that include a Central Processing Unit (CPU) and execute the instructions implemented in software (SW). Their main advantage is the great variety of devices present in the market, but, their main disadvantage is that other elements must be connected to the CPU for it to work properly. These devices are Random Access Memories (RAM), storage devices and the input/output peripherals, among others.

2. Microcontrollers: these devices include the CPU, the memories and input/output peripherals in the same IC or chip. Hence, no further elements are needed to implement the system, saving space and power, but their main disadvantage is that they are normally less powerful than the microprocessors. However, nowadays that difference in performance is decreasing.

3. Application-specific Integrated Circuits (ASIC): these ICs are specially designed to contain only the hardware (HW) that is needed for the processing of the environment, e.g. a special purpose core such as a Neural Network or a Markov Model, communication devices such as Universal Asynchronous Receiver/Transmitter (UART), Ethernet MACs or Controller Area Network (CAN) transceivers; and/or Analog to Digital converters. Their main advantage is their size and their performance. However, their main disadvantage is that their production is too expensive unless they are mass produced.

4. Programmable Logic Devices (PLD): these devices are ICs in which customized digital circuits can be implemented, with Field-Programmable Gate Arrays (FPGA) being the most recent devices. Compared with the ASICs, their main advantage is that they are cheaper with lower production numbers and that most current FPGA technologies are reconfigurable. However, their main disadvantage is that they have higher power consumption.

1.1.5 Example of Ambient Intelligence Environments

In the present section, some examples of IEs will be listed in order to understand better what an IE is. Three representative case examples are provided: smart houses, health environments, and the transport industry. There are, however, many more areas where AmI can be implemented.

Smart houses

A smart house is a good example for understanding the idea of an IE, after all, people spend a lot of time in houses and these are places where numerous sensors and actuators can be placed to make them smart. Today there are many projects involving smart houses; among them we can mention the following:

1. Ambient Lighting Assistance for an Ageing Population (ALADIN) [79]: this European project is focused on trying to understand how the intensity of the lighting of a house affects the wellbeing of the elderly people living there, for example the effects on sleep. The system consists in an open-loop control that reacts to the psycho-physiological data received from the user.

2. MavHome [3]: this system uses hierarchical Markov models of the user and the environment. It can automate the activities the user normally does manually, due to the capability of the system to predict those activities.

3. Gator Tech project [4]: this project was designed with the idea of creating an assistive environment for elderly or disabled people who aim to live independently. It also allows the users to be monitored remotely, giving the caregivers the opportunity of intervene remotely.

4. iSpace [5]: this project is a two-bedroom flat designed by the University of Essex. It emerged from the research obtained from the iDorm environment [80], a one-room dormitory. To control the devices in the apartment, learned Fuzzy rules were applied. The rules were developed using the observed activities of the inhabitant [30].

5. DOMUS [81]: this project was designed by the Computer Science Department of the University of Sherbrooke in Quebec, Canada. The main objective for the team responsible of this project is the creation of an environment based on pervasive assistants that can provide mobile orthosis.

Up to now, only projects from universities have been mentioned, but the private industry has also spent considerable resources on research to create efficient IEs. One example is the *Synco living* system [82] designed by Siemens, which has invested in the development of systems to improve entertaining, security and energy saving, for example by turning off all the lights and reducing the room temperature when all the inhabitants have left the house. Another example is the research done by Philips in its installations located in Eindhoven. There, the enterprise has developed a laboratory called *HomeLab* [83], an environment as close as possible to a real home. In the case of Microsoft, it has created a laboratory focused on how the Artificial Intelligence (AI) is capable of helping the users of the environment in their daily lives [84].

Health environments

There are many potential uses for AmI in health environments, but in this section two situations are mentioned: health monitoring at home, or transforming hospitals into intelligent environments (the entire building or only a pavilion of it).

1. Health monitoring at home: the idea is to help elderly or disabled people to lead independent lives in their own homes [6], [7]. This helps reduce their stress, because they do not need to rely on other people to perform daily activities. Also, it reduces the stress and workload of the caregivers [85].

2. Intelligent Hospitals: an example of an intelligent hospital is given by Kofod et al. [86] who describe the use of AmI to support health workers cooperating in the diagnosis of a patient and his/her treatment using context information. In [87], the idea is to create an intelligent hospital to help the doctors and nurses to manage their tasks better. Another example is described in [8], where the hospital is able to locate the doctors, warn them if a result they are waiting for has arrived and show it on the nearest terminal, saving time and effort.

Transportation

In the transportation industry AmI can be used in very different ways. One of these is to increase the comfort of the driver or the passengers. There are however, other different ways how AmI can be used in a vehicle.

One example of an AmI car is the *AwareCar* initiative which is being developed at the MIT AgeLab [88]. The AwareCar considers the driver as an active component in a state detection feedback system. It would detect driver state (fatigue or stress), display that information to the driver to improve the driver's situational awareness in relation to road conditions and their own 'normal' driving behaviours; and offer in-vehicle systems to refresh the driver, thereby improving performance and safety. The AwareCar is an instrumented vehicle built for evaluating new models and methods of monitoring driver state though physiology, visual attention, and driving performance in the field. The vehicle is used in different studies to assess the following: hands free cellular phone usage, surrogate measures of visual and cognitive distraction, driver health and wellness, as well as functional methods of assessing changes in workload, arousal and age-related stress [89].

Another important initiative is a recent NEDO [9] project that involves the international collaboration between universities in Japan (Nagoya Univ.), Italy (Univ. of Torino), Singapore (Nanyang Univ.), Turkey (Sabancı Univ.), and the USA (UT-Dallas). The main aim of the project is to understand driving behaviour using multi-channel sensor data from CAN-bus, GPS, video, audio and additional gas/brake pedal pressure signals. They apply state-of art signal processing methods to extract information and to build systems for driver identification, driver distraction detection and driver manoeuvre recognition. As a consequence of these efforts, future in-vehicle technologies will be designed to reduce the mental load of driving and decrease driver distraction via speech-prompted driver assist systems or semi-autonomous controller structures [31].

Talking about projects concerning the safety and wellness of the driver, in Europe between a 20-35% of accidents are due to driver fatigue [90], and that is the reason why the automotive industry has focused their goals in increasing the safety of the cars by monitoring the fatigue or the drowsiness of the driver. Companies like Mercedes-Benz (Attention Assist) [10], Ford (Driver Alert) [11], Volkswagen (Fatigue detection system) [12] or Volvo (Driver Alert Control) [13] among others, already have fatigue detectors in their cars to advise the driver that he/she should stop and rest.

Another factor in safety is helping people with heart problems. If, while driving they suffer a heart problem, for example a cardiac arrest, their chances of survival decrease with time [91]. To solve this problem, Ford in collaboration with the Rheinisch-Westfälische Technische Hochschule Aachen University [14] is developing a new system to detect wether the driver is suffering from a cardiac arrest. This system monitors the heart rate using sensors embedded in the seat that detects the electrical impulses of the heart through clothing. In the case of a heart attack, the car will stop safely and call the emergency services [15]. This system is still under development and the company has not published a roll-out date [92].

Other environments

There are other places where AmI can be used to create an IE. For example an intelligent classroom can use voice [93], gestures, or motion recognition techniques [94] to open the files of a presentation and/or to control lighting settings. Also it could be possible to use voice recognition to have the transcription of a lecture available minutes after it has been given [95], to help people with disabilities or problems taking notes. Benavides et al. propose in [96] an intelligent network called iNet to be used as a personal assistant in an intelligent university campus.

In the case of industry, an IE can be implemented in a factory so that the system is able to know if an accident has happened and take the needed measures, for example warning the factory supervisor [97]. Also a context aware factory, using a series of algorithms, can organize production depending on the demand for a product [61], [98], or based on the state of the factory, e.g. equipment malfunction [99].

1.2 Soft Computing: Artificial Neural Networks

In Section 1.1 an introduction to intelligent environments and the algorithms used to implement them has been made. This section begins with an introduction to Soft Computing (SC) techniques followed by an in-depth description of the Artificial Neural Networks (ANN). ANNs are very useful for modelling nonlinear systems, in particular, human behaviour.

1.2.1 Introduction to Soft Computing

SC is a series of techniques, according to Professor Lofti Zadeh, *"to define the computation that emulates the human mind's ability to reason and learn in an environment of uncertainty and imprecision"* [16]. This delimitation makes two statements, the first is that the data of the environment is imprecise, and the second that SC techniques are inspired by natural processes.

Due to the uncertainty present in the environment, classic or hard computing techniques cannot be used in this case, because their inputs and outputs must be clearly defined, so the presence of the lowest uncertainty in the data can cause the system to fail. Soft computing, on the other hand, offers a series of different techniques to compute this information in a simpler way, obtaining good results and also the capability to *"learn from experience, and adapt to changes in the operation conditions"* [100].

Some of the most important SC techniques are listed below:

- Fuzzy logic: fuzzy sets theory was established by Zadeh [101] in the 60s and it was the base for the development of fuzzy reasoning. Fuzzy logic's main feature is its capability to work with imprecise information and as a universal function approximator [102], [103], [104]. In recent years, Fuzzy Inference Systems (FIS) have been used in a wide area of applications, especially in control mechanisms [105]. A FIS is formed by an inference mechanism, a set of IF-THEN type linguistic rules and information converters: a fuzzifier [106] and a defuziffier [107], [108], [109], [110]. These converters are needed between the environment, which uses crisp information, and the fuzzy system in which the information is presented by fuzzy sets.

- Artificial Neural Networks: these are networks formed by processing elements (i.e. neurons) as proposed by McCulloch and Pitts in 1943. They are interconnected to emulate living beings' brains [17]. Their main advantage is their capability to learn and adapt from data samples [111]. Neural Networks are efficient in pattern recognition and classification [112], [113], function approximations [114], data clustering [115] and vector quantization [116].

- Genetic Algorithms: these are searching or learning algorithms based on the mechanics of natural selection, genetics and evolution. Their principles were first published by Holland in 1962 [117] and the mathematical framework was presented in 1975 [118]. In Genetic Algorithms, every chromosome is shown as a string of binary numbers. The learning is based on the process of *Selection* and *Reproduction* (where the chromosomes are recombined simulating sexual reproduction) and *Mutation*, in which a gene of the chromosome is altered randomly. They are used in parameter learning (e.g. the parameters of a neural network [119]), path planning [120], [121] or system control [122], [123].

These techniques can be combined to take advantage of each technique and overcome their disadvantages. An example is Neuro-Fuzzy systems [124], [48]. They have the advantage of showing information in a linguistic way, which is easily understandable as fuzzy systems and the learning capabilities of neural networks. Another way of using different SC techniques together is presented in [125]. Instead of combining both techniques in one core, a FIS is used to pre-processes the data, the contrast of fingerprint images, to increase the recognition rate of an ANN classifier.

1.2.2 Neural Networks

Biological Neurons

Neurons are the basis for any Neural Network, either biological or artificial. In the case of the biological neurons, they make the nervous system and their function is very different in comparison with the rest of the cells of the body. Basically, their function is to receive the signal from one or more neurons and to generate only one output. For that, the neuron has three different parts as can be seen in Figure 1.2, [126]:

- Dendrites: these are the inputs of the neuron, responsible for receiving the electric stimulus from other neurons. A neuron can have one or thousands of dendrites depending on the number of neurons which are connected to it.
- Soma: this is the body of the neuron where the nucleus and the rest of the organelles are located.
- Axon: this is the output of the neuron. As there is only one output per neuron, the axon will ramify to be able to connect to other neurons of the net it belongs to.

One important aspect about neurons is that a single neuron does not have intelligence; it is the combination of neurons, the neural network, which allows intelligence to surface.

Artifical Neurons

The mathematical model of an artificial neuron proposed by McCulloch and Pitts in 1943 [17], see Figure 1.3, is based on the morphology of a natural neuron.

- Dendrites (x_i): these are the inputs of the neuron. They receive the inputs of a system or the outputs of other neurons.
  - Weights (w_i): each input has associated to it a value, called weight, which is responsible for giving more or less strength to this input over the others [127]. If the value of the weight is 0, it means the dendrite is not connected to any other neuron.

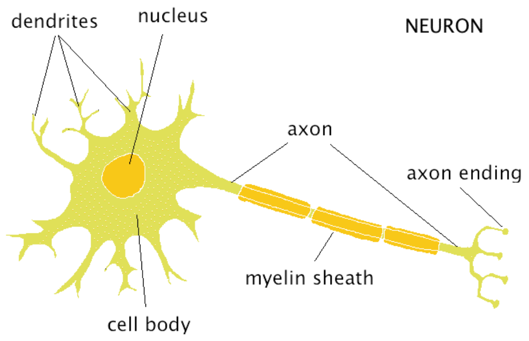


Figure 1.2: Biological neuron morphology

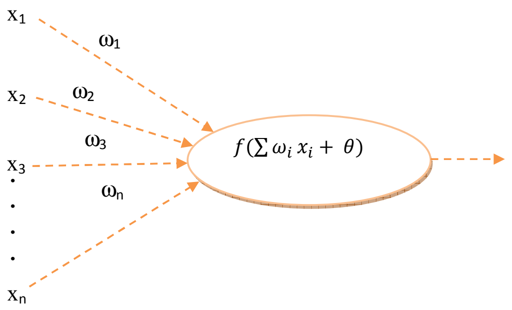


Figure 1.3: Artificial neuron morphology

- Soma: this is the body of the neuron. In the case of an artificial neuron, it is responsible for doing the sum of all the weighted inputs and an offset. After that, a transference function, also known as *activation function*, is applied to that value. The activation function can be of various types, but the most common are: lineal, step, unipolar sigmoid and bipolar sigmoid [127].
  - Offset (θ): this value is responsible for adding a threshold to the input/weight value set of the dendrites.
- Axon: this is the output of the neuron. It has no function except for being the path of the output data.

The output of the neuron is given by the following equation:

$$y = f(\sum_{i=1}^{n} x_i w_i + \theta) \tag{1.1}$$

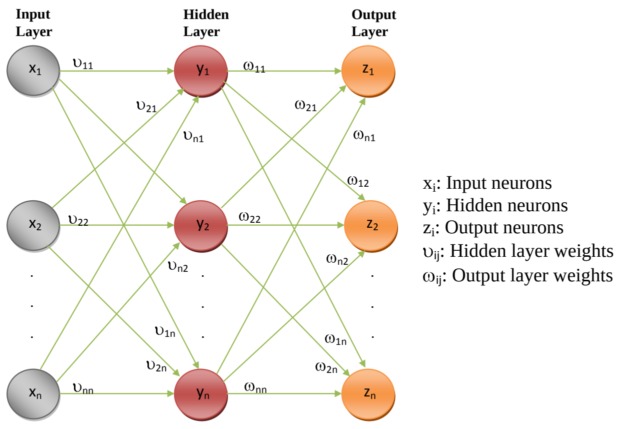


Figure 1.4: Artificial Neural Network scheme with one hidden layer

As with biological neurons, a single artificial neuron has limited capabilities. For example, a neuron with a step activation function, called basic perceptron, cannot solve an XOR function because perceptrons can only solve problems that are lineally separable [128].

Artificial Neural Network Structure

An ANN is just a number of neurons interconnected together, forming a structure capable of performing certain functions. In the case of a biological neural network, this function can be something so simple as a reflex reaction, like an anemone closing its tentacles when it is touched; or a more complex function like the human brain, where the number of neurons can be as many as 10^{11} organized in different scales and levels [129]. Nevertheless, the number of neurons of an ANN is closely related to the number of inputs and outputs of the network. This can be seen in Figure 1.4 in which an ANN is shown. A common way to represent ANNs is using graphs, where each node is a neuron and the arrows are connections between neurons. Neurons are commonly organized in layers. Each layer is independent, and this means that each of them can have neurons with different activation functions; for example, the neurons of one layer can have a lineal activation function, while the neurons of the following layer can have step activation functions.

The most common ways of organizing the layers are listed below:

- Input layer: the neurons of this layer receive the input of the system. They only have one dendrite, connected to the input, and their axons are connected to the inputs of all the neurons of the next layer.
  This layer acts as a buffer of the network, that is to say, there are no

weights or offset and the input does not change; it only connects the input to the internal layer.

- Hidden layers: there can be one, several or no hidden layers. These layers are the ones next to the input layer. The dendrites of its neurons come from all the axons of the neurons of the previous layers and in this case, the dendrites are weighted. Depending on the number of neurons present in this layer/s the performance of the neural network can vary significantly.

- Output layer: this layer is connected to the output of the system. The dendrites of the neurons of this layer come from all the axons of the neuron of the previous layer. In this case the axons of the output neurons do not ramify because they are connected to only one element, the outputs of the system.

ANNs Classification

Depending on the number of layers and the connections between them, ANNs can be grouped in three generic architectures:

- Single layer feed-forward networks
- Multi layer feed-forward networks
- Recurrent networks

1. **Single layer feed-forward networks:** these networks have only two layers, an input and an output layer. In this case, it is called *single layer* because the input layer acts as a buffer and only the output layer is responsible for performing the computations (see Figure 1.5). The data flow goes only in one way, from the input to the output, which is why this network is called *feed-forward.*
   Within this type of network architecture, the following networks can be cited: *Perceptron* [130], *Adaline* [131], *Hopfield* [132], [133] and *Associative Memory* [134], [111].

2. **Multi layer feed-forward networks:** in this architecture, presented in Figure 1.4, one or more hidden layers appear. In this case, both, hidden layers and output layers, perform computations. Multilayer Perceptron Neural Networks (MLP) are networks which consists of multiple layers of computational units, perceptrons, usually interconnected in a feed-forward way. These types of networks are described in-depth in the following sections.

3. **Recurrent networks:** in the previous architectures, the data flow goes from the input to the output, but if at least one feedback loop is added connecting the output with the input, then the architecture is known as Recurrent Network. An example is shown in Figure 1.6.

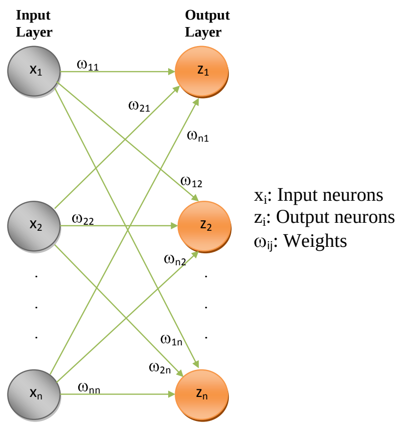


Figure 1.5: *Feed-forward* single layer perceptron neural network scheme

Adaptive Resonance Theory (ART) structures are one of the most important architectures of the recurrent neural network type. These networks respond to arbitrary input patrons, recognizing and self-organizing codes in real-time. Related to these architectures ART [135], ART2 [136] and ART3 [137] can be cited. The mentioned networks are used in pattern classification and also in forecasting. For example, to forecast the price or demand of electricity [138], [139] and [140], or the price of the stocks in the market [141], [142].

1.2.3 Learning Algorithms

There are two different types of learning strategies: parameter learning and structure learning. The most widely used is parameter learning, in which the values of the weights of the dendrites are modified. In the case of structure learning, the structure of the network is modified, that is to say, the number of neurons, the connections or the activation function are learned [143], [144], [26]. These learning methods can be used separately or together.

In the case of the parameter learning, three different categories can be distinguished:

1. Supervised learning: in this learning method, the system must provide the network with a set of desired input/output sample set, and then, a supervision mechanism determines the error between the output given by the network and the desired output (target), and adjusts the parameters of the network (weights and offsets), in order to reduce that error. This process is done with all pattern vectors and if the parameters are not correctly adjusted, another learning cycle starts. This can be done repeatedly until a maximum allowable error is reached or until a certain number of learning cycles is reached.

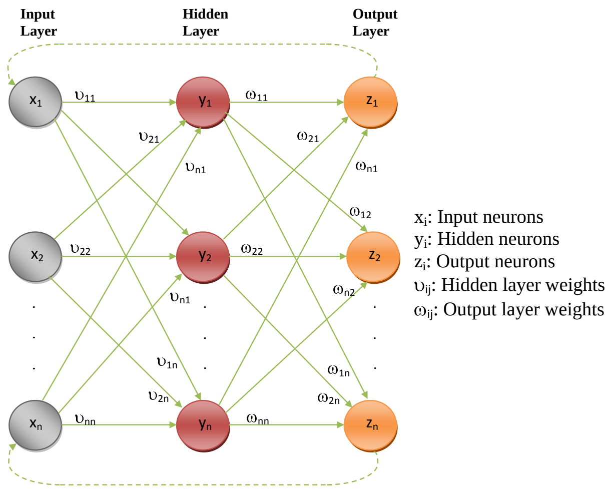


Figure 1.6: Recurrent neural network scheme

2. Unsupervised learning: in the case of unsupervised learning, the network does not receive any desired target, so the network must discover it by itself. To achieve that, a series of algorithms is available so the network can adapt the weights by characteristic search, correlations or categories in input data responding to the pattern that appears more frequently. This latter case is known as *probabilistic estimator*. Among the algorithms, Hebbian's [111] stands out, with several contributions over a number of years [145], [146], [147] and [148].
   Another unsupervised learning algorithm is the one applied in self-organizing systems. They modify their connection strengths based only on the characteristics of the input patterns. Kohonen's self-organized map is the simplest of the organizing systems [149]. It is a single layer network (called Kohonen layer) whose neurons are highly interconnected (lateral connections) within that layer and to the outside world using an input buffer layer that is fully connected to the Kohonen layer neurons using adjustable weights [150].

3. Reinforced learning: this method does not give an exact error between the given and desired output. Instead, it gives information similar to *output higher/lower than the desired one or a percentage of the outputs are correct* and in more critical cases, the indicator only informs if the output is correct or not with only one bit. The signal which transmits that information is known as *reinforcement signal*. Using the information given by the reinforcement signal, the system varies the weights to obtain the desired output using probabilistic criteria [151].
   This method is called sometimes *learning with a critic*, while the supervised learning is called *learning with a teacher*. In [152], [153] some pro-

posed learning architectures can be seen.

Backpropagation Algorithm

One of the main learning algorithms used in MLP supervised learning is the Backpropagation (BP) algorithm. This algorithm was initially presented by Werbos in 1974 [154] and afterwards was developed by Parker in 1982 [18] and Rumelhart et al. in 1986 [19]. It is based on the variation of the weights proportionally to the gradient of the error. This learning algorithm is called Backpropagation because the error spreads across the network but in the opposite way to the data flow, that is to say, from the outputs to the inputs.

The algorithm is based on the minimization of the existing error between the desired output and the real output of the net. Given a set of K learning samples, the error for a sample, E_k, can be expressed in its quadratic form:

$$E_k = \frac{1}{2}(y_k - t_k)^2 \tag{1.2}$$

where y_k is the output of the network for a x_k input, and t_k is the desired or target output. The variation of each weight of the network, $\triangle w_{ij}$, is obtained through the derivative of the error with respect to the weights. That is why this method is known as Gradient Descent.

$$\triangle w_{ij} = -\eta_i \frac{\partial E_k}{\partial w_{ij}}. \tag{1.3}$$

The parameter η_i represents the learning rate of the network associated to variable x_j that in some cases can be variable [155], [156], and $\frac{\partial E_k}{\partial w_{ij}}$ is the gradient of the error with respect to the weight w_{ij} of the network. A very low value of η_i makes the error convergence better, but very slow and high values make the convergence faster but oscillations around local minima can appear. Also, to simplify the algorithm, the same learning rate value is used for the entire network η.

Taking into account Equations 1.2 and 1.3, and applying the *Chain Rule,* the following equation is obtained:

$$\frac{\partial E_k}{\partial w_{ij}} = \frac{\partial E_k}{\partial y_k}\frac{\partial y_k}{\partial w_{ij}} = (y_k - t_k)\frac{\partial y_k}{\partial w_{ij}}, \tag{1.4}$$

in which y_k depends on w_{ij}, so, it is mandatory that fuction y_k would be derivable respect to the parameter w_{ij}. Some functions with points where the derivative is undefined (e.g. ramp or triangular functions) can be used if the algorithm avoids the derivative in those points [157]. Hence, using Equation 1.3 with the same learning rate for the entire network, the variation of the weights is given by the expression

$$w_{ij}^{'} = w_{ij} - \eta(y_k - t_k)\frac{\partial y_k}{\partial w_{ij}}, \tag{1.5}$$

where the superscript indicates the update of the parameter. With the new parameters obtained, the learning cycle can be repeated until a suitable error is reached.

If all input-output patterns are taken into account to compute the error, Equation 1.2 has the following expression

$$E = \frac{1}{2K}\sum_{k=1}^{K}(y_k - t_k)^2. \tag{1.6}$$

The learning can be done in two different ways. In the first one, only some training vectors are used and the process is carried out during the working state of the network using Equation 1.2. This algorithm is called *on-line learning*. However, when the entire number of learning vectors is available, the process can be carried out before the network enters in working state. In this case, Equation 1.6 is used and is called *off-line learning*.

As was said, the variation of the value of the parameter η makes the learning process to be faster or slower. However, there are other ways to accelerate it. One method consists in including a momentum in the variation of the weights, that is to say, adding in Equation 1.3 the value of $\triangle w_{ij}$ in the previous learning cycle, multiplied by a so-called Momentum Factor, α, as can be seen in Equation 1.7. There is no rule about selecting the value of α, but it is recommended not to use values above 0.9, which is also the recommended value to obtain the best results [150], [127].

$$w_{ij}^{'} = w_{ij} + \triangle w_{ij}^{'} + \alpha\triangle w_{ij} \tag{1.7}$$

Another method to speed up the convergence is to initialize the weights before the learning with random numbers. An evolution of this method is the Nguyen-Widrow algorithm [158]. It not only uses random numbers to initialize the weights of the network, but also takes into account the architecture of the network, that is to say, the number of inputs, hidden neurons and outputs of the network. The evolution of the error using this algorithm and compared with an initialization of the weights using constant and random numbers can be seen in [159].

First a random matrix weight is computed,

$$\mathbf{A} = (rand(L, M) - 0.5)2, \tag{1.8}$$

where L is the number of nodes in the layer and M is the number of nodes in the previous layer. Then, the sum of all values of the matrix are computed

$$p = \sum_{i=0,j=0}^{i=L,j=M} a_{ij}. \tag{1.9}$$

After that, a d value is obtained as

$$d = 0.7(M^{(1/N)}). \tag{1.10}$$

Finally, the weight matrix, $\boldsymbol{\omega}$, is obtained as

$$\boldsymbol{\omega} = ((d\mathbf{A})/\sqrt{p}). \tag{1.11}$$

In the case of the offsets, they are computed as

$$\boldsymbol{\theta} = (rand(L) - 0.5)2d. \tag{1.12}$$

Extreme Learning Machines

Extreme Learning Machine (ELM) is a novel supervised learning algorithm developed by Huang [20] in 2006 for Single-hidden-layer Feedforward Neural Networks (SLFN), with the peculiarity that the neurons of the output layer have no activation function nor offsets, and only one learning cycle is needed to obtain the value of the weights of the output layer, while the weights of the hidden layer are random numbers that do not change, making it faster than the BP algorithm.

Let us consider a generalized SLFN with n inputs, m outputs, and L nodes in the hidden layer. The network output for generalized ELM is

$$y(\mathbf{x}) = \sum_{i=1}^{L} b_i h_i(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta}. \tag{1.13}$$

Without loss of generalization, a single output node ($m = 1$) is taken in Equation 1.13. The vector of weights $\boldsymbol{\beta} = [\beta_1, \cdots, \beta_L]^T$ links the hidden nodes (i.e. random nodes) with the output node, and $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), \cdots, h_L(\mathbf{x})]$ is the output vector of the hidden layer for a given input $\mathbf{x} \in \mathbb{R}^n$.

The output of the ith hidden node is

$$h_i(\mathbf{x}) = S(\mathbf{a_i}, b_i, \mathbf{x}), \mathbf{a}_i \in \mathbb{R}^n, c_i \in \mathbb{R} \tag{1.14}$$

with $s(\mathbf{a_i}, b_i, \mathbf{x})$ being the sigmoid activation function, $\mathbf{a}_i$ the random weight vector connecting the inputs with the ith hidden node, and b_i the random offset of the ith hidden node. It is worth noting that any piecewise continuous function satisfying ELM universal approximation capability could be used as activation function.

Given a set of K training samples, $(\mathbf{x}_j, \mathbf{t}_j), 1 \leq j \leq K$ where $\mathbf{x}_j \in \mathbb{R}^n$ is the jth input vector, and $\mathbf{t}_j \in \mathbb{R}^m$ is the corresponding output vector (i.e. the target output), learning is performed by solving Equation 1.13 for the set of training samples

$$\mathbf{T} = \mathbf{H}(x)\mathbf{B} \tag{1.15}$$

where $\mathbf{H}$ is the hidden layer output matrix, as can be seen in Equation 1.16.

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}(\mathbf{x}_1) \\ . \\ . \\ . \\ \mathbf{h}(\mathbf{x}_K) \end{bmatrix} = \begin{bmatrix} h_1(\mathbf{x}_1) & . & . & h_L(\mathbf{x}_1) \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ h_1(\mathbf{x}_K) & . & . & h_L(\mathbf{x}_K) \end{bmatrix}_{KxL} \tag{1.16}$$

$$\mathbf{B} = [\beta_1 \cdots \beta_m]_{Lxm} \tag{1.17}$$

and

$$\mathbf{T} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_K \end{bmatrix}_{Kxm} . \tag{1.18}$$

Then, the matrix of output weights is

$$\mathbf{B} = \mathbf{H}^{-1}\mathbf{T}, \tag{1.19}$$

where $\mathbf{H}^{-1}$is the Moore-Penrose generalized inverse of matrix $\mathbf{H}$.

This algorithm demonstrates its utility for very large ANNs because only one learning cycle is needed to compute the output layer weights, whereas in the case of the BP algorithm thousands of cycles may be required. Also ELM does not present local minima problems and overfitting. As a result, even if the computation of the Moore-Penrose generalized inverse is very computationally demanding, it will be faster than BP because it is only computed once and the weights and offsets of the hidden layer nodes are not changed by the learning process.

Structure Learning

It is common practice to select the size of the hidden layer by taking into account user experience or empirical experiment. This approach, however, is unsuitable for developing autonomous adaptive systems. Different solutions have been proposed in the literature to select efficient MLP topologies:

1. Incremental, constructive or growing algorithms. These approaches start with a small number of hidden neurons and increase their number iteratively to satisfy the required modeling performance [160], [143]. The most commonly used performance criteria are the training error and the validation error. The former is less demanding from the computation viewpoint, while the latter provides better generalization capability.

2. Decremental or pruning algorithms. In contrast with the previous one, these methods start with a large hidden layer. Then, the less significant hidden neurons and their corresponding connection weights are iteratively

pruned. A simple prune method consists in evaluating the effect of each weight on the neural network performance. This brute-force or exhaustive method [161], is very time-consuming because it requires a systematic evaluation of each one of the connection weights of the network.

3. A combination of growing and pruning. This approach proposes a combination of the previous one [21]. It is suitable for the development of neural networks with adaptive topology; the neural network size can be adapted to efficiently fit changing conditions.

In addition, several topology selection algorithms, based on evolutionary algorithms, and sensitivity measure methods have also been proposed in the literature (see [162], and references therein), but they are outside the scope of this work since they are too complex for an adaptive FPGA-based implementation.

1.3 Artificial Neural Network Implementation

Neural networks can be implemented by software (SW), hardware (HW) or by a combination of both (HW/SW). In the present section, different approaches to the implementation of ANN will be analyzed.

1.3.1 Software Artificial Neural Networks

The main characteristic of software neural networks is that they are executed on microprocessors or in microcontrollers. As is said, in Section 1.1.4, these devices execute the software programs sequentially, instruction by instruction, without parallelism. One advantage is that the ANN can be defined using different notations, with the matrix notation being one of the most common [163], [164], [165]. Therefore, is very easy to develop a software application independent of the platform, making it portable, or with very small developing times if changes have to be made. Also, in the case of any change in the network, only the SW code has to be changed, making the SW implementation easier and more flexible. A SW approach is the best solution for applications with mild restrictions concerning size, power consumption and speed.

However their main disadvantage is their lack of real parallelism if high speed is required. In the case of using a processor with a Harvard architecture, the only possible parallelism is the simultaneous access to data and instruction memories at the same time, but in the case of a von Neumann architecture this is not possible. Another way to obtain a degree of parallelism is the use of multiple cores. Using this technique it is possible to have different cores doing different parts of the calculations obtaining some parallelism. In the case of PC based platforms, this is possible because there are many processors on the market with multiple cores, for example Intel's i7 cores [166]. However, in the case of the embedded platforms this is not very common. Today the most widely used embedded multiple core devices are the ARM devices with 2 or 4 processing cores [167], [168].

Another problem in a software solution is that the microprocessor executing the code must have a very powerful Arithmetic Logic Unit (ALU) to be able to compute the operations in a given time, otherwise to perform the calculations, the system must compute them using software functions using simpler operations, which is much slower. One example are Freescale's ColdFire 32 bit family processors: according to Freescale's webpage, only V4 generation microprocessors have a Floating Point Unit (FPU) (i.e. an ALU to perform floating point operations) integrated in the silicon [169]. This means their performance computing neural network equations will require a lot of processing. In some cases, the neural network has been implemented using Digital Signal Processor (DSP). These devices are equipped with fixed-point or floating point arithmetic units that perform the calculations faster than an embedded microprocessor [170], [171].

One last aspect is the length of the variables used to define the fractional numbers. For example, in C programming language, fractional numbers are defined in 32, 64 or, 128-bit variables in its standard library. It is therefore not possible to define fractional numbers with shorter word lengths and, in the case of numbers with low precision, using 32 bit variables is a waste of memory resources and execution time.

1.3.2 Hardware Artificial Neural Networks

HW implementations are used when low silicon area occupation, low power consumption and high speed are decisive factors [172] to cope with the system specifications.

There are two implementation possibilities, one using digital hardware, and the second one using analog hardware. Analog ANNs have the advantage that the devices have a non-linear behaviour, therefore, the activation functions can be implented more easily. Also, in the case of the size, analog devices are generally smaller than their digital conunterparts [173]. Unfortunately, analog ANN are very sensitive to temperature and voltage changes; some analog implementations are not easily programmable, and they have certain problems related to noise immunity [174]. On the other hand digital ANN, are robust against those changes.

The main advantage of using a hardware solution is the different types of parallelism:

- Node parallelism: this parallelism corresponds to the neurons and it is the most important level of parallelism. The neurons of each layer work in parallel computing their values simultaneously [173].
- Weight product parallelism: in this case, the input weight products in Equation 1.1 are computed in parallel.
- Bit-level parallelism: this parallelism is related to the communication used in the I/O data transfer of the network. This can be serial, serial-parallel, word-parallel, etc.

In the case of an analog ANN, the weights are stored using resistors [175], charge-coupled devices [176], capacitors [177] or floating gate EEPROMs [178]. Using variable analog devices it is possible to change the values of the weights allowing the implementation of a training algorithm.

In the case of digital ANNs, the main advantage is the number of devices present on the market to implement them, the well-known fabrication techniques, the storage of the weights in RAMs and the flexibility in the design of the ANN.

However, the main disadvantage of the digital solutions is the numbers of resources needed to perform the computations of the network and the learning algorithm. As can be seen in Equation 1.3, by using a Backpropagation learning algorithm, the derivative of the error must be computed, so increasing the complexity of the design. In [179], the learning algorithm is performed in hardware, and as a result of that, for a neural network of 8 inputs, two hidden layers with 6 neurons each and only one output, 92 18x18 bit multipliers are needed, when, normally, one multiplier per neuron is used. Other examples are also [180], [181], [182], [183].

To implement a HW neural network, one option is to use an ASIC [184], [185]. These devices have a fixed design and cannot be changed. Normally they are faster, they have lower unit costs for very high volume designs and a smaller form factor [186]. Another option to implement ANNs in HW is the use of PLDs, because there are reconfigurable devices on the market allowing for the possibility of changing the design.

Finally, there is another solution to try to overcome the problems of both designs: a hybrid analog/digital ANN, for example using a fast analog processing unit and storing the weights and the offsets digitally [187].

1.3.3 Hardware/Software Artificial Neural Networks

As has been said in the previous sections, the main problem of the SW neural networks is the way the code is executed, sequentially, making a parallelism nearly impossible. In the case of HW neural networks, the main problem is the number of resources needed to implement the networks and their learning algorithms, making them not suited for small device implementations.

When high speeds are required and the system must be small, the HW/SW co-design arises [188]. HW/SW co-design proposes the partition of the computation algorithms into HW and SW blocks by searching for the partition that optimizes the performance parameters of the whole system. This approach provides an optimal solution for many systems where a trade-off between versatility and performance is required.

An example of the HW/SW solution consists in using General-purpose Computing on Graphics Processing Units (GPGPU), e.g. Nvidia's CUDA platform [189]. These processing units were designed to compute computer graphics, floating point operations, freeing the CPU from that task using hundreds of cores that can handle thousands of threads simultaneously, see Figure 1.7 [190]. The SW partition, (i.e. executed in the CPU) deals with the high preci-

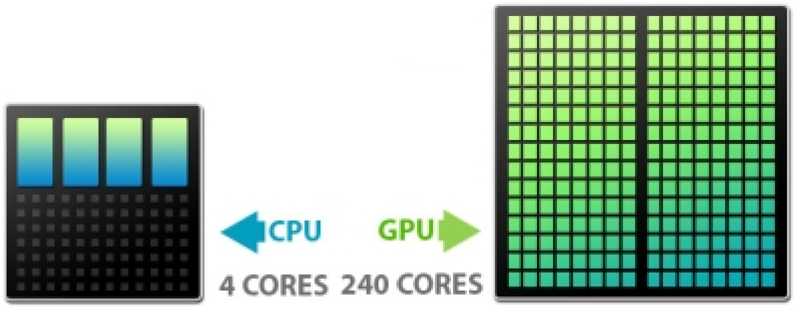


Figure 1.7: Difference between a CPU with four processing units and a GPU with 240 floating point processing units

sion operations (e.g., training algorithms) while the recursive and parallelizable computations are executed in the GPGPU. This solution is used for very large ANNs, very large learning sets or for networks with very complex architectures, e.g. convolutional neural networks [191]. Therefore it is possible to have a system having node and/or weight parallelism and specially designed to perform floating point operations.

As a result, the timing performance can be dramatically improved [192], [193], [194]. Unfortunately, in this case the GPGPUs are not integrated in the motherboard of the PC and must be connected to the processor using expansion slots.

In the case of using a single-chip solution, one of the newest options is the use of a System on Programmable Chip (SoPC), explained in Section 1.4. Using microprocessors, the software part can be implemented within the SoPC, for example, to perform the learning algorithm, and then, using the logic capabilities of the device, to design a custom hardware coprocessor to accelerate the computations. It could be also possible to perform weight parallelism, but in this case the number of resources needed by the HW part would be very big and the design, depending on the architecture of the network, could be too complex. Another advantage of the SoPCs is that all the HW is within the very same chip; no other elements are needed [195], [48], saving space and energy consumption.

1.4 Programmable Logic Devices

Programmable Logic Devices (PLD) can be defined as *"Standard, off-the-shelf parts that offer customers a wide range of logic capacity, features, speed, and voltage characteristics - and these devices can be changed at any time to perform any number of functions"* [196]. In the beginning, these devices were very simple allowing only the implementation of small designs, but over time, the devices became more complex, faster and easier to use.

The first PLDs were very simple, thus, they were known as Simple Pro-

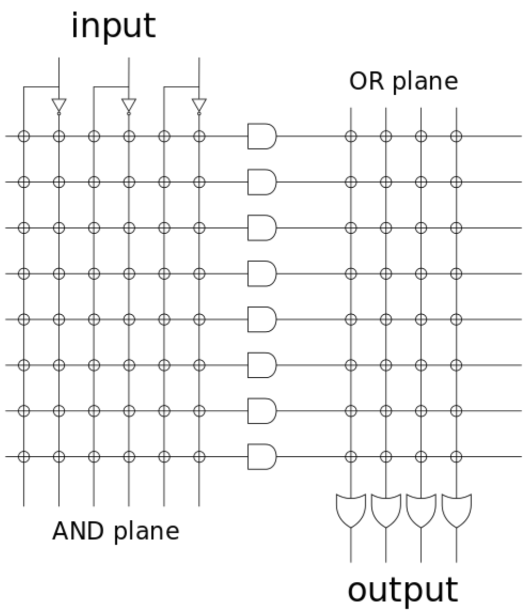


Figure 1.8: Programmable Logic Array AND and OR planes with their interconnection points

grammable Logic Devices (SPLD). Two examples of these devices are Programmable Logic Arrays (PLA) [197] and Programmable Array Logic (PAL) [198]. These devices had a set of AND gate planes followed by a plane of OR gates. In the case of the PALs, only the AND plane was programmable, whereas with PLAs both planes were programmable, see Figure 1.8 [199]. Later, macrocells were added to the PAL architecture. Using a 22V10 PAL as an example, the macrocell can be programmed to provide a registered or non-registered inverting or non-inverting output, Figure 1.9, [200].

The next evolution in PLDs came with the so-called Complex Programmable Logic Devices (CPLD). CPLDs are more complex and are formed by SPLDs interconnected by an internal net [201]. These devices are capable of performing more complex functions and, because of the fixed and predefined length of the interconnection net, the propagation time of the signals is predictable. CPLDs also have input/output blocks with memory registers (Type D Flip-Flops, FFD) and tri-state buffers.

The most recent evolution of the PLDs are the Field Programmable Gate Arrays (FPGAs). FPGAs do not implement the logic functions using logic planes as the SPLDs or the CPLDs; they use logic elements based on Look-up Tables (LUT) and multiplexers instead, as can be seen in Figure 1.10, [202]. This allows the implementation of complex designs, but unlike CPLDs, in this case the propagation time cannot be predicted. The logic elements (*Slices* in Xilinx's nomenclature and *Adaptive Logic Module* in Altera's nomenclature, the

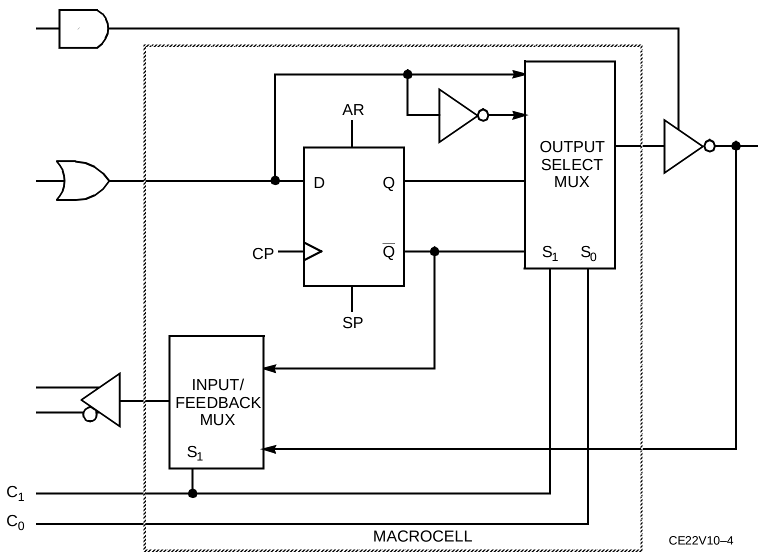


Figure 1.9: 22V10 PAL Macrocell, showing its register and the inverter of the output

principal FPGA manufacturers), are typically formed by a LUT that performs the combinational logic functions, bistable elements (i.e. FFDs) and the logic to perform the internal routing of the signals. Apart from these elements, the manufacturers began adding embedded cores in the silicon of the devices. In the beginning, hardware (HW) multipliers and RAM memory blocks were added. Later the multipliers were replaced by Digital Signal Processor (DSP) blocks, that implement high performance sum of products (see Figure 1.11, [203]). The use of these DSPs allowed the creation of more complex designs using fewer resources, achieving a decrease in size, resources and power consumption.

In some FPGA models such as Xilinx's Virtex 4 and Virtex 5, or Altera's Cyclone, a PowerPC or an ARM microprocessor was embedded respectively in the silicon. The evolution in the FPGAs also meant that it was possible to implement the communication buses within the FPGA and internal peripherals in the same chip, thus creating the so-called System on Programmable Chip (SoPC). However, the use of hard microprocessor cores embedded in the silicon has been discarded in favour of another evolution, the implementation of the microprocessors using the resources of the FPGA. These microprocessors are known as soft microprocessors, e.g. Altera's NIOS [22] or Xilinx's MicroBlaze [23]. These microprocessors are not as powerful as their hard counterparts, but, it is possible to create a system whose performance can be good enough to fulfil the requirements of a real-time environment using resources in a more efficient way. This opens up new possibilities: being able to include software partitions in the FPGAs, being able to take advantage of the SW applications, and of

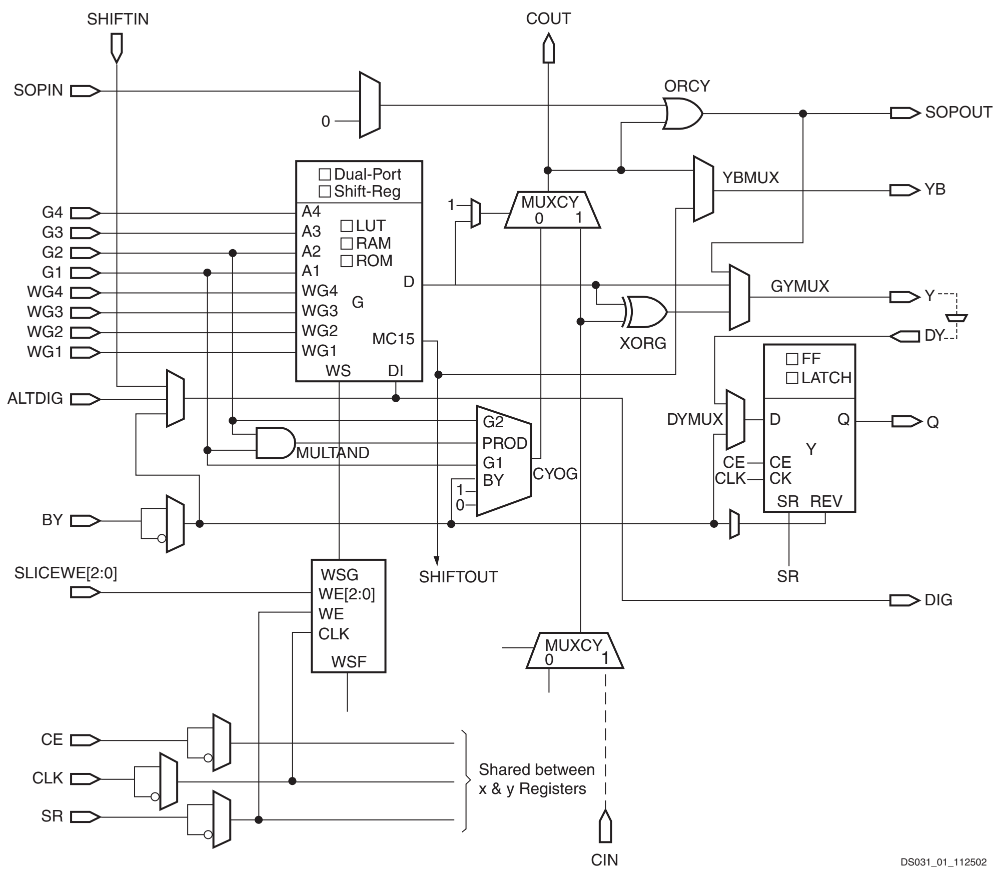


Figure 1.10: Virtex II Slice

being able to create custom HW.

Today FPGAs offered by Xilinx do not include a hard microprocessor in the Spartan 6 [204] and 7 Series FPGAs [205]. Recently, a new product has been offered, the so-called *All Programmable SoC* [206]. These devices have an ARM A9 [167] dual core processing unit with peripheral and memory interfaces embedded in the silicon, and attached to it, a small reconfigurable logic region based on the 7 Series architecture.

In the case of Altera, the approach is different. Their so-called *Altera SoC* products also have an ARM A9 dual-core attached, but, instead of implementing a small logic part, the processor is included within the FPGA of their Cyclone, Arria and Stratix families [207].

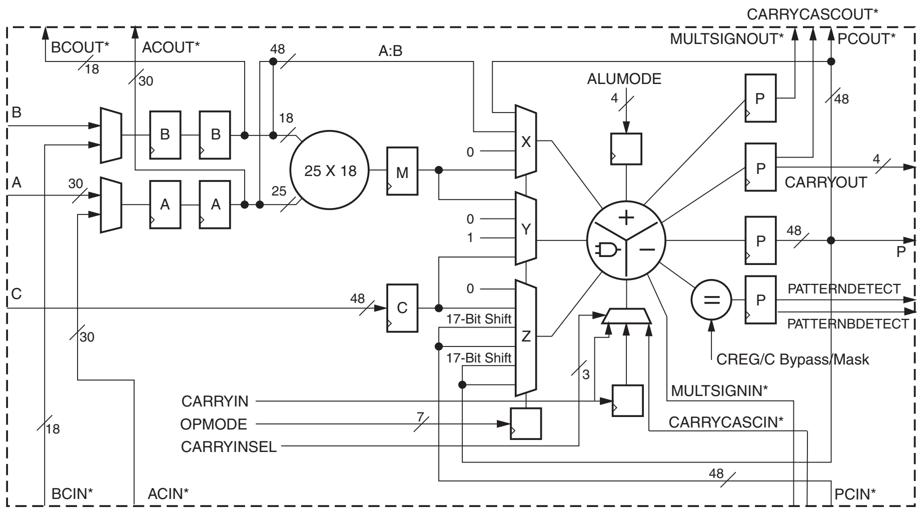


Figure 1.11: Xilinx DSP48E diagram. The core can be configured to perform addition, subtraction or logic operations

1.4.1 FPGA Configuration Technology

One way to classify FPGAs is according to the technology used to interconnect the logic elements. This is important because some methods are permanent, that is to say, after configuring the FPGA, the configuration cannot be changed. Other methods allow for changing the configuration, but they can be based on volatile or non-volatile technologies. The technologies are the following:

- SRAM: this method uses static (SRAM) memories to create the connections of the FPGA (see Figure 1.12, [208]), this means the configuration is erased when the system is shut down, so, it is mandatory to download the configuration bitstream every time the system is powered-up. Most of the SRAM FPGAs can be booted using an external or internal ROM memory [209], in which the bitstream is stored. This technology has been selected by the two main FPGA manufacturing companies (Xilinx [210] and Altera [210]).

- Anti-Fuse: this technology is One-Time Programmable (OTP), which means that after configuring the FPGA, its configuration is maintained permanently and cannot be altered again. This technology instead of melting the not needed connections, like the fuse technology used in SPLDs devices, creates the connections needed during the programming (see Figure 1.14, [208]). Their main advantage is that the device is Live-At-Power-Up (LAPU); it is operational at the precise moment the voltage supply reaches the minimum working value, because there is no need to load a

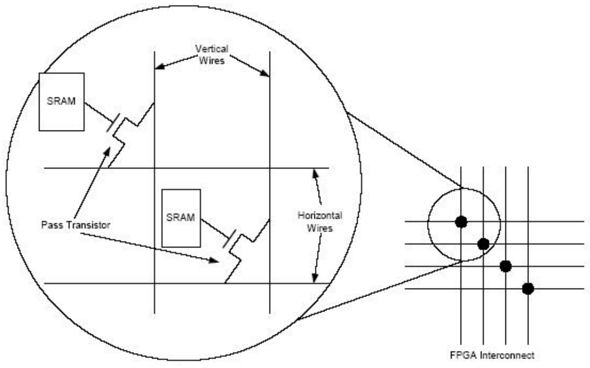


Figure 1.12: SRAM FPGA interconnection detail

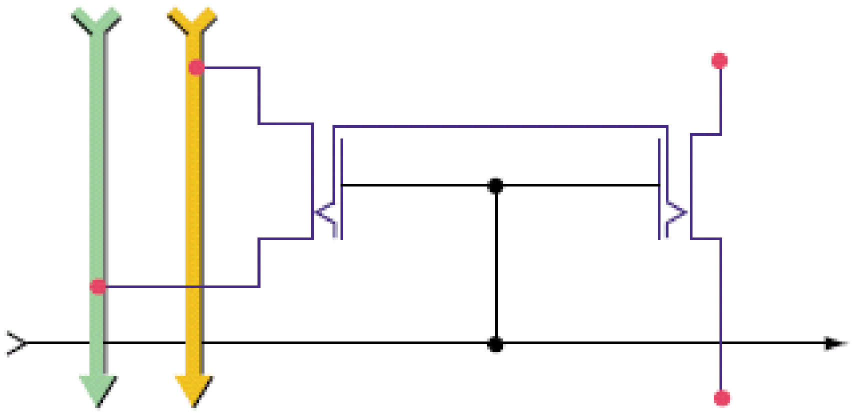


Figure 1.13: Flash FPGA interconnection detail

configuration bitstream every time the system is started-up. This technology is the one with the lowest power consumption, with the highest clock frequencies and also has the highest immunity to Single Event Upsets (SEU) [211], [212].

- Flash: this method uses Flash memories instead of anti-fuse connections. Using these memories it is possible to have a non-volatile but reprogrammable system (see Figure 1.13, [213]). The advantage of the Flash FPGAs is that the designer has a LAPU device but whose configuration can be changed whenever the designer wants. This technology has been selected by Microsemi and Lattice.

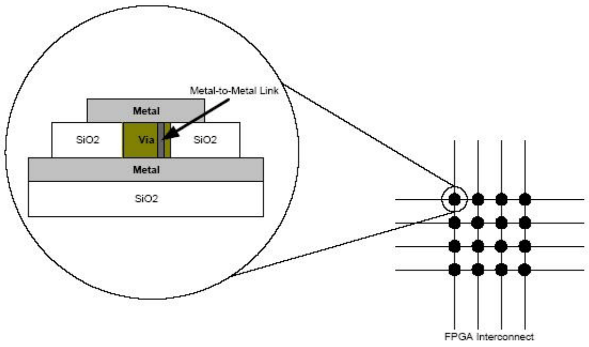


Figure 1.14: Anti-Fuse interconnection detail

1.4.2 Dynamic Partial Reconfiguration

Partial reconfiguration (PR) is the ability to change the configuration of one or several regions of the FPGA [214], [215]. There are two ways of doing this reconfiguration, the static one and the dynamic one. In the static reconfiguration, the system is stopped to proceed with the changes within the FPGA, even if only a part of the FPGA is reconfigured. After downloading the new configuration, the FPGA is reactivated again. In the case of Dynamic Partial Reconfiguration (DPR), the region that is not reconfigured, the 'static part', continues working while the reconfigurable region is being reprogrammed, Figure 1.15 depicts this process. The reconfiguration can be done externally or the static part can be responsible for performing it; this second case is called dynamic partial self-reconfiguration.

DPR is a capability of SRAM FPGAs. This is possible due to the nature of the memories used to control the connections of the different elements within the device such as, LUTs, DSP blocks, RAM blocks, etc. If a change in those memories is done, the connections are automatically changed. These memories can be rewritten on-the-fly, that is to say, the device must not be stopped or switched off to rewrite the memories. Firstly a full bitstream, with both static and reconfigurable partitions, is downloaded in the FPGA. Then, to perform the reconfiguration, a bitstream with the new configuration of a reconfigurable partition is downloaded in the device, as Figure 1.16 shows [215]; this bitstream is called 'partial bitstream'. Flash FPGAs can be in theory reconfigured, but main flash manufacturers do not design tools to perform dynamic reconfiguration in their devices.

There are different ways of downloading the partial bitstreams. The first one is using some external configuration ports used to download full bitstreams. These are: IEEE 1149.1 standard, also known as Joint Test Action Group (JTAG), SelectMap interface [216] or any other external interface such as Ethernet or PCI Express [215]. A second option is to perform the reconfiguration within the device using internal reconfiguration ports and/or internal

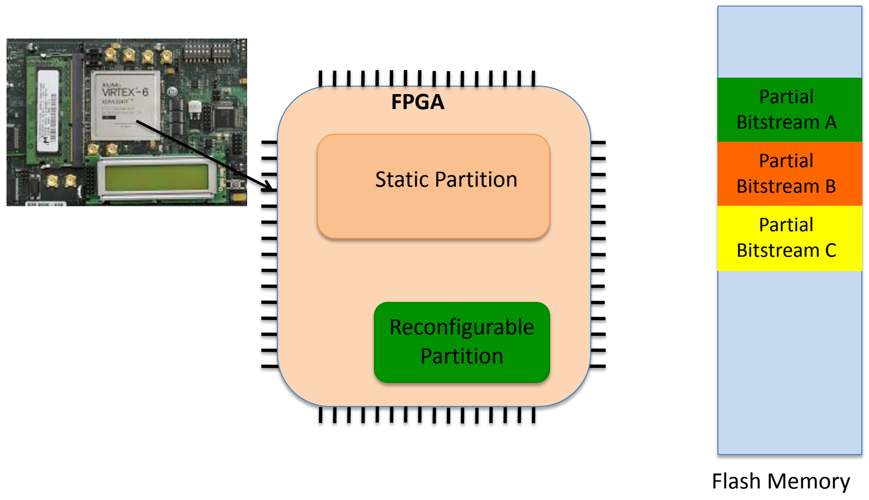


Figure 1.15: Dynamic Partial Reconfiguration working process using an external Flash memory to store the partial bitstreams

reconfiguration controllers. In Xilinx's case, its Virtex 4, Virtex 5, Virtex 6 and 7 Series FPGAs are provided with the Internal Configuration Access Port (ICAP). This port permits access to the internal configuration SRAM memories. A reconfiguration controller is also provided, the Hardware ICAP (HWICAP). This controller can be connected to the buses implemented in the system [217], [218] so soft and hard microprocessors can perform internally the reconfiguration of the system. Altera has a similar controller called *Partial reconfiguration control block* [219] that can be connected to a Nios soft microprocessor or to a core created by the user.

The advantages of using DPR in the designs are the following:

- Reduction in size: instead of implementing all the modules needed in the system, only those needed at each moment are implemented (i.e. time-multiplexing).

- Power consumption: because of sharing resources, the size of the FPGA can be reduced and so also the power consumption of the device. Also, non-used partition can be 'erased' to save power. The saved power is the dynamic power, the power the device consumes due to transitions on circuit signals and is defined by the equation

$$P = \frac{1}{2}CV^2 f, \tag{1.20}$$

where f is the working frequency, C is the capacitance of the node switching and V is the supply voltage [220].

- Uninterrupted operation even when a dynamic partition is reconfigured.

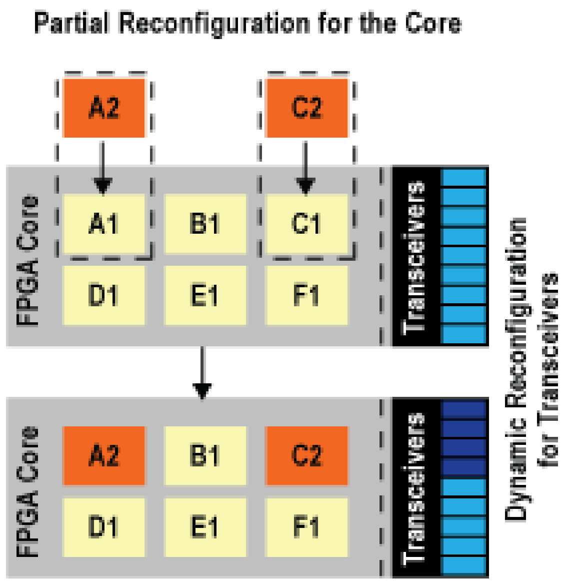

Partial Reconfiguration for the Core
A2
C2
FPGA Core
A1
B1
C1
D1
E1
F1
Transceivers
FPGA Core
A2
B1
C2
D1
E1
F1
Transceivers
Dynamic Reconfiguration for Transceivers

Figure 1.16: Reconfiguration of 2 internal modules of an FPGA using new partial bitstreams

Sadly this technology also has some drawbacks:

- The creation of the partial bitstreams can be very difficult if the manufacturer's tools have not been developed.
- If the reconfiguration is done using a microprocessor, the software partition can slow down significantly the reconfiguration process.
- Some Intellectual Properties (IP) cores cannot be reconfigured due to the restrictions imposed by the manufacturer [219].
- If the partial bitstream is not correctly read by the module responsible for performing the reconfiguration, the static part can be also altered.
- A memory to store the partial bistreams is needed. If the FPGA does not have an internal non-volatile memory, an external one must be added to the design.

1.4.3 Field Programmable Gate Array Manufacturers

FPGAs have become very useful to implement hardware and are widely used in the industry. But, even though the number of manufacturers is high, two companies comprise approximately 90% market share: Xilinx with 47% and Altera, 41%, both using SRAM technology. Due to this, other companies have focused their products on developing FPGAs using other technologies or focusing their products for specific areas.

Xilinx

It is the leader of the market with nearly a 50% share and it produces only SRAM FPGAs [221]. Xilinx has also developed a wide range of tools to implement hardware designs in their FPGAs and a wide range of IPs to implement in their products. Regarding Dynamic Partial Reconfiguration, it is one of the pioneers in this area and nearly all their new families support it [222], including the new 7 Series.

For low power applications, the Artix 7 devices can be used. Compared with the Spartan 6 family, these devices have, even in the smaller device, a huge number of DSPs. For applications where the performance is higher, Virtex 7 devices are the solution. Halfway between the Artix 7 and Virtex 7 devices Kintex 7 devices are present: they offer a good alternative when Artix 7 devices are not sufficiently powerful [223].

Altera

This is the second largest FPGA manufacturer in the world and Xilinx's main competitor. It also produces SRAM-based FPGAs and provides the designers with a wide variety of tools and IPs. Recently it has announced an agreement with Intel to use their factories to produce devices using Intel's 14 nm Tri-Gate transistor technology [224].

Microsemi

This manufacturer has focused its production on Flash technology-based FPGAs and low power applications. It has also radiation tolerant families certified for space applications, the RTAX and RTSX-SU families, based on anti-fuse technology, and the RT ProASIC3 Flash based FPGA family [225]. DPR is not supported in FPGAs using these configuration technologies.

Lattice

Lattice has focused its production on small size and low power SRAM and Flash FPGAs. Its biggest FPGA is 35 x 35 mm [226] and the smallest FPGA is 1.40 x 1.48 mm [227]. Due to the size and limited number of resources of these devices, they will not be taken into consideration in this thesis.

Atmel

Atmel had previously produced two different SRAM FPGAs, but today it only produces one FPGA model from 5K to 40K logic gates and 3V or 5V

[228]. Due to the small number of resources of this model, the implementation of the desired system is not possible even though it supports DPR.

Other manufacturers

Previously mentioned companies are the ones which dominate the market share. Nevertheless, there are other manufacturers, such as Acroflex, Quicklogic, SiliconBlue Technologies or Archronix. These companies have focused their products on very particular specifications, such as low power, small size or very high speed. Due to the objectives of this thesis, these manufacturers' products have not been taken into consideration during the project.

Chapter 2

A Hardware/Software Architecture for an Artificial Neural Network

In the present chapter, an FPGA-based HW/SW architecture for an Artificial Neural Network is presented. The system consists in a MicroBlaze soft microprocessor (i.e. the SW partition) and connected to it an Artificial Neural Network (ANN) hardware coprocessor (i.e. the HW partition) that has been designed using standard Very-High-Speed Integrated Circuits Hardware Description Language (VHDL) [229]. The system is suitable for the development of Ambient Intelligence (AmI) environments.

In the first part of the chapter, the global architecture of the system is presented. Then, the designed ANN coprocessor is described, including the scheme weights storage, activation function implementation and the global control by means of a Finite State Machine (FSM). The different elements of the SW partition are next described; the physical components, the algorithms implemented and also a proposal to accelerate by HW the most time-consuming computations. The final part of the chapter refers to a new method for implementing the activation function using a controlled accuracy method and its integration within the ANN core.

2.1 Global Hardware/Software Architecture

In this section, the scheme of the global system is presented. From the possible three ANN implementations presented in Section 1.3 (i.e. SW, HW or HW/SW), a HW/SW architecture has been selected to be implemented in an FPGA-based System on Programmable Chip (SoPC). Hence, the entire design, even the communication peripherals and memories, are integrated in the very same chip.

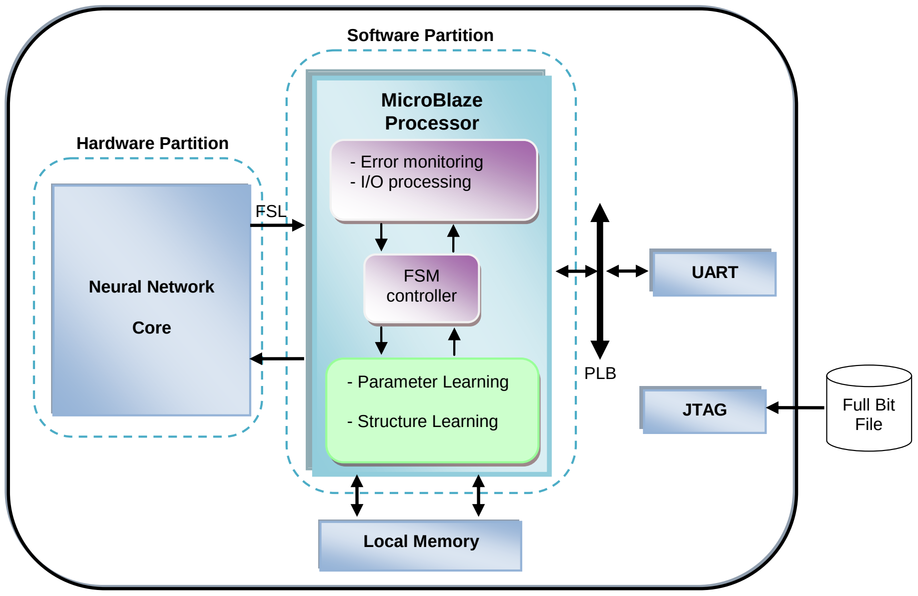


Figure 2.1: Block diagram of the HW/SW architecture of the system. The software partition is based on the MicroBlaze processor, while the ANN core represents the hardware partition.

2.1.1 General Scheme

The proposed HW/SW architecture is shown in Figure 2.1. The SW partition is based on a MicroBlaze soft microprocessor. It is responsible for the implementation of the learning algorithms and the control of the entire system. In contrast, the HW partition implements the ANN core and it is responsible for computing the outputs of the network after receiving the inputs from the processor. By using this method, it is possible to speed up the processing speed of the ANN, freeing the microprocessor from this task and hence, making the SW application smaller and simpler.

2.1.2 Communication Systems

To perform the communication between the core and the processor using Xilinx devices, (used in this project), there are two available options: Fast Simplex Link (FSL) [230] and Advanced eXtensible Interface v4 - Stream (AXI-Stream) [231] buses. They are point-to-point simplex communication buses with a word length of 32 bits, specifically designed for uni-directional data bursting. In the design of the ANN core, the FSL bus has been selected because it is supported almost entirely by Xilinx devices, while the AXI-Stream is only supported by the last families. FSL also has the following advantage: the First-Input

First-Output (FIFO) used to store the data can operate synchronously or asynchronously. This means that, by using the asynchronous mode, it is possible to have different clock rates in the processor and the ANN core. Finally, both buses are simplex; this means that the core must have two bus interfaces, as can be seen in Figure 2.2: one to receive data from the processor and another to send data to the processor.

2.1.3 Software Partition

The SW partition is built around a MicroBlaze soft processor core and is responsible for the ANN learning algorithms. These algorithms are better suited to being implemented in the SW partition due to their irregularity, high computational demands and high precision requirements. The SW partition is also in charge of the whole system control, the management of peripherals and co-processors, and the computation of non-recurrent tasks.

2.1.4 Hardware Partition

The hardware partition (i.e. special purpose hardware) is designed to perform the calculations of the feed forward neural network. Due to the parallel and regular nature of neural networks, they are better suited to being implemented in the HW partition, which allows the implementation of highly parallel architectures, optimized for real-time operation. Also, the rich variety of FPGA resources for digital signal processing greatly eases the development of high-performance low-power applications. In particular, most FPGA families sold on the market are equipped with DSP cores (e.g. Xilinx's Extreme DSP [203]). These hard cores perform mathematical operations such as products and/or sums faster than using logic elements.

2.2 Hardware Partition: Architecture of the Artificial Neural Network Core

2.2.1 Data Representation and Arithmentic

Before starting with the design of the network architecture, it is necessary to decide on the format to be used to represent the value of the numbers: fixed point or floating point representation.

Floating point representation is always more problematic, due to the complexity of the HW required in its arithmetic. In contrast, fixed point arithmetic is easier to handle and the designs are simpler, as can be seen in [232], where the design of floating and fixed point pipelined adders and multipliers are shown. Hence, except for some exceptions, fixed point representation is used to implement ANNs in PLDs [233], [234], [235], [236]. Floating point numbers can be used previously (for example in a PC) to train the parameters of the network or to test an algorithm before being implemented in the FPGA [237], [238],

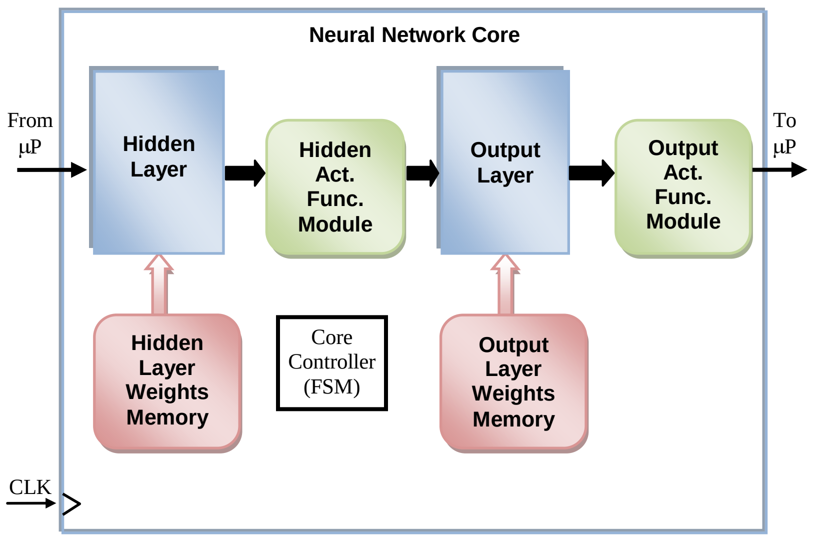


Figure 2.2: ANN core architecture diagram

[239]. For all these reasons, fixed point representation will be used in the HW partition.

2.2.2 Artificial Neural Network Core Scheme

The ANN core (HW partition) consists of a three-layer MLP, which communicates with the main processor by means a pair of FSLs. Figure 2.2 depicts an internal block diagram of the ANN core. The main modules of the core are the Hidden Layer, the Output Layer, the Activation Function Modules, the Weights Memories, and the Core Controller. In real-time mode operation, the core receives sequentially the inputs from the FSL FIFOs and the Hidden Layer performs the computation of all the hidden neurons in parallel. After that, the outputs of this layer are passed through the activation function filter (normally a sigmoid or a hyperbolic tangent function), implemented using a ROM memory per neuron. Finally, the Output Layer receives the values of the hidden layer sequentially and computes them in parallel. It includes as many neurons as core outputs, another activation function filter, and also a multiplexer (MUX) that sequences the transfer of the core outputs to the MicroBlaze. Finally, the Core Controller is a simple FSM responsible for the data pipelining through the data path.

The core is a standard VHDL module that can be sized in several dimensions by means of *GENERIC* parameters (i.e. word-length, number of inputs, number of outputs, and number of hidden layer of neurons). All these parameters are defined in a constant *PACKAGE* where the definitions of the internal connection parameters (i.e. connection buses) are also included.

2.2.3 Artificial Neuron Architecture

All neurons of the proposed architecture are based on the small functional module that is described in this section. First of all, let us take a look at the most common ways to implement a neuron [240]:

1. Serial Processing: In this model, a single Multiply-Accumulator (MAC) is used as the core of the neuron. In each cycle, the corresponding product of input and its weight is computed and the result added to the accumulator (see Figure 2.3).

2. Parallel Processing: In this case, all the inputs and their weights are connected to an array of multipliers, shown in Figure 2.4, so all input-by-weight products are done in only one clock cycle. Then, a full parallel adder is connected to the output of the multiplier array, so the outputs of the products are added in another clock cycle.

3. Partial Parallel Processing: This processing architecture uses an adder tree as can be seen in Figure 2.5.

For real-time environments, time is a key factor of the design, which is why the Partial Parallel Processing seems to be a good solution. However, in the case of using a coprocessor attached to a main microprocessor, there is one more factor to be taken into account: the communication buses between both devices.

The ANN core designed here uses a pair of FSL buses to perform the communication; this means that the inputs are sent one by one in a data burst to the ANN. As a consequence, an architecture where all inputs are available at the same time is not possible to implement. Therefore, only one of the three architectures presented above can be implemented in this project: the Serial Processing architecture, shown in Figure 2.3.

To implement the Serial Processing neuron, the DSP cores embedded in the silicon of FPGAs have been used [203], [241], as many neurons as DSPs available in the FPGA can be implemented. These cores allow MAC operations to be performed faster than using the logic of the FPGA.

In the design of the neuron, a word-length limiter in the output of the accumulator has been added so as not to exceed the number of bits after the MAC operations, see Figure 2.6.

After the *startup* or after the *reset signal* is disabled, the accumulator is initialized with the value of the offset (θ). After that, and while the *enable signal* is active, in each clock cycle the value of the input (x_k) is multiplied by the value of the weight (ϖ_k) and the obtained result is added to the value to the accumulator (y_k). The inputs of the neurons can be received from the input of the ANN or from another layer, whereas the weights and the offsets are received from memories connected to the neuron.

In this architecture, the number of cycles needed by the neuron to obtain the output is, in a best-case scenario, the number of nodes in the previous layer. For example, in a network with n inputs, L hidden neurons and m outputs, the

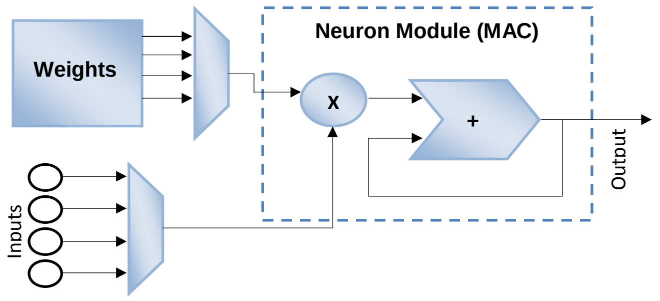


Figure 2.3: Serial Processing neuron scheme

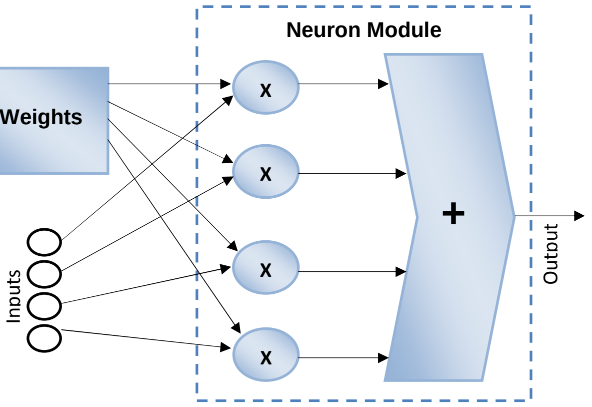


Figure 2.4: Parallel Processing neuron scheme

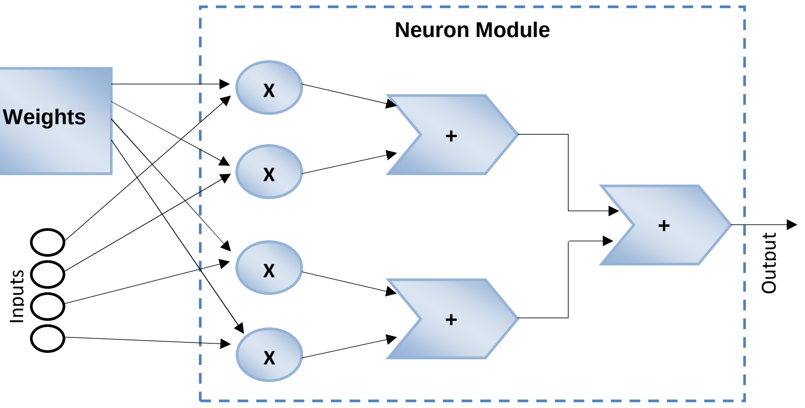


Figure 2.5: Partial Parallel Processing neuron scheme

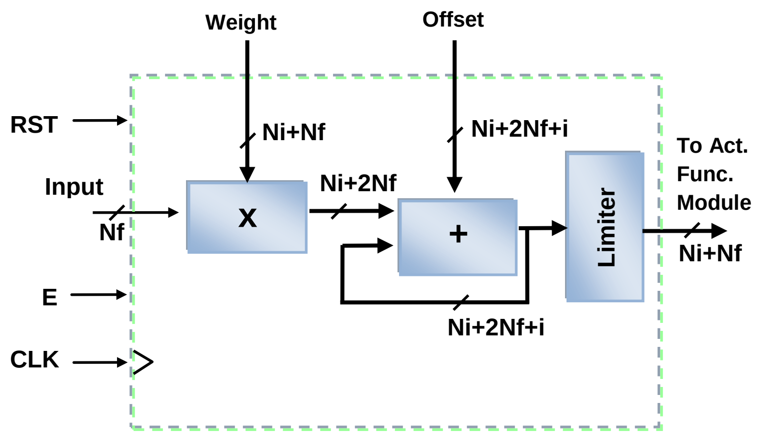


Figure 2.6: Proposed serial processing neuron design, where *Ni* is the length of the integer part and *Nf* is the length of the fractional part

hidden layer neurons need n clock cycles to perform the sum of products, while the output neuron requires L clock cycles.

2.2.4 Weight and Offset Storage

To store the weights and offsets of the ANN two options can be adopted: the use of Read Only Memories (ROM), implemented using the LUTs of the FPGA, or the use of Random Access Memories (RAM) using the registers or memory blocks of the FPGA.

The first method has the advantage of being very easy to implement and provides a very fast reading process. It has, however, a very important drawback: the memories cannot be modified dynamically. This aspect avoids changing dynamically the weights, hence, restricting the ANN capability to adapt to changes, which is a key aspect in neural network based systems.

In the case of RAM memories, the values of the memories can be modified dynamically, so, the ANN core gains the capability to adapt to changes. To implement the RAMs, the FPGAs provide two different ways. The first uses the distributed memory of the logic cells of the FPGAs, whereas the second uses the RAM blocks [242], [243] embedded in the silicon. Both methods have their advantages and disadvantages:

- Resources: In the case of the distributed memory, a large amount of logic cells can be needed, while in the case of embedded RAM blocks, depending on the size, only one block might be needed.

- Speed: The maximum frequency of the RAM implemented using distributed memory is much higher than the maximum frequency of a RAM block.

Taking into consideration the size of the memories needed to store the value of the weights, and that the key factor in the design is speed, it is the distributed memory implementation that has finally been selected. Moreover, due to the small size of the memories of the network, there could be problems implementing them in RAM blocks for some architectures and/or data word length.

To send the weights and offsets to be stored in the RAMs, two options were taken into consideration:

- Use the existing FSL bus: in this case, the same FSL bus used to send the inputs to the core is shared. To inform to the core that the data received are weights (and not inputs), a 1-bit port is added to the core.
- Add a new FSL bus: in this case a dedicated FSL bus from the processor to the core is implemented. Hence, the ANN core would implement three communication buses instead of the previous two.

Considering the above two options, the best solution is the use of the same data bus to send the values of the weights from the processing unit to the ANN core, a write enable signal, implemented in the core as a port independent of the communication bus.

2.2.5 Activation Function Implementation

To implement the activation function, different methods can be adopted. However, the most widely used is the LUT method, shown in Figure 2.8. In this case, the memory implemented contains the values of the activation function. The output of the neuron is used as address of the memory to obtain the correct function value. However, there is a problem with the amount of required LUTs if the precision of the data is very high. To overcome this problem, a new characteristic is added to the *word-length limiter of the neuron*, shown in Figure 2.6. As can be seen in Figure 2.7, some functions vary their values very slowly after a certain point (*Xs*), therefore, *the output of the neuron* is limited as follows:

$$\begin{cases} x, & if\ |x| < |Xs| \\ Xs, & if\ |x| \geq |Xs| \end{cases} \tag{2.1}$$

For example, for an hyperbolic tangent function with $Xs = \pm 4$, an initial word length of 14 bits (5 integer bits, 8 fractional bits and a sign bit), is reduced to 11 bits (2 integer bits, 8 fractional bits and a sign bit). Therefore, the ROM memory is reduced from 2^{14} to 2^{11} positions.

Another solution for avoiding the problem of the size of the memories is the so-called *Range Addressable Look-up Table* (RALUT), proposed originally in [244]. In this case, for every data value, there is a range of possible addresses instead of only one as can be seen in Figure 2.9. With this technique, the range of the values codified is wider, while the number of positions in the memory is smaller. Unfortunately, the main drawback of this technique is the

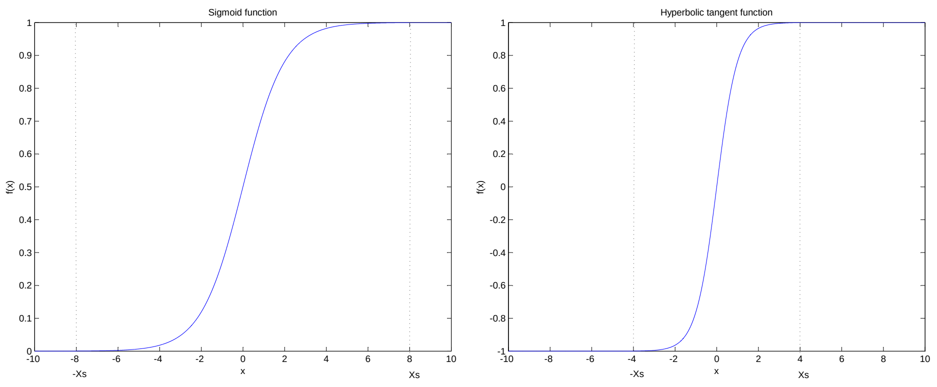


Figure 2.7: Waveforms of the sigmoid (Left) and hyperbolic tangent (Right) functions

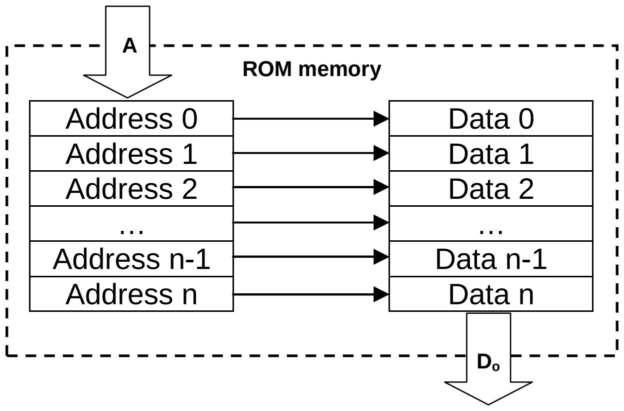


Figure 2.8: A ROM memory implemented using a LUT architecture

loss of accuracy. In this project, RALUT architecture was discarded. In the case of having very big and non-synthesizable LUT-based memories, the limiter operation could be changed to give an output with a smaller length, and hence, reduce the size of the ROM memories. This last option is better than the RALUT method.

2.2.6 Artificial Neural Network Core Control: Finite State Machine

Two different architectures are presented (i.e. RAM-based and ROM-based), therefore, two different Finite State Machines (FSM) were also designed.

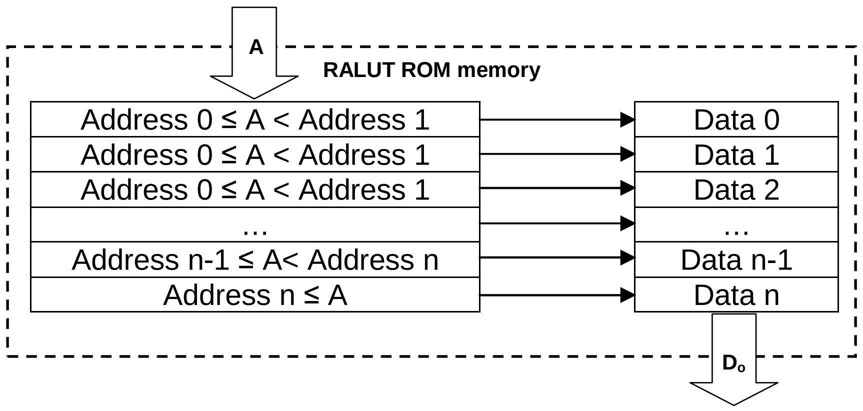

A
RALUT ROM memory
Address 0 ≤ A < Address 1
Address 0 ≤ A < Address 1
Address 0 ≤ A < Address 1
...
Address n-1 ≤ A< Address n
Address n ≤ A
Data 0
Data 1
Data 2
...
Data n-1
Data n
D0

Figure 2.9: A ROM memory implemented using a RALUT architecture

Finite State Machine for a Read Only Memory-based Core

In this architecture, the Finite State Machine is responsible for enabling the neurons of the layers of the ANN and controlling the communications signals of the buses to receive correctly the inputs from the processor or to send the outputs of the ANN to the processor. This flow control must be carefully implemented, because in some moments the inputs can be received one per clock cycle, while in other cases the inputs can be received having pauses of several clock cycles between them.

To perform this, the FSM has the following states (see Figure 2.10):

- *Idle*: This is the initial state of the FSM; it is entered after a reset. This state waits until a correct input is received from the microprocessor.

- *Hidden_Layer*: This is the state that is activated when in *Idle*, an input is received. In this state all hidden neurons are enabled so the input can be processed. Also, every time an input is received, a counter, initialized with the number of inputs, is decremented. When the counter reaches 0, the FSM jumps to the next state.

- *Hidden_act_function*: In this state, the activation function is applied to the outputs of the neurons of the hidden layer. It only lasts one clock cycle and jumps automatically to the next state.

- *Output_layer*: This state is similar to *Hidden_Layer* but in this case the neurons of the output layer are enabled. In this state, the values obtained in the hidden layer are sequentially multiplexed to the neurons of the output layer. A counter will be responsible for addressing the data and the weights. It has an initial value of *'0'*, and when the counter reaches the number of *hidden layer neurons* "-1", the FSM jumps to the next state.

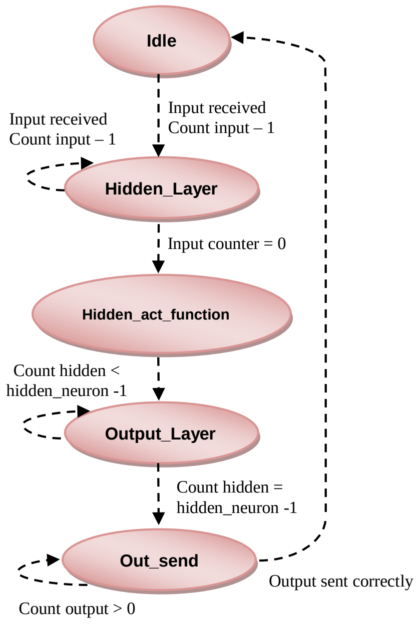


Figure 2.10: ANN core Finite State Machine for a ROM-based core

- *Out_send*: In this state, the activation function is applied to the output layer neuron values and the results are stored in the FSL bus queue to be sent to the microprocessor. The change of state is not performed until all outputs have been sent. For that purpose, a counter is used, which is decremented every time the bus sends an acknowledgement signal indicating that the operation has been done correctly.

Finite State Machine for a Random Access Memory-based Core

For the RAM-based architecture, the FSM has been modified as is explained below.

First, to write new values in the RAM, two strategies can be adopted:

- To send the weights of all the neurons sequentially from the processor. In this case the RAMs have no addresses, and internal counters are used within the ANN core so the weights are written in the corresponding memory. The main advantage of this method is that it is simple to design, but its main disadvantage is that all the weights and offsets must be sent even when only one value need to be changed.

- To implement RAMs with addresses: In this case, together with the values of the weights and the offset, the address of the memory must be also sent. The main advantage is that, when the weights of only one neuron need to be changed, the rest are not involved in the process. The main disadvantage is that the managing of the address makes the process a little more difficult to design.

Taking into consideration the advantages and disadvantages of both options, the second one is selected due to its performance. Hence, the writing process for a neuron is as follows:

- The microprocessor enables the WR signal.
- The address of the neuron is sent.
- The values of the weights of the neuron are sent one by one in order.
- The offset is sent.
- The WR signal is disabled.

This new writing process requires new states to be added to the FSM, as can be seen in Figure2.11. These are the new states:

- *Get_Addr*: To enter in this state the WR signal must be enabled (i.e. from the microprocessor) and new data must be received from the FSL bus. The first piece of data received is a memory address. This state determines whether the memory address belongs to a neuron of the hidden layer or of the output layer, and enables the *WR* signal of only that neuron.
- *Hidden_Memory_Write*: If the address received in *Get_Addr* corresponds to an address of a hidden layer neuron, the FSM enters in this state. Here, every piece of new data received from the bus will be stored in the correct memory and when all data have been received, the FSM will jump to the *Idle* state. A counter is used to check data reception. It works here as it works in the *Hidden_Layer* state but, with one exception, it counts one data more, the offset of the neuron.
- *Output_Memory_Write*: This state is the one after *Get_Addr* if the address received corresponds to an output layer neuron. The working process is similar to *Hidden_Memory_Write*, except that the jump condition to the *Idle* state is asserted when the number of values received equals the number of hidden layer neurons plus one.

Finally, when the system was implemented a problem arose: the working frequency was too low, below 100 MHz, in some devices such as the Virtex 5 XC5VLX110T. The reason was that the path between the RAM memories and the input of the neurons was too long; therefore, more modifications had to be made in the system. The best solution was to implement the memories inside the neuron. Hence, when the design was synthesized, the path between the two elements was shortened. By using this method the maximum working frequency was increased. However, it was still a little lower than 100 MHz.

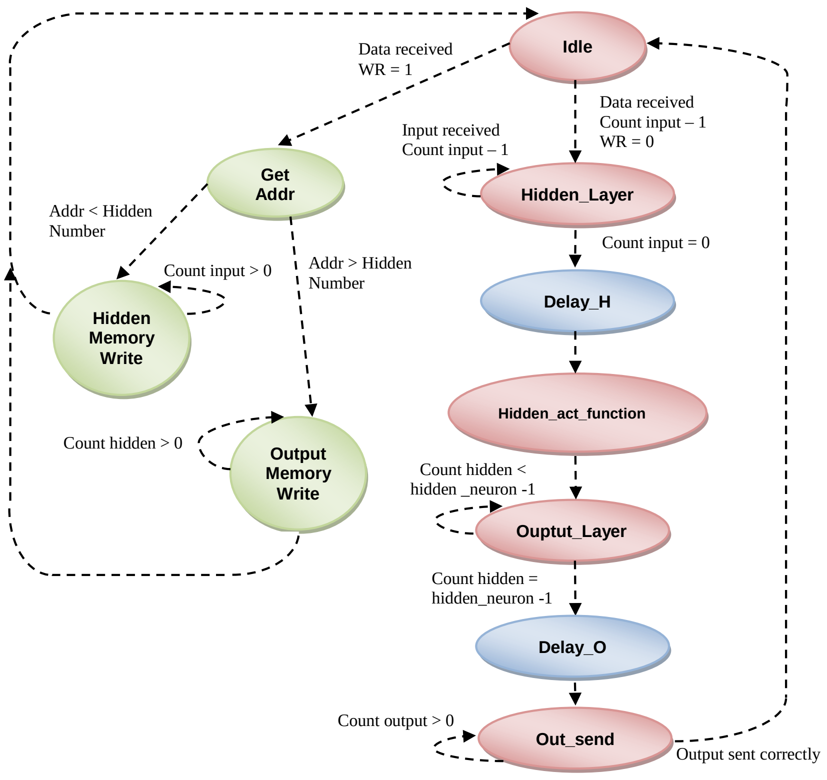


Figure 2.11: ANN core Finite State Machine with the new states to write the RAM memories

Another analysis of the path revealed a new source of delay in the path connecting the input FSL bus and the inputs of the neurons. The solution to this problem was to introduce a register in all neurons between the input of the neural network (the FSL bus) and the input of the neuron. In addition, to avoid control problems, the neuron enabling signals were also registered to delay them 1 clock cycle. As a result, two new states, *Delay_ H* and *Delay_ O* were introduced in the FSM to adjust the data flow to the new modifications (Figure 2.11). Hence, 2 clock cycles were added to the time needed by the ANN core to obtain an output.

Finite State Machine for a Random Access Memory-based Core with Structure Learning

Due to the possibility of using structure learning algorithms that change the number of hidden layer neurons, explained in more detail in Section 2.3, the system has to be able to warn the ANN core about how many neurons of

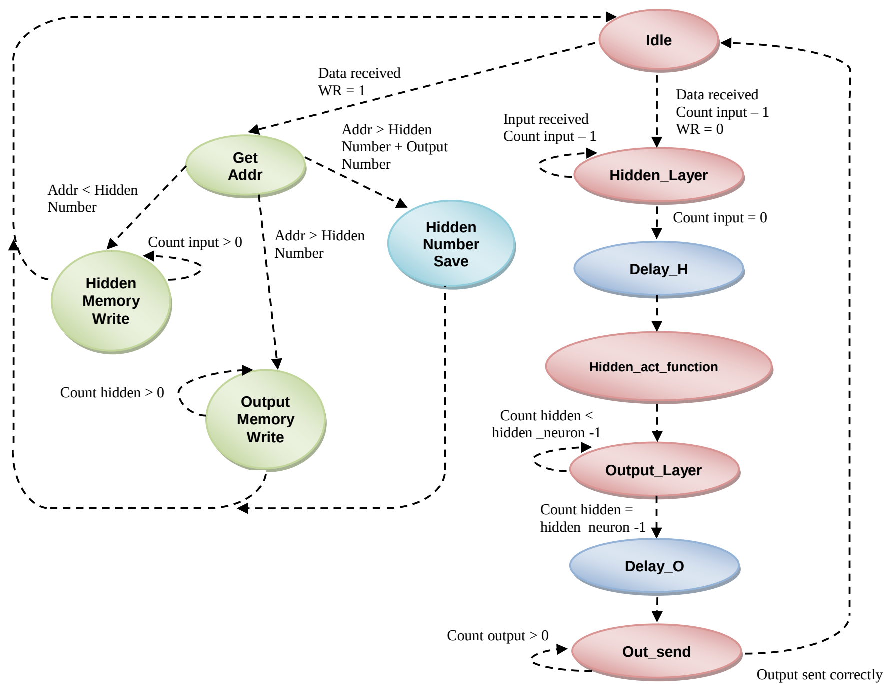


Figure 2.12: An ANN core Finite State Machine with the new states to update the number of active hidden layer neurons

the hidden layer are active to avoid problems during the real time operation mode. This is done by updating the value of a register that stores the number of active hidden neurons, which works as a limiter of the counters involved with the operation of the hidden layer, e.g. the control of the jump condition from state *Output_layer* to *Delay_O*. It also controls the enabling signals of the hidden neurons, thus, only the active neurons are actived.

To update this register, the same procedure applied to writing the weight RAMs is used, however, in this case its address is equal to *hidden neuron number + output number* and only one value is sent after the address. Hence, the FSM must be modified to perform this function and the result can be seen in Figure 2.12.

2.2.7 Artificial Neural Network Core Implementation

In Table 2.1 and Table 2.2, a comparison in the numbers of resources needed to implement an ANN with ROM and RAM memories is shown. In particular, a 7-input, 14-hidden neurons and 1-ouput topology has been selected for the implementation. This topology corresponds to an AmI experiment explained in detail in Chapter 4. Also, the data format has been set to 16-bit

Device	LUT	Registers	DSPs	Maximum Freq (MHz)
XC5VLX110T-1	1769 (2.56%)	604 (0.87%)	15 (24%)	120
XC6VLX240T-1	1080 (0.36%)	605 (0.4%)	15 (2%)	133
XC7K325T-2	1080 (0.26%)	604 (0.3%)	15 (1.8%)	176

Table 2.1: Resources needed by an ANN with ROM memories to store the weights with 7 inputs, 14 hidden neurons and 1 output implemented in Virtex 5 XC5VLX110T-1, Virtex 6 XC6VLX240T-1 and Kintex 7 XC7K325T-2 FPGAs

Device	LUT	Registers	DSPs	Maximum Freq (MHz)
XC5VLX110T-1	2360 (3.41%)	878 (1.27%)	15 (24%)	113
XC6VLX240T-1	1534 (0.51%)	881 (0.6%)	15 (2%)	129
XC7K325T-2	1534 (0.37%)	880 (0.43%)	15 (1.8%)	168

Table 2.2: Resources needed by an ANN with RAM memories to store the weights with for 7 inputs, 14 hidden neurons and 1 output implemented in Virtex 5 XC5VLX110T-1, Virtex 6 XC6VLX240T-1 and Kintex 7 XC7K325T-2 FPGAs

signed numbers, with an 8-bit fractional part. The devices selected for the implementation belong to the three most widely used families of Xilinx: Virtex 5 XC5VLX110T-1, Virtex 6 XC6VLX240T-1 and Kintex 7 XC7K325T-2 FPGAs.

As was expected, the number of resources is higher and the maximum frequency lower when using the RAM memories. However, to have the capability of changing the weights, and thus the capability of implementing the learning algorithm within the system, is an important issue. The number of resources needed in the case of the ANN with RAM memories is between a 33% and a 45% higher, compared with the ANN with ROMs, depending on the device, but this number is still low enough to be implemented in small FPGAs.

After verifying that the HW can be implemented correctly in the devices, the ANN core must be tested. First, the test has been carried out with the ROM-based core. The process has been done with the Xilinx Chipscope tool. This tool has the ability to show the behaviour of every internal signal during run-time operation. In particular, the test has been done with an ANN with 7 inputs, 14 hidden neurons, one output and a word length of 15 bits (8 fractional bits, 6 integer bits and a sign bit). First, the neuron is checked to know if it works correctly. To make the visualization easier, the input values go from 1 to 7 and different values are used in the weights of the first neuron. To check the accumulator initialization, the offset has been set to '4'. Finally, the limiter has been set to give a range of ± 8.

As can be seen in Figure 2.13, the value of the accumulator (i.e. signal *result*) is '0' until the first input arrives, then the *startup* signal is deactivated and the value of the output of the accumulator is the value of the offset plus the

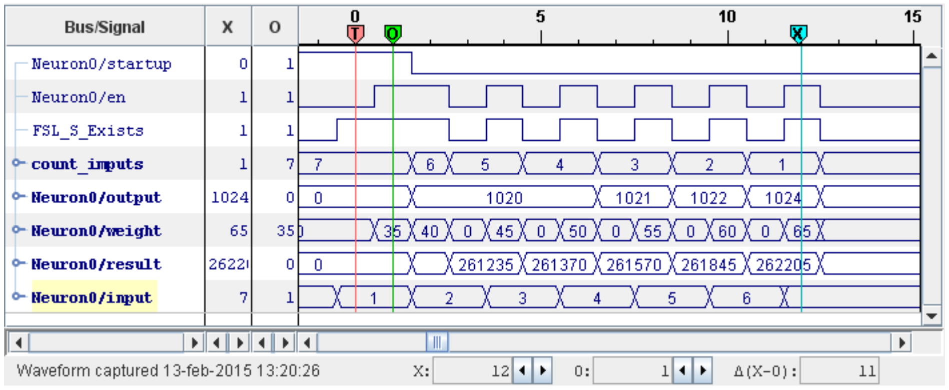


Figure 2.13: Chipscope capture of the first hidden layer neuron working process for a ROM-based ANN with 7 inputs

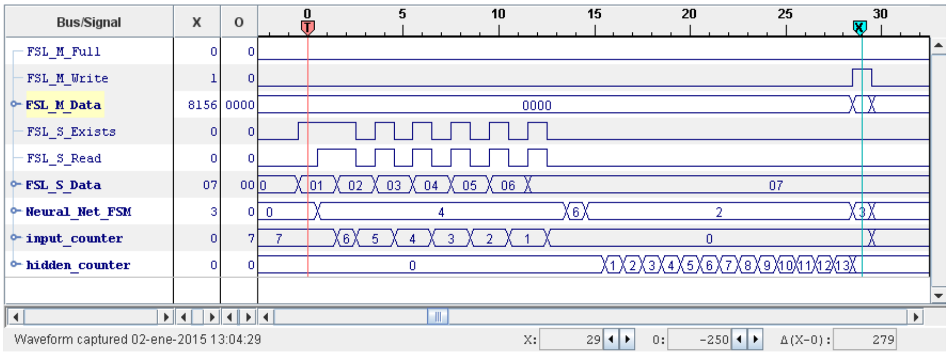


Figure 2.14: Chipscope capture of the FSL bus signals, FSM and the control counters of a ROM-based ANN core

input by the weight. Every time a new input is received, the neuron enable signal (i.e. *en*) is enabled and the input-weight product is added to the accumulator. As can be seen, the limiter also works correctly because the output is the value 4 codified with 8 bits in the fractional part.

Once it has been verified that the neuron works correctly, the ANN control has to be checked. To do this, we examine the FSM signals and the control counters.

Figure 2.14 shows that when the first input is received, the FSM changes from *Idle* state to *Hidden_Layer* and the control counter of the state starts decreasing. When it reaches '0', the FSM changes to the *Hidden_act_function* state for only one cycle and then changes to the *Output_layer* state. When all the outputs of the hidden layer have been processed, the FSM changes to the state *Out_send* and, if the FSL bus is ready, the output of the ANN is sent back to the processor.

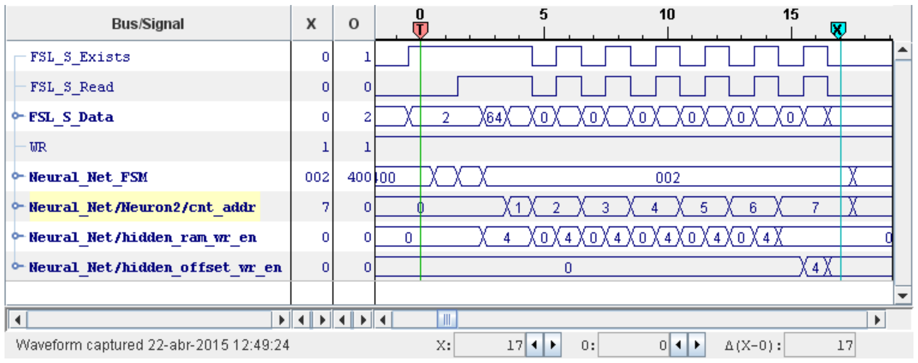

Bus/Signal
X
O
FSL_S_Exists
FSL_S_Read
FSL_S_Data
WR
Neural_Net_FSM
Neural_Net/Neuron2/cnt_addr
Neural_Net/hidden_ram_wr_en
Neural_Net/hidden_offset_wr_en
Waveform captured 22-abr-2015 12:49:24
X: 17
O: 0
Δ(X-O): 17

Figure 2.15: Chipscope capture of the FSL bus signals, FSM and memories write enable signals during a memory writting process of a RAM-based core

Finally, the required clock cycles are checked. With this ANN architecture the ideal time to process the ANN is 23 clock cycles, although in reality, the ANN needs 29 clock cycles. As Figure 2.14 shows, the reason is in the FSL bus. This bus does not send always the inputs one per cycle (sometimes a delay is added); despite this, the HW coprocessor is capable of performing the computations extremely fast.

Finally, the writing process for the ANN core with RAM memories is checked. The result of writing the weights in the memory of the third hidden layer neuron can be seen in Figure 2.15. This figure shows how the first data received from the FSL bus, with the port *WR* enabled, is checked by the FSM and is understood as the address of the memory, (in this case the third neuron of the hidden layer). Subsequently, every piece of data received is written in the correct memory. This can be seen because only the first bit of the *hidden_ram_wr_en* signal is enabled every time an input is correctly received. After the weights are written, the offset is received and stored. This can be seen in the figure because the third bit of the signal *hidden_offset_wr_en* is enabled. After the process has ended, the FSM returns to the state *Idle*. The writing process would theorically require 11 clock cycles; however, the figure shows that more cycles are needed due to the extra cycles in the FSL bus, as happened in the input-sending process.

2.3 Software Partition

The SW partition is responsible for performing several tasks: system control, the management of peripherals and co-processors and the computation of the learning algorithms. Two different types of learning strategies are implemented: parameter and structure learning. For the parameter learning, two algorithms have been selected, Backpropagation and Extreme Learning Machine. In the case of the structure learning, the selected algorithm is a combination of

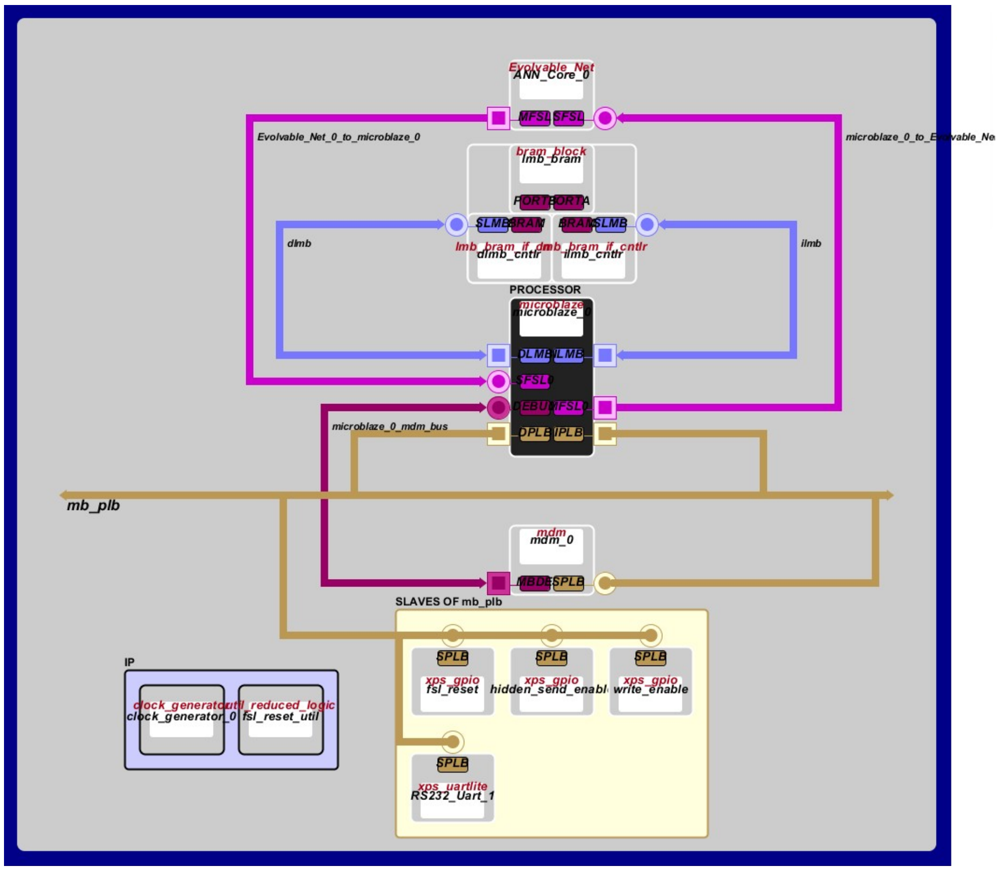


Figure 2.16: Block diagram of the system showing the SW partition, including its peripherals and the HW partition connected by means of a pair of FSL buses

growing and pruning methods [21]. The application of each learning algorithm is presented in Chapter 4.

In the next section, several aspects concerning the SW partition are discussed.

2.3.1 Components of the Software Partition

The SW partition is based on a 32-bit MicroBlaze processor (see Figure 2.1). The processor is equipped with a Floating Point Unit (FPU) to meet the performance and the precision requirements of learning algorithms. Its main functions are: system control, resource management, input/output (I/O) handling, data pre and post-processing (scaling and fixed-point codification), HW/SW communication, and training of the ANN. The program to be executed in the microprocessor has been developed using C programming language, written to be independent of the platform and the operating system as much as possible. The only hardware-dependable functions are those needed to communicate with the HW partition, and the I/O send/receive functions.

The Microblaze processor has, attached to it, a dual-port memory for data and instructions. The access to this memory is carried out with 2 Local Memory Buses (LMB). The size of this memory is up to 64 kB in the case of the Spartan devices, up to 128 kB in Virtex 4 devices and up to 256 kB in the case of the Virtex 5, Virtex 6 and 7 Series devices. Also, connected to the peripheral bus (i.e. AXI or PLB bus) it is possible to include on-chip RAM blocks (BRAM), which are also limited to those sizes.

In Figure 2.16 the block diagram of the system, generated with the synthesis tool, is shown. Here, the Microblaze connected to the memory, the peripherals and the ANN core using the above-mentioned communication buses can be seen.

2.3.2 Implementation Issues

The memory map of the Microblaze is divided as follows: a code section, an initialized data section and an uninitialized data section (this latter includes the heap and the stack of the processor). It is possible to include each section in different memories. However, if one of the sections is bigger than the maximum size of the memory, it must be stored in an external memory, such as a DDR memory, which requires a memory controller. Hence, the timing performance will decrease due to the accesses to the external memory, making mandatory the use of cache memories to improve the execution time.

One element that enables a reduction in memory consumption is a proper selection of the variable types of the SW. In this sense, C language provides a wide range of variables with different word lengths. To limit the size of the program, short memory variables have been selected whenever this has been possible.

Regarding the FPU of the Microblaze, it can be enabled or disabled during the design process. If there is an FPU, this can be a basic or an extended one. Furthermore, there is the possibility of implementing 32 or 64-bit integer multipliers and dividers, and a barrel shifter.

2.3.3 Learning Algorithm Acceleration Using the Hardware Partition

Even with a Microblaze fully equipped to perform mathematical operations and with the program stored in internal memory, considerable time is required for the learning algorithms to be executed. In the case of a BP algorithm for a training set using 408 training samples, (e.g. the first AmI application that will be explained in Chapter 4) each learning cycle requires 1.27×10^6 clock cycles for only 1 neuron in the hidden layer, and 7.33×10^6 clock cycles for 14 neurons.

Up to now, the ANN core has only been used after the learning has finished. However, the ANN core can also be used during the learning process to accelerate the algorithm. To do this, the ANN core is slightly modified to be able to send the outputs of the hidden layer neurons to the SW partition, whose values are needed by both BP and ELM. This avoids the SW algorithms computing those values.

After performing the above mentioned changes in the HW and SW partitions, the BP algorithm is run again to measure the time needed by each learning cycle. The result shows that 0.49×10^6 and 3.08×10^6 clock cycles are now required for 1 and 14 hidden neurons respectively. Hence, the use of the ANN core as a learning coprocessor speeds up the BP algorithm between a *57.87%* and a *61.83%*, which means that it is possible to perform more than the double of learning cycles in the same time. Therefore, the execution time of the SW partition is drastically decreased. For ELM, however, the effects of the HW acceleration are negligible because the main part of the algorithm, Equation 1.19, requires most of the computing time and it cannot be accelerated using our ANN core. In this case, another solution should be implemented [245].

2.4 Controlled Accuracy Activation Function Implementation

The sigmoid and hyperbolic tangent functions are two of the most widely used activation functions, mainly due to their differentiable nature which makes them suitable for on-line training. However, these functions are costly to implement in digital hardware because they require the calculation of an exponentiation and a division. To avoid this problem, a number of approximation techniques have been proposed over the years. The most commonly used are look-up tables (LUTs), explained in Section 2.2.5, bit-level mapping, Piecewise Linear (PWL) methods [246], lattice algebra-based Centred Recursive Interpolation (CRI) algorithm [247], Taylor series expansion [248], and hybrid methods [249].

The selection of the approximation method and its hardware implementation are the aspects that constrain the accuracy of the activation function, and consequently, the learning and generalization capabilities of the whole ANN [250]. Moreover, too low accuracy leads to poor performance, whereas an excess of it unnecessarily increases the hardware resources and reduces the processing speed. Despite the importance of proper specification of the accuracy of the activation function, very few works incorporate it as a design parameter [248]. To tackle this problem, a novel approximation scheme is proposed. The scheme is based on Taylor's theorem and the Lagrange form of the remainder or the error. A systematic design methodology which guarantees the accuracy of the approximation is provided. The proposed methodology is independent of the function it is going to be approximated.

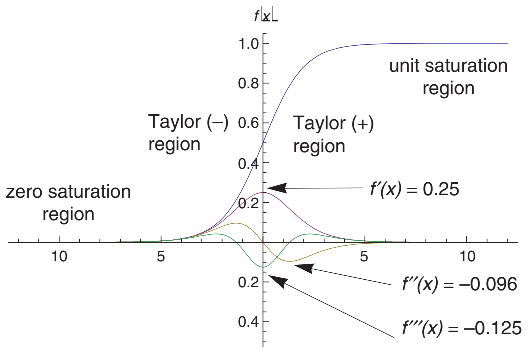


Figure 2.17: Approximation scheme input regions, sigmoid function and its first three derivative extreme values

2.4.1 Function Approximation Scheme Using Taylor Series

This scheme is going to be used in S-shaped functions (e.g. sigmoid and hyperbolic tangent). Hence, this approximation scheme divides the input range into two region types, saturation regions, and Taylor regions, as is shown in Figure 2.17. The saturation region gives the same value for any input, whereas Taylor regions approximate the function using an n^{th} degree Taylor polynomial given by the expression

$$f(x) \backsimeq f(x_o)+f'(x_o)(x-x_o)+f''(x_o)\frac{(x-x_o)^2}{2!}+\ldots+f^{(n)}(x_o)\frac{(x-x_o)^n}{n!}, \quad (2.2)$$

in any interval I containing x_o, where $I=(x_o-r, x_o+r)$, being $f(x)$ n times differentiable at the point.

The error of the approximation of the function in I can be bounded by using the Lagrange form of the remainder or the error:

$$|R_n(x)| \leq \frac{|x-x_o|^{n+1}}{(n+1)!}M_n, \quad (2.3)$$

where

$$|f^{(n+1)}(x)| \leq M_n. \quad (2.4)$$

Equation 2.3 is very useful because it gives a means for dealing with the approximation error ε as a design parameter.

Another point that must be taken into account is the properties of the sigmoid and hyperbolic tangent functions, where

$$f_{sigmoid}(-x) = 1 - f_{sigmoid}(x) \quad (2.5)$$

and

Allowed error (ε)	Taylor region width (t)	Intervals first-order series	Intervals second-order series
0.1	2.19	2	1
0.01	4.59	6	3
0.001	6.91	24	10
0.0001	9.21	102	28

Table 2.3: Design parameters (minimum values) as a function of allowed error for a sigmoid function

$$f_{tansig}(-x) = -f_{tansig}(x). \tag{2.6}$$

Using these properties, the functions are only computed for the positive semi-axis of the inputs, and for a negative input, Equations 2.5 or 2.6 are applied, therefore simplifying the design of the approximator.

The selection of the positive saturation region depends on the required accuracy. To determine where it starts, it is necessary to find the value where the gap between the value of the function and the maximum value of the function, '1' for the sigmoid and tangential functions, equals the maximum error:

$$1 - f(t) = \varepsilon. \tag{2.7}$$

Solving Equation 2.7, and using as example a maximum error of 10^{-3}, the saturation region for a sigmoid function starts for values $t > 6.91$, and $t > 3.8$ for a hyperbolic tangent function.

The positive Taylor region, with a $[0, t]$ range, is split into ni intervals of width *2*r, then $r = t/2ni$. By substituting in Equation 2.3 and taking into account that $|x - x_o| \leq r$, $\forall x \in I$, the minimum number of intervals for a given error results:

$$ni \geq \frac{1}{2}\left(\frac{M_n}{\varepsilon(n+1)!}\right)^{1/(n+1)} t \tag{2.8}$$

with

$$|R_n(x)| = \varepsilon. \tag{2.9}$$

Table 2.3 provides the boundary between the Taylor region and the saturation region according to 2.7, and the number of intervals in the Taylor region for a first- and second-order polynomial for a sigmoid function. As can be seen, the larger the polynomial, the lower the number of intervals in the Taylor region. Hence, to increase the accuracy of the approximation, a larger polynomial can be used or the number of intervals can be increased.

The results shown in Table 2.3 are for a sigmoid function, but applying the correct values in Equations 2.7, 2.8 and 2.9 for any other S-shaped function, e.g. a hyperbolic tangent function, the values of t and ni for it can be also obtained.

2.4.2 Selection of the Word Length

The proposed scheme has been implemented using a fixed-point signed fractional data format. The length of the integer part (*Ni*) depends on the width of the Taylor region: $2^{Ni} \geq t$, which can be rewritten as:

$$Ni \geq \frac{\ln t}{\ln 2}, \tag{2.10}$$

where $Ni \in \mathbb{Z}$.

The fractional part of the input (*Nj*) should be enough to represent the changes in the inputs $\triangle x$ that produce the changes in the function $\triangle f$ equal to the maximum allowable error:

$$\frac{\triangle f}{\triangle x} = \frac{\varepsilon}{2^{-Nj}} \tag{2.11}$$

and for small increments the derivative $f'(x)$ is

$$\frac{\triangle f}{\triangle x} \cong f'(x). \tag{2.12}$$

Taking the maximum value of the first derivative

$$2^{-Nj} \leq \frac{\varepsilon}{f'(x)_{max}} \tag{2.13}$$

then

$$Nj \geq -\frac{\ln f'(x)_{max}}{\ln 2} + \frac{|\ln \varepsilon|}{\ln 2} \tag{2.14}$$

where $Nj \in \mathbb{Z}$.

To obtain the length of the fractional part of the output (*No*), assuming that the signal quantisation is performed by truncation, the maximum quantisation error is $\varepsilon_t = 2^{-No}$. Therefore:

$$No \geq \frac{|\ln \varepsilon_t|}{\ln 2} \tag{2.15}$$

Using Equations 2.10, 2.14 and 2.15 it is possible to select the word length of the module that implements the scheme.

This module will be integrated in the ANN core. The word lengths of the function approximation module and the ANN core must, therefore, be compatible. To select the correct values of the integer part, the value of the integer part of the weights and *Ni* are checked and the highest one is selected, otherwise it would not be possible to codify the weights correctly. In the case of the fractional part, the same is done; the biggest value is selected between *Nj*, *No* and the length of the fractional part used in the ANN core (*Nf*).

2.4.3 Hardware Implementation of the Taylor Module

A 2nd order approximation of the activation function was implemented. The hardware implementation was done following the same ideas used in the implementation of the ANN core: standard VHDL, use of the DSPs to reduce the size of the core and a working frequency higher than the maximum frequency of the microprocessor.

To implement the scheme using the lowest number of resources, Equation 2.2 is decomposed in equation

$$f(x) = y_1(x - x_o) + f(x_o) \tag{2.16}$$

where

$$y_1 = \frac{f''(x_o)}{2!}(x - x_o) + f'(x_o). \tag{2.17}$$

Hence, following this decomposition, the 2nd order Taylor polynomial can be implemented in an easy way using only one DSP. Also, this scheme to decompose Equation 2.2 allows the implementation of a Taylor polynomial of any order.

Storage of the Taylor Coefficients

To store the coefficients of each interval (i.e. x_o, $f(x_o)$, $f'(x_o)$ and $f''(x_o)$) ROM memories are used, one for each variable, as shown in Figure 2.20. In this case, a RALUT method is implemented (see Figure 2.9), and the memory words are addressed by means of the Most Significant Bits (MSB) of the absolute value of the input data, plus an offset depending on the value where the saturation region starts, as is shown in Figures 2.18 and 2.19.

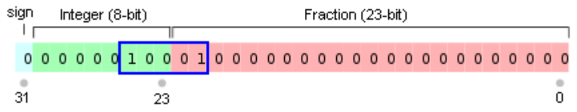


Figure 2.18: Address of the ROM memories for a (0,8) range and 32 intervals

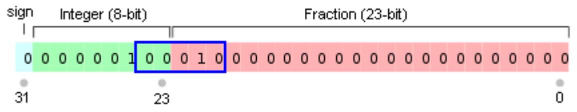


Figure 2.19: Address of the ROM memories for a (0,4) range and 32 intervals

Module Input/Output Ports

To design the Taylor module, a Handshake protocol is used to ensure a correct communication. Hence, the module not only includes the input and the output, it also includes the control signals of the protocol:

- *stb_i*: Strobe signal that warns the module that a new input is available.
- *ready*: Strobe signal used by the module to warn that the output is available. It also works as acknowledgement of the *stb_i* signal.
- *out_read*: Acknowledgement signal used to tell the module that the output has been successfully read.

The final architecture of the core, including the reset and clock signals, can be seen in Figure 2.20.

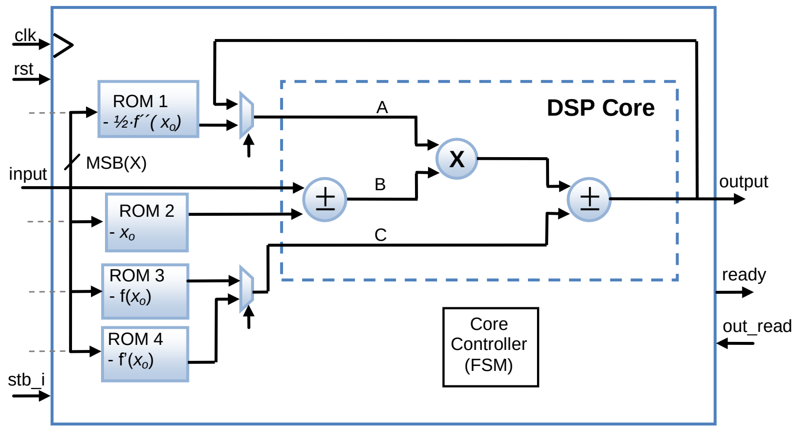


Figure 2.20: Taylor module architecture (positive semi-axis)

Finite State Machine of the Taylor Module

To control the module, a Finite State Machine has been designed (shown in Figure 2.21) that in 5 clock cycles is capable of obtaining the output of the approximation. It has the following states:

- *Idle*: In this state the module waits until an input is received. This is known when the port *stb_i* is active.
- *Is_minus*: Here the sign bit is checked. If the sign bit is '1', then a flag is set and, as working value, the absolute value of the input is taken.
- *Load_val*: Using the value given in the previous state, the values of the memories are loaded and the value of $(|x| - x_o)$ is obtained.
- *y1_compute*: In this state, Equation 2.17 is computed.
- *y_compute*: In this state, Equation 2.16 is computed.
- *Output_ready*: Here Equations 2.5 or 2.6 are applied if the input is negative and the output is trimmed to adapt it to the same data representation

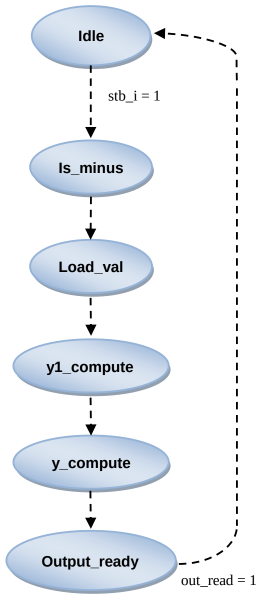


Figure 2.21: Taylor module Finite State Machine

of the input. At the same time the module indicates that the output is ready. The change to the *Idle* state is not completed until *out_ read* signal is '1'.

2.4.4 Simulations of the Proposed Scheme

Before implementing the design, the module has been simulated with Matlab tool using its Fixed-Point Toolbox [251]. To perform the simulation a Matlab script has been written to simulate the Taylor Module. The Taylor module is implemented in a digital HW platform, therefore, the number of intervals (ni) and the Taylor region width (t) should be powers of 2. For example, to obtain an error lower that 10^{-3}, Table 2.3 shows that t should be *6.91* and *ni 24*, however in the HW implementation the value of t is *8* and *32* intervals are selected. Also, the result of Equation 2.17 is trimmed to be used as input parameter in Equation 2.16. The idea is to check if it is possible to reach an error of 10^{-3} with values similar to the ones obtained in Table 2.3, selecting as the fractional part the highest value between Equation 2.14 and Equation 2.15. Hence, the simulation is done with a Taylor region of 16 and 32 intervals. According to Equations 2.14 and 2.15, the fractional part of the number must have 12 bits. If the error is not reached, the number of bits in the fractional part will be increased until the targeted error is reached. If more bits have to be added, it is checked if the new length of the fractional part is acceptable or

Number of intervals (0,8)	Maximum absolute error ε		
	$N_j = 10$	$N_j = 11$	$N_j = 12$
16	1.2×10^{-3}	6.1×10^{-4}	4.86×10^{-4}
32	9.69×10^{-4}	5.20×10^{-4}	2.70×10^{-4}

Table 2.4: Maximum error vs. fractional bits (N_j) for 16 and 32 intervals in the case of a sigmoid function

not.

As can be seen in Table 2.4, to obtain the same error that was obtained in Table 2.3, the fractional part must be at least 11 bits long. The only reason why having even less bits than the ones needed (the error is lower than 10^{-3}) is because the number of intervals is much higher than the required minimum, and therefore, the effect of having fewer bits is counteracted by having more intervals.

The simulations demonstrate that the system, even with a trimming in the approximation process, is capable of approximating the sigmoid function with the required accuracy.

2.4.5 Field Programmable Gate Array Implementation of the Taylor Module

After simulating the proposed scheme, the module is implemented in an FPGA using standard VHDL language. Using this implementation method it is possible to create a system that can be implemented in any FPGA, making the module independent of the platform.

To test the module, an implementation has been done using 16 intervals ($ni = 16$) and 12 fractional bits ($N_j = 12$). Then an extra module has been implemented, an FSL bus wrapper, so the function approximator can communicate with the microprocessor directly by using a pair of FSL buses.

First of all, the approximator has been simulated using Modelsim simulation tool [252], and using the very same code of the implementation. The inputs of the approximator are the codified numbers in the range [0,9], therefore, it is also necessary to check if the approximator saturates the output as it should do. During the simulations, the FSL wrapper has not been used.

The absolute value of the absolute error between the value of the sigmoid function and the value of the HW approximator captured using the simulation tool can be seen in Figure 2.22. The maximum error between the two signals is 4.76×10^{-4}, a value less than the one present in Table 2.4 for 16 intervals and 12 bits in the fractional part of the number.

After checking that the module works correctly on a simulation, the hardware is implemented using Xilinx's ISE Design Suite 14.6. The number of resources of the implemented hardware module can be seen in Table 2.5 for Virtex 5 XC5VLX110T-1, Virtex 6 XC6VLX240T-1 and Kintex 7 XC7K325T-2 devices.

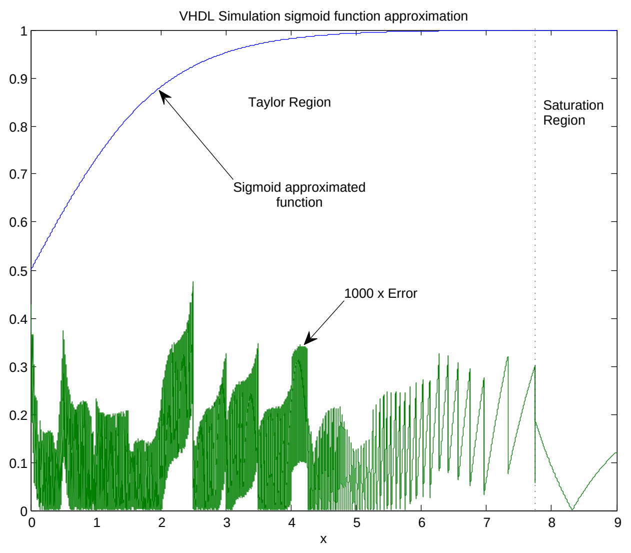


Figure 2.22: Absolute error of a simulated sigmoid function using the Taylor module

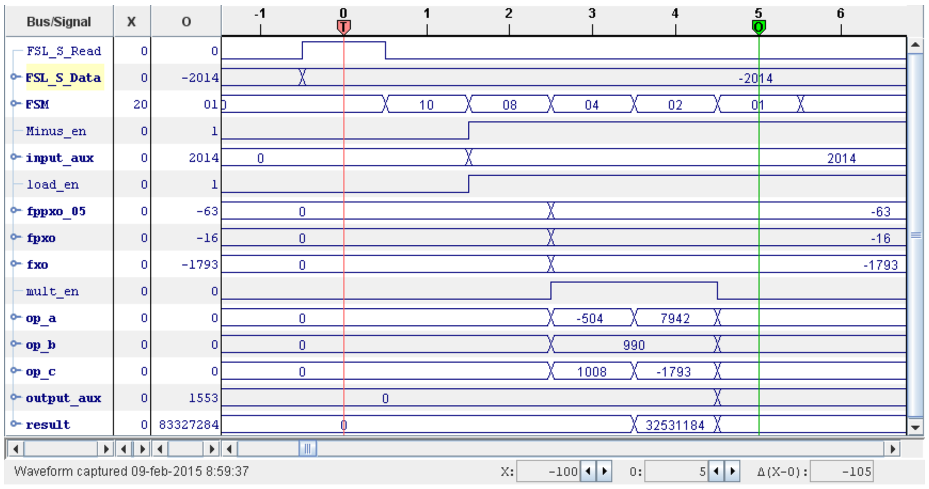
Device	Registers	LUTs	DSPs	Fmax
XC5VLX110T-1	96 (0.13%)	201 (0.29%)	1 (1.56%)	117 MHz
XC6VLX240T-1	96 (0.032%)	190 (0.13%)	1 (0.13%)	120 MHz
XC7K325T-2	96 (0.023%)	191 (0.94%)	1 (0.12%)	166 MHz

Table 2.5: Taylor module occupation for Virtex 5 XC5VLX110T, Virtex 6 XC6VLX240T-1 and Kintex 7 XC7K325T-2 devices

After synthesizing the module integrated in the FSL wrapper, it is connected to a Microblaze microprocessor and implemented in an FPGA with a simple SW partition. The SW partition sends to the module the same inputs used in the simulation, receives the output value and sends both of them to a PC connected to the board using a serial link.

Firstly the system was debugged using Xilinx ChipScope tool. This tool allows viewing of the signals of the modules implemented in the FPGA during the runtime. In Figure 2.23, it is possible to see how, after receiving an input, the number -4.07, the system detects that it is a negative number, takes the absolute value (i.e. signal *input_ aux*), and with it does the calculations. Then, in the final state, Equation 2.5 is applied to the result to obtain the correct output value. It is also checked that in only 5 clock cycles the output is ready to be sent to the microprocessor.

In Figure 2.24, the same input is introduced in the function approximation module, but in this case with a positive sign. The module does not enable signal *minus_ en,* so, in the final part of the process, Equation 2.5 is not applied to



Bus/Signal	X	O
FSL_S_Read	0	0
FSL_S_Data	0	-2014
FSM	20	01
Minus_en	0	1
input_aux	0	2014
load_en	0	1
fppxo_05	0	-63
fpxo	0	-16
fxo	0	-1793
mult_en	0	0
op_a	0	0
op_b	0	0
op_c	0	0
output_aux	0	1553
result	0	83327284

Waveform captured 09-feb-2015 8:59:37 X: -100 O: 5 Δ(X-O): -105

Figure 2.23: Chipscope capture of the Taylor module with a negative input

the result.

To compare the results, the same procedure used in the simulation was carried out. In this case, the maximum error is also 4.76×10^{-4}, the same value as the simulation, so there are no discrepancies between the simulation and the real implementation of the HW in the FPGA.

To sum up, it is possible to approximate a sigmoid function with high accuracy using a 2$^{\text{nd}}$ order Taylor polynomial. Also, our module requires less than a 2% of the available resources present in the smallest FPGA used in the implementation, a Virtex 5 XC5VLX110T, which is the medium size device within its family. Hence, our system can be implemented as part of a system, or as a stand-alone implementation in any device, even in the smallest ones.

2.5 An Artificial Neural Network Architecture Using the Taylor Module to Implement the Activation Fucntion

The ANN architecture presented in Section 2.2 has been enhanced using the Taylor module to implement the sigmoid activation function instead of using a LUT approach based on ROM memories.

When using ROM memories to implement the activation function, it was possible to use one sigmoid function approximator per neuron. However, depending on the number of DSPs present in the FPGA, it might be recommended to use one Taylor module per layer, with the neuron outputs multiplexed, to avoid the use of bigger FPGAs. As consequence, 5 clock cycles per neuron have to be added to the execution time.

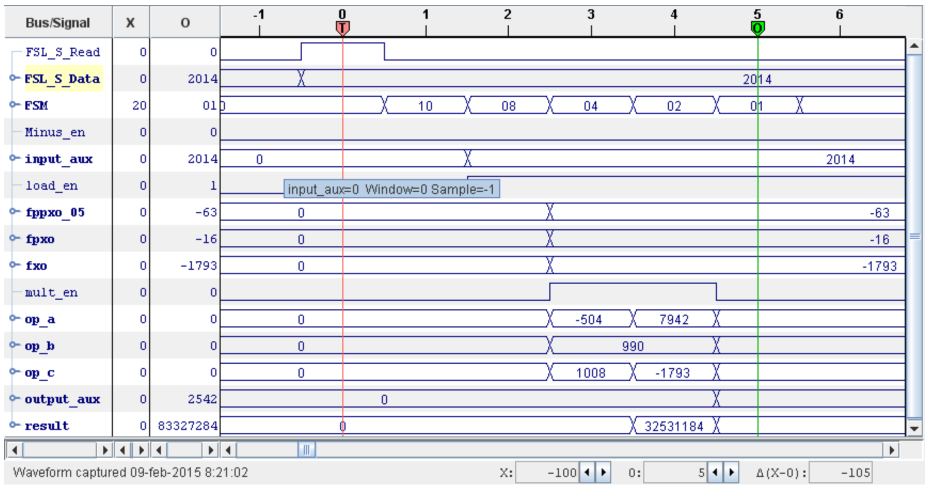


Figure 2.24: Chipscope capture of the Taylor module with a positive input

The option of using one module per neuron has the main advantage of using the same FSM but with minimum changes. It is just necessary to wait until all *out_ready* signals are enabled, and the activation functions of all neurons are computed in only 5 clock cycles. However, to use only one function approximator per layer requires making changes in the control of the ANN core to control the multiplexing of the inputs of the function approximator in both the hidden and output layers.

2.5.1 An Artificial Neural Network Core with One Taylor Module per Layer

To implement this solution, several changes have to be made in the ANN core. Now the inputs of the function approximator are not connected to the outputs of the neurons directly, but to a multiplexer to receive those outputs one by one. Regarding the output of the Taylor module, in the case of the one located in the hidden layer, its output is connected to the inputs of the output layer neurons. In contrast, with the output of the Taylor module located in the output layer, its output is connected to the FSL bus.

Due to the changes in the architecture of the ANN core, the FSM has to be changed too. The new FSM, shown in Figure 2.25, has the following changes:

- *Hidden_act_function*: The FSM does not jump to the next state until the signal *ready* of the function approximation module is active. This state will also enable the *stb_i* signal of the hidden layer function approximation module.

- *Ouptut_Layer*: instead of continuing in this state until the counter of hidden layer neurons reaches '0', the FSM will jump to the state *Hid-*

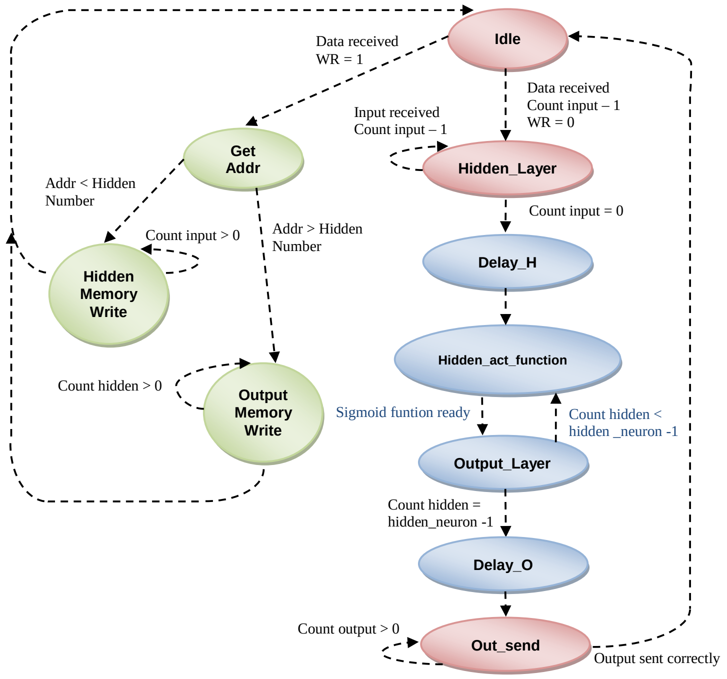


Figure 2.25: FSM of the ANN with one function approximation module per layer

den_ act_function in the next clock cycle if the counter is not '0'. Otherwise, it jumps to the state *Delay_ O*. Also, the enable signal of the output layer neurons is connected to the signal *out_ read* of the hidden layer function approximation module.

- *Out_ send*: The jump condition in the previous design was for the counter of output layer neurons to be '0' and the queue of the FSL bus not full. Now another condition is added to the decrement of the counter and to the jump to the next state of the FSM: the signal *ready* of the Taylor module of the output layer to be '1'.

The ANN endowed with the Taylor module is capable of performing the computations but with an increment in the computational time. Now, for each neuron in the core, 5 clock cycles must be added to computational time. For example, an ANN with parallel ROMs with n inputs, L hidden neurons and m outputs needs $n + L + m + 3$ cycles to compute the outputs of the network. However, the core with shared function approximation modules requires $n + 6L + 5m + 2$ clock cycles.

The main advantage of the present architecture compared with the implementation of the activation function using a LUT approach is the high accuracy that can be obtained. In systems that require very high accuracy, for example a fractional part $N_f \geq 14$, the size of the ROM memories have $2^{N_f+N_i+1}$ positions of N_f+1 bits. Hence, the system could not be implemented in the desired FPGA due to the number of resources. It is in these scenarios that the Taylor module shows its advantage, because with 4 small ROMs to store the coefficients of the intervals and a DSP (see Table 2.5), it is capable of obtaining higher accuracies than implementing the activation function in a LUT-based ROM. This is very useful when the training needs to be accelerated using the ANN core as a coprocessor, because the algorithm requires high accuracy. Hence, the use of the Taylor module is a better solution than using a bigger device. This is true even when an ANN with a single Taylor module per layer, the slowest solution, is used.

2.6 Conclusions

In the present chapter, an FPGA-based HW/SW architecture for an Artificial Neural Network to be implemented in an FPGA-based System on Programmable Chip is presented.

The hardware partition is a fully scalable Multilayer Perceptron Neural Network core implemented using standard VHDL, which means that the system can be implemented in any FPGA independent of the manufacturer. The neurons of the layers of the ANN are implemented using the DSP cores of the FPGA, decreasing the number of resources needed to implement them, and they work in parallel to obtain a faster design. In the case of not having a device with DSPs, the neurons can be implemented using the logic resources of the FPGA.

Two ANN core architectures have been designed, one with ROM memories and another one with RAM memories, to store the weights. The storage of the activation function has been also designed using ROM memories. Also, following the scheme used in the design the implementation of an ANN with more than one hidden layer is a straightforward task.

The SW partition is built around a MicroBlaze soft microprocessor core using the internal memory of the FPGA to make the system faster and to avoid the use of external elements to reduce the size of the entire system. The processor is responsible of performing the control of the system and the execution of the different learning algorithms implemented (i.e. Backpropagation Gradient Descent, Extreme Learning Machine and growing/pruning algorithm). The code of these learning algorithms are written in C programming language as independently as possible of the HW partition, therefore, the code can be reused in different devices performing only minimum changes. Talking about the performance of the algorithms, ELM is a very fast algorithm; however, BP is much slower, hence, the algorithm is accelerated using the ANN core as coprocessor achieving accelerations up to 61%.

In addition, a new method to obtain the activation function has been presented based on Taylor's theorem. Using a 2$^{\text{nd}}$ order Taylor polynomial and the Lagrange form of the remainder, it is possible to approximate the sigmoid and hyperbolic tangent functions with controlled accuracy.

To sum up, a flexible and scalable HW/SW architecture has been designed that can be adapted to different situations. It is capable of covering a wide range of specifications related to size, speed and accuracy. The system has been implemented correctly in different devices; low, middle and high-performance devices, showing that its size requirement is not very demanding, therefore, making it suitable to be implemented in environments where size and/or performance are one of the most important constraints in the design.

Chapter 3

Dynamic Partial Reconfiguration

In Section 1.4.2 Dynamic Partial Reconfiguration (DPR) technology was described in detail. In the present chapter, different applications of DPR in the HW/SW architecture for an Artificial Neural Network (ANN) proposed in Chapter 2 are presented and discussed. The main advantages of DPR are exploited: FPGA size reduction and power consumption reduction.

In the first part of the chapter, a review of the use of this technology in the Intelligent Environments and in the implementation of Soft Computing hardware is done. Afterwards, the different limitations of the designing tool used during this project are presented. In the second part of the chapter, how DPR can be applied to the proposed architecture is explained. As it is shown, partial reconfiguration can be applied either to the entire core or to some parts of the core. All the details about these issues are carefully analyzed. Finally, a description of another alternative to DPR for power reduction is provided.

3.1 Dynamic Partial Reconfiguration in Intelligent Environments

In the last two decades, DPR has been used in the implementation of systems for Intelligent Environments (IE) and Soft Computing (SC) techniques. In the present section, some examples of these implementations are shown.

In 2005 Mermoud et al. [253] applied DPR to create evolving fuzzy systems implemented in FPGAs to provide adaptation at structural changes and parameter tuning. In particular, DPR was used to modify Look-up-Table functions, and hence, to change the implemented design. In 2006 Chalhoub et al. [254] presented a multilayer ANN in an FPGA where only the modules of one layer are implemented, and then by using time multiplexing reconfiguration, the resources are reused by all the layers successively. Its main advantage is the resource saving, however, the main disadvantage is the time needed to perform a reconfiguration. This method makes the system very slow, specially if as the authors say, up to 32 hidden layers can be implemented. Also in 2006, Starzyk et al. proposed the use of DPR to dynamically reconfigure the neuron architecture of a Self-Organizing Learning Array [255], based on a modified Xilinx PicoBlaze microcontroller [256]. The main disadvantage of this implementation is the use of a PicoBlaze (i.e. a complex versatile core) for the implementation of just each single neuron. Instead, more efficient hardware-based methods could be used (e.g. Multiply-Accumulator type cores).

More recently, in 2011, Harbt et al. [257] proposed the use of DPR to change the configuration parameters of a Multiple Target Tracking Driver Assistant System depending on the driving conditions and the changes in the environment. Echanobe et al. [258] proposed in 2012 the use of DPR to reconfigure an ANFIS-like System in an AmI scenario. In 2014 Grantner and Nguyen [259] developed a fuzzy automaton-based decision support system for handicapped children -to test their eye-hand coordination- in which DPR is used to reconfigure the automaton's Hybrid-Fuzzy-Boolean Finite State Machine. In the case of these last two examples, what they propose is something similar to what it is proposed in this chapter to save resources, instead of implementing all the cores. However, these solutions are limited to change the configuration of their systems, and therefore, the cores cannot be replaced by other ones when they are not used.

To sum up, the use of DPR is not widely spread in the area of the IEs and most of the research was done using devices that today are obsolete. Also, most of the use of DPR in IEs and SC is based on changing the configuration of the system but maintaining their original structure with a lack of flexibility. In this project, the main objectives of using DPR are to save resources, power consumption and also to make the system more adaptable.

3.2 Restrictions Introduced by the Design Tool

In Section 1.4.2 DPR was described as a technology that allows changing the configuration of one or several regions of the FPGA while the rest are still working. This ability provides the ANN core described in Section 2.2 new functionality without having to configure again and again the entire system.

As it was mentioned in Section 1.4.3, in this project Xilinx and Altera products have been taken into account due to the variety and resources of their devices. Both companies offer support and tools to reconfigure their FPGAs. In the case of Altera, the DPR design is performed using their Quartus II tool [219], while in the case of Xilinx, PlanAhead tool is used [25], and the Vivado Design Suite [260] since the 2013.3 version.

Unfortunately, DPR presents some limitations when the PlanAhead tool provided by Xilinx is used (this manufacturer's devices are the ones finally used in this project), which at the moment has not enough capabilities. These limitations have implications in the design of the HW modules that are to be reconfigured and therefore, they are described in first place:

1. Some elements of the FPGA cannot be reconfigured individually. For example, in the case of the DSP slices they can only be selected by pairs (even if only an odd number of DSPs slices want to be reconfigured). This is important because this project design contains a DSP slice per neuron, and hence, the reconfiguration of a single neuron is affected by this limitation.

2. It is not possible to have 2 different reconfigurable partitions in the same column of a clock region. Hence, the partitions must be placed in different columns or clock regions, which it normally implies a decay in the maximum working frequency due to the length of the connections.

3. Xilinx's tools do not allow the reconfiguration of internal components of a module. That is to say, to apply DPR in a functional module (e.g. the neurons or the activation functions), this must be implemented as an independent module. This implies some changes in the design, because the modules that are to be reconfigured should be treated as separate entities from the beginning of the design.

An example of the first two restrictions can be seen in Figure 3.1. The partition in the upper side of the image occupies three DSPs (a green rectangle has 2 DSPs) but only two could be reconfigured. Also, both partitions are located in the same columns, which means that they use the same clock region. Therefore, the synthesis will fail due to this last error.

3.3 Reconfiguring the Entire Hardware Partition

This strategy is the simplest way to apply DPR in the design because no changes have to be done neither in the system nor in the ANN core. Also, another advantage is that the core can be replaced by any other core that implements FSL communication buses as our ANN core does. Three different possibilities are discussed: to reconfigure with another ANN topology, to reconfigure it with another Soft Computing core, and to change the accuracy of the core.

3.3.1 Changing the Artificial Neural Network Topology

The idea here is to have stored a collection of synthesized ANN cores (i.e. bitstreams), which model different situations in AmI environments. For example, different periods of the year or different users of the environment. Each core may have not only different parameters (weights, offsets) but also different topologies (layers, number of neurons, outputs). In this scenario, the system configures only the ANN-core that is required in each moment. As a result, power consumption could be saved thanks to those periods in which small networks would be implemented.

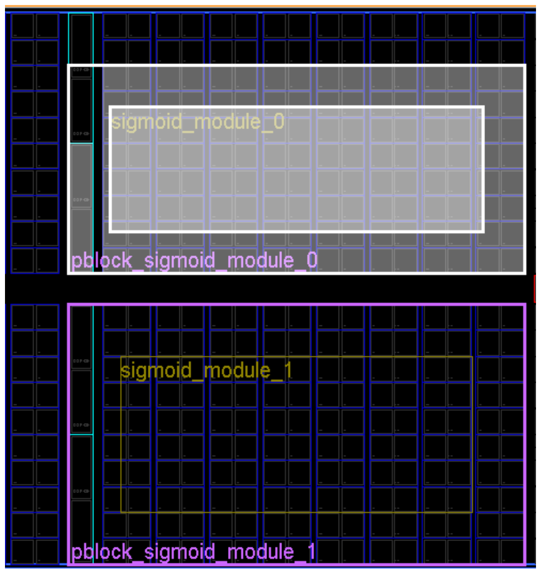


Figure 3.1: Erroneous selection of the reconfigurable regions due to the restrictions of the designing tool. The two regions share a column, i.e. they are placed in the same clock region.

If a smart house is taken as example, an ANN core trained to control the window blinds and the air conditioning could be implemented during the daylight, while during the night time the core can be reconfigured to implement a smaller ANN (and therefore less power demanding), which has been trained to control only the lighting of the house (see Figure 3.2).

The above idea can be extended to other Soft Computing techniques. Using DPR it is possible to change the HW partition to replace the ANN by another core. For example, a multi-output Fuzzy Inference System, a Genetic Algorithm, or any other solution that could be better suited for the actual context (see Figure 3.3). This solution gives more flexibility to the system than the proposals made in [257] and [259].

Reconfiguration Results

To test this proposal, two ANN core examples with different topologies have been implemented in a Virtex6 XC6VLX240T-1 device and with the ICAP controller connected to an AXI-Lite bus: a 7–32–4 ANN and a smaller 7–8–1 ANN. Table 3.1 shows the results of this test. As can be seen the number of resources saved by changing the topology of the ANN core (i.e. by reconfiguring the core) is around a third. Also the dynamic power consumption is reduced in the same proportion. Regarding the reconfiguration time, the system requires about 9.3×10^6 *clock cycles*; *93 ms* for a working frequency of 100 MHz.

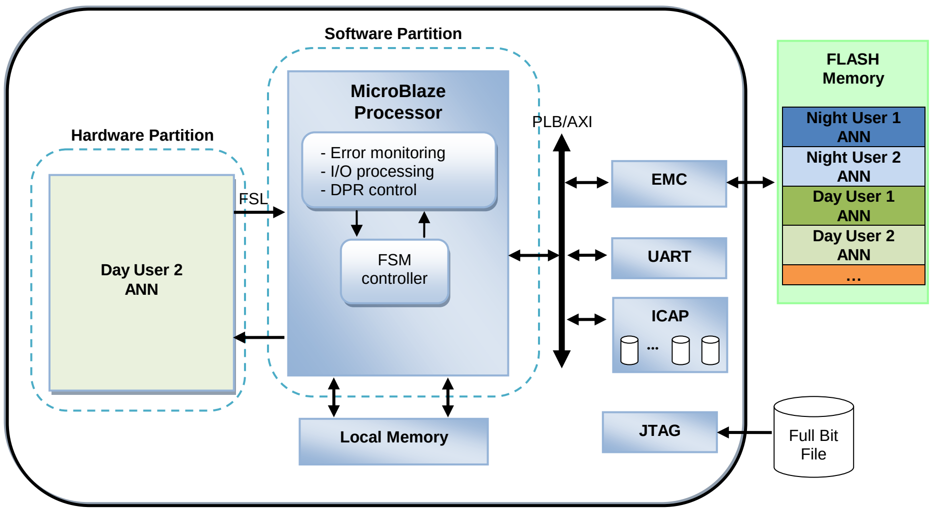


Figure 3.2: Block diagram of the HW/SW architecture with a reconfigurable HW partition to implement ANN cores prepared to be used in different period of the day, or periods of the year

ANN topology	Registers	LUTs	DSPs	Dynamic power consumption (mW)
7–32–4	2362 (0.9%)	3792 (2.5%)	38 (5%)	39
7–8–1	868 (0.29%)	1283 (0.85%)	11 (1.43%)	13

Table 3.1: Resource and dynamic power comparison between an ANN with 7 inputs, 32 hidden neurons and 4 outputs and an ANN with 7 inputs, 8 hidden neurons and 1 output implemented in a Virtex 6 XC6VLX240T-1 device

3.3.2 Changing the Accuracy of the Activation Function

In Chapter 2 it was explained that the size of the ROM memories that are needed to store the activation function values with the precision required in the learning process (i.e. when using the core as coprocessor), can be prohibitive for a given device. A solution, based on the use of the Taylor module to implement an activation function, was also proposed in the same chapter. However, both the size of the core and the execution time are increased, which could be a drawback to be used in on-line mode. To overcome this problem, the reconfiguration of two different ANN cores is proposed: one to be used in the learning process and another one in the on-line stage. The most suitable cores to be used in each stage are the following:

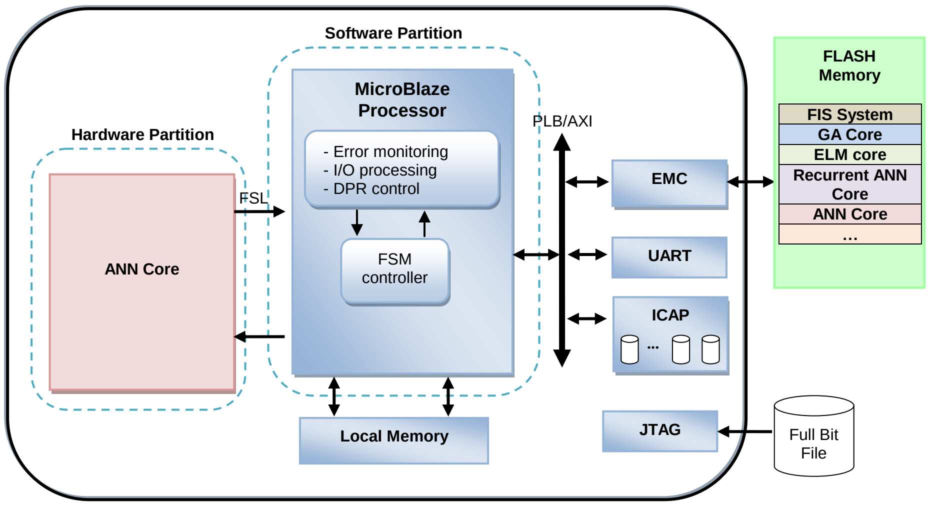


Figure 3.3: Block diagram of the HW/SW architecture with a reconfigurable HW partition to implement different Soft Computing cores

- Learning stage: During this stage, the system is reconfigured to use an ANN core with Taylor modules, one per neuron or one per layer depending on the availability of DSPs in the FPGA. Thus, the system is capable of performing the learning processes using the ANN core as coprocessor, which is much faster than performing the training only by SW.
- On-line stage core: In this stage, the system is reconfigured with a core that uses ROM memories for the activation function. The accuracy is lower but enough for the on-line stage and the system becomes much faster because there is one ROM per neuron.

Reconfiguration Results

Table 3.2 and Table 3.3 show the differences in size and velocity between the two options for a *7–14–4* ANN example and for different devices. As can be seen, a system with a Taylor module per layer and a word length with 14 bits in the fractional part has a processing latency of 122 clock cycles. However, the same core but with only 8 fractional bits and one ROM activation function per neuron, has a processing latency of 30 clock cycles (more than 4 times faster). Also, the number of registers is about 24% lower in the case of using ROM memories and the number of LUTs is about an 8% higher in the case of the Virtex 6 and Kintex 7 devices. In the case of the Virtex 5 device the number of LUTs needed is about the double, however, it only represents the 6% of the LUTs of this FPGA.

Regarding the reconfiguration time, both cores have a 178.52 kB bitstream, therefore, the reconfigurations requires about *77 ms* at 100 MHz.

Device	Registers	LUTs	DSPs	Processing time
XC7K325T-2	1498	2150	20	122 Clock cycles
XC5VLX110T-1	1502	2241	20	122 Clock cycles
XC6VLX240T-1	1498	2146	20	122 Clock cycles

Table 3.2: Resources and response time of an ANN core with one 32-interval Taylor module per layer, 7 inputs, 14 hidden neurons, 4 outputs and 14 fractional bits

Device	Registers	LUTs	DSPs	Processing time
XC7K325T-2	1102	2206	18	30 Clock cycles
XC5VLX110T-1	1106	4740	18	30 Clock cycles
XC6VLX240T-1	1102	2206	18	30 Clock cycles

Table 3.3: Resources and response time of an ANN core with parallel ROM activation function, 7 inputs, 14 hidden neurons, 4 outputs and 8 fractional bits

3.4 Reconfiguring the Internal Modules of the Artificial Neural Network

In this section, we explain how DPR can be used to change some internal elements of the ANN. However, as it was mentioned in the introduction of the chapter, all the modules that are going to be reconfigured have to be implemented from the beginning as independent modules. In particular, the ANN elements that can be reconfigured are the activation function modules and the neurons of the hidden layer.

3.4.1 Changing the Activation Functions

Along this thesis, different methods to train and configure an ANN have been explained. On the one hand, different methods to set the values of the parameters of the neurons (i.e. the weights and the offsets) have been described (e.g. BP and ELM). On the other hand, a method to establish a determined structure, changing the number of neurons in the hidden layer, has been also presented. However, there is another element in the ANN that is susceptible to be changed during the training process, allowing the ANN to have better results: the activation function. As it was mentioned in Section 1.2.2, ANNs can use different activation functions; therefore, to obtain a more refined learning, different functions should be tested. In fact, that is what has been done in the different experiments presented in this project.

However, if different activation functions want to be used when the ANN core is used to accelerate the learning process (see Section 2.3.3), all of them must be implemented in the core. Here is where the problem arises: the implementation of all the activation functions that want to be tested require a huge number of resources of the FPGA, hence, bigger devices should be used, with a consequent increase in power consumption, price, etc. In this section a solution to that problem is presented by using DPR. The goal is to have implemented only one activation function module at a time.

Two different designs to perform the reconfiguration of the activation function modules have been carried out in this work. The first one is a design in which for each layer an activation function module is implemented, while in the second case only one shared module is implemented for all the different layers presented in the network. In the first design, the main advantage is the ability to test all possible activation function combinations, however, its main disadvantage is the high number of reconfigurations that must be done. For example, in case of having n activation functions available and an ANN with one hidden layer and the output layer, the number of reconfigurations is 2^n. However, in the second design the number of reconfigurations is much lower, only one per function, but the main disadvantage is that the same function is used in all the layers, so, the best possible combination might not be found.

Reconfiguration Results

The proposed reconfiguration strategy is tested in a Virtex 6 XC6VLX240T-1 device, and with the ICAP controller connected to an AXI-Lite bus, using a Taylor module per layer with a word length of 19 bits, 14 bits in the fractional part, and 16 intervals. The partial bitstreams of the Taylor module for each function is 22 kB long and the system requires about *928000* clock cycles to perform the reconfiguration of each module, therefore, to reconfigure the two modules about *19 ms* are needed if the system works at 100 MHz.

3.4.2 Removing and Adding the Neurons of the Hidden Layer

In Section 1.2.3 it was explained that the ANN-based system could perform a structure learning to select the best hidden layer size. However, in the case of the original ANN core described in Section 2.2, it is mandatory to implement the maximum number of hidden layer neurons the algorithm can reach. The main problem of this design is the waste of power consumption of the system, especially, when the size of the hidden layer obtained through the structure learning is very small compared with the maximum number of neurons available.

A solution for reducing the power consumption of the ANN core (see Equation 1.20 in Section 1.4.2), can be adopted if we remove -using DPR- the neurons that are not being used. Also, they can be added later if the learning algorithm -after an adaptation process- decides to increase the number of neurons in the hidden layer.

In order to make the reconfiguration of individual neurons possible, they have to be implemented now as independent modules (see Figure 3.4). Thanks to this implementation the partial bitstreams are as small as possible, however, the routing of the design can be quite troublesome.

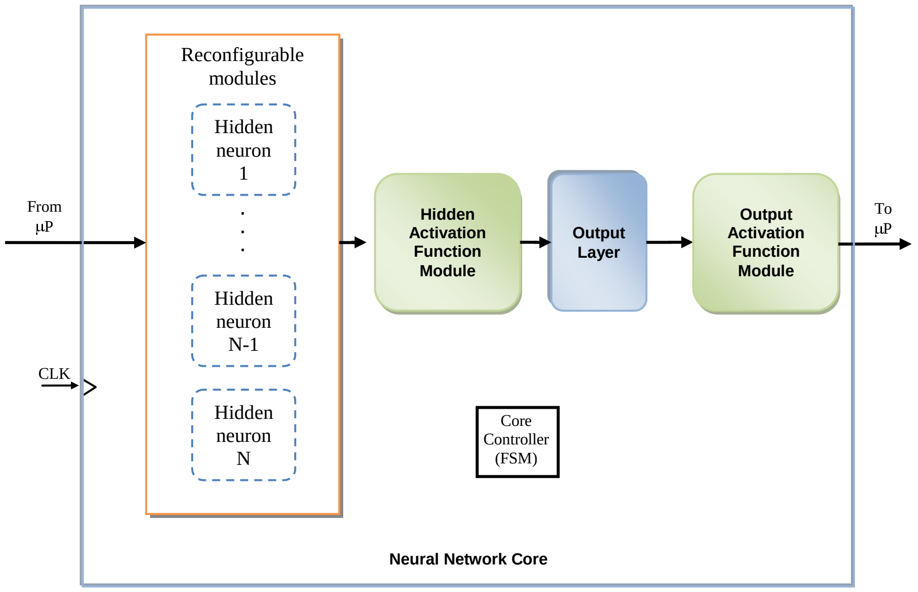


Figure 3.4: Block diagram of the ANN core with reconfigurable hidden-layer neurons

Although the ANN core has been designed to allow the reconfiguration of individual neurons, the PlanAhead tool limitations make preferable the reconfiguration by pairs, in order to avoid wasting resources. Hence, the ANN core has been slightly modified. As a consequence, the system has a slight loss of efficiency. There is also another problem due to the design restrictions: some FPGAs have only one DSP column per clock region (see Figure 3.5). Therefore, in those devices each reconfigurable partition must be implemented in a different clock region, so, the reconfigurable partitions are in different places of the FPGA, which means that the routing and the timing of the design can be seriously affected. This happens, for example in the Virtex 5 XC5VLX110T-1 device; it has 64 DSPs and not all the clock regions are fitted with a DSP column. The device has only DSPs in 8 clock regions, so; only 16 neurons could be made reconfigurable.

In sum, each partition has its bitstream to implement one or two neurons within it, and another bitstream to remove the partition. The size of the partial bitstreams is 20 kB and a reconfiguration time of about *10 ms* is needed to reconfigure each partition in a system with the ICAP controller connected to a PLB bus and with a working frequency of 83 MHz.

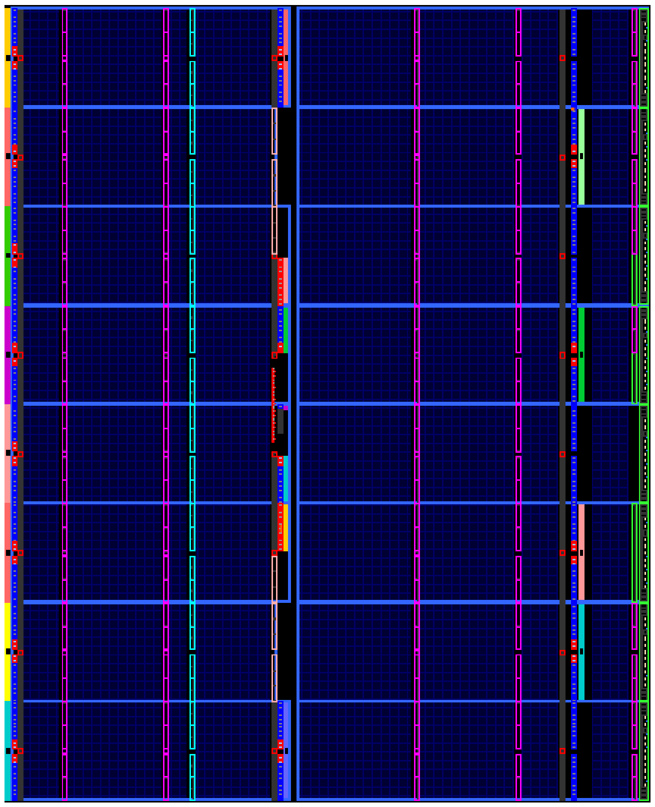


Figure 3.5: Internal architecture of a Virtex 5 XC5VLX110T-1. The DSPs, the light blue rectangles, are located in only one column in the clock regions of the left side

3.5 Power Reduction by Means of Gated Clocks

Another alternative to DPR for power reduction is available in new FPGA families. As Equation 1.20 reveals, dynamic power depends on the working frequency of the system, therefore, another way to reduce that power consumption consists in disconnecting the clocks of the elements or modules that are not used. The main problem when working with clock signals is that they use dedicated low-skew networks to make sure that the difference between two clock signals in different parts of the FPGA have a skew of few picoseconds. However, clock networks do not naturally allocate logic gates, therefore, the place & route tools will place the gated clock signal in the routing network resulting in a drastic increase on the skew of the clock [261]. The glitches in the output of the logic gate are also another problem, because they can cause non desired transitions in the registers.

In the last few years, the FPGA manufacturers have started introducing clock enable signals in their clock networks. These tri-state buffers have been specially designed to be glitch-free and they are located in the clock network, hence, the resulting gated clock signal is routed through the low-skew path. Xilinx has implemented in most of its FPGAs different tri-state clock buffers [262] to control the global clocks or the clocks within a clock region. Altera has also its IP to control the clock signals, including an Enable/Disable signal, called *Clock Control Block* [263]. According to [264] up to a 30% of dynamic power reduction can be achieved in the new Xilinx 7 series FPGAs.

Apart from the manufacturers, there has been a series of research works to create efficient clock gating techniques. In [265] several techniques based on *AND* and *NOR* gates are reviewed, however, their tests demonstrate that the glitches are still unavoidable depending on the technique used. Also, the clock network of the FPGA is not used, hence, they can be only applied to low frequency designs to avoid any problem with the skew of the clocks.

3.6 Conclusions

In the present chapter different applications of Dynamic Partial Reconfiguration technology have been presented to give new functionalities to the proposed HW/SW architecture.

A reconfiguration strategy has been presented for those situations where high accuracy ANN cores are required during the training process, but low accuracies are enough for on-line operation. The solution is given by reconfiguring the core with a low-accuracy ANN core and with a high-accuracy ANN respectively. The high-accuracy core requires more resources and is slower, that is why a fast, but low-accuracy, version is reconfigured in on-line mode.

It has been explained how internal components of the ANN can be reconfigured individually. In particular, the ANN elements that have been reconfigured are the activation function modules and the neurons of the hidden layer. By reconfiguring the activation function modules, the system can select the best activation function to obtain the lowest possible error. This is useful when the system has an on-line adaptation mechanism that decides to change the activation functions to better cope with the environment changes.

In addition, a methodology to reconfigure the neurons of the hidden layer using the structure learning described in Section 2.3 is presented. Although the proposed methodology allows reconfiguring the neurons individually, the limitations of the devices (specially the small ones) and also of the designing tools, make very difficult to fulfil this goal completely. Instead, the neurons have to be reconfigured by pairs or even in larger modules, specially if the number of neurons that must be reconfigured is too high and the workstation used in this design is not powerful enough to handle the number of reconfigurable partitions.

As a final conclusion it can be said that DPR provides the ANN core with more capabilities and functionalities without needing to implement all the cores or modules, hence saving space, and therefore, price and power consumption. However, it must be taken into account that the inclusion of the partial bitstream storage device and the rest of elements needed to perform the reconfiguration (i.e. the ICAP controller) can, sometimes, make the power consumption higher than the one saved or the power saving negligible. Hence, the use of the clock enable signals of the new FPGA devices and new design techniques (e.g. the *Divide-and-Conquer Strategy* presented in [27]) could be in some cases a better solution than using DPR to decrease the dynamic power consumption, making also the design process simpler.

Chapter 4

Applications of the Hardware/Software Artificial Neural Network Architecture to Real-world Intelligent Environments

In this chapter the applications in Intelligent Environments carried out with the architecture presented in Chapters 2 and 3 are shown. The proposed architecture is applied in two different scenarios, an intelligent inhabited environment, the iDorm dormitory developed by the University of Essex, and a smart car, the AmI-car (see Section 1.1.5), where AmI stands for Ambient Intelligence applied to the car environment [36].

4.1 Development of an Embedded Agent for an Intelligent Inhabited Environment

A major challenge in Ambient Intelligence research deals with the complexity of intelligent systems for human behaviour modelling in inhabited environments. To face this problem, some researchers proposed a distributed solution based on embedded agents [266].

In a broad sense, an intelligent agent is a computer system, SW and/or HW-based, that is able to perform autonomous actions driven by a goal [267]. An intelligent agent for AmI environments must be capable of perceiving the environment through sensors, take decisions and act by means of actuators (see Section 1.1.2). It is widely accepted that autonomy, reactivity and proactivity are the main capabilities that characterize the behaviour of intelligent agents.

In addition, some kind of social ability might be expected, that is to say, the capability to interact with other agents (and even humans) in order to satisfy their design objectives.

ANNs have been shown to play a key role in agent reactivity - the ability of the agent to perceive an environment and respond to the changes that occur in it according to its objectives. In [268] and [269] radial basis function networks are used for modelling the behaviour of robotic agents. A health monitor intelligent agent based on ANNs is proposed in [270] in which the agent is able to classify symptom patterns into medical conditions and suggest appropriate actions in a ubiquitous healthcare environment. Other ANN-based intelligent agents can be found in sectors such as control systems for energy and comfort management in buildings [271].

The above research works validates the suitability of ANNs for modelling agent reactivity in complex decision making processes. Furthermore, they highlight the capability of ANN-based agents to learn from the environments and to dynamically adapt to changing situations. Most of those agents are software-based computer systems implemented on personal computers. These kind of approaches is unsuitable when restrictive design specifications such as high performance, reduced size, or low power consumption are required. In this sense, present Field Programmable Gate Arrays (FPGAs) provide a suitable technology to materialize single-chip adaptive intelligent agents based on ANNs [48]. FPGAs provide a wide variety of resources that enable the adaptation of the agent at different levels, ranging from the physical level to the software level, including the architectural ones (see Section 1.3).

In this section, a multilevel adaptive ANN scheme for the development of intelligent agents is proposed. The scheme has been successfully implemented in a single-chip FPGA with DPR capability. Software learning algorithms are applied to adapt the agent behaviour (i.e. neural network parameters) at the system level, while DPR is used to modify the agent at the physical and architectural level (i.e. neural network topology). As a case application, we present the development of an intelligent agent for real-time control of an AmI environment. The FPGA-based adaptive agent was tested with experimental data obtained in a real inhabited AmI space - the intelligent dormitory (iDorm), a monitored living space developed by researchers of the University of Essex, briefly mentioned in Section 1.1.5.

4.1.1 iDorm Intelligent Environment

The iDorm AmI space is a one-room dormitory (see Figure 4.1) at the University of Essex, which was develop by the Intelligent Environment Group [28], [29], [30].

Researchers collected the thousands of data from the interaction with the environment of different users during periods of several days in different seasons of the year. This is not a specially demanding application in terms of processing speed, but in any other aspect of the requirements of an embedded intelligent agent application (i.e. small size, low-power, multiple inputs and

Figure 4.1: Pictures of the iDorm dormitory at the University of Essex

Input	Variable	Output	Actuator
In1	Internal light level	Out1	Dimmable spotlight 1
In2	External light level	Out2	Dimmable spotlight 2
In3	Internal ambient temperature	Out3	Dimmable spotlight 3
In4	External ambient temperature	Out4	Dimmable spotlight 4
In5	Pressure in chair (binary)	Out5	Blind state
In6	Pressure in bed (binary)	Out6	Bed light state
In7	Hour	Out7	Desk light state
		Out8	Heat state

Table 4.1: Description of the inputs and outputs of the iDorm environment

outputs, adaptability to a changing environment and autonomy) this is a very suitable real-world data set for the development of this research. In fact, trying to model and predict human activity and preferences over time, even for simple human-environment interactions, is an extremely challenging problem.

The environment uses 7 input variables obtained from the sensors distributed around the dormitory, and 8 outputs corresponding to as many actuators. Inputs and outputs are summarized in Table 4.1.

4.1.2 Topology and Parameters of the Intelligent Agent

The kernel of the proposed intelligent agent is a Multilayer Perceptron Neural Network (MLP) with an adaptive topology, and learning capabilities. Concerning the ANN topology selection, a three-layer interconnected structure, with a variable-size hidden layer, has been devised. The size of the hidden layer (i.e. number of hidden neurons) is a critical design parameter as it has a great impact on the modelling capability of the neural network. It is well known that too few hidden neurons provide poor performances, while an excess of hidden neurons could blur the generalization capability of the network.

The topology selection is based on a growing/pruning structure algorithm, while the parameter training is performed using a Backpropagation Gradient Descent (BP) algorithm (see Section 2.3). When the topology and the parameters of the ANN have been obtained, the non-needed neurons can be erased by means of DPR as it was explained in Section 3.4.2. The operation of the agent comprises two different operation modes, the off-line stage and the on-line stage.

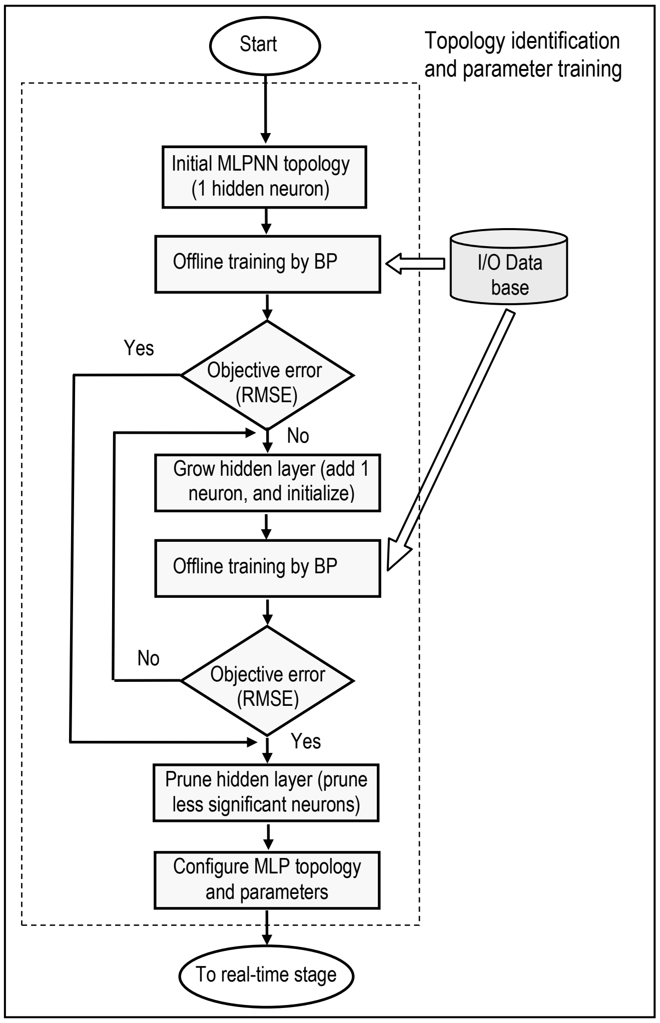

Start
Topology identification and parameter training
Initial MLPNN topology (1 hidden neuron)
Offline training by BP
I/O Data base
Yes
Objective error (RMSE)
No
Grow hidden layer (add 1 neuron, and initialize)
Offline training by BP
No
Objective error (RMSE)
Yes
Prune hidden layer (prune less significant neurons)
Configure MLP topology and parameters
To real-time stage

Figure 4.2: Flowchart of the of the agent operation in the off-line mode

Off-line Stage

In the off-line stage (see Figure 4.2) the growing/pruning algorithm is used to develop an initial ANN topology. This stage starts with the definition of a simple ANN topology composed of an n-neuron input layer, an m-neuron output layer, and a hidden layer with a single neuron. This ANN is trained using the BP algorithm with a set of input-output (I/O) training data. If the ANN is able to model the agent behaviour in a satisfactory way - with a RMSE less than a given value, then the ANN topology is validated; the agent behaviour is evaluated using a validation data set instead of the training data set. After that, the agent is able to perform real-time control of the environment. On the contrary, if the initial topology is too simple to provide the required modelling capability, a new neuron is added to the hidden layer, and the learning algorithm is performed again. The initialization of the new weights and bias is done using the Nguyen-Widrow method (see Equations 1.8 to 1.12). The hidden layer is grown, one neuron at a time, until the desired performance is reached. Finally, the ANN topology is optimized with the aim of configuring the agent with the simplest topology able to fulfil the modelling requirements. This task is performed by means of the so-called simplified brute-force pruning method [21]. It evaluates the significance of each hidden neuron, comparing it to the fully grown network, and eliminates the irrelevant ones and all their corresponding connection weights. This topology will be used to configure the initial intelligent agent.

On-line Stage

In the on-line stage (see Figure 4.3) the agent performs real-time control of the environment. The agent behaviour in this stage is continuously monitored to guarantee that the system performs properly. Whenever a degradation of the modelling capability of the agent is detected, the on-line BP is activated (adaptation at the system level). In most cases this action is enough to restore agent performance. However, in a lifelong operation mode, a deeper adaptation of the agent affecting its topology could be required. In these situations, the growing/pruning method used to develop the initial topology is activated again but starting from the present topology (adaptation at the architectural level).

To test the proposed methodology, the agent has been firstly implemented in software and its behaviour has been simulated using Matlab programming tools. The MLP and the adaptation algorithms, (i.e. topology identification and parameter training) have been simulated as a prior step to their implementation on a FPGA. The following section presents two representative experiments which have been selected from a more comprehensive study performed with the same datasets, but considering different modelling criteria (i.e. different target RMSE).

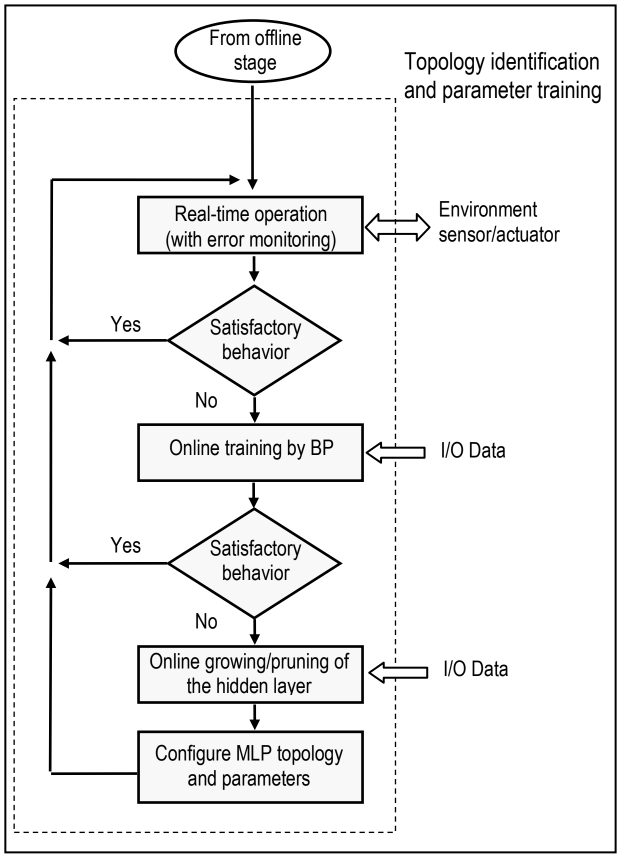

From offline stage
Topology identification and parameter training
Real-time operation (with error monitoring)
Environment sensor/actuator
Satisfactory behavior
Yes
No
Online training by BP
I/O Data
Satisfactory behavior
Yes
No
Online growing/pruning of the hidden layer
I/O Data
Configure MLP topology and parameters

Figure 4.3: Flowchart of the of the agent operation in the on-line mode

4.1.3 Simulation Results

The experimental data used are two different iDorm datasets, *Set1* and *Set2*, corresponding to the same user but different environmental conditions. The agent uses the 7 inputs mentioned in Table 4.1, however, in a first experiment only the first four outputs are used, the four dimmable spotlights, while in a second experiment all outputs are used.

The procedure followed with the datasets to evaluate the adaptive-ANN was similar in both experiments. First, the 408 input/output samples of the dataset *Set1* were randomized. Then, each random set was split into a training set and a validation/test set comprising 50% of the samples respectively. In both experiments the second dataset *Set2*, has been used to rigorously test the agent behaviour.

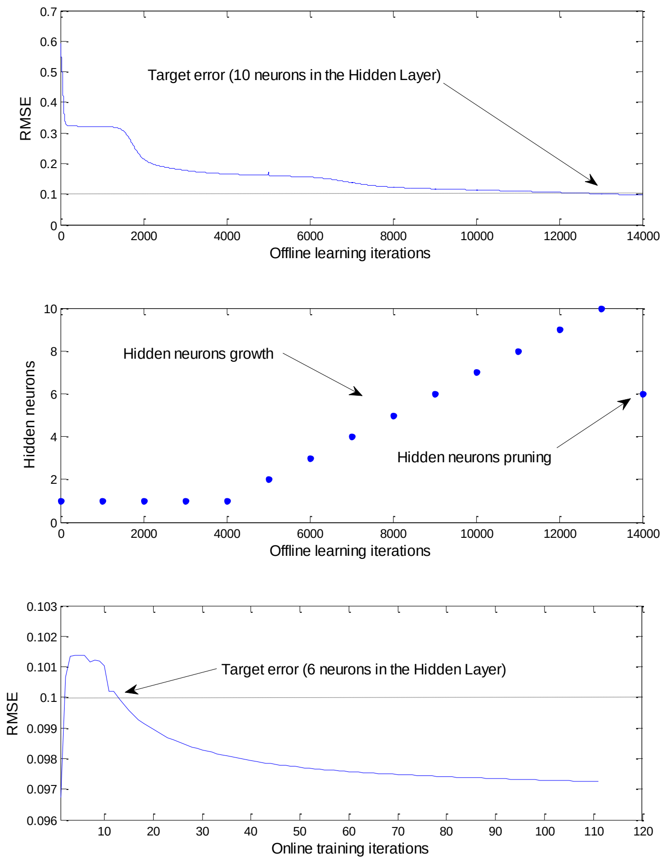


Figure 4.4: a) Evolution of the RMSE, and (b) the corresponding number of hidden neurons during the off-line stage of Experiment 1. (c) Evolution of the RMSE during the on-line stage of Experiment 1

Experiment 1

Figures 4.4 a) and 4.4 b) show the RMSE evolution during the off-line learning stage, and the corresponding number of neurons in the hidden layer respectively. First, a 7–1–4 ANN topology was trained during 5000 epochs. However, the target error, which has been set at 0.1, was not reached. Therefore, the ANN was grown by adding a single neuron to the hidden layer. Each new neuron entails a 1000 iteration learning cycle. As can be seen, the growth process finished with a 7–10–4 topology which was able to match the target error. Then, the pruning process reduced the hidden layer to only 6 neurons, that is to say, the off-line stage concluded with a 7–6–4 topology.

Figure 4.4 c) presents the evolution of the RMSE during a particular on-line situation where the agent has to cope with data of the same user but in a different context (i.e. a different season of the year). As can be seen, due to the changes produced in the user habits, the objective error is slightly surpassed. The agent, who is aware of the system behaviour, activates the on-line training procedure. This procedure performs 100 iterations of the BP algorithm with the aim of reducing the RMSE to an acceptable level. A few iterations were enough to return the system to the satisfactory behaviour zone; no additional neurons were required.

In Figure 4.5 the four outputs of the system for a 3 day period can be seen in blue and their target values in green. It can be checked that both plots are very similar, as a result, the system can be implemented in a real scenario like iDorm. In this experiment, a relatively high value has been set as maximum allowable error, however, lower errors can be achieved if the number of learning cycles is increased. This will result in higher execution times of the off-line stage. If Table 4.3 is checked it can be seen that the duration of the training is not too high, and it can be performed during downtime periods.

Experiment 2

In the second round of experiments, the whole set of inputs and outputs were considered. Figures 4.6 a) and 4.6 b) present the results obtained during the off-line learning stage. The initial topology, a 7–1–8 ANN, was trained and grown until a satisfactory behaviour of the agent was obtained. Then, the hidden layer was pruned by removing 5 irrelevant neurons. The off-line stage finished with a 7–14–8 ANN topology. Please note that the number of epochs had to be doubled in order to cope with the complexity of the experiment (i.e. eight outputs instead of four) without degrading the agent modelling capabilities. In the on-line stage, the same procedure as in the previous experiment was followed. In this case, the same changes in the user habits had greater impact on the agent behaviour (see Figures 4.6 c) and 4.6 d)). The experiment required three new neurons in the hidden layer. In the on-line stage, the number of iterations remains equal to 100, as in the previous experiment, to avoid unaffordable real-time training periods.

4.1.4 Hardware/Software Architecture of the Intelligent Agent

To achieve the requirements of the above adaptive agent system, we propose a hardware/software solution based on FPGAs with DPR capability. The HW/SW architecture presented in Section 2.2 has been adapted to implement the adaptive intelligent agent. Without limiting the generality of the architecture, let us consider a seven-input agent compatible with the experiments presented in Section 4.1.3. Figure 4.7 depicts a system-level block diagram of the intelligent agent.

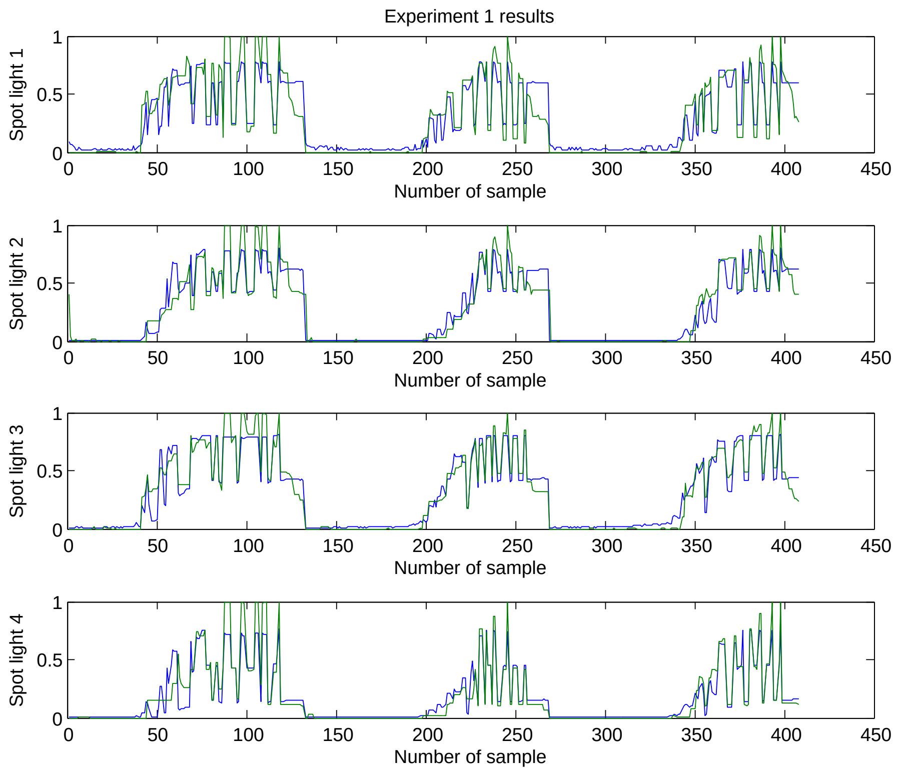


Figure 4.5: Outputs of the Experiment 1 after the off-line stage (blue) and their target values (green) for a 3 day period between 9am and 11pm

Software Partition

The HW/SW architecture shown in Figure 4.7 is based on the block diagram proposed in Figures 3.2 and 3.3. The software partition also performs the control of the whole system and several specific tasks: I/O processing, signal conditioning, system monitoring, parameter training by means of BP, growing/pruning evaluation, and multilevel system adaptation. The software partition has been developed on a MicroBlaze microprocessor, enhanced with a floating point unit (FPU) to accelerate the computation of the software modules. The architecture has been customized with several internal peripherals (I/O peripheral, UART, user switches, and a timer), 256 KB on-chip memory, and a FLASH memory where the partial bit files for dynamic reconfiguration of the agent are stored. The FLASH memory is interfaced with the MicroBlaze via the external memory controller (EMC) and the PLB bus. Finally, a key component of the proposed Microblaze system is the Internal Configuration Access Port (ICAP), as it has been explained in Section 1.4.2. This peripheral enables the FPGA to be self-reconfigured. It receives from the microprocessor

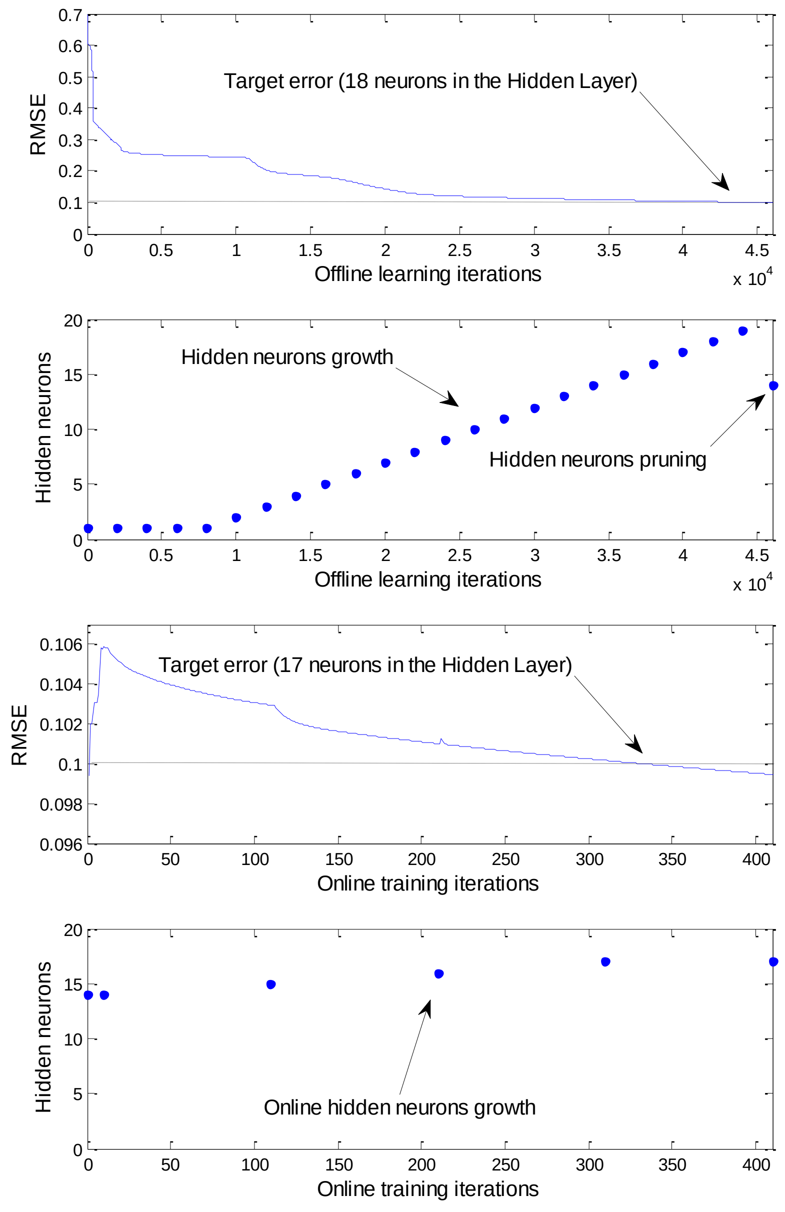


Figure 4.6: (a) Evolution of the RMSE, and (b) the corresponding number of hidden neurons during the off-line stage of Experiment 2. (c) Evolution of the RMSE, and (d) the corresponding number of hidden neurons during the on-line stage of Experiment 2

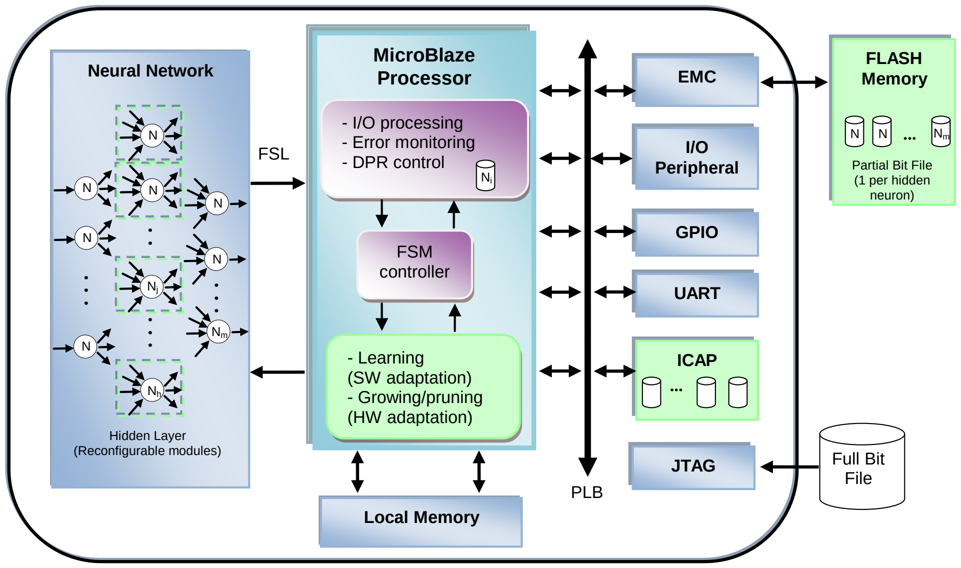


Figure 4.7: Block diagram of the HW/SW architecture of the adaptive intelligent agent. The software partition is developed on the MicroBlaze processor, while the reconfigurable ANN core is implemented in the hardware partition

a partial bitstream, previously loaded from FLASH memory, and reconfigures the corresponding hardware module.

Hardware Partition

The HW partition is the ANN core designed in Section 2.2 using ROM memories for the activation function, one per neuron, and a word length of 13 bits, with 8 bits in the fractional part. To be able to reconfigure the neurons of the hidden layer, these ones were implemented as independent modules as it was mentioned in Section 3.4.2.

The proposed architecture was implemented using the ml605 board which features an FPGA of Xilinx's Virtex-6 LXT family [272]. Table 4.2 summarizes the resources required to synthesis a 7-input 4-output agent with different configurations of the hidden layer. An adaptive agent with 32 hidden neurons (i.e. the largest Hidden Layer) uses less than 30% of the resources available in the smallest device of this family, so more complex agents could be developed.

4.1.5 Implementation Considerations and Results

The single-chip intelligent agent is intended for real-time control of the environment and on-line multilevel adaptation. Therefore, there are three main time-critical processes involved in the agent operation that have to be analyzed:

Component	LUTs	Registers	DSPs	36-Kbit RAM blocks
Intelligent agent	9086 (6%)	6786 (2.25%)	41 (5.34%)	65 (15.63%)
MicroBlaze	2819 (1.87%)	2463 (0.82%)	5 (0.65%)	64 (15.39%)
32 hidden neurons	2304 (1.53%)	1568 (0.52%)	32 (4.17%)	-
8 hidden neurons	576 (0.38%)	392 (0.13%)	8 (1.04%)	-

Table 4.2: Resources report for the 7-input 4-output agent for a Virtex 6 XC6VLX240T-1 device

1. On-line software algorithms. This category includes the BP training algorithm as well as the growing/pruning algorithm. The most time-consuming algorithm is by far the BP training. Figure 4.8 shows time required by a MicroBlaze to perform a single training iteration, as a function of the neural network size, for a different number of output variables.

2. Real-time hardware computation of the feed-forward ANN. The computation of the three layers of an ANN core with n inputs, L hidden neurons and a single output is performed in $n + L + 5$ clock cycles, where the FSL communication delay has been included. If the ANN has more than one output, as many cycles as additional outputs have to been added in order to account for FSL pipelining.

3. Partial reconfiguration of the ANN core. The reconfiguration time depends on the size of the partial bit-stream which in turn depends on the size of the reconfigurable block. The size of the reconfigurable regions has been selected in order to allow the configuration of, at most, 8 hidden neurons per region. The corresponding bit-stream occupies 45.089 bytes, and its reconfiguration time is 20.9 ms.

The above measures were performed with an 83.33 MHz clock (the maximum frequency allowed in this design by the hardware ICAP). Table 4.3 summarizes the above timing considerations for different MLP topologies. As can be seen, a 100 iteration training cycle could be performed in a few seconds (e.g. a 7–8–1 ANN requires 3.6 second, while a 7–32–4 ANN requires less than 20 seconds). Concerning the feed-forward ANN, the ANN core requires only *0.22* μs to evaluate a 7-8-1 ANN and *0.57* μs to compute a 7–32–4 ANN, respectively. This performance allows true real-time operation of the agent. On the contrary, an embedded system based on a whole software implementation of the feed-forward ANN would have increased the computation time more than three magnitude orders.

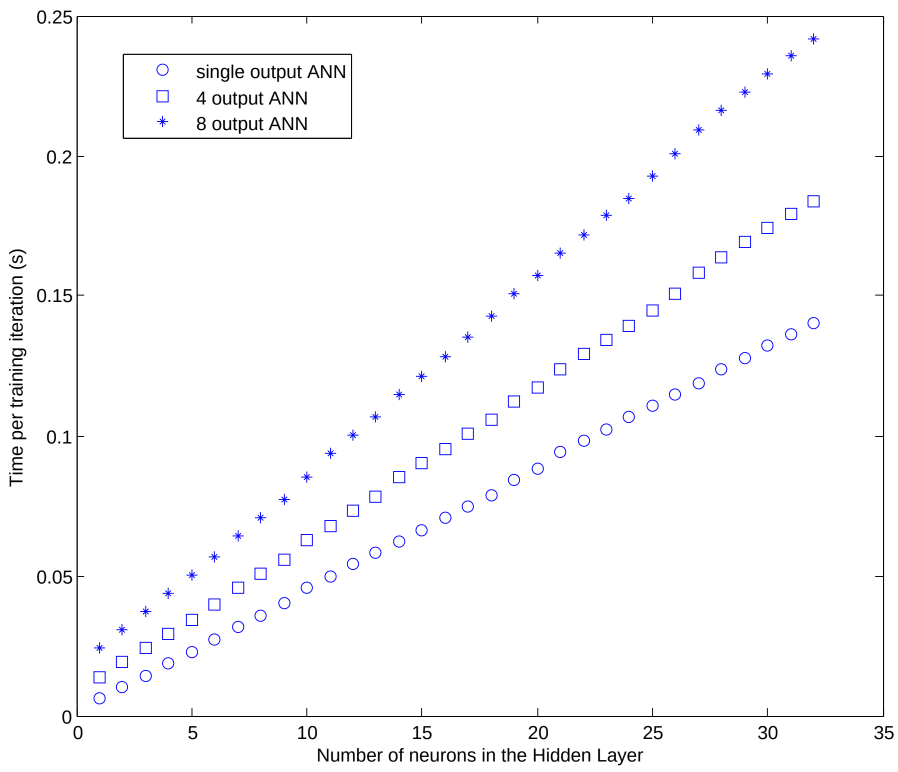


Figure 4.8: Time required performing a single training iteration with the MicroBlaze processor as a function of the neural network size

Process	7–8–1 ANN	7–16–1	7–32–1	7–32–4
100 iterations BP (SW)	3.6 s	7.1 s	14 s	18 s
ANN (HW)	216 ns	390 ns	520 ns	570 ns
ANN (SW)	0.065 ms	0.128 ms	0.25 ms	0.32 ms
DPR (hidden neurons)	20.9 ms	20.9 ms	20.9 ms	20.9 ms

Table 4.3: Timing considerations of the agent performance

To summarize, concerning the adaptation of the topology, the reconfiguration time of the selected reconfiguration regions is slightly greater than 20 ms, while the BP training algorithm requires several seconds to perform a 100 iteration cycle. These timing constraints can be assumed in most typical AmI environments, where the adaptation of the agent can be performed on-line but during downtime periods. This is true in practical intelligent environments (e.g. smart homes, intelligent buildings, etc.), and even in more restrictive environments such as, for example, smart cars. It can be concluded that the adaptive intelligent agent is suitable for the implementation of a wide variety of applications demanding high performance in real-time.

4.2 Development of a Real-time Driver Identification Embedded System for Ambient Intelligence Applied to the Car Environment

Ambient intelligence (AmI) concepts and technologies developed during the last decade have great potential to improve in-vehicle comfort and safety, mainly due to their non-intrusive nature. The car can be viewed as an intelligent environment (i.e. smart car) and the ideas presented in previous chapters concerning AmI can be applied to the car and its occupants with the aim of improving driver performance and the overall traffic safety. The main challenge consists in adapting AmI concepts to the vehicle, its occupants, and the surrounding environment [273], [274].

Innovation in car safety over recent decades has undoubtedly contributed to a reduction in traffic accidents, even though the number of cars on the roads in the developed countries continues to rise. As a consequence of continuous technological advances, mainly in the areas of microelectronics and communications, new safety systems are being developed and incorporated into cars as standard equipment [275], [276], [277]. However, the main source of insecurity in a car is the driver himself, and many traffic accidents are wholly or partly caused by the driver. The availability of Advanced Driver Assistance Systems (ADAS), for safety and well-being, is becoming increasingly important in order to avoid traffic accidents caused by fatigue, stress, or distractions, especially since the driving population is getting older [89], [278]. The identification of the driver and his/her behaviour can be used in different ways in an AmI-Car, for example, in terms of comfort adapting the environment of the car to the needs of the user; or in terms of safety and security, developing personalized ADAS.

The aim of this work is to model individual differences in driving behaviour of a group of drivers, and identify the driver in real-time by using the developed models. Next the main characteristics of the data collection are introduced and the selection of signals, from the whole set of driving behaviour signals, is justified.

4.2.1 Uyanik Instrumented Car

In the last decade there has been increasing research activity concerning driving behaviour signals and their potential application in the development of ADAS [274], [31], [32], [279]. These particular signals can be obtained in a non-intrusive manner, without disturbing the driver, as opposed to video or audio signals which are the basis of some current ADAS [280], [281].

The driving behaviour data collection was supplied by the “Drive-Safe Consortium”. It was collected in Istanbul with an instrumented car called

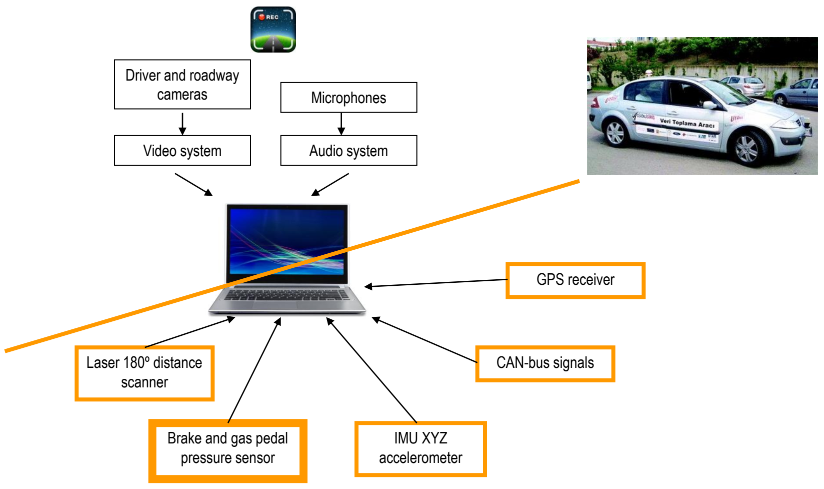

Driver and roadway cameras
Microphones
Video system
Audio system
GPS receiver
CAN-bus signals
Laser 180° distance scanner
Brake and gas pedal pressure sensor
IMU XYZ accelerometer

Figure 4.9: Picture of the instrumented car *Uyanik* and the signals captured through the CAN-bus (in orange), video and audio recorders

Uyanik, which is a sedan car equipped with different sensors [31], [32] (see Figure 4.9). The complete data set (84 male and 17 female) includes audio and video recordings, CAN-bus signals, pedal-sensor recordings, 180° laser range finder, and XYZ accelerometer recordings.

The car route is around 25 km (about 40 minutes), and includes different kinds of sections: city, very busy city, highway, highway with less traffic, a university campus, etc. The route is the same for all drivers; however, the road conditions differ depending on traffic and weather. Approximately half of the driving sessions include driving while completing specific tasks with the aim of disturbing the attention of the drivers: signboard and plate reading, different types of dialogs on mobile phones, and conversations with passengers. To avoid additional noise sources, these driving periods were not considered.

4.2.2 Driver Identification Based on Extreme Learning Machines and Statistical Analysis

Although BP-based ANNs have been successfully applied to solve numerous problems, as much for classification as for regression applications, they present some drawbacks that make them unsuitable for an increasing number of cutting-edge applications. It is well known that the design of BP based ANNs is a time-consuming task that depends on the skills of the designer to obtain effective solutions. The designer has to select the most suitable network topology, optimize the parameters to avoid over-fitting, and be aware of local minima. As a consequence, applications requiring autonomy (i.e. no human intervention) and real-time adaptation are difficult to manage using this approach. These drawbacks are especially difficult to overcome when a single-chip embedded system is required. Other approaches in the literature are oriented to globally optimize the structure and parameters together. However, they normally include trial-and-error steps or are based on iterative progress, which make them not appropriate for the implementation of embedded systems [282], [283]. Another mature machine learning technique is Support Vector Machine (SVM) [284]. SVM is free of local minimum, and is able to improve the generalization performance of traditional ANNs for some important application domains (e.g. machine vision, handwritten character recognition, medicine and bioinformatics applications, among others) [285]. However, autonomy and real-time adaptation are also difficult to achieve with an SVM embedded solution because of the strong dependency of this paradigm on its design parameters.

In Section 1.2.3 Extreme Learning Machines (ELM) were presented. This new algorithm has demonstrated its value because it outperforms conventional BP-ANNs and SVM in some aspects [20], [286].

The development of hardware for embedded ELM is still in an early stage. In [287] the authors provide a training procedure for ELM that aims at reducing the size of the network. This approach introduces a cost function that favours sparse solutions, and the network neurons can therefore be pruned without loss of accuracy. In that work, a Single-hidden-layer Feedforward Neural Networks (SLFN) is implemented using reconfigurable devices. Two technologies are tested: FPGAs and Complex Programmable Logic Devices (CPLD). However, in both FPGA and CPLD approaches the learning stage is performed out of the chip. In a recent work [288], an exhaustive analysis of three different computation architectures for the learning algorithm of ELM is presented. The authors provide speed of operation, bit-length accuracy in fixed point arithmetic, and logic resource operation based on an FPGA. The circuit has been developed as a hardware IP (Intellectual Property) core for integration with other digital modules.

In this section, a high-performance embedded system for ELM is presented. The proposed solution is a scalable HW/SW architecture based on the system proposed in Chapter 2. It provides high speed, small size, low power consumption, autonomy, and true capability for real-time adaptation (i.e. the learning stage is performed on-chip). The developed system is able to deal with a highly demanding multiclass classification problem such as the driver identification system for smart car proposed in this application.

Signal Selection and Data Processing

The data used in this experiment are obtained from the database presented in Section 4.2.1. The data used are provided by an Inertial Measurement Unit (IMU): yaw rate (deg/s), X-axis acceleration (g), Y-axis acceleration (g), and Z-axis acceleration (g). These signals were obtained in a non-intrusive way and provided useful information about the driving style of the drivers. The selected signals were sampled at 32 Hz and a pre-processing step of the data was performed before delivering them to the SLFN inputs. Firstly the samples of the signals were windowed using a 64-sample Hamming window (i.e. a bell-

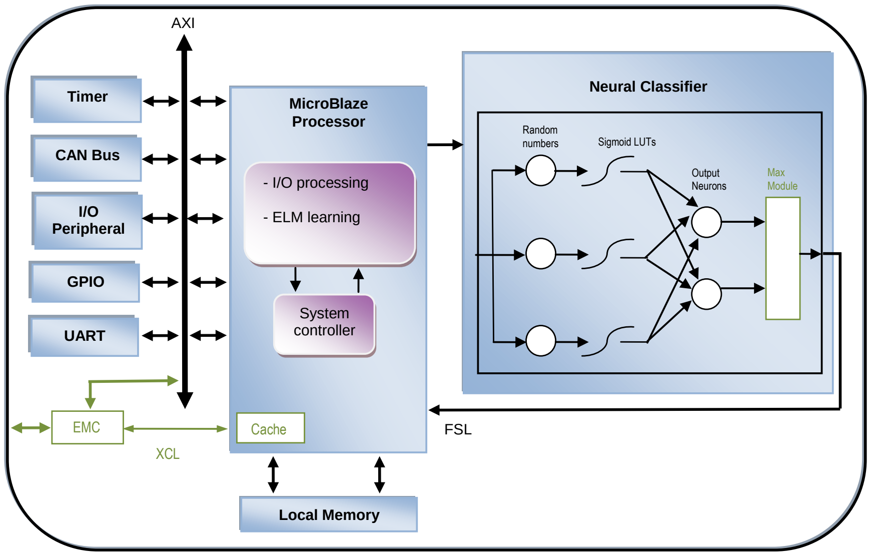


Figure 4.10: Block scheme of the HW/SW architecture for embedded ELM. The synthesis of the resources in green is optional. The data cache and the External Memory Controller (EMC) are included only if external memory is used, while the maximum module is used to identify a single class when the individual classification rates are not required

shaped function) to avoid the effects of using a short sequence of samples; an overlapping of 60 samples was used. Then, the computation of four statistical variables of the windows was made: the standard deviation, the mean value, the sum of absolute values, and the maximum absolute value.

4.2.3 Hardware/Software Architecture

Figure 4.10 depicts a block diagram of the proposed architecture for FPGA-based embedded ELM implemented in a Kintex 7 XC7K325T-2 device. Compared with Figure 4.7, this architecture does not implement any DPR related device and a CAN-bus module is included. In addition, an External Memory Controller (EMC) to control a SDRAM, which is required for medium/large size applications, and a timer for timing measures have been included.

The SW partition is in charge of the system control, the management of peripherals and coprocessors, and the computation of ELM learning algorithm. In particular, in this application an ELM classifier is used to identify drivers. It has been developed using C programming language. The code has been written to be independent of the platform and the operating system as much as possible. The only hardware-dependable functions are those needed to communicate with the HW partition, and the I/O send/receive functions.

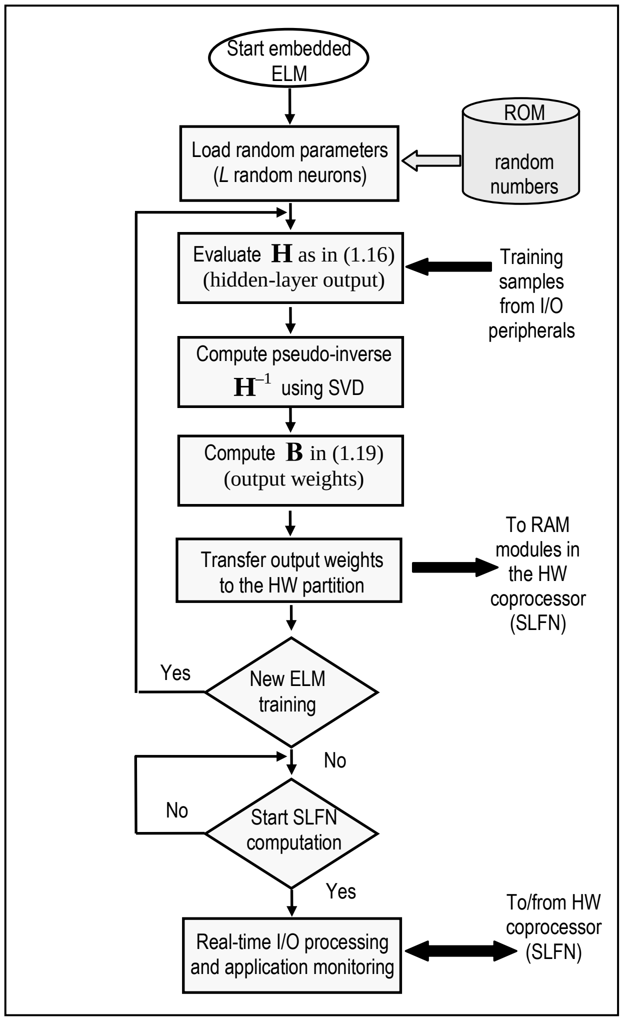

Start embedded ELM
Load random parameters (L random neurons)
ROM
random numbers
Evaluate H as in (1.16) (hidden-layer output)
Training samples from I/O peripherals
Compute pseudo-inverse H^-1 using SVD
Compute B in (1.19) (output weights)
Transfer output weights to the HW partition
To RAM modules in the HW coprocessor (SLFN)
New ELM training
Yes
No
Start SLFN computation
No
Yes
Real-time I/O processing and application monitoring
To/from HW coprocessor (SLFN)

Figure 4.11: Flow chart of the main tasks performed by the microprocessor

The basic real-time sequence of tasks, viewed from the SW partition, is depicted in Figure 4.11. For simplicity, secondary tasks which depend on particular applications have been omitted. Two main operation modes can be distinguished: ELM learning, and computation of the neural network (i.e. SLFN). The first mode involves the learning process (Equations 1.15 and 1.19), and the transfer of the weights of the output layer to RAM modules in the HW partition. Although different methods can be applied to solve Equation 1.19, in this work, Singular Value Decomposition (SVD) will be used.

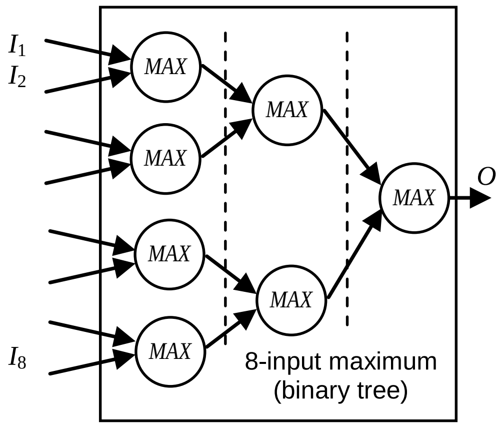


Figure 4.12: Scheme of an 8-input maximum circuit implemented by means of two-input single-cycle maximum modules. The maximum is structured into a binary tree of 3 layers. In the general case, an m-input multiplier consists of log2m layers

Once the learning stage is completed, the SLFN computation is ready to be activated. In this second operation mode, the microprocessor reads data from the peripheral inputs, sends them to the HW coprocessor using the FSL bus, and receives the SLFN output for further management according to the application specifications. The latency of a SLFN with n inputs, L hidden neurons, one activation function ROM per neuron and m outputs is

$$l_{clk} = (n+1) + (L+1) + m + 1 = n + m + L + 3 \tag{4.1}$$

where l_{clk} is the number of clock cycles required to perform the computation of the network, without the maximum module.

The maximum module is implemented as a tree of binary maximum circuits. The module is structured into a binary tree of log2m layers; if the number of classes or outputs (m) is not a power of 2, then the next power of 2 is used to evaluate the required layers. Figure 4.12 depicts a typical tree-like structure that implements the maximum operation. Each layer will add an additional clock cycle to Equation 4.1. In addition, the HW/SW communication interface (i.e. FSL busses) introduces an extra delay of two clock cycles.

Finally, it is worth noting that the learning stage could be activated at any time with the aim of adapting the system in real-time (i.e. on-line training).

4.2.4 Simulation and Experimental Results

The performance of ELM was firstly evaluated using Matlab programming tools, and the source code provided in ELM Web Portal [289].

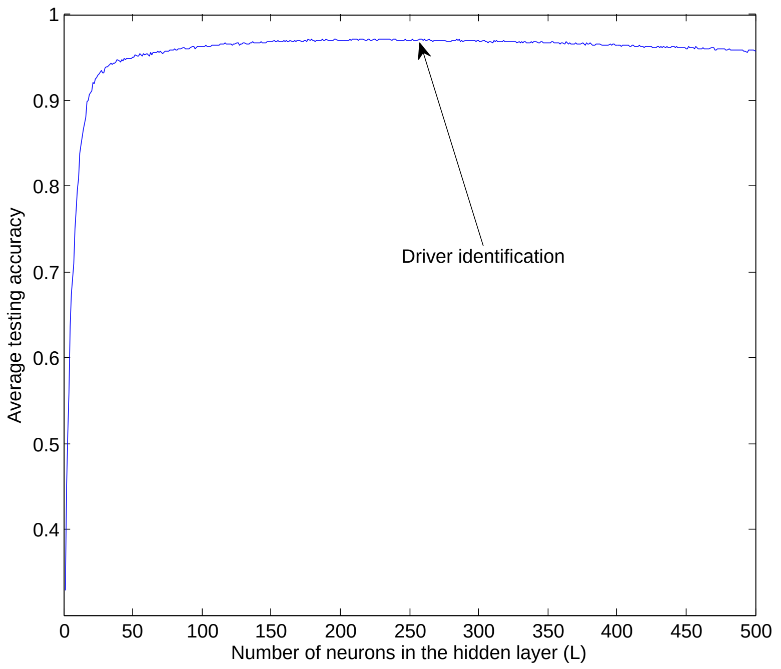


Figure 4.13: Average testing accuracy as a function of the number of neurons in the hidden layer (Equation 1.13) of an SLFN ELM core for AmI-driver

In this experiment, we considered groups of three drivers, and a dataset composed of 800 patterns (i.e. windows) with 16 attributes (i.e. 4 statistical variables of the 4 IMU measurements) plus the driver label. The procedure followed with the dataset to evaluate the performance of ELM was similar to that applied in the previous experiment. In addition, several experiments using SVM with Gaussian kernel were performed in order to provide comparative results. After a comprehensive exploration of SVM parameters, classification rates close to 90% were obtained. The evolution of the classification accuracy with the number of neurons of the hidden layer (L) using ELM is depicted in Figure 4.13. It can be seen that 20 random neurons are enough to obtain recognition rates that outperform the 90% obtained with SVM (e.g. $L = 25$, $ACC = 0.929$, $DEV = 0.011$; $L = 50$, $ACC = 0.951$, $DEV = 0.007$; $L = 100$, $ACC = 0.963$, $DEV = 0.005$; $L = 500$, $ACC = 0.958$, $DEV = 0.008$), where ACC stands for Accuracy and DEV stands for Standard Deviation. Moreover, more than 150 neurons in the hidden layer do not improve the performance of the system. On the contrary, the driver identification success rate decreases due to over-fitting effects.

The experimental results provided by the FPGA-based embedded system (see Table 4.4) agree with those obtained with Matlab.

Neurons in the hidden layer (L)	Training time (s) (MB at 100MHz)	ANN Computation time (μs) at 100 MHz	Mean recognition rate (%)
10	1.22	0.33	68.7
25	4.16	0.48	93.0
50	12.10	0.73	95.3
100	39.44	1.20	96.8

Table 4.4: Performance of the ELM algorithm

4.2.5 Timing Considerations and Resource Utilization

The architecture of the embedded system depicted in Figure 4.10 was sized to solve the driver identification application. The system was implemented using the XC7K325T Kintex7 device and an external memory was required. The MicroBlaze processor was configured with a 4 KB data cache to improve the processing speed of the learning algorithm. The code segment and the uninitialized data segment were stored in internal memory, while the initialized data section had to be stored in external SDRAM memory.

The timing performance of the HW partition is summarized in Table 4.5. The coprocessor is able to compute the SLFN operating at frequencies over 200 MHz for the neural networks with 10, 25, 50 and 100 hidden neurons. However, the largest network ($L = 500$) reports a maximum frequency of 119 MHz.

Table 4.4 summarizes the timing performance obtained with different sizes of ELM embedded systems for real-time driver identification. As in the previous experiment, the microprocessor and the coprocessor were configured to operate at 100 MHz. The training stage is performed extremely fast, in the worst case (i.e. $L = 100$) the training algorithm is computed in less than 40 seconds, and the SLFN core is evaluated in *1.2* μs. This performance allows for on-line training of the neural network classifier. The main advantage of this result is that the proposed HW/SW architecture can be used to develop an autonomous System on Programmable Chip (SoPC) suitable for in-vehicle driver identification. Real-time driver identification could be used to develop advanced driver assistance systems, and to improve both security and comfort in smart cars [33].

Table 4.6 summarizes the resources required to synthesize the 16-input 3-class SLFN for the driver identification system. The greater percentage of resource usage corresponds to the DSP embedded cores. Moreover, although the number of logic elements used to implement the RAM and ROM memories is high, the system can be implemented in a low range FPGA. Even in the worst case scenario (see Table 4.6), the coprocessor only needs 59.9% DPSs, 7.4% registers, and 41.1 % LUTs.

In [33] a driver classification system is performed with a Multilayer Perceptron Neural Network with two hidden layers and training the system using BP. If its results are compared with Table 4.4, even with only 25 neurons in the

Neurons in the hidden layer (L)	Achieved frequency (MHz)	Computation time (μs)
10	221	0.15
25	220	0.22
50	219	0.33
100	213	0.58
500	119	4.39

Table 4.5: Timing performance of the SLFN coprocessor. Device: Kintex 7 XC7K325T-2

Neurons in the hidden layer (L)	Registers	LUTs	DSPs
10	932 (<1%)	1191 (<1%)	13 (1.5%)
25	1914 (<1%)	2102 (1%)	18 (3.3%)
50	3545 (<1%)	3948 (1.9%)	53 (6.3%)
100	6039 (1.5%)	7053 (3.5%)	103 (12.3%)
500	30027 (7.4%)	83719 (41.1%)	503 (59.9%)

Table 4.6: Hardware resources of the ANN coprocessor. Device: Kintex 7 XC7K325T-2

hidden layer, ELM outperforms the MLP classifier. In terms of HW resources and computation time ELM is also a better solution than a MLP based ANN.

4.3 Conclusions

In this chapter two real-world Artificial Neural Network-based applications for AmI environments have been shown.

First a multilevel adaptive Artificial Neural Network scheme for the development of intelligent agents is proposed. The scheme has been successfully implemented in a single-chip FPGA with DPR capability. Software learning algorithms are applied to adapt the agent behaviour (i.e. neural network parameters) at the system level, while DPR is used to modify the agent at the physical and architectural level (i.e. Neural Network topology).

The adaptive agent is able to behave in an optimum way for a given behaviour performance (i.e. maximum allowed error). This is achieved due to the BP learning algorithm and a growing/pruning algorithm applied to the system architecture. Those algorithms also allow the agent to adapt to the changes of the behaviour of the environment user, not only adapting the parameters of the net but also adapting the architecture of the net itself if it is needed. To perform the modelling, the system needs a few seconds to compute 100 learning cycles, which is why the learning/adaptation should be done in downtime periods.

Another important point is the use of DPR. Using this technology only the needed neurons of the hidden layer are implemented in the hardware partition of the system, so, a decrease in power consumption is also achieved. In the case of the time needed to reconfigure hidden layer neurons in the worst case scenario, the reconfiguration of all 32 neurons lasts 83.6 ms. This is sufficient for iDorm where the operation time -defined by the user-machine interaction- is a matter of seconds.

The second application proposed is a driver identifier based on the use of Extreme Leaning Machines and statistical values of an Inertial Measurement Unit implemented in FPGAs. ELM improves traditional Backpropagation Gradient Descent learning algorithm, and is free of local minimum. In addition, its performance depends on a single design parameter, the size of the hidden layer. Therefore, autonomy and real-time adaptation are easier to achieve with ELM than using other well-known machine learning techniques. This work contributes to the development of ADAS with a driver-centred perspective which aims at improving the driver's awareness and driving performance in a personalized way.

In future works we are going to improve the identification performance of the ELM classifier by adding new driving behavioural signals. In addition, the performance of the FPGA-based system will be improved in order to enable on-line training. This new capability of the system would allow the adaptation of the reference driving style models in the long term. Also the capabilities of the embedded system to perform both ELM classification and ELM regression application will be enhanced. The main computational effort to implement ELM is the SVD algorithm used to obtain the generalized inverse matrix. In this work, SVD is developed in the software partition and, as a consequence, a large amount of internal memory is required. With the aim of reducing the computational effort of the MicroBlaze processor, and accelerating even more the learning stage of ELM, an SVD hardware coprocessor will be developed. In addition, the use of feature extraction techniques, based on Principal Component Analysis (PCA), will be investigated in order to reduce the dimensionality of the training data.

Chapter 5

Final Conclusions and Future Work

5.1 Conclusions

In this project a solution to the problem of the design of embedded systems for Ambient Intelligence environments has been proposed. Those systems must be small-size, low-cost and low-power electronic devices with also high processing speed to execute the algorithms needed to provide intelligence to the environment. In the present thesis a system implementing a hybrid hardware/software (HW/SW) architecture is presented to achieve these specifications. In the following paragraphs the main conclusions and the major contributions of this thesis are presented.

To tackle the above objective, a HW/SW Artificial Neural Network (ANN), based on a Field Programmable Gate Array for Ambient Intelligence environments has been developed. Both hardware and software partitions have been implemented in the same device, as a result no other external devices are needed. Hence, the implementation is done on a System on Programmable Chip.

The Artificial Neural Network core (HW partition) has been designed using standard VHDL language, therefore, the core is independent of the device in which is implemented. Another advantage is its scalability: that is to say, the code is designed to adapt easily to changes in the number of inputs, hidden layer neurons, outputs or word-length. The main advantage of the HW ANN core is the high degree of parallelism, which increases its computational performance.

Different types of learning algorithms have been implemented (SW partition), thus, the system is able to learn during the off-line stage and adapt to changes during the on-line stage. On the one hand, parameter learning algorithms (i.e. Backpropagation and Extreme Learning Machines) have been implemented. On the other hand, a structure learning algorithm has been also designed (i.e. growing/pruning), to select the best hidden layer size. All these algorithms have been implemented using standard C programming language independent of the platform as much as possible. The SW implementation of the Backpropagation (BP) algorithm cannot be as fast it is desired depending on the network architecture and the number of learning cycles. Hence, to speed up the algorithm, the ANN core has been modified so it can be used also as hardware accelerator of BP algorithm, achieving accelerations up to a *61.83%*.

In some cases, the accuracy required by the learning algorithm is very high and, in consequence, the memories to store the activation functions need to also be very large. To overcome this problem, a new method to obtain the activation function has been proposed based on Taylor's theorem and the Lagrange form of the remainder. This method allows obtaining activation functions with high accuracies using fewer resources than the memory-based methods.

The use of Dynamic Partial Reconfiguration technology has been studied with the objective of achieving a reduction of size and/or power consumption. Concerning size reduction, DPR allows implementing only the modules needed in each moment, and thus, avoiding the implementation of all the cores or internal modules required by the system. As a result, smaller devices can be selected for the system implementation. On the other hand, regarding power reduction, the results show that DPR provides small benefit, especially in last generation FPGAs (i.e. Xilinx's Family 7 devices) available in the market. Instead, the reduction of power in these last devices can be obtained by means of new design techniques and resources (e.g. the use of clock enable signals).

To demonstrate the suitability of the system for AmI environments two real-world applications have been developed. The first application is an adaptive agent for real-time control of an inhabited Intelligent Environment. This agent is capable of modelling the activity of the user of the environment not only selecting the best parameters of the ANN using BP algorithm but also selecting the best size of the hidden layer, using a growing/pruning algorithm. It is also able to dynamically adapt to changing situations if needed during the on-line stage. The second application developed is the implementation of a driver identifier based on the use of Extreme Leaning Machines. This system contributes to the development of a driver-centred Advanced Driver Assistance System, which is becoming increasingly important in the automotive industry, as a way to decrease the number of accidents caused by fatigue, stress, or distractions. The system is designed to use the signals provided by the car, therefore, no extra equipment has to be included in the vehicles, in contrast with other proposed solutions that require the use of cameras or techniques intrusive to the driver.

Finally, during the realization of these thesis, contacts have been established with different entities both national, the Automotive Intelligence Center, and international, the University of Coventry in the United Kingdom and the Drive-Safe Consortium in Istanbul, Turkey.

5.2 Future Work

This project has opened new research topics that will be developed in the future. Firstly the topics related to Ambient Intelligence are mentioned followed by the ones related to the FPGA technology:

1. Emotion sensing and affective computing is gaining more importance in Ambient Intelligence [2]. Further research in this area should be done to study its implementation in both inhabited environments and AmI-cars.

2. The powerful capabilities of Extreme Learning Machines for multiclass classification problems have been verified; in future research its regression properties in AmI environments will be also investigated.

3. The project has focused on Multilayer Perceptron Neural Networks. It should be studied if the ANN core can be redesigned to implement other ANNs such as recurrent neural networks.

4. A large part of the learning algorithms is still performed in SW. In the future, the impact in term of processing speed of performing the learning process in a HW coprocessor will be studied.

5. Up to now only DPR has been taken into consideration to reduce the power consumption. In future work, the use of the new clock enable signals of the last generation FPGAs will be studied as a power reduction technique.

6. During the last part of this project, Programmable Logic Devices including ARM processing units (e.g. Xilinx's Zynq-7000 and Altera's SoC family) have been gaining popularity. Hence, a migration of the communication bus from Fast Simplex Link (Xilinx's Intellectual Property) to AXI-Stream (part of the ARM's AMBA open-standard) will give the opportunity of using the ANN core in those devices.

7. Finally, in recent years new generations of FPGAs have appeared in the market. It is important to keep updated about the new capabilities of these FPGAs, such as the clock enable signals, and the new designing tools developed by the manufacturers and their new features.

5.3 Publications

Journals

del Campo, I.; Finker, R.; Echanobe, J.; Basterretxea, K., "Controlled accuracy approximation of sigmoid function for efficient FPGA-based implementation of artificial neurons," Electronics Letters, vol.49, no.25, pp.1598,1600, December 5 2013

Congresses

Echanobe, J.; del Campo, I.; Finker, R.; Basterretxea, K., "Dynamic Partial Reconfiguration in Embedded Systems for Intelligent

Environments," 2012 8th International Conference on Intelligent Environments (IE), vol., no., pp.109,113, 26-29, Guanajuato, Mexico, June 2012

del Campo, I.; Echanobe, J.; Finker, R.; Doctor, F.; Basterretxea, K.; Martinez, M. V.; Tarela, J., “Using Ambient-Intelligence Techniques for Reducing Traffic Accidents Caused By Driver Error”, Congreso internacional de Seguridad Vial, Santander, Spain. pp. 67 , 2013.

Finker, R.; del Campo, I.; Echanobe, J.; Doctor, F., "Multilevel adaptive neural network architecture for implementing single-chip intelligent agents on FPGAs," The 2013 International Joint Conference on Neural Networks (IJCNN), vol., no., pp.1,9, 4-9, Dallas, TX, USA, Aug. 2013

del Campo, I.; Finker, R.; Martinez, M.V.; Echanobe, J.; Doctor, F., "A real-time driver identification system based on artificial neural networks and cepstral analysis," 2014 International Joint Conference on Neural Networks (IJCNN), vol., no., pp.1848,1855, 6-11, Beijing, People’s Republic of China, July 2014

Martinez-Corral, U.; Basterretxea, K.; Finker, R., "Scalable parallel architecture for singular value decomposition of large matrices," 2014 24th International Conference on Field Programmable Logic and Applications (FPL), vol., no., pp.1,4, 2-4, Munich, Germany, Sept. 2014

Finker, R.; del Campo, I.; Echanobe, J.; Martinez, V., "An intelligent embedded system for real-time adaptive extreme learning machine," 2014 IEEE Symposium on Intelligent Embedded Systems (IES), vol., no., pp.61,69, 9-12, Orlando, FL, USA, Dec. 2014

Echanobe, J.; Finker, R.; del Campo, I., “A Divide-and-Conquer Strategy for FPGA Implementations of Large Neural Network-based Classifiers” The 2015 International Joint Conference on Neural Networks (IJCNN), Killarney, Ireland, Jul. 2015, Accepted

Bibliography

[1] J. C. Augusto, "Ambient intelligence: the confluence of ubiquitous/pervasive computing and artificial intelligence," in *Intelligent Computing Everywhere* (A. J. Schuster, ed.), pp. 213–234, London: Springer, 2007. http://link.springer.com/chapter/10.1007%2F978-1-84628-943-9_11.

[2] F. Doctor, R. Iqbal, and V. Zamudio, "Introduction to the thematic issue on affect aware ubiquitious computing," *Journal of Ambient Intelligence and Smart Environments*, vol. 7, pp. 3–4, January 2015. http://content.iospress.com/articles/journal-of-ambient-intelligence-and-smart-environments/ais295.

[3] G. M. Youngblood, D. J. Cook, and L. B. Holder, "Managing adaptive versatile environments," *Pervasive and Mobile Computing*, vol. 1, no. 4, pp. 373 – 403, 2005. http://www.sciencedirect.com/science/article/pii/S1574119205000465.

[4] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen, "The gator tech smart house: a programmable pervasive space," *Computer*, vol. 38, pp. 50–60, March 2005. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1413118&tag=1.

[5] University of Essex, *iSpace webpage*, Accesed 2014. http://cswww.essex.ac.uk/iieg/idorm2/index.htm.

[6] M. E. Pollack, "Intelligent technology for an aging population: The use of ai to assist elders with cognitive impairment," *AI Magazine*, pp. 9–24, 2005. http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/1810.

[7] M. Saito, "Expanding welfare concept and assistive technology," in *Proceedings of the IEEK Annual Fall Conference*, pp. 156–161, 2000. http://web.cecs.pdx.edu/~mperkows/Rehabilitation_Robots/Saito-paper.pdf.

[8] M. D. Rodríguez, J. Favela, A. Preciado, and A. Vizcaíno, “Agent-based ambient intelligence for healthcare,” *AI Commun.*, vol. 18, pp. 201–216, Aug. 2005. http://dl.acm.org/citation.cfm?id=1218875.1218880.

[9] New Energy and Industrial Technology Development Organization, *NEDO Webpage*, Accessed 2014. http://www.nedo.go.jp/english/index.html.

[10] Daimler, *Attention Assist*, Accessed 2014. Mercedes Attention Assist Web.

[11] Ford Motor Company, *Driver Alert*, Accessed 2014. http://technology.fordmedia.eu/documents/newsletter/FordTechnologyNewsletter082010.pdf.

[12] VolksWagen, *Fatigue Detection*, Accessed 2014. http://www.volkswagen.com.au/en/technology_and_service/technical-glossary/fatigue-detection.html.

[13] Volvo Cars, *Volvo Driver Alert Control*, Accessed 2014. http://www.volvocars.com/uk/top/my_volvo/videos/pages/volvo-driveralertcontrol.aspx.

[14] Ford Motor Company, *Ford Develops Heart Rate Monitoring Seat; Adds New Element to Health and Wellness Research*, Accessed 2014. http://corporate.ford.com/news-center/press-releases-detail/pr-ford-develops-heart-rate-34664.

[15] The Independent, *Ford unveils a car seat which detects when a driver is having heart attack*, Accessed 2014. http://www.independent.co.uk.

[16] L. A. Zadeh, “Fuzzy logic, neural networks, and soft computing,” *Commun. ACM*, vol. 37, pp. 77–84, Mar. 1994. http://www.cs.berkeley.edu/~zadeh/papers/Fuzzy%20Logic,%20Neural%20Networks,%20and%20Soft%20Computing-1994.pdf.

[17] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bulletin of mathematical biology*, vol. 52, no. 1, pp. 99–115, 1990. http://link.springer.com/article/10.1007%2FBF02478259.

[18] D. B. Parker, “Learning logic,” invention report. s81-64, File Office of Technology Licensing, Stanford University, Oct. 1982.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Parallel distributed processing: explorations in the microstructure of cognition, vol. 1,” ch. Learning internal representations by error propagation, pp. 318–362, Cambridge, MA, USA: MIT Press, 1986. http://mitpress.mit.edu/books/parallel-distributed-processing.

[20] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489 – 501, 2006. http://www.sciencedirect.com/science/article/pii/S0925231206000385.

[21] M. Bortman and M. Aladjem, "A Growing and Pruning Method for Radial Basis Function Networks," *IEEE Transactions on Neural Networks*, vol. 20, pp. 1039–1045, June 2009. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4926121.

[22] Altera, *Nios II Processor Reference Handbook*, ver 11.0 ed., May 2011. www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf.

[23] Xilinx, *MicroBlaze Processor Reference Guide*, v8.50b ed., June 2013. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_6/mb_ref_guide.pdf.

[24] I. del Campo, R. Finker, J. Echanobe, and K. Basterretxea, "Controlled accuracy approximation of sigmoid function for efficient FPGA-based implementation of artificial neurons," *Electronics Letters*, vol. 49, pp. 1598–1600, December 2013. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6678448.

[25] Xilinx, *PlanAhead Design and Analysis Tool*, Accessed 2014. http://www.xilinx.com/tools/planahead.htm.

[26] R. Finker, I. del Campo, J. Echanobe, and F. Doctor, "Multilevel adaptive neural network architecture for implementing single-chip intelligent agents on FPGAs," in *International Joint Conference on Neural Networks (IJCNN)*, (Dallas, TX, USA), pp. 1–9, Aug 2013. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6706760.

[27] J. Echanobe, R. Finker, and I. del Campo, "A Divide-and-Conquer Strategy for FPGA Implementations of Large Neural Network-based Classifiers," in *International Joint Conference on Neural Networks (IJCNN)*, (Killarney, Ireland), July 2015. Accepted.

[28] F. Doctor, H. Hagras, and V. Callaghan, "A fuzzy embedded agent-based approach for realizing ambient intelligence in intelligent inhabited environments," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 35, pp. 55 – 65, jan. 2005. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1369345&tag=1.

[29] F. Doctor, H. Hagras, and V. Callaghan, "A type-2 fuzzy embedded agent to realise ambient intelligence in ubiquitous computing environments," *Information Sciences*, vol. 171, no. 4, pp. 309 – 334, 2005. http://www.sciencedirect.com/science/article/pii/S0020025504003123.

[30] H. Hagras, F. Doctor, V. Callaghan, and A. Lopez, "An incremental adaptive life long learning approach for type-2 fuzzy embedded agents in ambient intelligent environments," *IEEE Transactions on Fuzzy Systems*, vol. 15, pp. 41–55, Feb 2007. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4088986.

[31] C. Miyajima, T. Kusakawa, T. Nishino, N. Kitaoka, K. Itou, and K. Takeda, "On-going data collection of driving behavior signals," in *In-Vehicle Corpus and Signal Processing for Driver Behavior* (K. Takeda, H. Erdogan, J. Hansen, and H. Abut, eds.), pp. 45–54, Springer US, 2009. http://dx.doi.org/10.1007/978-0-387-79582-9_4.

[32] P. Angkititrakul, J. Hansen, S. Choi, T. Creek, J. Hayes, J. Kim, D. Kwak, L. Noecker, and A. Phan, "Utdrive: The smart vehicle project," in *In-Vehicle Corpus and Signal Processing for Driver Behavior* (K. Takeda, H. Erdogan, J. Hansen, and H. Abut, eds.), pp. 55–67, Springer US, 2009. http://dx.doi.org/10.1007/978-0-387-79582-9_5.

[33] I. del Campo, R. Finker, M. Martinez, J. Echanobe, and F. Doctor, "A real-time driver identification system based on artificial neural networks and cepstral analysis," in *International Joint Conference on Neural Networks (IJCNN)*, (Beijing, People's Republic of China), pp. 1848–1855, July 2014. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6889772.

[34] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.-C. Burgelman, "Scenarios for Ambient Intelligence in 2010, Final Report," tech. rep., European Commission IST Advisory Group, Feb 2001. ftp://ftp.cordis.lu/pub/ist/docs/istagscenarios2010.pdf.

[35] "Ambient Intelligence: From Vision to Reality," tech. rep., European Commission IST Advisory Group, 2003. ftp://ftp.cordis.europa.eu/pub/ist/docs/istag-ist2003_consolidated_report.pdf.

[36] F. Sadri, "Ambient Intelligence: A Survey," *ACM Computing Surveys (CSUR)*, vol. 43, p. 36, October 2011. http://doi.acm.org/10.1145/1978802.1978815.

[37] M. Friedewald, O. Da Costa, *et al.*, "Science and technology roadmapping: Ambient intelligence in everyday life (ami@ life)," *Karlsruhe: Fraunhofer-Institut für System-und Innovationsforschung (FhG-ISI)*, June 2003. http://foresight.jrc.ec.europa.eu/documents/SandT_roadmapping.pdf.

[38] G. Acampora and V. Loia, "A proposal of ubiquitous fuzzy computing for ambient intelligence," *Information Sciences*, vol. 178, no. 3, pp. 631 – 646, 2008. http://www.sciencedirect.com/science/article/pii/S002002550700401X.

[39] S. D. Glaser and A. Tolman, "Sense of sensing: from data to informed decisions for the built environment," *Journal of infrastructure systems*, vol. 14, no. 1, pp. 4–14, 2008. http://ascelibrary.org/doi/abs/10.1061/%28ASCE%291076-0342%282008%2914%3A1%284%29.

[40] L. Benini, E. Farella, and C. Guiducci, "Wireless sensor networks: Enabling technology for ambient intelligence," *Microelectronics Journal*, vol. 37, no. 12, pp. 1639 – 1649, 2006. http://www.sciencedirect.com/science/article/pii/S0026269206001728.

[41] J. Augusto and P. McCullaugh, "Ambient Intelligence: Concepts and Applications," *Computer Science and Information Systems (ComSis)*, vol. 4, pp. 1–28, 2007. http://www.doiserbia.nb.rs/img/doi/1820-0214/2007/1820-02140701001A.pdf.

[42] Witura. http://www.witura.com/wifi-smart-home-management-system.html.

[43] Maxin Integrated, *iButtons*. http://www.maximintegrated.com/en/products/comms/ibutton.html.

[44] ZigBee Alliance, *ZigBee specification*, 2006. http://www.zigbee.org/Specifications.aspx.

[45] *Bluetooth Special Interest Group webpage*, 2014. https://www.bluetooth.org/.

[46] Wi-Fi Alliance. http://www.wi-fi.org/.

[47] Greenvity Communications, *Home Gateways*, 2014. http://greenvity.com/home-gateways.htm Accesed 2014.

[48] I. del Campo, K. Basterretxea, J. Echanobe, G. Bosque, and F. Doctor, "A system-on-chip development of a neuro-fuzzy embedded agent for ambient-intelligence environments," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. PP, no. 99, pp. 1 –12, 2011. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6046142.

[49] N. Sridevi and P. Subashini, "Combining Zernike moments with Regional features for classification of handwritten ancient Tamil scripts using Extreme Learning Machine," in *International Conference on Emerging Trends in Computing, Communication and Nanotechnology (ICE-CCN)*, (Amsterdam, Netherlands), pp. 158–162, March 2013. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=652848.

[50] F. Amigoni, N. Gatti, C. Pinciroli, and M. Roveri, "What planner for ambient intelligence applications?," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 35, pp. 7–21, Jan 2005. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1369341.

[51] V. Zamudio and V. Callaghan, "Facilitating the ambient intelligent vision: A theorem, representation and solution for instability in rule-based multi-agent systems," *International Transactions on Systems Science and Applications*, vol. 4, no. 1, 2008. http://cswww.essex.ac.uk/staff/vic/papers/2008_TSSA08%28FacilitatingTheAmbient%29.pdf.

[52] V. Zamudio and V. Callaghan, "Unwanted periodic behaviour in pervasive computing environments," in *ACS/IEEE International Conference on Pervasive Services*, (Lyon, France), pp. 273–276, June 2006. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1652240.

[53] E. Amaro, J. Lopez, V. Zamudio, R. Baltazar, M. Casillas, and V. Callaghan, "Innovative locking in ami: Efficiently removing instabilities in multi-agent systems," in *Intelligent Environments (IE), 2011 7th International Conference on*, (Nottingham, United Kingdom), pp. 135–141, July 2011. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6063377.

[54] Fagor S. Coop., *Maior-Vocce, Interfaz de Voz para Maior-Domo®*, 2008. http://www2.fagor.com/domotica/pub/cast/catalogo/maior_vocce.pdf.

[55] Fagor S. Coop., *Maior-Domo® PRO*, Accessed 2014. www2.fagor.com/domotica/_bin/cast/maiordomo.php.

[56] S. Ge, Y. Yang, and T. Lee, "Hand gesture recognition and tracking based on distributed locally linear embedding," *Image and Vision Computing*, vol. 26, no. 12, pp. 1607 – 1620, 2008. http://www.sciencedirect.com/science/article/pii/S0262885608000693.

[57] M. Pantic, "Face for ambient interface," in *Ambient intelligence in everyday life* (J. Abascal and Y. Cai, eds.), vol. 3864 of *Laecture Notes on computer Science*, pp. 32–66, Springer-Verlag Berlin Heidelberg: Springer, 2006 ed., December 2006. http://www.springer.com/computer/ai/book/978-3-540-37785-6.

[58] E. Leon, G. Clarke, V. Callaghan, and F. Sepulveda, "A user-independent real-time emotion recognition system for software agents in domestic environments," *Engineering Applications of Artificial Intelligence*, vol. 20, no. 3, pp. 337 – 345, 2007. http://www.sciencedirect.com/science/article/pii/S0952197606001011.

[59] A. R. Aguiñaga, M. L. Ramírez, A. A. Garza, R. Baltazar, and V. Zamudio, "Emotion analysis through physiological measurements.," in *Intelligent Environments (Workshops)* (J. A. Botía and D. Charitos, eds.), vol. 17 of *Ambient Intelligence and Smart Environments*, pp. 97–106, IOS Press, 2013. http://ebooks.iospress.nl/publication/33851.

[60] U. Esnaola and T. Smithers, “Whistling to machines,” in *Ambient Intelligence in Everyday Life* (Y. Cai and J. Abascal, eds.), vol. 3864 of *Lecture Notes in Computer Science*, pp. 198–226, Springer Berlin Heidelberg, 2006. `http://dx.doi.org/10.1007/11825890_10`.

[61] D. J. Cook, J. C. Augusto, and V. R. Jakkula, “Ambient intelligence: Technologies, applications, and opportunities,” *Pervasive and Mobile Computing*, vol. 5, no. 4, pp. 277 – 298, 2009. `"http://www.sciencedirect.com/science/article/pii/S157411920900025X"`.

[62] Y. Chen, G. Liang, K. K. Lee, and Y. Xu, “Abnormal Behavior Detection by Multi-SVM-Based Bayesian Network,” in *International Conference on Information Acquisition, 2007. ICIA '07*, (Seogwipo, South Korea), pp. 298–303, July 2007. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4295746`.

[63] O. Brdiczka, P. Reignier, and J. Crowley, “Detecting individual activities from video in a smart home,” in *Knowledge-Based Intelligent Information and Engineering Systems* (B. Apolloni, R. Howlett, and L. Jain, eds.), vol. 4692 of *Lecture Notes in Computer Science*, pp. 363–370, Springer Berlin Heidelberg, 2007. `http://link.springer.com/content/pdf/10.1007%2F978-3-540-74819-9_45.pdf`.

[64] E. M. Tapia, S. S. Intille, and K. Larson, “Activity recognition in the home using simple and ubiquitous sensors,” in *Pervasive Computing* (A. Ferscha and F. Mattern, eds.), vol. 3001, pp. 158–175, Cambridge Center, 4FL, Cambridge, MA 02142 USA: Springer, 2004. `https://stuff.mit.edu/afs/athena/dept/cron/group/house_n/documents/Tapia03.pdf`.

[65] D. Cook and M. Schmitter-Edgecombe, “Assessing the quality of activities in a smart environment,” *Methods of information in medicine*, vol. 48, p. 480, May 2009. `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2759863/`.

[66] L. Liao, D. Fox, and H. Kautz, “Location-based activity recognition using relational markov networks,” in *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, (San Francisco, CA, USA), pp. 773–778, Morgan Kaufmann Publishers Inc., 2005. `http://ijcai.org/papers/1572.pdf`.

[67] U. Maurer, A. Smailagic, D. Siewiorek, and M. Deisher, “Activity recognition and monitoring using multiple sensors on different body positions,” in *International Workshop on Wearable and Implantable Body Sensor Networks*, (Cambridge, MA, USA), pp. 4 pp.–116, April 2006. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1612909&tag=1`.

[68] N. Oliver and E. Horvitz, “A comparison of hmms and dynamic bayesian networks for recognizing office activities,” in *User Modeling 2005* (L. Ardissono, P. Brna, and A. Mitrovic, eds.), vol. 3538 of *Lecture Notes*

in Computer Science, pp. 199–209, Springer Berlin Heidelberg, 2005. http://link.springer.com/content/pdf/10.1007%2F11527886.pdf.

[69] X. Wang and Q. Ji, "Learning dynamic bayesian network discriminatively for human activity recognition," in *21st International Conference on Pattern Recognition (ICPR), 2012*, (Tsukuba, Japan), pp. 3553–3556, Nov 2012. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=6460932.

[70] E. Nazerfard, B. Das, L. B. Holder, and D. J. Cook, "Conditional random fields for activity recognition in smart environments," in *Proceedings of the 1st ACM International Health Informatics Symposium*, IHI '10, (New York, NY, USA), pp. 282–286, ACM, 2010. http://dl.acm.org/citation.cfm?id=1883032.

[71] D. Cook and S. Das, *Smart environments: technology, protocols and applications*, vol. 43. Wiley-Interscience, 2004.

[72] B. Gottfried, H. Guesgen, and S. Hübner, "Spatiotemporal reasoning for smart homes," in *Designing Smart Homes* (J. Augusto and C. Nugent, eds.), vol. 4008 of *Lecture Notes in Computer Science*, pp. 16–34, Springer Berlin Heidelberg, 2006. http://dx.doi.org/10.1007/11788485_2.

[73] J. Allen and G. Ferguson, "Actions and events in interval temporal logic," in *Spatial and Temporal Reasoning* (O. Stock, ed.), pp. 205–245, Springer Netherlands, 1997. link.springer.com/chapter/10.1007/978-0-585-28322-7_7.

[74] B. Gottfried, "Reasoning about intervals in two dimensions," in *2004 IEEE International Conference on Systems, Man and Cybernetics*, vol. 6, (The Hague, The Netherlands), pp. 5324–5332 vol.6, Oct 2004. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1401040.

[75] M. Vilain, H. Kautz, and P. van Beek, "Readings in qualitative reasoning about physical systems," ch. Constraint Propagation Algorithms for Temporal Reasoning: A Revised Report, pp. 373–381, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990. http://www.sciencedirect.com/science/article/pii/B9781483214474500341.

[76] S. P. Rao and D. J. Cook, "Predicting inhabitant action using action and task models with application to smart homes," *International Journal on Artificial Intelligence Tools*, vol. 13, no. 01, pp. 81–99, 2004. http://www.worldscientific.com/doi/abs/10.1142/S0218213004001533.

[77] K. Basterretxea, I. del Campo, M. Martinez, and J. Echanobe, "Dynamic significant feature extraction for embedded intelligent agent implementations," in *2013 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE)*, (Singapore), pp. 39–46, April 2013. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6595770.

[78] A. Krause, D. Siewiorek, A. Smailagic, and J. Farringdon, "Unsupervised, dynamic identification of physiological and activity context in wearable computing," in *Seventh IEEE International Symposium on Wearable Computers, 2003.*, (White Plains, NY, USA), pp. 88–97, Oct 2003. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1241398.

[79] "Aladin," Accesed 2014. http://www.2020-horizon.com/ALADIN-Ambient-lighting-assistance-for-an-ageing-population%28ALADIN%29-s14805.html.

[80] University of Essex, *iDorm project*. http://cswww.essex.ac.uk/iieg/idorm.htm.

[81] J. Bauchet, H. Pigot, S. Giroux, D. Lussier-Desrochers, Y. Lachapelle, and M. Mokhtari, "Designing judicious interactions for cognitive assistance: The acts of assistance approach," in *Proceedings of the 11th International ACM SIGACCESS Conference on Computers and Accessibility*, Assets '09, (New York, NY, USA), pp. 11–18, ACM, 2009. http://doi.acm.org/10.1145/1639642.1639647.

[82] Siemens, Accesed 2014. http://www.buildingtechnologies.siemens.com/bt/global/en/buildingautomation-hvac/home-automation-system-synco-living/system/pages/system.aspx.

[83] Philips, *HomeLab Webpage*, Accesed 2014. http://www.research.philips.com/technologies/projects/ami/background.html.

[84] E. Horvitz, P. Koch, and J. Apacible, "Busybody: Creating and fielding personalized models of the cost of interruption," in *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, CSCW '04, (New York, NY, USA), pp. 507–510, ACM, 2004. http://doi.acm.org/10.1145/1031607.1031690.

[85] P. P. Vitaliano, D. Echeverria, J. Yi, P. E. Phillips, H. Young, and I. C. Siegler, "Psychophysiological mediators of caregiver stress and differential cognitive decline.," *Psychology and aging*, vol. 20, pp. 402–411, September 2005. http://www.ncbi.nlm.nih.gov/pubmed/16248700.

[86] A. Kofod-Petersen and A. Aamodt, "Contextualised ambient intelligence through case-based reasoning," in *Advances in Case-Based Reasoning* (T. Roth-Berghofer, M. Göker, and H. Güvenir, eds.), vol. 4106 of *Lecture Notes in Computer Science*, pp. 211–225, Springer Berlin Heidelberg, 2006. http://dx.doi.org/10.1007/11805816_17.

[87] J. Corchado, J. Bajo, and A. Abraham, "Gerami: Improving healthcare delivery in geriatric residences," *IEEE Intelligent Systems*, vol. 23, pp. 19–25, March 2008. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4475855&tag=1.

[88] MIT AgeLab, *AwareCar Webpace*, Accessed 2014. http://agelab.mit.edu/awarecar.

[89] J. Coughlin, B. Reimer, and B. Mehler, "Monitoring, managing, and motivating driver safety and well-being," *IEEE Pervasive Computing*, vol. 10, pp. 14–21, July 2011. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5958684.

[90] European Comission, *HARKEN Report Summary.* http://cordis.europa.eu/result/rcn/58023_en.html.

[91] National Heart, Lung and Blood Institute, *What Is a Heart Attack?*, Accessed 2014. http://www.nhlbi.nih.gov/health/health-topics/topics/heartattack/.

[92] Daily Mail, *The car seat that detects HEART ATTACKS: Ford plans to monitor drivers' pulses to prevent accidents*, Acessed 2014. http://www.dailymail.co.uk.

[93] A. Othman and M. Riadh, "Speech recognition using scaly neural networks," *World Academy of Science, Engineering and Technology*, vol. 38, pp. 253–258, 2008. http://www.waset.org/publications/7057.

[94] D. Franklin, "Cooperating with people: the intelligent classroom," in *Proceedings of Fifteenth National Conference on Artificial Intelligence*, (Madison, WI, USA), pp. 555–560, 1998. http://infolab.northwestern.edu/media/papers/paper10073.pdf.

[95] R. Ranchal, T. Taber-Doughty, Y. Guo, K. Bain, H. Martin, J. Robinson, and B. Duerstock, "Using Speech Recognition for Real-Time Captioning and Lecture Transcription in the Classroom," *IEEE Transactions on Learning Technologies*, vol. 6, pp. 299–311, Oct 2013. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6529071.

[96] D. Benavides, E. Fuentes, and V. Zamudio, "inet: An intelligent network as a personal assistant in an icampus environment.," in *Intelligent Environments (Workshops)* (J. C. Augusto, H. K. Aghajan, V. Callaghan, D. J. Cook, J. O'Donoghue, S. Egerton, M. Gardner, B. D. Johnson, Y. Kovalchuk, R. López-Cózar, P. Mikulecký, J. W. P. Ng, R. Poppe, M. Wang, and V. Zamudio, eds.), vol. 10 of *Ambient Intelligence and Smart Environments*, pp. 534–539, IOS Press, 2011. http://ebooks.iospress.nl/publication/28078.

[97] J.-M. Yang, J.-Y. Park, and R.-D. Oh, "Context awareness acquisition for safety sensor data processing on industrial sensor network," in *Advanced Computing, Networking and Security* (P. Thilagam, A. Pais, K. Chandrasekaran, and N. Balakrishnan, eds.), vol. 7135 of *Lecture Notes in Computer Science*, pp. 38–47, Springer Berlin Heidelberg, 2012. http://dx.doi.org/10.1007/978-3-642-29280-4_5.

[98] D. Lucke, C. Constantinescu, and E. Westkämper, "Smart factory - a step towards the next generation of manufacturing," in *Manufacturing Systems and Technologies for the New Frontier* (M. Mitsuishi, K. Ueda, and F. Kimura, eds.), pp. 115–118, Springer London, 2008. `http://link.springer.com/content/pdf/10.1007%2F978-1-84800-267-8_23.pdf`.

[99] A. Smirnov and N. Shilov, "Context-aware smart sustainable factories: Technological framework," in *Sustainable Manufacturing* (G. Seliger, ed.), pp. 151–156, Springer Berlin Heidelberg, 2012. `http://dx.doi.org/10.1007/978-3-642-27290-5_23`.

[100] L. A. Zadeh, "Making Computers Think Like People," *IEEE Spectrum*, vol. 21, no. 8, pp. 26–32, 1984. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6370431`.

[101] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338 – 353, 1965. `http://www.sciencedirect.com/science/article/pii/S001999586590241X`.

[102] L.-X. Wang, "Fuzzy systems are universal approximators," in *IEEE International Conference on Fuzzy Systems*, (San Diego, CA, USA), pp. 1163–1170, Mar 1992. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=258721`.

[103] V. Kreinovich, H. T. Nguyen, and Y. Yam, "Fuzzy systems are universal approximators for a smooth function and its derivatives," *International Journal of Intelligent Systems*, vol. 2000, pp. 565–574, 1999. `http://digitalcommons.utep.edu/cgi/viewcontent.cgi?article=1514&context=cs_techrep`.

[104] R. Rovatti, "Fuzzy Piecewise Multilinear and Piecewise Linear Systems As Universal Approximators in Sobolev Norms," *IEEE Transactions on Fuzzy Systems*, vol. 6, pp. 235–249, May 1998. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=669022`.

[105] S. Sanchez-Solano, A. Barriga, C. Jimenez, and J. Huertas, "Design and application of digital fuzzy controllers," in *Fuzzy Systems, 1997., Proceedings of the Sixth IEEE International Conference on*, vol. 2, pp. 869–874 vol.2, Jul 1997. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=622824`.

[106] D. Dubois and H. Prade, "Unfair Coins and Necessity Measures: Towards a Possibilistic Interpretation of Histograms," *Fuzzy Sets Syst.*, vol. 10, pp. 15–20, January 1983. `http://dx.doi.org/10.1016/S0165-0114(83)80099-2`.

[107] T. Runkler, "Selection of appropriate defuzzification methods using application specific properties," *IEEE Transactions on Fuzzy Systems*, vol. 5, pp. 72–79, Feb 1997. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=554449`.

[108] M. Braae and D. A. Rutherford, "Fuzzy relations in a control setting," *Kybernetes*, vol. 7, pp. 185–188. `http://www.emeraldinsight.com/doi/abs/10.1108/eb005482`.

[109] R. Yager, "Fuzzy sets and approximate reasoning in decision and control," in *IEEE International Conference on Fuzzy Systems*, (San Diego, CA, USA), pp. 415–428, Mar 1992. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=258652`.

[110] D. P. Filev and R. R. Yager, "A generalized defuzzification method via bad distributions," *International Journal of Intelligent Systems*, vol. 6, no. 7, pp. 687–697, 1991. `http://onlinelibrary.wiley.com/doi/10.1002/int.4550060702/abstract`.

[111] G. Clark, "The organization of behavior: A neuropsychological theory. d. o. hebb. john wiley and sons, inc., new york, 1949, 335 pages, 19 illustrations, 288 references.," *The Journal of Comparative Neurology*, vol. 93, no. 3, pp. 459–460, 1950. `http://dx.doi.org/10.1002/cne.900930310`.

[112] K. Mao, K. C. Tan, and W. Ser, "Probabilistic neural-network structure determination for pattern classification," *IEEE Transactions on Neural Networks*, vol. 11, pp. 1009–1016, Jul 2000. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=857781`.

[113] J. Pradeep, E. Srinivasan, and S. Himavathi, "Neural network based handwritten character recognition system without feature extraction," in *2011 International Conference on Computer, Communication and Electrical Technology (ICCCET)*, (Tamilnadu, India), pp. 40–44, March 2011. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5762513`.

[114] J. Parri and S. Ratti, "Trigonometric function approximation neural network based coprocessor," in *2nd Microsystems and Nanoelectronics Research Conference*, (Ottawa, ON, Canada), pp. 148–151, Oct 2009. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5338938`.

[115] J. Vesanto and E. Alhoniemi, "Clustering of the self-organizing map," *IEEE Transactions on Neural Networks*, vol. 11, pp. 586–600, May 2000. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=846731`.

[116] N. Nasrabadi and Y. Feng, "Vector quantization of images based upon the kohonen self-organizing feature maps," in *IEEE International Conference on Neural Networks, 1988.*, (San Diego, CA, USA), pp. 101–108 vol.1, July 1988. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=23837`.

[117] J. H. Holland, "Outline for a Logical Theory of Adaptive Systems," *J. ACM*, vol. 9, pp. 297–314, July 1962. `http://doi.acm.org/10.1145/321127.321128`.

[118] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975. `http://ieeeexplore.com/xpl/bkabstractplus.jsp?bkn=6267401`.

[119] F. H. F. Leung, H. Lam, S. Ling, and P.-S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Transactions on Neural Networks*, vol. 14, pp. 79–88, Jan 2003. `ieeeexplore.com/xpl/articleDetails.jsp?tp=&arnumber=1176129`.

[120] C. W. Ahn and R. S. Ramakrishna, "A genetic algorithm for shortest path routing problem and the sizing of populations," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 566–579, Dec 2002. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1134124`.

[121] C.-T. Cheng, K. Fallahi, H. Leung, and C. Tse, "An AUVs path planner using genetic algorithms with a deterministic crossover operator," in *IEEE International Conference on Robotics and Automation (ICRA), 2010*, (Anchorage, AK, USA), pp. 2995–3000, May 2010. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5509335`.

[122] M. Reformat, E. Kuffel, D. Woodford, and W. Pedrycz, "Application of genetic algorithms for control design in power systems," *IEEE Proceedings - Generation, Transmission and Distribution*, vol. 145, pp. 345–354, Jul 1998. `ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=707075`.

[123] J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 16, pp. 122–128, Jan 1986. `ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4075583`.

[124] I. del Campo, J. Echanobe, G. Bosque, and J. Tarela, "Efficient hardware/software implementation of an adaptive neuro-fuzzy system," *IEEE Transactions on Fuzzy Systems*, vol. 16, pp. 761–778, June 2008. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4392481`.

[125] M. A. Lopez, A. A. Garza, B. Y. Marquez, K. Romero, M. del Rosario Baltazar-Flores, and V. Zamudio, "Optimization of a fuzzy contrast method for a pattern recognition system," in *Intelligent Environments (Workshops)* (J. A. Botía, H. R. Schmidtke, T. Nakashima, M. R. Al-Mulla, J. C. Augusto, A. Aztiria, M. Ball, V. Callaghan, D. J. Cook, J. Dooley, J. O'Donoghue, S. Egerton, P. A. Haya, M. J. Hornos, E. Morales, J. C. Orozco, O. Portillo-Rodríguez, A. R. González, O. Sandoval, P. Tripicchio, M. Wang, and V. Zamudio, eds.), vol. 13 of *Ambient Intelligence and Smart Environments*, (Guanajuato, Mexico), pp. 142–153, IOS Press, 2012. `http://ebooks.iospress.nl/publication/28219`.

[126] C. G. D. Boeree, "The neuron," 2009. http://webspace.ship.edu/cgboer/theneuron.html.

[127] C.-T. Lin and C. S. G. Lee, *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems.* Prentice Hall PTR, 1996.

[128] M. Minsky and S. A. Papert, *Perceptrons: An Introduction to Computational Geometry.* 1969.

[129] "Human brain project." https://www.humanbrainproject.eu/.

[130] F. Rosenblatt, "The Perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, pp. 386–408, 1958. http://psycnet.apa.org/index.cfm?fa=buy.optionToBuy&id=1959-09865-001.

[131] B. Widrow and M. E. Hoff, "Adaptive Switching Circuits," in *1960 IRE WESCON Convention Record, Part 4*, (New York, NY, USA), pp. 96–104, IRE, 1960. http://www-isl.stanford.edu/people/widrow/papers/c1960adaptiveswitching.pdf.

[132] J. J. Hopfield, "Neurocomputing: Foundations of Research," ch. Neural Networks and Physical Systems with Emergent Collective Computational Abilities, pp. 457–464, Cambridge, MA, USA: MIT Press, 1988.

[133] R. Nielsen, *Neurocomputing.* New Horizons in Technology Series, Addison-Wesley, 1990.

[134] S. Rajasekaran and G. Pai, *Neural Networks, Fuzzy Logic and Genetic Algorithms.* PHI Learning Private Limited, 2011.

[135] S. Grossberg, "Adaptive pattern classification and universal recording II. Feedback, expectation, olfaction, illusions," *Biological Cybernetics*, vol. 23, pp. 187–202, 1976.

[136] G. A. Carpenter and S. Grossberg, "ART 2: Self-Organization of Stable Category Recognition Codes for Analog Input Patterns," *Applied Optics*, vol. 26, pp. 4919–4930, 1987. http://cns.bu.edu/~steve/CarGro1987AppliedOptics.pdf.

[137] G. A. Carpenter and S. Grossberg, "ART 3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architectures," *Neural Networks*, vol. 3, no. 2, pp. 129 – 152, 1990. http://link.springer.com/chapter/10.1007%2F978-94-009-0643-3_93.

[138] S. Anbazhagan and N. Kumarappan, "Day-Ahead Deregulated Electricity Market Price Forecasting Using Recurrent Neural Network," *IEEE Systems Journal*, vol. 7, pp. 866–872, Dec 2013. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6397562.

[139] Z. Lin, L. Gao, and D. Zhang, "Predictions of System Marginal Price of Electricity Using Recurrent Neural Network," in *The Sixth World Congress on Intelligent Control and Automation*, vol. 2, (Dalian, People's Republic of China), pp. 7592–7595, 2006. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1713442.

[140] G. Khan, A. Khattak, F. Zafari, and S. Mahmud, "Electrical load forecasting using fast learning recurrent neural networks," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, (Dallas, TX, USA), pp. 1–6, Aug 2013. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6706998.

[141] G. Luhasz, M. Tirea, and V. Negru, "Neural Network Predictions of Stock Price Fluctuations," in *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, (Timisoara, Rumania), pp. 505–512, Sept 2012. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6481072.

[142] S. Pattamavorakun and S. Pattamavorakun, "Determination the Number of Hidden Nodes of Recurrent Neural Networks for River Flow and Stock Price Forecasting," in *5th ACIS International Conference on Software Engineering Research, Management Applications*, (Busan, South Korea), pp. 184–194, Aug 2007. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4296935.

[143] T.-Y. Kwok and D.-Y. Yeung, "Constructive algorithms for structure learning in feedforward neural networks for regression problems," *IEEE Transactions on Neural Networks*, vol. 8, pp. 630–645, May 1997. ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=572102.

[144] D. Tian, Y. Liu, and J. Wang, "SLNN: A neural Network for Fuzzy Neural Network's Structure Learning," in *Sixth International Conference on Intelligent Systems Design and Applications, 2006. ISDA '06*, vol. 1, (Jinan, People's Republic of China), pp. 919–924, Oct 2006. ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4021562.

[145] J. A. Anderson, "Cognitive and psychological computation with neural models," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 13, pp. 799–815, 1983. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6313074.

[146] B. Kosko, "Bidirectional associative memories," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 18, pp. 49–60, Jan. 1988. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=87054.

[147] R. Linsker, "Self-Organization in a Perceptual Network," *Computer*, vol. 21, pp. 105–117, Mar. 1988. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=36.

[148] R. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, vol. 4, pp. 4 –22, apr 1987. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1165576`.

[149] T. Kohonen, *Self-organization and Associative Memory: 3rd Edition*. New York, NY, USA: Springer-Verlag New York, Inc., 1989. `http://link.springer.com/book/10.1007%2F978-3-642-88163-3`.

[150] H. T. Lefteri and R. E. Uhrig, *Fuzzy and Neural Approaches in Engineering*. New York, NY, USA: John Wiley & Sons, Inc., 1st ed., 1996. `http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0471160032.html`.

[151] A. G. Barto, *Neural networks for control*. Cambridge, MA, USA: MIT Press, 1990. Chapter 1: Connectionist Learning for Control.

[152] J. Jang, C. Sun, and E. Mizutani, *Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence*. MATLAB curriculum series, Prentice Hall, 1997.

[153] R. S. Sutton, "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming," in *Proceedings of the Seventh International Conference on Machine Learning*, (San Diego, CA, USA), pp. 216–224, Morgan Kaufmann, 1990. `http://papersdb.cs.ualberta.ca/~papersdb/uploaded_files/505/paper_sutton-90.pdf`.

[154] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[155] G. D. Magoulas, M. N. Vrahatis, and G. S. Androulakis, "Improving the convergence of the backpropagation algorithm using learning rate adaptation methods," *Neural Computation*, vol. 11, no. 7, pp. 1769–1796, 1999. `http://www.mitpressjournals.org/doi/pdf/10.1162/089976699300016223`.

[156] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," *CoRR*, vol. abs/1212.5701, 2012.

[157] R. Rojas, *Neural Networks: A Systematic Introduction*. No. 1, Berlin, Germany: Springer-Verlag Berlin Heidelberg, 1996. `http://www.springer.com/us/book/9783540605058`.

[158] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," in *International Joint Conference on Neural Networks, 1990., 1990 IJCNN*, (San Diego, CA, USA), pp. 21 –26 vol.3, jun 1990. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5726777`.

[159] R. Finker de la Iglesia, "Diseño e implementación de una RNA integrando la tecnología de "Reconfiguración Dinámica Parcial". aplicación en entornos de Inteligencia Ambiental," Master's thesis, University of the Basque country, 2012.

[160] R. Setiono and L. C. K. Hui, "Use of a quasi-Newton method in a feed-forward neural network construction algorithm," *IEEE Transactions on Neural Networks*, vol. 6, pp. 273–277, Jan 1995. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=363426.

[161] R. Reed, "Pruning algorithms-a survey," *IEEE Transactions on Neural Networks*, vol. 4, pp. 740–747, Sep 1993. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=248452.

[162] J.-C. Li, W. Ng, P. Chan, and D. Yeung, "A growing architecture selection for Multilayer Perceptron Neural Network by the L-GEM," in *2010 International Conference on Machine Learning and Cybernetics (ICMLC)*, vol. 3, (Qingdao, People's Republic of China), pp. 1402–1407, July 2010. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5580850.

[163] A. R. Omondi and J. C. Rajapakse, *FPGA Implementations of Neural Networks*. New York, NY, USA: Springer, 2006. http://www.springer.com/engineering/circuits+%26+systems/book/978-0-387-28485-9.

[164] K.-S. Oh and K. Jung, "GPU implementation of neural networks," *Pattern Recognition*, vol. 37, no. 6, pp. 1311 – 1314, 2004. http://www.sciencedirect.com/science/article/pii/S0031320304000524.

[165] J. Hines, L. H. Tsoukalas, and R. E. Uhrig, *Matlab Supplement to Fuzy and Neural Approaches in Engineering*. Wiley-Interscience Publication, 1997.

[166] Intel corporation, *Intel Core Processor Family*, 2013. http://www.intel.es/content/dam/www/public/us/en/documents/product-briefs/4th-gen-core-desktops-brief.pdf.

[167] A. Holdings, *The ARM Cortex - A9 Processors*. http://www.arm.com/files/pdf/ARMCortexA-9Processors.pdf.

[168] A. Holdings, *Cortex-R Series Webpage*. http://www.arm.com/products/processors/cortex-r/index.php Accessed 2014.

[169] Freescale, *Coldfire Processors Table*. http://www.freescale.com/webapp/sps/site/taxonomy.jsp?code=CFMPU.

[170] M. Mohamadian, E. Nowicki, F. Ashrafzadeh, A. Chu, R. Sachdeva, and E. Evanik, "A novel neural network controller and its efficient dsp implementation for vector-controlled induction motor drives," *IEEE Transactions on Industry Applications*, vol. 39, pp. 1622–1629, Nov 2003. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1248245.

[171] N. Kim, N. Kehtarnavaz, M. Yeary, and S. Thornton, "DSP-based hierarchical neural network modulation signal classification," *IEEE Transactions on Neural Networks*, vol. 14, pp. 1065–1071, Sept 2003. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1243710.

[172] G. Bosque, I. del Campo, and J. Echanobe, "Fuzzy systems, neural networks and neuro-fuzzy systems: A vision on their hardware implementation and platforms over two decades," *Engineering Applications of Artificial Intelligence*, vol. 32, no. 0, pp. 283 – 331, 2014. http://www.sciencedirect.com/science/article/pii/S0952197614000384.

[173] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, no. 3, pp. 239 – 255, 2010. Artificial Brains.

[174] S. Draghici, "Neural networks in analog hardware-Design and implementation issues," *International journal of neural systems*, vol. 10, no. 01, pp. 19–42, 2000. http://www.worldscientific.com/doi/abs/10.1142/S0129065700000041.

[175] K. Someya, H. Shinozaki, and Y. Sekine, "Pulse-type hardware chaotic neuron model and its bifurcation phenomena," *Neural Networks*, vol. 12, no. 1, pp. 153 – 161, 1999. http://www.sciencedirect.com/science/article/pii/S0893608098000999.

[176] A. Agranat, C. Neugebauer, and A. Yariv, "A ccd based neural network integrated circuit with 64k analog programmable synapses," in *International Joint Conference on Neural Networks (IJCNN)*, (San Diego, CA, USA), pp. 551–555 vol.2, June 1990. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5726583.

[177] T. Morishita, Y. Tamura, and T. Otsuki, "A bicmos analog neural network with dynamically updated weights," in *IEEE International Solid-State Circuits Conference, 1990. Digest of Technical Papers. 37th ISSCC*, (San Francisco, CA, USA), pp. 142–143, Feb 1990. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=110167.

[178] M. Holler, S. Tam, H. Castro, and R. Benson, "An electrically trainable artificial neural network (ETANN) with 10240 'floating gate' synapses," in *International Joint Conference on Neural Networks (IJCNN)*, (Washington, DC, USA), pp. 191–196 vol.2, 1989. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=118698.

[179] T. Orlowska-Kowalska and M. Kaminski, "FPGA Implementation of the Multilayer Neural Network for the Speed Estimation of the Two-Mass Drive System," *IEEE Transactions on Industrial Informatics*, vol. 7, pp. 436–445, Aug 2011.

[180] M. Kaminski and T. Orlowska-Kowalska, "FPGA Implementation of ADALINE-Based Speed controller in a Two-Mass System," *IEEE Transactions on Industrial Informatics*, vol. 9, pp. 1301–1311, Aug 2013. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6340338`.

[181] P. Domingos, F. Silva, and H. Neto, "An efficient and scalable architecture for neural networks with backpropagation learning," in *International Conference on Field Programmable Logic and Applications, 2005*, (Tampere, Finland), pp. 89–94, Aug 2005. `ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1515704`.

[182] A. Gomperts, A. Ukil, and F. Zurfluh, "Development and Implementation of Parameterized FPGA-Based General Purpose Neural Networks for Online Applications," *IEEE Transactions on Industrial Informatics*, vol. 7, pp. 78 –89, feb. 2011. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5607329`.

[183] R. G. Girones, R. C. Palero, J. C. Boluda, , and A. S. Cortes, "FPGA implementation of a Pipelined On-Line Backpropagation," *The Journal of VLSI Signal Processing*, vol. 40, pp. 189–213, 2005. `http://dx.doi.org/10.1007/s11265-005-4961-3`.

[184] A. Momoi, S. Akimoto, S. Sato, and K. Nakajima, "Implementation of a large scale hardware neural network system based on stochastic logic," in *IEEE International Joint Conference on Neural Networks (IJCNN)*, vol. 4, (Budapest, Hungary), pp. 2671–2675 vol.4, July 2004. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1381070`.

[185] J. Schemmel, A. Grubl, K. Meier, and E. Mueller, "Implementing Synaptic Plasticity in a VLSI Spiking Neural Network Model," in *International Joint Conference on Neural Networks (IJCNN)*, (Honolulu, HI, USA), pp. 1–6, 2006. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=171606`.

[186] Xilinx, *FPGA vs ASIC*, 2014. `http://www.xilinx.com/fpga/asic.htm` Accessed 2014.

[187] P. Masa, K. Hoen, and H. Wallinga, "A high-speed analog neural processor," *IEEE Micro*, vol. 14, no. 3, pp. 40–50, 1994. `ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=285223`.

[188] J. Teich, "Hardware/software codesign: The past, the present, and predicting the future," *Proceedings of the IEEE*, vol. 100, pp. 1411–1430, May 2012. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6172642`.

[189] Nvidia, "Compute Unified Device Architecture." `http://www.nvidia.com/object/cuda_home_new.html` Accessed 2014.

[190] Aberdeen Stirling, *Aberdeen Stirling 464G - Xeon DP GPU-Optimized HPC Server.* http://www.aberdeeninc.com/abcatg/stirling-464g.htm Accesed 2014.

[191] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980. http://dx.doi.org/10.1007/BF00344251.

[192] H. Perez-Sanchez, G. Guerrero, J. Garcia, J. Pena, J. Cecilia, G. Cano, S. Orts-Escolano, and J. Garcia-Rodriguez, "Improving drug discovery using a neural networks based parallel scoring function," in *International Joint Conference on Neural Networks (IJCNN)*, (Dallas, TX, USA), pp. 1–5, Aug 2013. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6706909.

[193] D. Ciresan, U. Meier, J. Masci, and J. Schmidhuber, "A committee of neural networks for traffic sign classification," in *International Joint Conference on Neural Networks (IJCNN)*, (San Jose, CA, USA), pp. 1918–1921, July 2011. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6033458.

[194] C.-F. Juang, T.-C. Chen, and W.-Y. Cheng, "Speedup of Implementing Fuzzy Neural Networks With High-Dimensional Inputs Through Parallel Processing on Graphic Processing Units," *IEEE Transactions on Fuzzy Systems*, vol. 19, pp. 717–728, Aug 2011. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5744114.

[195] K. Basterretxea, I. del Campo, and J. Echanobe, "A semi-active suspension embedded controller in a FPGA," in *International Symposium on Industrial Embedded Systems (SIES)*, (Trento, Italia), pp. 69 –78, july 2010. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5551387.

[196] Xilinx, "What is programmable logic?." http://www.xilinx.com/company/about/programmable.html Accessed 2014.

[197] J. Turner and G. Josephson, "Programmable logic array," August 1988. http://www.google.com/patents/US4766569.

[198] J. Birkner and H. Chua, "Programmable array logic circuit," November 1978. http://www.google.com/patents/US4124899.

[199] Wikipedia, "Programmable logic array — wikipedia, the free encyclopedia," 2015. Accessed 2014 http://en.wikipedia.org/w/index.php?title=Programmable_logic_array&oldid=657045322.

[200] Cypress, *PAL CE22V10 Datasheet*, September 1996. http://www.engr.uky.edu/$\sim$jel/misc/d481/info/PAL/ce22v10.pdf.

[201] Xilinx, *Complex Programmable Logic Device*, Accesed 2014. http://www.xilinx.com/cpld/.

[202] Xilinx, *Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet*, June 2011. http://www.xilinx.com/support/documentation/data_sheets/ds083.pdf.

[203] Xilinx, *XtremeDSP 48 Slice*. http://www.xilinx.com/technology/dsp/xtremedsp.htm Accessed 2014.

[204] Xilinx, *Spartan 6 product brief*. http://www.xilinx.com/publications/prod_mktg/Spartan6_Product_Table.pdf Accesed 2014.

[205] Xilinx, *7 Series FPGAs Overview*, v1.3 ed., July 2013. http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf.

[206] Xilinx, *Zynq-7000 All Programmable SoC Overview*, v1.6 ed., December 2013. http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf.

[207] Altera, *SoC Overview*. http://www.altera.com/devices/processor/soc-fpga/overview/proc-soc-fpga.html Accesed 2014.

[208] J. K. Lew, "Low Power System Design Techniques Using FPGAs," *EE TImes*, 2004. http://www.eetimes.com/document.asp?doc_id=1271155.

[209] Xilinx, *Spartan-3AN FPGA Family Data Sheet*, 2011. www.xilinx.com/support/documentation/data_sheets/ds557.pdf.

[210] Xilinx, *Xilinx Webpage*. http://www.xilinx.com/.

[211] Xilinx, *SEU Strategies for Virtex-5 Devices*, 2010. http://www.xilinx.com/support/documentation/application_notes/xapp864.pdf.

[212] C. Carmichael, E. Fuller, P. Blain, and M. Caffrey, "SEU mitigation techniques for Virtex FPGAs in space applications," in *Proceeding of the Military and Aerospace Programmable Logic Devices International Conference*, (Laurel, MD, USA), p. 99, 1999. http://www.xilinx.com/appnotes/VtxSEU.pdf.

[213] Design & Reuse, *Design Security in Nonvolatile Flash and Antifuse FPGAs*, 2003. http://www.design-reuse.com/articles/6529/design-security-in-nonvolatile-flash-and-antifuse-fpgas.html.

[214] Xilinx, *Partial Reconfiguration User Guide*, April 2013. http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/ug702.pdf.

[215] Altera, *Stratix V FPGAs: Ultimate Flexibility Through Partial and Dynamic Reconfiguration.* http://www.altera.com/devices/fpga/stratix-fpgas/stratix-v/overview/partial-reconfiguration/stxv-part-reconfig.html Acessed 2014.

[216] Xilinx, *Difference-Based Partial Reconfiguration*, December 2007. http://www.xilinx.com/support/documentation/application_notes/xapp290.pdf.

[217] Xilinx, *LogiCORE IP XPS HWICAP (V5.00a)*, July 2010. http://www.xilinx.com/support/documentation/ip_documentation/xps_hwicap.pdf.

[218] Xilinx, *LogiCORE IP AXI HWICAP (V2.02a)*, April 2012. http://www.xilinx.com/support/documentation/ip_documentation/axi_hwicap/v2_02_a/ds817_axi_hwicap.pdf.

[219] Altera, *Design Planning for Partial Reconfiguration*, June 2014. http://www.altera.com/literature/hb/qts/qts_qii51026.pdf.

[220] Xilinx, *Power Consumption in 65 nm FPGAs*, February 2007. http://www.xilinx.com/support/documentation/white_papers/wp246.pdf.

[221] Xilinx. http://www.xilinx.com/about/company-overview/index.htm.

[222] Xilinx. http://www.xilinx.com/tools/partial-reconfiguration.htm Accessed 2014.

[223] Xilinx, *All Programmable FPGAs*, 2014. http://www.xilinx.com/products/silicon-devices/fpga/index.htm Accessed 2014.

[224] Altera, "14 nm and 20 nm process technology." http://www.altera.com/technology/system-tech/next-gen/process-technologies.html Accessed 2014.

[225] M. Corporation, *Spaceflight FPGAs*, 2012. http://www.microsemi.com/document-portal/doc_download/131352-spaceflight-fpgas-catalog.

[226] L. Semiconductor, *Lattice ECP3 FPGA Family Table.* http://www.latticesemi.com/Products/FPGAandCPLD/LatticeECP3.aspx Accessed 2014.

[227] L. Semiconductor, *Lattice iCE40 Family Table.* http://www.latticesemi.com/Products/FPGAandCPLD/iCE40.aspx Accessed 2014.

[228] A. Corporation, "AT40KAL Series co-processor FPGAs." http://www.atmel.com/products/other/field_programmable_gate_array/default.aspx Accessed 2014.

[229] *VHDL Analysis and Standardization Group (VASG).* http://www.eda.org/vasg/.

[230] Xilinx, *LogiCORE IP Fast Simplex Link (FSL) V2.0 Bus*, April 2010. http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf.

[231] A. Holdings, *AMBA AXI4-Stream Protocol Specification*, 2010. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0051a/index.html Accessed 2014.

[232] X. Li, M. Moussa, and S. Areibi, "Arithmetic formats for implementing artificial neural networks on FPGAs," *Canadian Journal of Electrical and Computer Engineering*, vol. 31, pp. 31–40, Winter 2006. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4028882.

[233] A. Damak, M. Krid, and D. Masmoudi, "Neural network based edge detection with pulse mode operations and floating point format precision," in *3rd International Conference on Design and Technology of Integrated Systems in Nanoscale Era*, (Tozeur, Tunisia), pp. 1–5, March 2008. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4540255.

[234] H. Hikawa, "Pulse mode multilayer neural network based on floating point number representation," in *IEEE International Symposium on Circuits and Systems*, vol. 3, (Geneva, Switzerland), pp. 145–148 vol.3, 2000. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=856017.

[235] H. Hikawa, "Pulse mode multilayer neural network with floating point operation and on-chip learning," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, vol. 2, (Geneva, Switzerland), pp. 71–76 vol.2, 2000. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=857877.

[236] H. El-Madany, F. Fahmy, N. El-Rahman, and H. Dorrah, "Design of FPGA based neural network controller for earth station power system," *Telkomnika*, vol. 10, no. 2, pp. 281–290, 2012. http://iaesjournal.com/online/index.php/TELKOMNIKA/article/view/681.

[237] M. Hoffman, P. Bauer, B. Hemrnelman, and A. Hasan, "Hardware synthesis of artificial neural networks using field programmable gate arrays and fixed-point numbers," in *IEEE Region 5 Conference*, (San Antonio, TX, USA), pp. 324–328, April 2006. ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5507410.

[238] J. Vasquez, S. Perez, C. Travieso, and J. Alonso, "Meteorological Prediction Implemented on Field-Programmable Gate Array," *Cognitive Computation*, vol. 5, no. 4, pp. 551–557, 2013. `http://link.springer.com/article/10.1007%2Fs12559-012-9158-z`.

[239] H. Madokoro and K. Sato, "Hardware implementation of back-propagation neural networks for real-time video image learning and processing," *Journal of Computers (Finland)*, vol. 8, no. 3, pp. 559–566, 2013. `http://www.ojs.academypublisher.com/index.php/jcp/article/view/jcp0803559566`.

[240] A. Savich, M. Moussa, and S. Areibi, "The Impact of Arithmetic Representation on Implementing MLP-BP on FPGAs: A Study," *IEEE Transactions on Neural Networks*, vol. 18, pp. 240–252, Jan 2007. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4049835`.

[241] Altera, *Variable Precision DSP Blocks in Arria 10 Devices*, December 2013. `http://www.altera.com/literature/hb/arria-10/a10_dsp.pdf`.

[242] Xilinx, *7 Series FPGAs Memory Resources*, May 2014. `http://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf`.

[243] Altera, *Internal Memory (RAM and ROM) User Guide*, November 2013. `http://www.altera.com/literature/ug/ug_ram_rom.pdf`.

[244] R. Muscedere, V. Dimitrov, G. Jullien, and W. Miller, "Efficient techniques for binary-to-multidigit multidimensional logarithmic number system conversion using range-addressable look-up tables," *IEEE Transactions on Computers*, vol. 54, pp. 257–271, March 2005. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1388191`.

[245] U. Martinez-Corral, K. Basterretxea, and R. Finker, "Scalable parallel architecture for singular value decomposition of large matrices," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, (Munich, Germany), pp. 1–4, Sept 2014. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6927393`.

[246] C.-W. Lin and J.-S. Wang, "A digital circuit design of hyperbolic tangent sigmoid function for neural networks," in *IEEE International Symposium on Circuits and Systems*, (Seattle, WA, USA), pp. 856–859, May 2008. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4541553`.

[247] K. Basterretxea, J. Tarela, and I. Del Campo, "Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons," *IEEE Proceedings Circuits, Devices and Systems*, vol. 151, pp. 18–24,

Feb 2004. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1267679.

[248] B. Zamanlooy and M. Mirhassani, "Efficient VLSI Implementation of Neural Networks With Hyperbolic Tangent Activation Function," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, pp. 39–48, Jan 2014. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6409494.

[249] A. Armato, L. Fanucci, E. P. Scilingo, and D. De Rossi, "Low-error digital hardware implementation of artificial neuron activation functions and their derivative," *MICROPROCESSORS AND MICROSYSTEMS*, vol. 35, pp. 557–567, AUG 2011. http://www.sciencedirect.com/science/article/pii/S0141933111000731.

[250] K. Basterretxea, J. Tarela, I. del Campo, and G. Bosque, "An experimental study on nonlinear function computation for neural/fuzzy hardware design," *IEEE Transactions on Neural Networks*, vol. 18, pp. 266–283, Jan 2007. ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4049811.

[251] T. M. Inc., *Fixed-Point Designer*, 2014. www.mathworks.es/products/fixed-point-designer/ Accessed 2014.

[252] Mentor Graphics, *ModelSim, ASIC and FPGA design*. www.mentor.com/products/fpga/simulation/modelsim Accessed 2014.

[253] G. Mermoud, A. Upegui, C. A. Peña-Reyes, and E. Sanchez, "A dynamically-reconfigurable fpga platform for evolving fuzzy systems," in *IWANN* (J. Cabestany, A. Prieto, and F. S. Hernández, eds.), vol. 3512 of *Lecture Notes in Computer Science*, pp. 572–581, Springer, 2005. http://www.researchgate.net/publication/221581963_A_Dynamically-Reconfigurable_FPGA_Platform_for_Evolving_Fuzzy_Systems.

[254] N. Chalhoub, F. Muller, and M. Auguin, "FPGA-based generic neural network architecture," in *International Symposium on Industrial Embedded Systems*, (Antibes Juan-Les-Pins, France), pp. 1 –4, oct. 2006. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4197498.

[255] J. Starzyk, Y. Guo, and Z. Zhu, "Dynamically reconfigurable neuron architecture for the implementation of self- organizing learning array," in *18th International Parallel and Distributed Processing Symposium*, (Santa Fe, NM, USA), pp. 143–, April 2004. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1303122.

[256] Xilinx Inc., *PicoBlaze 8-bit Embedded Microcontroller User Guide*, June 2011. http://www.xilinx.com/support/documentation/ip_documentation/ug129.pdf.

[257] N. Harb, S. Niar, M. Saghir, Y. Hillali, and R. Atitallah, "Dynamically reconfigurable architecture for a driver assistant system," in *IEEE 9th Symposium on Application Specific Processors (SASP)*, (San Diego, CA, USA), pp. 62–65, June 2011. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5941079.

[258] J. Echanobe, I. Del Campo, R. Finker, and K. Basterretxea, "Dynamic Partial Reconfiguration in Embedded Systems for Intelligent Environments," in *8th International Conference on Intelligent Environments (IE), 2012*, (Guanajuato, Mexico), pp. 109–113, June 2012. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6258510.

[259] J. Grantner and C. Nguyen, "Flexible decision support system using dynamic partial reconfiguration technology," in *2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, (Beijing, People's Republic of China), pp. 2270–2276, July 2014. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6891850.

[260] Xilinx Inc., *Partial Reconfiguration in the Vivado Design Suite*, Accessed 2015. http://www.xilinx.com/products/design-tools/vivado/implementation/partial-reconfiguration.html.

[261] D. Amos, A. Lesea, and R. Richter, "Getting the design ready for the prototype," in *FPGA-based Prototyping Methodology Manual*, ch. Chapter 7, Synopsys Press, 2011.

[262] Xilinx, *7 Series FPGAs Clocking Resources*, November 2014. http://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf.

[263] Altera, *Clock Control Block (ALTCLKCTRL) IP Core*, August 2014. http://www.altera.com/literature/ug/ug_altclock.pdf.

[264] F. Rivoallon and J. Balasubramanian, *Reducing Switching Power with Intelligent Clock Gating*. Xilinx. http://www.xilinx.com/support/documentation/white_papers/wp370_Intelligent_Clock_Gating.pdf.

[265] J. Kathuria, M. Ayoubkhan, and A. Noor, "A review of clock gating techniques," *MIT International Journal of Electronics and Communication Engineering*, vol. 1, pp. 106–114, August 2011. http://mitpublications.org/yellow_images/1315565167_logo_13.pdf.

[266] H. Hagras, V. Callaghan, M. Colley, G. Clarke, A. Pounds-Cornish, and H. Duman, "Creating an ambient-intelligence environment using embedded agents," *IEEE Intelligent Systems*, vol. 19, pp. 12–20, Nov 2004. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1363729.

[267] M. Woolridge and M. J. Wooldridge, *Introduction to Multiagent Systems.* New York, NY, USA: John Wiley & Sons, Inc., 2001.

[268] M.-J. Lee, G.-H. Hwang, W.-T. Jang, and K.-H. Cha, "Robotic agent control based on adaptive intelligent algorithm in ubiquitous networks," in *Agent and Multi-Agent Systems: Technologies and Applications* (N. Nguyen, A. Grzech, R. Howlett, and L. Jain, eds.), vol. 4496 of *Lecture Notes in Computer Science*, pp. 539–548, Springer Berlin Heidelberg, 2007. `http://dx.doi.org/10.1007/978-3-540-72830-6_56`.

[269] P. Vidnerova, S. Slusny, and R. Neruda, "Evolutionary trained radial basis function networks for robot control," in *10th International Conference on Control, Automation, Robotics and Vision*, (Hanoi, Vietnam), pp. 833–838, Dec 2008. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4795625`.

[270] M. Salvo, R. Mateo, J. Lee, and M. Lee, "Health monitor agent based on neural networks for ubiquitous healthcare environment," in *Agent and Multi-Agent Systems: Technologies and Applications* (A. Hakansson, N. Nguyen, R. Hartung, R. Howlett, and L. Jain, eds.), vol. 5559 of *Lecture Notes in Computer Science*, pp. 380–388, Springer Berlin Heidelberg, 2009. `http://dx.doi.org/10.1007/978-3-642-01665-3_38`.

[271] R. Yang and L. Wang, "Multi-agent based energy and comfort management in a building environment considering behaviors of occupants," in *2012 IEEE Power and Energy Society General Meeting*, (San Diego, CA, USA), pp. 1–7, July 2012. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6345671`.

[272] Xilinx, *ML605 Hardware User Guide*, October 2012. `http://www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf`.

[273] S. R. Garzon, "Intelligent in-car-infotainment systems: A contextual personalized approach," in *8th International Conference on Intelligent Environments (IE)*, (Guanajuato, Mexico), pp. 315–318, June 2012. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6258541`.

[274] J. H. L. Hansen, P. Boyraz, K. Takeda, and H. Abut, eds., *Digital Signal Processing for In-Vehicle Systems and Safety.* Springer Publishing Company, Incorporated, 2011. `http://www.springer.com/gp/book/9781441996060`.

[275] A. Schmidt, J. Paradiso, and B. Noble, "Automotive pervasive computing," *IEEE Pervasive Computing*, vol. 10, no. 3, pp. 12–13, 2011. `http://doi.ieeecomputersociety.org/10.1109/MPRV.2011.45`.

[276] E. Shang, J. Li, X. An, and H. He, "A real-time lane departure warning system based on fpga," in *14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, (Washington, DC, USA), pp. 1243–

1248, Oct 2011. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6082815.

[277] Z. Stamenkovic, K. Tittelbach-Helmrich, J. Domke, C. Lorchner-Gerdaus, J. Anders, V. Sark, M. Eric, and N. Sira, "Rear view camera system for car driving assistance," in *28th International Conference on Microelectronics (MIEL)*, (Nis, Serbia), pp. 383–386, May 2012. ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6222882.

[278] A. Eskandarian and A. Mortazavi, "Unobtrusive driver drowsiness detection system and method," May 2010. http://www.google.com/patents/US20100109881.

[279] H. Abut, A. Ercil, H. Erdogan, B. Çürüklü, H. C. Koman, F. Tas, A. Ö. Argunsah, S. Cosar, B. Akan, H. Karabalkan, E. Cökelek, R. Ficici, V. Sezer, S. Danis, M. Karaca, M. Abbak, M. G. Uzunbas, K. Eritmen, C. Kalaycioglu, M. Imamoglu, C. Karabat, M. Peyic, and B. Arslan, "Data collection with uyanik: Too much pain; but gains are coming," in *Proc. of the Biennial on DSP for In-Vehicle and Mobile Systems*, (Istanbul, Turkey), June 2007. http://www.iss.mdh.se/index.php?choice=publications&id=1831.

[280] Q. Ji, Z. Zhu, and P. Lan, "Real-time nonintrusive monitoring and prediction of driver fatigue," *IEEE Transactions on Vehicular Technology*, vol. 53, pp. 1052–1068, July 2004. ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1317209.

[281] L. Bergasa, J. Nuevo, M. Sotelo, R. Barea, and M. Lopez, "Real-time system for monitoring driver vigilance," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, pp. 63–77, March 2006. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1603553.

[282] M. Rocha, P. Cortez, and J. Neves, "Evolutionary design of neural networks for classification and regression," in *Adaptive and Natural Computing Algorithms* (B. Ribeiro, R. Albrecht, A. Dobnikar, D. Pearson, and N. Steele, eds.), pp. 304–307, Springer Vienna, 2005.

[283] B. Correa and A. Gonzalez, "Evolutionary algorithms for selecting the architecture of a mlp neural network: A credit scoring case," in *IEEE 11th International Conference on Data Mining Workshops (ICDMW)*, (Vancouver, BC, Canada), pp. 725–732, Dec 2011. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6137452.

[284] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995. http://link.springer.com/article/10.1023%2FA%3A1022627411411.

[285] K. P. Bennett and C. Campbell, "Support vector machines: Hype or hallelujah?," *SIGKDD Explor. Newsl.*, vol. 2, pp. 1–13, dec 2000. `http://doi.acm.org/10.1145/380995.380999`.

[286] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 42, pp. 513–529, April 2012. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6035797`.

[287] S. Decherchi, P. Gastaldo, A. Leoncini, and R. Zunino, "Efficient digital implementation of extreme learning machines for classification," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 59, pp. 496–500, Aug 2012. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6236105`.

[288] J. Martanez-Villena, A. Rosado-Munoz, and E. Soria-Olivas, "Hardware implementation methods in random vector functional-link networks," *Applied Intelligence*, vol. 41, no. 1, pp. 184–195, 2014. `http://link.springer.com/article/10.1007%2Fs10489-013-0501-1`.

[289] Nanyang Technological University, *Extreme Learning Machine*, Accessed 2014. Available at: `http://www.ntu.edu.sg/home/egbhuang/`.