UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Arquitectura escalable SIMD con conectividad jerárquica y reconfigurable para la emulación de SNN

Mireya Zapata Rodríguez

Departament d'Enginyeria Electrònica

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Arquitectura Escalable SIMD con Conectividad Jerárquica y Reconfigurable para la Emulación de SNN

Tesi doctoral presentada per a l' obtenció
del títol de doctor

MIREYA ZAPATA RODRÍGUEZ

DIRECTOR: DR. JORDI MADRENAS BOADAS

Barcelona, Septiembre 2017

Índice general

Agradecimientos

La culminación de esta tesis ha sido posible gracias al apoyo de mi supervisor, familiares y amigos.

Sinceramente agradezco al Dr. Jordi Madrenas por su invaluable guía y apoyo en cada una de las etapas de esta investigación.

A mi madre, por su amor incondicional y aliento en todos estos años.

A mi esposo, por su enorme paciencia, comprensión y amor.

A mis hermanos y sobrinos, por tener siempre una sonrisa que compartir.

A mis amigos, y en especial a Saoni Banerji por su cálida amistad y todos los días que compartimos y caminamos juntas al C5-101.

A Taho Dorta, Sergi Moreno, Malavika Suresh y Adithya Krish por su aporte al desarrollo de esta tesis.

Mis agradecimientos a la Secretaría Nacional de Educación Superior de Ecuador SENESCYT, por concederme la beca para la realización de este doctorado.

Índice de figuras

Índice de cuadros

Abreviaturas

AHA	Advanced Hardware Architectures
AER	Address Event Representation
AER-SRT	Address Event Representation over Synchronous Serial Ring
ALU	Arithmetic Logic Unit
ANNs	Artificial Neural Networks
ARM	Advanced RISC Machine
AM	Associative Memory
ASIC	Application Specific Integrated Circuit
BRAM	Block Random Access Memory
CAM	Content Addressable Memory
CPE	Configurable Processing Element
CPU	Central Processing Unit
CPh	Configuration Phase
DPh	Distribution Phase
EPh	Execution Phase
EvPh	Evolution Phase
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GPU	Graphic Processing Unit
GT	Gigabit Transceiver
HEENS	Hardare Emulator of Evolved Neural System
ILA	Integrated Logic Analyzer
IPh	Initialization Phase
IZ	Izhikevich
LIF	Leaky Integrate and Fire
LSB	Least Significant Bit

LTP	Long-term potentiation
LUT	Lookup Table
MC	Master Chip
MP	Multi-Processor
NC	Neuromorphic Chip
PE	Processing Element
PL	Programmable Logic
PS	Processing System
PSoC	Programmable System on Chip
RNA	Artificial Neural Network
SIMD	Single Instruction Multiple Data
SNAVA	Spiking Neural-network Architectures for Versatile Applications
SNNs	Spiking Neural Networks
STDP	Spike Timing Dependent Plasticity
SFC	Synfire Chain
VIO	Virtual Input Output
VHDL	Very high speed integrated circuit Hardware Description Language

Resumen

Un sistema neuronal biológico consiste de millones de neuronas altamente integradas con múltiples funciones dinámicas operando en coordinación entre sí. Su organización estructural se caracteriza por contener agrupaciones altamente jerárquicas. Dichas agrupaciones se distinguen por conexiones localmente densas y globalmente dispersas comunicadas a través de pulsos transitorios (spikes) que viajan por el axón hasta la neurona destino. En el último siglo, aproximarse a la complejidad biológica del cortex mediante arquitecturas de hardware continúa siendo un desafío todavía inalcanzable. Esto se debe, no sólo al masivo procesamiento paralelo con soporte para la comunicación entre neuronas en redes de gran escala, sinó también a la necesidad de mecanismos que permitan la evolución de la red neuronal de forma eficiente.

En este marco, esta tesis contribuye al desarrollo de una arquitectura denominada HEENS (Emulador de Hardware para Sistemas Neuronales Evolutivos, Hardware Emulator of Evolved Neural System) que soporta conectividad inter-chip; con una topología de anillo entre un chip que actúa de master (MC) y uno o más Chips Neuromórficos (NCs). El MC está implementado en un dispositivo PSoC que integra un CPU ARM Dual Core junto con lógica programable comunicados a través del protocolo estándar de bus AXI4. El procesador se encarga de configurar el anillo de comunicación y de ejecutar la aplicación de software que controla el envío de información de configuración del algoritmo y los parámetros neuronales a todos los NCs de la red. Durante la fase de ejecución, cuando se está procesando el algoritmo neuronal, el MC es el encargado de activar el modo de evolución de la red, así como de gestionar el envío de datos de reconfiguración a cualquiera de los nodos.

Cada NC a su vez, está compuesto por un arreglo 2D parametrizable de Elementos de Procesamiento (Processing Elements, PEs) con un esquema de procesamiento tipo SIMD implementado sobre una FPGA Kintex7. Los NCs son multiprocesadores SIMD que soportan la ejecución de cualquier algoritmo neuronal basado en spikes. Se cuenta con un set de instrucciones personalizadas diseñadas específicamente para esta arquitectura. Imitando la configuración estructural del cerebro, los NC soportan un esquema jerárquico con spikes locales y globales. Los spikes locales establecen la conectividad inter-neuronal dentro de un mismo chip, y los globales la comunicación inter-modular entre diferentes chips. Los NC cuentan con neuronas fijas tipo hub que procesan spikes locales y globales que permiten la conectividad inter e intra módulos. La conexión en anillo, con enlaces punto a punto, y la definición de spikes locales y globales permite desarrollar arquitecturas jerárquicas multi-nivel que se inspiran en las topologías del cerebro y ofrecen una escalabilidad excelente.

La propagación de spikes a través de la red multi-chip es soportada por una pila de protocolos Aurora/AER-SRT. El protocolo Aurora encapsula y desencapsula los paquetes transmitidos a través del enlace serial de alta velocidad que comunica la plataforma. Mientras que el protocolo Síncrono de Representación de Eventos de Dirección (AER-SRT) gestiona los datos (eventos de dirección) y los paquetes de control que permiten sincronizar la operación de la red neuronal. Cada evento encapsula la dirección de la neurona que genera un spike como resultado del procesamiento del algoritmo neuronal.

La definición de topología sináptica local y global es implementada usando bloques de memoria RAM on-chip, lo que reduce los requerimientos de lógica combinacional y, además de facilitar la configuración del conexionado sin modificar el hardware, permite el desarrollo de aplicaciones evolutivas al soportar la reconfiguración on-line tanto del algoritmo neuronal como de los parámetros neuronales y sinápticos. HEENS también admite retardos programables de axón, lo cual incorpora características dinámicas a la red.

Abstract

A biological neural system consists of millions of highly integrated neurons with multiple dynamic functions operating in coordination with each other. Its structural organization is characterized by highly hierarchical assemblies. These assemblies are distinguished by locally dense and globally dispersed connections communicated by spikes traveling through the axon to the target neuron. In the last century, approaching the biological complexity of the cortex by means of hardware architectures has continued to be a challenge still unattainable. This is not only due to the massively parallel processing with support for the communication between neurons in large-scale networks, but also for the need of mechanisms that allow the evolution of the neural network efficiently.

In this context, this thesis contributes to the development of an architecture called HEENS (Hardware Emulator of Evolved Neural System), which supports inter-chip connectivity with a ring topology between a Master Chip (MC) controlling one or more Neuromorphic Chips (NCs). The MC is implemented in a PSoC device that integrates a CPU ARM Dual Core together with programmable logic. The ARM is responsible for setting up the communication ring and executing the software application that controls the data configuration transmission from the algorithm and the neural parameters to all NCs in the network. Besides, the MC is in charge of activating the evolution mode of the network, as well as managing the dispatching of reconfiguration data to any of the nodes during the execution.

Each NC, in turn, consists of a configurable 2D array of Processing Elements (PEs) with a SIMD-like processing scheme implemented on a Kintex7 FPGA. NCs are SNN multiprocessors that support the execution of any neural algorithm based on spikes. A set of custom instructions was designed specifically for this architecture. The NCs support a hierarchical scheme of local and global spikes to mimic the brain structural configuration. Local spikes establish inter-neuronal connectivity within a single chip and the global ones allow inter-modular communication between different chips. The NCs have fixed hub neurons that process local and global spikes, thus allowing inter-modular and intra-modular connectivity. This definition of local and global spikes allows the development of multi-level hierarchical architectures inspired by the brain topologies, and offers excellent scalability.

The spike propagation through the multi-chip network is supported by an Aurora / AER-SRT protocol stack. The Aurora protocol encapsulates and de-capsulates the packets transmitted through a high-speed serial link that communicates the platform, while the Synchronous Address Event Representation (AER-SRT) protocol manages the data (address events) and controls packets that allow

synchronization of the operation of the neural network. Each event encapsulates the address neuron that fires a spike as result of the neural algorithm execution.

The definition of local and global synaptic topology is implemented using on-chip RAM blocks, which reduces the combinational logic requirements and, in addition to allowing the dynamic connectivity configuration, permits the development of evolutionary applications by supporting the on-line reconfiguration of both the neural algorithm or the neural and synaptic parameters. HEENS also supports axon programmable delays, which incorporates dynamic features to the network.

Lista de Contribuciones

Congresos

- S. Hori, **M. Zapata**, J. Madrenas, T. Morie, H. Tamukoh, "An implementation of spiking neural network using digital spiking silicon neuron model on a SIMD processor", 26th International Conference on Artificial Neural Networks. ICANN 2017, Alghero - Italia, Sep. 2017.
- **M. Zapata**, Jordi Madrenas, "Flexible spike delay controller for neural processing based on FPGA", 4th International Symposium on Brainware LSI, Research Institute of Electrical Communication, Tohoku University, Sendai-Japón, Feb. 2017.
- J. Madrenas, **M. Zapata**, H. Akima, S. Sato, "An Evolvable and Configurable SIMD Architecture for Spiking Neural Emulation",5th RIEC International Symposium on BRAIN Functions and Brain Computer, Research Institute of Electrical Communication, Tohoku University, Sendai-Japón, Feb. 2017
- **M. Zapata**, J. Madrenas, "Synfire Chain Emulation by Means of Flexible SNN Modeling on a SIMD Multicore Architecture", 25th International Conference on Artificial Neural Networks. ICANN 2016, Barcelona-España, Sep. 2016.
- **M. Zapata**, J. Madrenas, "Compact Associative Memory for AER Spike Decoding in FPGA-Based Evolvable SNN Emulation", 25th International Conference on Artificial Neural Networks. ICANN 2016, Barcelona-España, Sep. 2016.
- **M. Zapata**, J. Madrenas, "AER Spike Detection using Parameterized Associative Memory on BRAMs for SNN Hardware Implementations", 3rd International Symposium on Brainware LSI, Research Institute of Electrical Communication, Tohoku University, Sendai-Japón, Feb. 2016.
- **M. Zapata**, J. Madrenas, "Scalable Communication Model for Configurable Hardware Architectures of Large-Scale Spiking Neural Networks",2nd International Symposium on Brainware LSI, Research Institute of Electrical Communication, Tohoku University, Sendai- Japón, Mar. 2016.
- J. Madrenas, **M. Zapata**, G. Sánchez, "Advances on SIMD Architectures for Spiking Neural Emulators",2nd International Symposium on Brainware LSI, Research Institute of Electrical Communication, Tohoku University, Sendai-Japón, Feb. 2015.

Publicaciones

- T. Dorta, **M. Zapata**, J. Madrenas , "AER-SRT: Scalable spike distribution by means of synchronous serial ring topology address event representation", Neurocomputing, Vol 171, pp 1684-1690, https://doi.org/10.1016/j.neucom.2015.07.080, Jan 2016.

Capítulo 1

Estado del Arte

1.1. Introducción

El estudio del cerebro humano ha sido motivo de múltiples investigaciones con el fin de entender cómo trabaja y poder imitar su comportamiento. Sin embargo, hasta el día de hoy no es posible afirmar que se tiene un conocimiento exacto de cómo realiza sus funciones y entre ellas principalmente cómo logra aprender debido a su extrema complejidad [1] [2]. No obstante, la aplicación de ingeniería inversa reproduciendo la inteligencia humana, ha tenido como resultado una revolución de la ciencia en varios campos con un amplio abanico de aplicaciones como reconocimiento de voz, robótica, dispositivos inteligentes, navegación, visión, etc. [3].

En consecuencia, en la comunidad científica se mantiene un creciente interés por contar con plataformas eficientes que exhiban paralelismo masivo, eficiencia energética, escalabilidad con conectividad eficiente, plasticidad [4]; todas estas características presentes en el cerebro. Existen muchas y diversas aproximaciones reportadas en este campo que concentran sus esfuerzos en alcanzar un equilibrio entre realismo, escala, velocidad, etc., a un costo de pérdida de flexiblidad [5] en muchos de los casos.

En este contexto, la presente tesis aporta al estado del arte una arquitectura hardware de SNN de propósito general, diseñada para proporcionar al usuario un entorno flexible de investigación con un alto realismo biológico. Se distingue por su escalabilidad, reconfigurabilidad y programabilidad para describir modelos neuronales.

1.2. Neuronas Biológicas

El cortex está constituido por billones de neuronas interconectadas que se comunican por medio de impulsos eléctricos (spikes) que se transmiten a través de las sinapsis. Las neuronas están conformadas por tres partes funcionalmente distintas: dendritas, soma y axón (ver Fig. 1.1).

Las dendritas son equivalentes al canal de entrada de información que recoge señales de otras

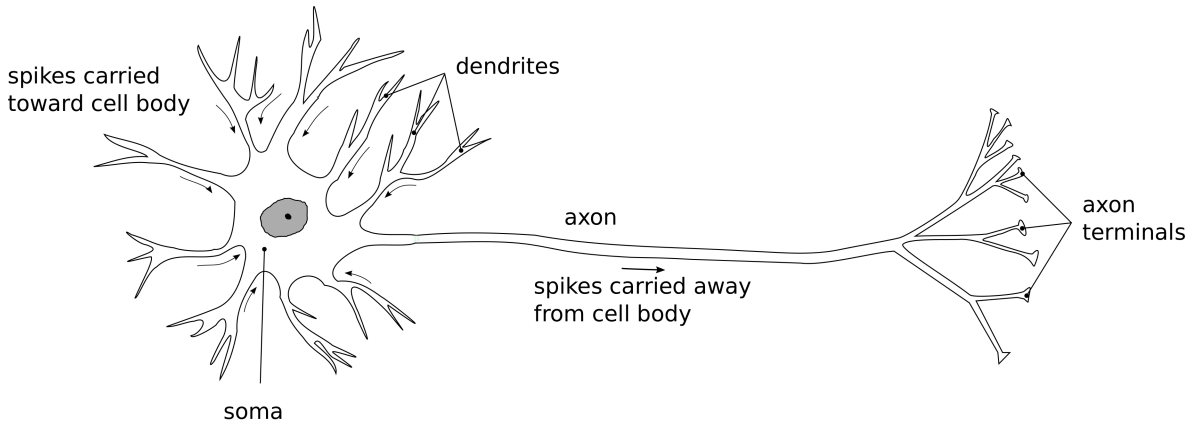


Figura 1.1: Estructura de una neurona

neuronas a través de las conexiones sinápticas y las transmite al soma. El soma es análogo a la unidad central de procesamiento encargada de controlar toda la actividad de la célula. El axón constituye el canal de salida que transmite los spikes generados hasta su otro extremo para conectarse vía sinapsis a las dendritas de las neuronas destino. La sinapsis es la región entre la ramificación axonal y las dendritas que comunica dos neuronas por medio de neurotransmisores. Estos se encargan de potenciar o inhibir la transmisión de spikes. A la neurona que libera neurotransmisores se la conoce como pre-sináptica y a la que los recibe se la denomina post-sináptica.

1.2.1. Redes Neuronales tipo Spiking

La gran complejidad que presenta el estudio de las neuronas biológicas y sus conexiones ha determinado que, conforme se va ampliando el conocimiento de su comportamiento y funcionalidad, vayan surgiendo nuevos modelos de Redes Neuronales Artificiales (RNA) con la intención de reproducir de forma más realista el sistema nervioso.

En los últimos años, un modelo aproximado que está siendo objeto de múltiples investigaciones son las Redes Neuronales tipo Spiking (SNN) consideradas como la tercera generación de las RNA. Estas redes se caracterizan por tener un alto realismo biológico y emplear información espacio-temporal en sus cálculos.

Las SNN han sido inspiradas a partir de un tren de potenciales de acción [6] generados por las células del sistema nervioso que se pueden analizar como señales temporales, cuya frecuencia y magnitud contiene información de procesamiento y percepción neuronal [7]. Una SNN modela dicho tren de eventos en función de un conjunto de pulsos temporales de entrada ocurridos en $\{{t_i}^1, {t_j}^1, ..., {t_k}^1\}$, donde, ${t_i}^n$ describen el instante en que el n-ésimo pulso llega a la i-ésima neurona como se observa en la Fig. 1.2. El pulso de salida *(spike)* o también denominado potencial de acción es determinado en función del patrón temporal de entrada, las conexiones sinápticas y sus pesos. Se ha demostrado matemáticamente que las SNN son computacionalmente más eficientes que los

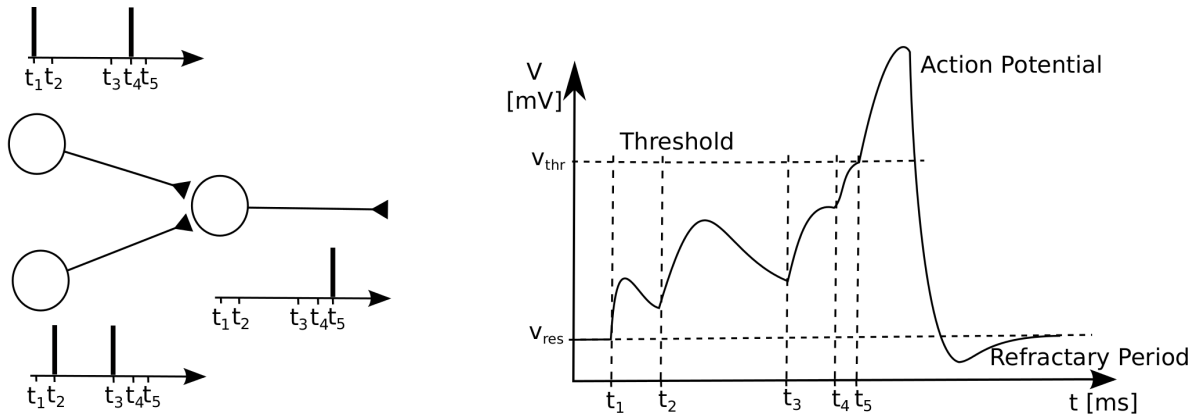


Figura 1.2: Neuronas tipo spiking - potencial postsináptico

modelos tradicionales basados en umbral [8]. Además, sus aplicaciones [9], [10], [11], entre otras, se distinguen por su bajo consumo de potencia y su rápida velocidad de convergencia. Para ello emplean diferentes técnicas de aprendizaje, con más o menos neuronas y algoritmos neuronales de mayor o menor plausibilidad biológica.

1.3. Redes Neuronales Modulares

Siendo el cerebro una red muy compleja (red de redes) se ha caracterizado acorde a propiedades topológicas de redes tales como: modularidad, jerarquía, centralización, localización de nodos, etc. Esto ha permitido establecer la relación entre la topología del cerebro y la dinámica de procesamiento neuronal [12] [13]. En este sentido, la configuración anatómica del cortex se distingue por su organización modular con patrones topológicos que abarcan conectividad inter-neural hasta inter-regional como se muestra en la Fig. 1.3. Su organización estructural presenta una densa conexión local, con relativamente escasa conectividad entre módulos o regiones. Existen neuronas con funciones especiales como las denominadas hub, que habilitan la conectividad inter e intra módulos. Estas proporcionan una contribución a la conectividad global entre regiones, que además deriva en una alta complejidad dinámica, así como actividad limitada en la red. Además, se ha reportado formaciones de nodos que exhiben un comportamiento centralizado, esto difiere del tradicional concepto de redes homogéneas y según [12] esto conlleva a decir que la organización del cerebro tiene una estructura jerárquica con funciones especializadas. Al respecto, Simon [14] planteó que un sistema complejo conformado por módulos escasamente interconectados permite una adaptación o evolución más rápida en respuesta a los cambios del entorno. Uno o varios módulos pueden evolucionar independientemente sin riesgo de perder las funciones en otros que se encuentran ya adaptados. Esta característica representa una de las principales ventajas de cualquier sistema dinámicamente adaptable.

Así también, se ha reportado que agrupaciones módulares a nivel biológico exhiben características

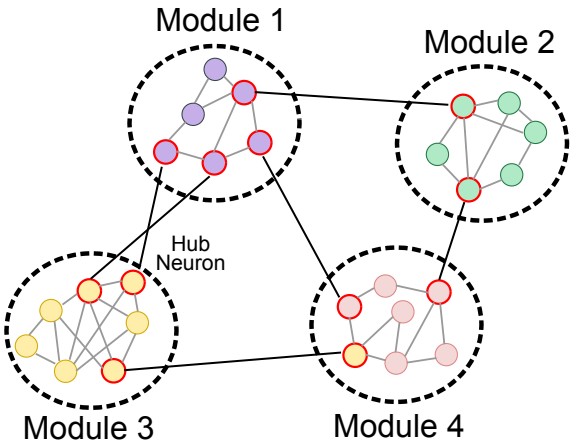


Figura 1.3: Los módulos exhiben un alto grado de conectividad local, y una relativamente baja conectividad intra-modular. Nodos Hubs (borde rojo) se conectan a nodos de otros módulos de la red.

de redes "small-world"(SW) [15]. Watts y Strogatz [16] definen las SW como la combinación de altas agrupaciones de neuronas con interconexiones de corta longitud entre ellas [17]. Se distinguen principalmente por permitir una conectividad entre cualquier par de nodos de la red en un número muy reducido de saltos que crece logarítmicamente con el número de nodos. La ventaja de este tipo de redes en el sistema nervioso es que la alta concentración de conexionado neuronal en un mismo módulo favorece el procesamiento local de funciones especializadas como la visión; mientras que el conexionado global permite el procesamiento de funciones más genéricas [18] facilitando la integración y la dispersión de las señales. No obstante este tipo de redes no explica la presencia de nodos altamente conectados cumpliendo la función de hubs. Al respecto, Barabasi y Albert [19] proponen el modelo de red "Scale-Free"(SF) que utiliza una función con un grado de distribución exponencial que justifica la presencia de pocos nodos altamente conectados y muchos con escasa conexión. Las redes SF son ampliamente utilizadas en la investigación de redes tanto a nivel biológico como tecnológico.

1.4. Evolución en Redes Neuronales

Una de las cualidades más importantes del cerebro humano es la capacidad de aprender y adaptarse al medio. Estos procesos han sido ampliamente estudiados, y aún cuando no han sido totalmente entendidos se han propuesto varios métodos de evolución en RNA [20] que reproducen en diferente grado un comportamiento inteligente. Las investigaciones realizadas tienen como principal objetivo obtener sistemas complejos que modifiquen parámetros y por ende funcionalidades de acuerdo a los estímulos del entorno.

En este contexto, la plasticidad es una de las características neuronales ampliamente estudiadas en Neurociencia, para entender la adaptación de las neuronas durante los procesos de aprendizaje [21]. Entre los métodos más utilizados en evolución están : cambio de parámetros entre ellos el peso

sináptico [22] [23] (el más común en los procesos de aprendizaje), velocidad de propagación de spikes entre neuronas, cambios estructurales (topológicos) [24] creando o eliminando redes o patrones de conexionado, etc.

Se han realizado aplicaciones que utilizan mecanismos como ”growing”, ”self-replication”, ”self-repair”, ”pruning”[25] [26], o algoritmos evolutivos [27] [24] para conseguir cambios topológicos en la red. Sin embargo, pueden llegar a representar un enorme costo computacional junto con un relativamente alto tiempo de convergencia dependiendo de la complejidad de los algoritmos de aprendizaje.

1.5. Emulación vs Simulación

El concepto de emulación y simulación a menudo tiende a manejarse como un sinónimo, sin embargo en el campo de las Ciencias de la Computación es importante distinguir la diferencia entre ambas terminologías.

Las simulaciones están dirigidas a las implementaciones por software, donde se abstrae ciertas propiedades del sistema para estudiar su comportamiento genérico [28]. En el caso que nos concierne, las simulaciones de SNN no persiguen tiempo real y, por lo general su aplicación se limita al análisis de modelos.

Por el contrario la emulación está estrictamente basada en hardware y, trata de capturar o replicar las propiedades biológicas de las SNNs persiguiendo tiempo real, considerando cada función y sus relaciones [28] [29].

1.6. Modelamiento de Conexionado Neuronal

La complejidad de la conectividad neuronal ha impedido hasta el momento que el progreso del hardware alcance SNN de escala biológica debido al número de conexiones y de neuronas requeridas. Una neurona típicamente presenta de 1,000 a 10,000 sinapsis [4]. Los dispositivos de silicio, a pesar del avance tecnológico alcanzado hoy en día, presentan un fan-in y fan-out limitado y dispersión en las señales; con lo cual conexiones dedicadas a todas las neuronas ubicadas a cualquier distancia en términos de una escala biológica, son inviables debido a la magnitud del problema y limitaciones tecnológicas. Estas a su vez se acentúa cuando se pretende emular la red neuronal de gran escala dentro de la ventana temporal típica de *1 ms* considerada como tiempo real [30]. Para solventar este problema, los sistemas digitales modernos explotan las técnicas de multiplexación digital para establecer la conectividad entre neuronas a cualquier distancia con máxima capacidad de transmisión de spikes por unidad de tiempo. En consecuencia, en una implementación SNN, mientras mayor sea el número de neuronas, el tráfico generado se incrementará en relación, haciendo cada vez más difícil obtener tiempo real sin pérdida de información. Además, a mayor número de neuronas y sinapsis, se necesitará mayor número de circuitos o funciones para la comunicación con lo cual, el consumo de potencia aumentará en proporción.

1.6.1. Representación de eventos de direcciones (AER)

Una de las aproximaciones más utilizadas para modelar la gran cantidad de interconexiones neuronales en plataformas bio-inspiradas tipo spiking, es el protocolo AER *"Address Event Representation"* propuesto por Mahowald y Sivilotti [31]. Se basa en la suposición de que la conectividad permanente entre neuronas no es necesaria para una correcta emulación, ésta solo se requiere cuando una neurona se dispara. Bajo este contexto, AER fue introducido como una alternativa eficiente de comunicación punto a punto (P2P) de eventos impulsionales (spikes) entre arreglos de neuronas. Un evento representa el código o dirección de la neurona que ha disparado. En la versión original, los eventos se multiplexan y son transmitidos por broadcast de forma asíncrona por un único bus compartido cuando una neurona dispara. No obstante, la mayor limitación de este enfoque se presenta cuando varias neuronas disparan al mismo tiempo, produciéndose una pérdida o degradación temporal de información como se ilustra en la Fig. 1.4(a).

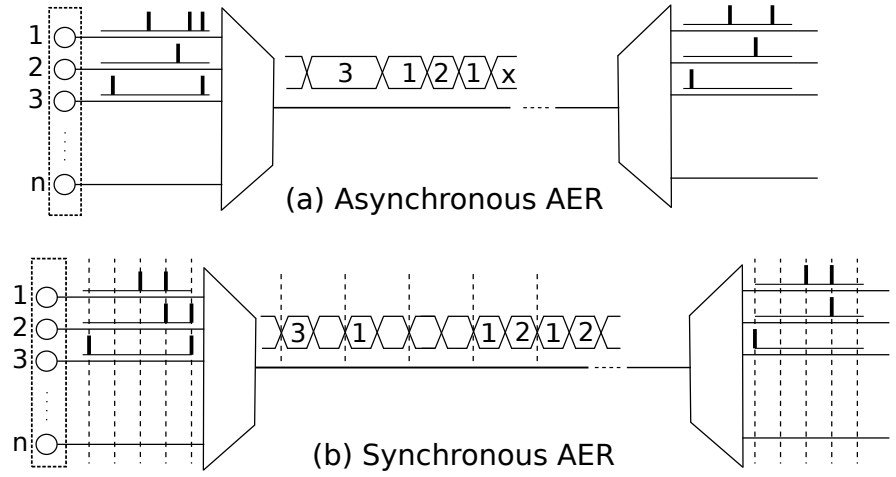


Figura 1.4: Representación de los fundamentos del bus AER

A partir de la versión original, [32] [33] proponen la emulación por ventanas temporales transmitidos por un bus síncrono AER (ver Fig. 1.4(b)). Desde este punto de vista, los eventos son asignados en ventanas de tiempo que posteriormente se multiplexan antes de ser transmitidos por el bus AER. Esta técnica asegura una transmisión por el canal libre de colisiones, no obstante la resolución se limita por el ancho de la franja temporal. Siempre que este ancho de banda sea suficiente para la dinámica neuronal requerida, y que el número de spikes pueda acomodarse dentro del intervalo temporal, el sistema de multiplexado será suficiente.

1.7. Modelos Neuronales SNN

Se ha desarrollado una amplia variedad de modelos que describen el comportamiento neuronal a diferentes niveles de abstracción. La elección de uno de ellos dependerá del tipo de fenómeno

neurológico a simular/emular, así como de los parámetros requeridos dependiendo de la aplicación a implementar. Estos generalmente son descritos a través de ecuaciones diferenciales ordinarias. Entre los más utilizados en la implementación de SNN tenemos los descritos a continuación.

1.7.1. Leaky Integrate and Fire (LIF)

LIF es uno los modelos neuronales más utilizados en la realización hardware de SNN. Este captura las propiedades más importantes de una neurona cortical con un bajo coste computacional [30]. El circuito eléctrico equivalente se basa en una red RC en paralelo, que exhibe un decaimiento de la carga una vez que la neurona dispara el spike [34]. Como se reporta en [30], está definido por la siguiente ecuación:

$$v' = I + a - bv \qquad \text{Si: } v \geq v_{thr} \quad \text{Entonces: } v \longleftarrow v_{res} \tag{1.1}$$

Donde cada variable representa:

v: Potencial de membrana
v': Derivada temporal de v
I: Corriente de entrada
v_{thr}: Voltaje de threshold
v_{res}: Voltaje de restitución
a, b: Parámetros dinámicos

Un spike se dispara cuando el potencial de membrana alcanza el voltaje de threshold. A continuación v se resetea al valor del voltaje de restitución.

1.7.2. Izhikevich (IZ)

Es un modelo sencillo y eficiente que tiene como principal ventaja la capacidad de reproducir 20 diferentes patrones de disparo observados en las neuronas biológicas [30]. Utiliza dos ecuaciones diferenciales:

$$v' = 0{,}004v^2 + 5v + 140 - u + I \tag{1.2}$$

$$u' = a(bv - u) \tag{1.3}$$

Donde:

- v es el potencial de membrana.
- u es el mecanismo de recuperación de membrana.
- I es la corriente sináptica, es decir la contribución de las neuronas post-sinápticas.

Cuando la tensión de membrana supera el voltaje de threshold, las variables u y v se resetean a los siguientes valores:

$$\text{Si: } v \geq v_{thr}, \qquad \text{entonces } \begin{cases} v \longleftarrow c \\ u \longleftarrow u + d \end{cases} \tag{1.4}$$

Donde *a, b, c, d* son parámetros adimensionales utilizados para ajustar el comportamiento del algoritmo para obtener diferentes patrones de disparo.

1.7.3. Hodgkin Huxley

Es uno de los modelos más importantes en el ámbito de la Neurociencia computacional ya que está basado en parámetros biológicos reales que describen el comportamiento de la conductancia iónica y sináptica para cada neurona. Está descrito por las siguientes 4 ecuaciones diferenciales que en conjunto emulan las principales características del potencial de acción [35] [30].

$$C\frac{dv}{dt} = I_{Na} + I_K + I_L + I_{ext} \tag{1.5}$$

$$m\frac{dm}{dt} = -m + m_\infty \tag{1.6}$$

$$n\frac{dn}{dt} = -n + n_\infty \tag{1.7}$$

$$h\frac{dh}{dt} = -h + h_\infty \tag{1.8}$$

Donde:

v: Potencial de membrana
I_{ext}: Corriente externa
I_{Na}: Corriente de sodio
I_K: Corriente de potasio
I_L: Corriente de fuga
m: Variable de puerta de activación del sodio
h: Variable de puerta de inactividad del sodio
n: Variable de puerta de activación del potasio

Éste modelo es importante no solo porque sus parámetros son biofísicamente significativos y medibles, sino también, porque permiten investigar efectos y relaciones biológicas relacionadas con la dinámica neuronal. Además, se caracteriza por el alto costo computacional que requiere, dificultando la implementación en hardware.

Existen otros modelos que reproducen características biológicas, como por ejemplo FitzHugh Nagumo [36], Hindmarsh-Rosee [30] [37], Morris Lecar [38] etc. La decisión de emplear uno u otro modelo, conlleva a un compromiso entre la eficiencia computacional y plausibilidad biológica. Para estudios que requieran una descripción precisa de la biofísica real de la neurona, el Hodgkin Huxley sería la mejor opción. Sin embargo, el alto costo computacional limita la cantidad de neuronas, especialmente si se quiere obtener simulaciones/emulaciones en tiempo real. Por el contrario, para emular desde cientos hasta miles de neuronas se podría emplear modelos de costo computacional más bajo, siendo el LIF el más eficiente no obstante, este exhibe las propiedades más básicas de las neuronas.

Tabla 1.1: Resumen de las Arquitecturas hardware SNN más relevantes en el Estado del Arte.

Plataforma	Dispositivo	Neuronas/ Core	# Cores	Modelo	Sinapsis/ Neurona	Resolución	Topol.	Conectiv.
FACET (2007)	ASIC*	510	352	AEIF	224	1 us	Malla	CC
Cassidy (2008)	FPGA	256	1	IZ	128	200 ns	-	CC
Thomas y Luk (2009)	FPGA	1,024	1	IZ	1,024	10 us	-	CC
SpiNNaker (2013)	ARM	100	18	PGen	1,000	1 ms	Toroidal	CC
Neurogrid (2014)	ASIC*	65k	16x16	QIF	7,980	0,1 ms	Bus Árbol	Aleat
TrueNorth (2014)	ASIC*	256	64x64	LIF	256	1 ms	Malla	Aleat.
IFAT-HiAER (2016)	ASIC*	2,400	4	LIF	436	1 ms	Arbol	CC
NeuroFlow (2016)	FPGA	98,304	6	PGen	1,000-10,000	1 ms	Toroidal	Aleat.
Pani (2017)	FPGA	1,440	1	IZ	256	0,1 ms	-	CC

AIF: Adaptative Exponential Integrate and Fire. QIF: Quadratic Integrate and Fire.
PGen: Arquitecturas de propósito general. ASIC*: Plataformas de señal mixta.
CC: Conectividad completa (el origen se puede conectar a cualquier nodo en la red).
Aleat. La conectividad se basa en una función de distribución de probabilidad.

1.8. Trabajos Relacionados

Debido al creciente y activo interés por desarrollar modelos SNN con un alto grado de plausibilidad biológica, se han desarrollado varias plataformas computacionales basadas en software y hardware que cubren el ámbito analógico, digital y se señal mixta.A continuación se presenta algunos de los trabajos más representativos en el estado del arte.

Los sistemas basados en software se distinguen por soportar varios grados de programabilidad, así como un gran número de neuronas y sinapsis ya que utilizan los recursos de memoria y procesamiento del sistema computacional. Sin embargo, los procesadores convencionales de un solo núcleo están limitados por un comportamiento secuencial y por la cantidad de memoria. Simuladores como BRIAN [39], NEST [40], NEURON [41] son ampliamente usados como herramientas de investigación en el campo de la Neurociencia para pocas neuronas. No obstante, para modelos neuronales biológicamente plausibles en redes de gran escala, presentan problemas de escalabilidad relacionada estrechamente con el objeto de la simulación, lo cual deriva en tiempos de ejecución demasiado lentos [42].

Como alternativa a esta limitación se encuentran las aproximaciones que emplean Graphic Processing Unit(GPU) como las reportadas por Fidjeland [43], Richert [2], etc. En términos generales, presentan un alto grado de programabilidad y flexibilidad ya que permiten una amplia variedad de modelos neuronales, sinápticos y de plasticidad. Dentro de este ámbito también se encuentra el

Blue Brain Project [44], desarrollado para estudiar regiones del cerebro, en el que se emplea una supercomputadora (Blue Gene de IBM) conformado por 8192 CPUs y 32 TB de memoria jerárquica, capaz de emular 10,000 neuronas y algoritmos como Hodgking-Huxley de alto grado de plausibilidad. Sin embargo, su velocidad está lejos de alcanzar las fronteras de tiempo real, así también su ingente consumo de potencia en el orden de 100,000 veces más que el del cerebro humano son sus principales desventajas.

En el otro extremo, se encuentran las aproximaciones por hardware. Para redes SNN de gran escala de SNN ofrecen tiempos de ejecución entre 2 y hasta 3 órdenes de magnitud más rápidos que las aproximaciones de software debido al inherente paralelismo tanto en sistemas analógicos como digitales [45]. En la Tabla 1.1 se presenta un resumen de características de las arquitecturas SNN hardware más relevantes en el estado del arte, categorizadas de acuerdo a la tecnología y densidad de neuronas y sinapsis que soportan, así como, por el modelo de comunicación que utilizan para la distribución de spikes en la red.

Las basados en ASIC (Circuitos Integrados de Aplicación Específica), se caracterizan por presentar la más alta velocidad y bajo consumo entre el resto de aproximaciones de procesamiento neuronal. Sin embargo, los modelos spiking son fijos y las conexiones sinápticas en muchos de los casos no presentan plasticidad, y limitan los patrones de conexión que pueden implementar como en [46]. La falta de flexibilidad, programabilidad y alto coste de manufactura son sus principales inconvenientes.

Existen configuraciones híbridas que combinan ASIC con circuitos digitales para lograr una infraestructura de comunicación flexible como es el caso de FACETS [47] [48]. FACET es un dispositivo de señal mixta configurable que alcanza densidades de 10^6 neuronas analógicas. La comunicación se implementa mediante buses jerárquicos para conectividad neuronal interna a una tasa de transmisión de 500 Mevent/s, y routers implementados sobre FPGAs para enlaces externos inter-oblea de 1.28 Gevent/s. Las altas tasas de transmisión se obtienen por la alta frecuencia de reloj utilizada (500 Mhz), y los 4 transceivers de 10 Gbps sobre varios enlaces paralelos. Presenta una penalidad en el consumo de potencia de 1 kW por cada oblea.

Con este mismo enfoque, Neurogrid [49] utiliza 16x16 cores integrados en una placa. El arreglo de neuronas en silicio está interconectado en una red con topología en bus y en árbol mediante múltiples buses AER P2P a 234 Mevent/s. No permite plasticidad sináptica debido a restricciones en su modelo de comunicación que limita su flexibilidad.

Así también, SpiNNaker [42] una de las plataformas más representativas en la emulación de las SNN, es una arquitectura escalable multimodelo diseñada para SNN de gran escala. Basa su procesamiento neuronal en eventos y trabaja en tiempo real. Está compuesta por una malla hexagonal de procesadores ARM de propósito general. Cada chip o nodo SpiNNaker contiene hasta 18 procesadores ARM con 128 MB de memoria compartida. A su vez, 48 nodos están dispuestos en una PCB. Las PCB se comunican a través de enlaces seriales de alta velocidad a una tasa de 1 Gevent/s. Utiliza NoC (Network on Chip) para la comunicación multi-cast que soporta la conectividad neuronal. Además, dado que emplea un esquema de comunicación asíncrono los eventos de dirección llegan al destino de forma no determinista, y dependiendo de la actividad de la red se puede llegar a perder paquetes para evitar la saturación del canal.

En otro contexto de implementación que ha ganado mucho interés entre la comunidad científica, tenemos las basadas en FPGAs, mismas que se benefician de las características de paralelismo, reconfigurabilidad, bajo costo y en especial la flexibilidad para realizar una rápida actualización o modificación del diseño original. Tienen los beneficios de las plataformas a medida (como las ASIC) y la flexibilidad de las soluciones de software [50]. Además la presencia de IP cores permite optimizar y acelerar la descripción de hardware.

En esta categoría se encuentra NeuroFlow [51], una plataforma SNN escalable compuesta por 6 FPGAs. Cada implementación debe ser rediseñada y resintetizada con la penalidad de un tiempo muy alto de síntesis ligado al tamaño de la red y la complejidad del algoritmo. Se reportan tiempos de 17 h a 20 h para una red de 589k neuronas Izhikevich. Usa PyNN [52] (lenguaje genérico basado en Python de descripción de redes neuronales) para configurar el procesador. Soporta un número muy alto de neuronales y sinapsis dentro de la escala de tiempo real. Emplea una tasa de transmisión de eventos mayor a 1 Gevent/s. La conectividad sináptica está dada en función de la distancia entre nodos y obedece a una función de distribución Gausiana.

Así tambien, otras plataformas como las reportados por Cassidy [53] [54], Thomas y Luk [55], Pani [5] ofrecen gran flexibilidad a cambio de optimizar su diseño a una clase específica de neurona como se observa en la tabla referida anteriormente.

Finalmente, en arquitecturas como Neurogrid, TrueNorth [56], IFAT [57] [58], se han desarrollado paralelamente modelos de comunicación para enfrentar las limitaciones de ancho de banda del canal impuesto por el tráfico de los eventos de spike en redes de gran escala, factor que restringe el tamaño de la red. Sus propuestas ofrecen diferentes alternativas de enrutamiento de spikes utilizando diferentes topologías, así como diferente grado de flexibilidad para establecer la conexión de las direcciones de origen pre-sinápticas con las de destino post-sinápticas.

1.9. Objetivos y Estructura de la Tesis

Dentro del campo de la Neurociencia, se han reportado varias aproximaciones de sistemas de simulación/emulación de SNN. No obstante, logran alcanzar buenas prestaciones ha sido a un costo de pérdida de flexibilidad. El presente trabajo de tesis pretende contribuir al estado del arte con la presentación de una nueva alternativa que aborda dos de los principales desafíos en la emulación de las redes neuronales la escalabilidad y la conectividad sobre una plataforma que proporciona un alto grado de flexibilidad para el usuario. Para enfrentar estos retos, el desarrollo de esta tesis se ha dividido en dos etapas:

- La optimización de la arquitectura SNAVA, punto de partida de esta investigación para mejorar sus características de procesamiento neuronal y establecer la interoperabilidad con el modelo de comunicación y sus funciones.

- El diseño e implementación de un modelo eficiente de comunicación jerárquico escalable, en una plataforma multi-chip con soporte de reconfiguración dinámica.

Aunque se han abarcado otros tópicos, los objetivos mencionados son considerados los más relevantes en esta tesis. A continuación se presentan los capítulos que conforman esta tesis:

Capítulo 2: Presenta un resumen de las principales características de la investigación previamente reportada para el desarrollo de SNAVA. Posteriormente, como primera contribución, se describe los resultados de una aplicación de Cadena Synfire, realizada para comprobar la funcionalidad y estimar las limitaciones de SNAVA.

Capítulo 3: En este capítulo se detalla la segunda contribución de esta tesis, con una nueva arquitectura denominada HEENS conformada por dos bloques principales: Chip Neuromórfico y el Controlador AER-SRT. El primero es cubierto en este capítulo con la explicación de su diseño y las nuevas características añadidas en relación con su predecesora SNAVA.

Capítulo 4: Aquí se expone la tercera contribución de la tesis respecto al modelo de comunicación jerárquico de HEENS. Se explica en detalle el diseño que da soporte a la conectividad jerárquica de la la red, así como los modos de programabilidad y reconfiguración que habilitan la evolución neuronal de los chip neuromórficos.

Capítulo 5: En este capítulo se presentan los resultados de una aplicación a modo de test de verificación sobre HEENS. Se demuestra la funcionalidad de la arquitectura y su flexibilidad para implementar en hardware aplicaciones de redes neuronales tipo spiking. A continuación se presenta la caracterización de la arquitectura completa y el análisis de escalabilidad y consumo de potencia.

Capítulo 6: Se presentan las conclusiones de la tesis y lineas futuras de trabajo. Se resumen las contribuciones al estado del arte realizadas con esta investigación hacia las arquitecturas SNN hardware basadas en FPGA.

Referencias

[1] O. Sporns, "Networks of the brain", *Cambridge, MA: The MIT Press*, 2011.

[2] M. Richert, J. M. Nageswaran, N. Dutt y J. L. Krichmar, "An Efficient Simulation Environment for Modeling Large-Scale Cortical Processing", *Frontiers in Neuroinformatics*, vol. 5, n.º September, págs. 1-15, 2011, ISSN: 1662-5196. DOI: 10.3389/fninf.2011.00019. dirección: http://journal.frontiersin.org/article/10.3389/fninf.2011.00019/abstract.

[3] A. Pannu y M. T. Student, "Artificial Intelligence and its Application in Different Areas", *Certified International Journal of Engineering and Innovative Technology*, vol. 9001, n.º 10, págs. 2277-3754, 2008, ISSN: 1687-7047. DOI: 10.1155/2009/251652.

[4] R. Cattell y A. Parker, "Challenges for brain emulation: why is building a brain so difficult?", *Natural Intelligence*, feb. de 2012.

[5] D. Pani, P. Meloni, G. Tuveri, F. Palumbo, P. Massobrio y L. Raffo, "An FPGA Platform for Real-Time Simulation of Spiking Neuronal Networks", *Frontiers in Neuroscience*, vol. 11, n.º February, 2017, ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00090.

[6] S. Ghosh-Dastidar y H. Adeli, "Third generation neural networks: spiking neural networks", en *Advances in Computational Intelligence*, W. Yu y E. N. Sanchez, eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, págs. 167-178, ISBN: 978-3-642-03156-4. DOI: 10.1007/978-3-642-03156-4_17. dirección: http://dx.doi.org/10.1007/978-3-642-03156-4_17.

[7] J. Ramírez y M. Chacón, "Redes neuronales artificiales para el procesamiento de imágenes, una revisión de la última década", *RIEE&C*, vol. 9, jul. de 2011, ISSN: ISSN 1870 9532.

[8] W. Maass, "Networks of spiking neurons: The third generation of neural network models", *Neural Networks*, 1997.

[9] S. M. Bohte y J. N. Kok, "Applications of spiking neural networks", *Information Processing Letters*, vol. 95, n.º 6 SPEC. ISS. Págs. 519-520, 2005, ISSN: 00200190. DOI: 10.1016/j.ipl.2005.05.018.

[10] I. Paz y H. Gress, "Pattern recognition with spiking neural networks", en *Advances in Soft Computing and Its Applications: 12th Mexican International Conference on Artificial Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, págs. 279-288, ISBN: 978-3-642-45111-9. DOI: 10.1007/978-3-642-45111-9_25.

[11] B. Meftah y A. Benyettou, "Image Segmentation with Spiking Neuron Network", págs. 15-19, 2008.

[12] E. T. Bullmore y O. Sporns, "Complex brain networks: graph theoretical analysis of structural and functional systems.", *Nature reviews. Neuroscience*, vol. 10, n.º 3, págs. 186-198, 2009, ISSN: 1471-0048. DOI: 10.1038/nrn2575. dirección: http://www.ncbi.nlm.nih.gov/pubmed/19190637.

[13] M. Newman, "Modularity and community structure in networks", *Proceedings of the National Academy of Sciences*, vol. 103, n.º 23, pág. 8577, 2006, ISSN: 0027-8424. DOI: 10.1073/pnas.0601602103. arXiv: 0602124 [physics]. dirección: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1482622%7B%5C&%7Dtool=pmcentrez%7B%5C&%7Drendertype=abstract$%5Cbackslash$nhttp://www.pnas.org/content/103/23/8577.short.

[14] H. Simon, "The architecture of complexity", *Proc. Am. Philos. Soc. 106*, págs. 467-482, 1962.

[15] D. Samu, A. K. Seth y T. Nowotny, "Influence of Wiring Cost on the Large-Scale Architecture of Human Cortical Connectivity", *PLoS Computational Biology*, vol. 10, n.º 4, 2014, ISSN: 15537358. DOI: 10.1371/journal.pcbi.1003557.

[16] D. Watts y S. Strogatz, "Collective dtynamics of small world networks", *Nature 393(684)*, 1998.

[17] D. Meunier, R. Lambiotte y E. T. Bullmore, "Modular and hierarchically modular organization of brain networks", *Frontiers in Neuroscience*, vol. 4, n.º DEC, págs. 1-11, 2010, ISSN: 16624548. DOI: 10.3389/fnins.2010.00200.

[18] D. Sporns O. Chialvo y et.al., "Organization development and function of complex rain networks", *Trends Cogn. Sci. 8*, págs. 418-425, 2004.

[19] A. Barabasi y R. Albert, "Emergence of scaling in random networks", *Science 286)*, 1999.

[20] X. Yao, "Evolving artificial neural networks", *Proceedings of the IEEE 87(9)*, 1999.

[21] C. G. Antonopoulos, S. Srivastava, S. E. d. S. Pinto y M. S. Baptista, "Do Brain Networks Evolve by Maximizing Their Information Flow Capacity?", *PLoS Computational Biology*, vol. 11, n.º 8, págs. 1-29, 2015, ISSN: 15537358. DOI: 10.1371/journal.pcbi.1004372. arXiv: 1508.03527.

[22] N. García-Pedrajas y C. Hervás-Martínez, "Multi-objective cooperative coevolution of artificial neural networks (multi-objective cooperative networks)", *Neural Networks*, vol. 15, n.º 10, págs. 1259-1278, 2002, ISSN: 0893-6080. DOI: http://dx.doi.org/10.1016/S0893-6080(02)00095-3.

[23] A. Upegui, C. A. Peña-Reyes y E. Sanchez, "An FPGA platform for on-line topology exploration of spiking neural networks", *Microprocessors and Microsystems*, vol. 29, n.º 5, págs. 211-223, 2005, ISSN: 01419331. DOI: 10.1016/j.micpro.2004.08.012.

[24] A. Vandesompele, F. Walter y R. Florian, "Neuro-Evolution of Spiking Neural Networks on SpiNNaker Neuromorphic Hardware", 2016.

[25] H. Shayani, P. Bentley y a.M. Tyrrell, "Hardware Implementation of a Bio-plausible Neuron Model for Evolution and Growth of Spiking Neural Networks on FPGA", *2008 NASA/ESA Conference on Adaptive Hardware and Systems*, págs. 236-243, 2008. DOI: 10.1109/AHS.2008.13. dirección: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4584279.

[26] J. Wang, A. Belatreche, L. Maguire y T. M. McGinnity, "Dynamically Evolving Spiking Neural network for pattern recognition", *Proceedings of the International Joint Conference on Neural Networks*, vol. 2015-September, 2015. DOI: 10.1109/IJCNN.2015.7280649.

[27] S. Soltic, S. G. Wysoski y N. K. Kasabov, "Evolving spiking neural networks for taste recognition", *Proceedings of the International Joint Conference on Neural Networks*, págs. 2091-2097, 2008, ISSN: 1098-7576. DOI: 10.1109/IJCNN.2008.4634085.

[28] E. Gordon, *Integrative Neuroscience: Bringing Together Biological, Psychological and Clinical Models of the Human Brain*. Taylor & Francis, 2003, ISBN: 9780203304761. dirección: https://books.google.es/books?id=3MpFw214Yc4C.

[29] A. Sandberg y N. Bostrom, "Whole brain emulation: a roadmap", *Technical Report*, vol. #2008-2003, Oxford, UK: Future of Humanity Institute, Oxford U, 2008.

[30] E. M. Izhikevich, "Which model to use for cortical spiking neurons?", *IEEE Transactions on Neural Networks*, vol. 15, n.º 5, sep. de 2004.

[31] M. Mahowald, "VLSI Analogs of Neuronal Visual Processing: What does the retina know about a Synthesis of Form and Function", Tesis doct., California Institute of Technology y Pasadena, 1992.

[32] J. Madrenas y J. M. Moreno, "Strategies in SIMD computing for complex neural bioinspired applications", *Proceedings - 2009 NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2009*, págs. 376-381, 2009. DOI: 10.1109/AHS.2009.31.

[33] J. Moreno, J. Madrenas y L. Kotynia, "Synchronous digital implementation of AER communication scheme for emulating large-scale spiking neural networks models", *NASA-ESA Conference on Adaptive Hardware and Systems, AHS 2009*, págs. 186-196, 2009.

[34] P. Dayan y L. F. Abbott, "Computational and mathematical modeling of neural system", *Theoretical Neuroscience*, vol. 1st Edition, dic. de 2001.

[35] A. L. Hodgkin y A. F. Huxley, "A quantitative description of membrane current and application to conduction and excitation in nerve", *J. Physiol*, vol. 117, dic. de 1954.

[36] R. FitzHugh, "Impulses and physiological states in models of nerve membrane", *Biophys. J.*, vol. 1, 1961.

[37] R. M. Rose y J. L. Hindmarsh, "The assembly of ionic currents in a thalamic neuron. the three-dimensional model", *Proc. R. Soc. Lond.*, 1989.

[38] H. L. Morris, "Voltage oscillations in the barnacle giant muscle fiber", *Biophysical Journal*, vol. 35, 1981.

[39] D. Goodman y R. B. Brette, "BRIAN: a simulator for spiking neural networks in Python", *Front. Neuroinform*, 2008. DOI: 10.3389/neuro.11.005.2008.

[40] M.-O. Gewaltig y M. Diesmann, "Nest (neural simulation tool)", *Scholarpedia*, vol. 2, n.º 4, pág. 1430, 2007.

[41] N. Carnevale y M. Hines, *The NEURON Book. Cambridge: Cambridge University Press*, 2006. dirección: http://www.neuron.yale.edu/neuron/docs.

[42] A. D. Rast, X. Jin, F. Galluppi, L. A. Plana, C. Patterson y S. Furber, "Scalable event-driven native parallel processing: the spinnaker neuromimetic system", CF '10, págs. 21-30, 2010. DOI: 10.1145/1787275.1787279. dirección: http://doi.acm.org/10.1145/1787275.1787279.

[43] a.K. Fidjeland y M. Shanahan, "Accelerated simulation of spiking neural networks using GPUs", *Neural Networks (IJCNN), The 2010 International Joint Conference on*, 2010.

[44] H. Markram, "The blue brain project", *Nature Rev Neuroscience*, vol. 7, 2006. DOI: http://dx.doi.org/10.1038/nrn1848.

[45] S. Cawley, F. Morgan y col., "Hardware spiking neural network prototyping and application", *Genet Program Evolvable Mach*, 2011.

[46] A. Joubert, B. Belhadj, O. Temam y R. Heliot, "Hardware spiking neurons design: Analog or digital?", *The 2012 International Joint Conference on Neural Networks (IJCNN)*, págs. 1-5, 2012, ISSN: 2161-4393. DOI: 10.1109/IJCNN.2012.6252600. dirección: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6252600.

[47] M. Ehrlich, C. Mayr, H. Eisenreich, S. Henker, A. Srowig, A. Grübl, J. Schemmel y R. Schüffny, "Wafer-scale {VLSI} implementations of pulse coupled neural networks", *Proceedings of 4th IEEE International Multi-Conference on Systems, Signals & Devices SSD07, electronic publication, Abstract on page 409 of Proc.*, 2007.

[48] J. Schemmel, J. Fieres y K. Meier, "Wafer-scale integration of analog neural networks", *Proceedings of the International Joint Conference on Neural Networks*, págs. 431-438, 2008, ISSN: 1098-7576. DOI: 10.1109/IJCNN.2008.4633828.

[49] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla y K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations", *Proceedings of the IEEE*, vol. 102, n.º 5, págs. 699-716, 2014, ISSN: 00189219. DOI: 10.1109/JPROC.2014.2313565.

[50] B. Glackin, T. M. McGinnity, L. P. Maguire, Q. X. Wu y A. Belatreche, "A novel approach for the implementation of large scale spiking neural networks on FPGA hardware", *Proceedings of IWANN 2005 computational intelligence and bioinspired systems*, págs. 552-563, 2005.

[51] K. Cheung, S. R. Schultz y W. Luk, "NeuroFlow: A general purpose spiking neural network simulation platform using customizable processors", *Frontiers in Neuroscience*, vol. 9, págs. 1-15, 2016, ISSN: 1662453X. DOI: 10.3389/fnins.2015.00516.

[52] A. Davison, D. BrÃ¼derle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet y P. Yger, "PyNN: a common interface for neuronal network simulators", *Frontiers in Neuroinformatics*, vol. 2, pág. 11, 2009, ISSN: 1662-5196. DOI: 10.3389/neuro.11.011.2008. dirección: http://journal.frontiersin.org/article/10.3389/neuro.11.011.2008.

[53] A. Cassidy, S. Denham, P. Kanold y A. Andreou, "FPGA Based Silicon Spiking Neural Array", *2007 IEEE Biomedical Circuits and Systems Conference*, n.º 1, págs. 75-78, 2007, ISSN: 1424415241. DOI: 10.1109/BIOCAS.2007.4463312.

[54] A. Cassidy y A. G. Andreou, "Dynamical digital silicon neurons", *2008 IEEE-BIOCAS Biomedical Circuits and Systems Conference, BIOCAS 2008*, págs. 289-292, 2008. DOI: 10.1109/BIOCAS.2008.4696931.

[55] D. B. Thomas y W. Luk, "FPGA accelerated simulation of biologically plausible spiking neural networks", *Proceedings - IEEE Symposium on Field Programmable Custom Computing Machines, FCCM 2009*, págs. 45-52, 2009. DOI: 10.1109/FCCM.2009.46.

[56] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar y D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface", *Science*, vol. 345, n.º 6197, págs. 668-673, 2014, ISSN: 0036-8075. DOI: 10.1126/science.1254642.

[57] J. Park, T. Yu, S. Joshi, C. Maier y G. Cauwenberghs, "Hierarchical Address Event Routing for Reconfigurable Large-Scale Neuromorphic Systems", *IEEE Transactions on Neural Networks and Learning Systems*, págs. 1-15, 2016, ISSN: 21622388. DOI: 10.1109/TNNLS.2016.2572164.

[58] J. Vogelstein, U. Mallik y col., "Dynamically Reconfigurable Silicon Array of Spiking Neurons With Conductance Based Synapses", *IEEE Transactions on Neural Network*, 2007.

Capítulo 2

Mapeado de Redes Neuronales Spiking sobre SNAVA

2.1. Introducción

La implementación de arquitecturas hardware SNN ha sido abordada bajo diferentes enfoques. Entre ellos, SNAVA (Spiking Neural Network Architecture for Versatile Applications) [1] ha sido propuesta como un contribución significativa al estado del arte. SNAVA es una arquitectura configurable bio-inspirada para emular SNN de gran escala, la misma que constituye el punto de partida de esta tesis. Este capítulo, primero resume sus principales características y a continuación se presenta los resultados de una aplicación de Cadena Synfire (SFC); realizada para estudiar su desempeño y analizar sus limitaciones.

2.2. Arquitectura SNAVA

En este apartado se presenta la arquitectura SNAVA, punto de partida de esta tesis. Por ello es necesario explicar sus detalles con el fin de facilitar la comprensión de las contribuciones de la tesis.

2.2.1. Breve descripción de la arquitectura SNAVA

La architectura SNAVA es un plataforma compacta y multi-modelo con características bio-inspiradas que permite la emulación de SNN y la implementación de aplicaciones neuronales en hardware reconfigurable y de procesamiento paralelo. Está constituida por un arreglo de Elementos Procesasdores (Processing Element, PE) ha sido implementada sobre FPGA, aunque se podría migrar fácilmente hacia un ASIC. Es una arquitectura síncrona con un enfoque computacional basado en intervalos temporales de forma que cada ciclo de emulación neuronal está dividido en dos fases: ejecución y distribución.

- En la fase de ejecución se procesa y calcula la dinámica sináptica-neuronal en los PE. Los cálculos sinápticos se realizan multiplexados en el tiempo, mientras que las neuronas se calculan en paralelo. Se soporta la posibilidad de multiplexar también las neuronas (virtualización de neuronas). Los potenciales de acción son representados por eventos de dirección (Address Event Representation, AER). Estos se definen por la posición de fila y columna del PE donde se generó el spike, así como de la capa virtual de la neurona que emula el PE, así como el identificador de chip (ID).
- En la fase de distribución, los spikes post-sinápticos (eventos) generados son propagados a toda la red neuronal.

Los módulos que conforman la arquitectura se muestran en la Fig. 2.1. El arreglo fijo de 10x10 PEs utiliza un esquema de cómputo del tipo Single Instruction Multiple Data (SIMD) con una única unidad de control correspondiente al Módulo de Ejecución (Execution Module). La razón de emplear una aproximación SIMD es el importante ahorro de recursos y memoria de programa, ya que el algoritmo para todos los PEs es el mismo. Ello permite obtener sistemas mucho más eficientes que empleando arquitecturas "Multiple Instruction, Multiple Data"(MIMD). El módulo de ejecución se encarga de buscar, decodificar y transmitir las mismas instrucciones para todos los PE, que cada uno ejecuta, pero con su propio conjunto de datos. Además, esta unidad gestiona las señales de control para sincronizar las operaciones del resto de módulos del sistema [2].

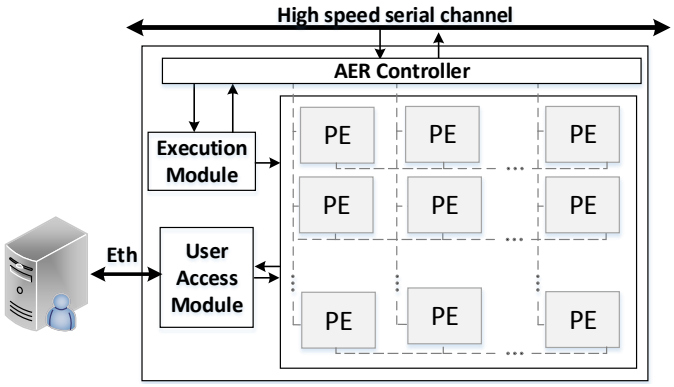


Figura 2.1: Diagrama de Bloques de la Arquitectura SNAVA [2].

El PE realiza el cálculo matemático de los algoritmos neuronales y sinápticos por medio de un banco de registros y una ALU de 16 bits que ejecuta operaciones lógicas y aritméticas de punto fijo, incluyendo un multiplicador paralelo-paralelo. En el set de instrucciones en lenguaje ensamblador, además de las operaciones lógicas y aritméticas, también incluye instrucciones de control de flujo para administrar las fases de procesamiento, almacenamiento de datos y monitoreo de la aplicación.

Tal como se ha indicado, los PEs tienen la capacidad de emular más de una neurona a la vez, multiplexando en el tiempo su procesamiento (virtualización) sin necesidad de consumir recursos extra de hardware a cambio de un incremento en el tiempo de ejecución.

Como se muestra en la Fig 2.2 la organización interna del PE [1] consiste de:

- BRAM Sináptica: Se utiliza para el almacenamiento de valores sinápticos, a cada sinapsis le corresponde una palabra de memoria.
- CAM(Content Addressable Memory) y SPIKE REGISTER: La topología sináptica se implementa en registros que contienen la dirección origen de las neuronas con las cuales se ha definido una conexión. Cuando se reciben los spikes pre-sinápticos, éstos son leídos por todos los PE y comparados con las direcciones de las neuronas presinápticas propagadas en la CAM, que opera como memoria asociativa. Solamente en aquellos donde exista una coincidencia de dirección se procesa el evento almacenando su valor en el SPIKE REGISTER.
- Central Processing Element: Corresponde a la unidad aritmético-lógica del PE.

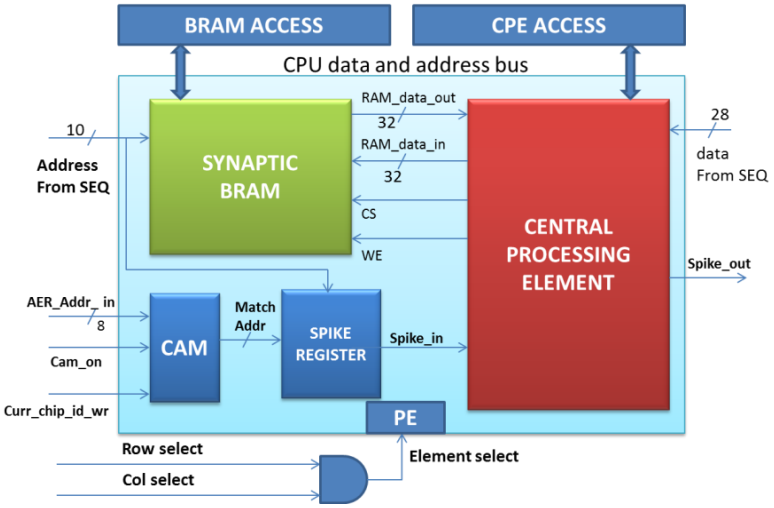


Figura 2.2: Elementos de procesamiento configurable [1]

Finalmente, regresando a la Fig. 2.1, la arquitectura SNAVA dispone de un controlador de bus AER que permite realizar la distribución de spikes en la red neuronal. En el siguiente capítulo se detalla las características y funciones del controlador, ya que el desarrollo de un esquema de comunicación AER eficiente forma parte de los objetivos planteados en esta tesis.

2.2.2. Aplicación: Implementación de una Cadena Synfire sobre SNAVA [ICANN]

En este apartado se detalla la implementación de una red SFC sobre una implementación de SNAVA con FPGA. Dicha implementación demuestra la funcionalidad de SNAVA como arquitectura multi-modelo capaz de procesar un algoritmo neuronal con la exactitud suficiente para reproducir la actividad de una cadena Synfire. El concepto de cadenas Synfire (SFC) introducido por Abeles [3]

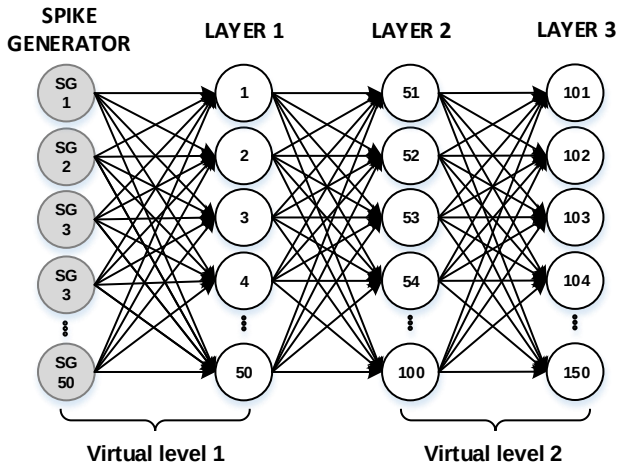


Figura 2.3: Topología de la SFC

se fundamenta en el hecho de que neuronas corticales que comparten un conjunto poco disperso y suficientemente grande de estímulos de entrada tienden a alinear sus potenciales de acción al retransmitirse por las subsiguientes capas neuronales [4], [5]. El grado de dispersión temporal de los spike entre capas determina si una vez conseguido el sincronismo del conjunto de estímulos de entrada, éste se puede mantener hasta el final de la cadena o si, en su defecto la excitación se dispersa y eventualmente se desvanece [4].

La SFC implementada consiste en una red de neuronas excitatorias dispuestas en una configuración tipo feed-fordward como se muestra en la Fig 2.3, con una capa de entrada que genera un grupo de 50 spikes aleatorios con distribución gausiana de valor medio y desviación estándar igual a 140 y 12 ciclos, respectivamente.

La SFC utiliza un modelo neuronal modificado de Leaky Integrate and Fire (LIF), cuyas ecuaciones diferenciales fueron extraídas de [6] y se listan a continuación:

$$\tau_{mem}\frac{dv}{dt} = x(t) - (V(t) - V_r) \tag{2.1}$$

$$\tau_{rft}\frac{dx}{dt} = -x(t) + y(t) \tag{2.2}$$

$$\tau_{rft}\frac{dy}{dt} = -y(t) + \tau_{rft} \cdot 25{,}27\ mV + I_s + B. \tag{2.3}$$

donde *v(t)* corresponde al potencial de membrana; x y y son variables que representan las corrientes de los canales iónicos; V_r es el voltaje de restitución; τ_{rft} y τ_{mem} son constantes de tiempo refractario y de membrana respectivamente. El modelo usa una componente de ruido de fondo $(B, Brackground\,noise)$ que emula la actividad aleatoria de la red neuronal. El voltaje de umbral (V_{thr}) es de -70 mV y a todas las conexiones sinápticas se aplica el mismo peso de 11 mV. La corriente sináptica obtenida como la suma de todas las contribuciones sinápticas activas es representada por I_s y aplicada a $y(t)$. Para resolver las ecuaciones diferenciales se empleó la aproximación de Euler con condiciones

iniciales $v(t=0)$ =valor aleatorio entre $(V_{thr} - V_r)$, $x(t=0) = 0$ and $y(t=0) = 0$. Se obtuvieron las siguientes ecuaciones discretas usadas para la programación del algoritmo en SNAVA.

$$v(t+1) = v(t)(1 - \frac{\Delta t}{\tau_{mem}}) + (V_r + x(t))\frac{\Delta t}{\tau_{mem}} \tag{2.4}$$

$$x(t+1) = x(t)(1 - \frac{\Delta t}{\tau_{rft}}) + y(t)\frac{\Delta t}{\tau_{rft}} \tag{2.5}$$

$$y(t+1) = y(t)(1 - \frac{\Delta t}{\tau_{rft}}) + (25{,}27 + \frac{B}{\tau_{rft}})\Delta t + I_s \tag{2.6}$$

Implementación y Resultados

La implementación de la SFC en SNAVA se realizó en una FPGA Kintex 7 XC7K325T con un arreglo de 10x10 PE y dos niveles de virtualización (Fig 2.3) para obtener las 150 neuronas más las 50 generadores de la entrada requeridos para esta aplicación. Cada PE emula un generador y una neurona o 2 neuronas al multiplexar en el tiempo su ejecución. Para el mapeo de la topología se definieron 7500 conexiones sinápticas, ya que hay una conectividad completa entre capas.

Las operaciones matemáticas utilizan una representación de números binarios en un formato de punto fijo con signo de 16 bits. Para el caso de esta aplicación se realizó un análisis de precisión y del rango dinámico para evitar desbordamientos y garantizar que las operaciones puedan ejecutarse con suficiente exactitud. Cómo resultado se usó 4 bits para x(t) y y(t), 7 bits para v(t) en la parte fraccionaria y el resto de los 16 bits se designaron a la parte entera, es decir, 12 bits para x(t) y y(t) y 9 bits para v(t).

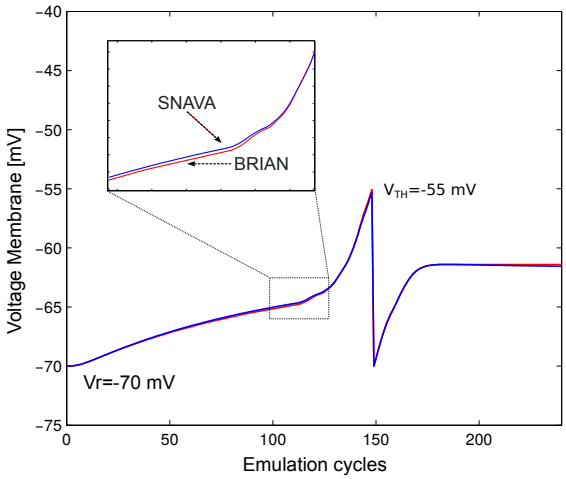


Figura 2.4: Voltaje de Membrana en SNAVA y BRIAN de la neurona 45

Como demostración, se muestra a continuación los resultados para una ejecución de 300 ciclos de emulación. En la Fig 2.4 se muestra una comparación de resultados del voltaje de membrana implementado en SNAVA y CPU. La implementación en CPU se realizó con el simulador de redes

neuronales spiking BRIAN [6] (implementado con Python con datos de tipo punto flotante). Como se observa, los resultados obtenidos en ambas implementaciones son muy similares con diferencias producidas debido a errores de truncamiento.

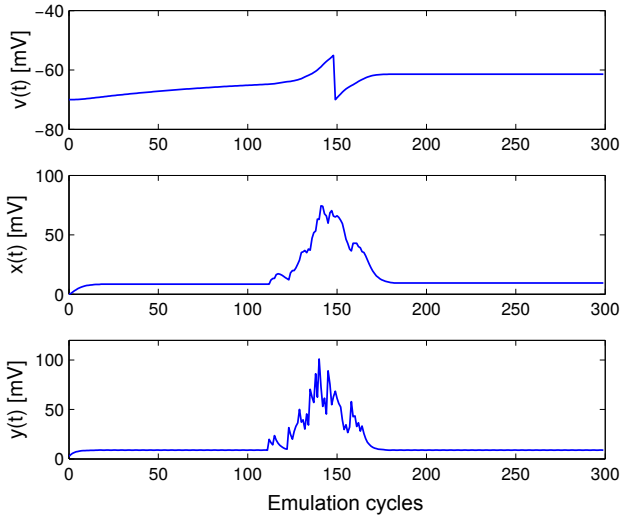


Figura 2.5: Variables de estado de la SFC

En la Fig 2.5 se presenta la actividad de las 3 variables de estado de una neurona en la capa 1. Se observa cómo el ruido y los estímulos de entrada con valor medio en el ciclo 140 influyen en $y(t)$. Además, la actividad de $y(t)$ y $x(t)$ interviene en la polarización y despolarización del potencial de membrana observado en $v(t)$.

El gráfico de raster de la Fig 2.6 ilustra la ejecución de la aplicación y la actividad de toda la red durante 200 ciclos de emulación. En la primera capa se observan los estímulos de entrada y cómo éstos se van alineando a medida que se propagan en las subsiguientes capas. Cada ciclo de emulación se ejecuta en 29.26 us, lo cuál está dentro de la ventana de tiempo de 1 ms considerada como tiempo real para aplicaciones neuronales.

En la Tabla 2.1 se reporta la utilización de recursos de hardware para esta aplicación luego de síntesis. Cómo se puede observar las LUTs son el recurso de mayor consumo. Esto se debe a que la definición del mapeo sináptico se realiza a base de registros, los cuáles al sintetizarse consumen recursos de LUTs. Además, cabe notar que una vez sintetizada la topología, ésta se mantiene fija. El tiempo de síntesis e implementación utilizando la herramienta de Xilinx Vivado 2013.1 fue de 17 horas.

2.3. Limitaciones de la arquitectura SNAVA

A partir de la experiencia de la implementación de la aplicación SFC, se identificaron las siguientes limitaciones en la arquitectura SNAVA:

- **Arreglo de PE:**

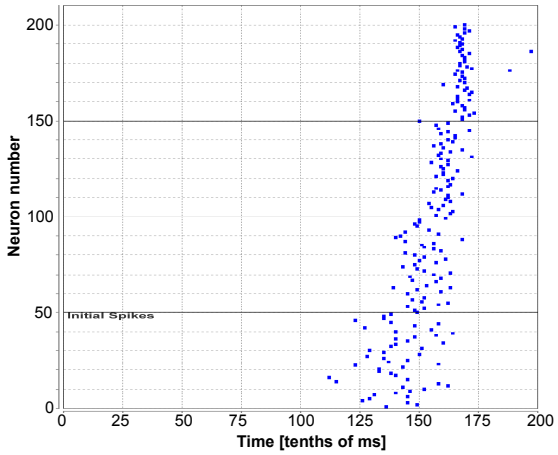


Figura 2.6: Raster Plot de la SFC

Tabla 2.1: Utilización sobre la FPGA Kintex XC7K325T para la implementación de la SFC

Lógica	Porcentaje de utilización	Disponible
LUTs	159,739 (78 %)	203,800
Slice registers	89,841(22 %)	407,600
Block RAMs	119(9 %)	1,335
DSP	100(12 %)	840

- Conexionado sináptico fijo, su modificación requiere volver a sintetizar SNAVA.
- Alto consumo de LUTs debido en gran parte al tipo de conexionado sináptico implementado a base de lógica combinacional.
- Las LUTs limitan la escalabilidad de SNAVA.
- El tiempo total de síntesis e implementación es considerablemente alto debido al uso de LUTs en gran parte de la arquitectura. Esto obliga al sintetizador (Vivado 2013.2) a invertir gran cantidad de tiempo en el mapeo de conexiones de la FPGA.
- Arreglo de PEs no parametrizable.
- Violaciones de tiempo (slacks negativos) en síntesis e implementación.

- **Secuenciador:**
  - Simplificación en los accesos a memoria de datos mediante direcciones fijas que limita el soporte multimodelo de SNAVA.
  - Juego de instrucciones no optimizado.

- **AER:**
  - Modelo de comunicación plano.

  - Se requiere n conexiones ethernet para configurar y acceder a n chips del anillo. En la aplicación de la SFC se utilizó una sola tarjeta.

- **Soporte de Software:**
  - Herramientas de software para compilación de algoritmos neuronales poco flexibles.
  - Configuración de la aplicación neuronal mediante conexión ethernet con errores de transmisión/recepción de datos.

2.4. Conclusiones

Con los resultados obtenidos se valida la funcionalidad de SNAVA como una arquitectura multi-modelo capaz de reproducir en tiempo real y con suficiente precisión algoritmos neuronales en general, y en particular el de la SFC para 200 neuronas y 7500 sinapsis. Esto es posible gracias a que el set de instrucciones de SNAVA es lo suficientemente flexible como para implementar algoritmos neuronales no lineales complejos. Así también la multiplexación en el tiempo (virtualización) implementada para el procesamiento neuronal permite compartir recursos de hardware e incluso ejecutar diferentes algoritmos por cada nivel virtual.

En lo concerniente a consideraciones de diseño, uno de los principales inconvenientes encontrados en SNAVA es que una vez sintetizada, el mapeado sináptico es fijo ya que las conexiones se realizan mediante lógica combinacional. Cambios en las conexiones implican sintetizar de nuevo el hardware. Esta es una limitación muy importante debido a que le resta realismo biológico a la arquitectura al no ser capaz de crear o destruir conexiones sinápticas durante procesos de aprendizaje (plasticidad). Además, también fundamental, limita la escalabilidad del sistema. Por otro lado, los retardos axonales que también contribuyen al realismo de la red y posiblemente al proceso de aprendizaje tampoco han sido implementados en SNAVA.

Referencias

[1] G. Sanchez, "Efficient multiprocessing architectures for spiking neural network emulation based on configurable devices", Tesis doct., Universitat Potecnica de Catalunya, 2014.

[2] M. Zapata y J. Madrenas, "Synfire chain emulation by means of flexible snn modeling on a simd multicore architecture", English, *25th International Conference on Artificial Neural Networks ICANN 2016*, vol. 8681, n.º September, págs. 222-229, 2016, ISSN: 16113349. DOI: 10.1007/978-3-319-11179-7. eprint: 1412.7927.

[3] M. Abeles, "Corticonics: neural circuits of the cerebral cortex", *Cambridge University Pres*, 1991.

[4] M. Diesmann, M. Gewaltig y et.al., "Stable propagation of synchronous spiking in cortical neural networks", *Nature*, vol. 402, págs. 599-533, 1999.

[5] M. Abeles, G. Hayon y D. Lehmann, "Modeling compositionality by dynamic binding of synfire chains", *Journal of Computational Neuroscience*, vol. 17, n.º 2, págs. 179-201, 2004, ISSN: 09295313. DOI: 10.1023/B:JCNS.0000037682.18051.5f.

[6] D. Goodman y R. B. Brette, "BRIAN: a simulator for spiking neural networks in Python", *Front. Neuroinform*, 2008. DOI: 10.3389/neuro.11.005.2008.

Capítulo 3

Arquitectura HEENS

Con base en las contribuciones de SNAVA, de entre ellas las más relevantes: el dar soporte a modelos neuronales programables, el uso de memoria distribuida, y el incremento del número de neuronas/chip al multiplexar en el tiempo el procesamiento de los PE; en este capítulo se presenta una nueva generación de dicha arquitectura. La plataforma Hardware Emulator of Evolving Neural System (HEENS) incorpora características que proporcionan mayor flexibilidad y realismo biológico que serán descritas y detalladas a lo largo de este capítulo.

A continuación la sección 3.1 presenta la estructura de HEENS y se lista sus principales características y fases de procesamiento que gobiernan la ejecución de todo el sistema. La sección 3.2 presenta el Multiprocesador HEENS, aspectos de diseño y funciones, seguido por la implementación, el análisis de resultados y por último las conclusiones del capítulo.

3.1. Descripción de la arquitectura HEENS

HEENS es una arquitectura planteada para implementaciones multi-chip destinada a la emulación de SNN en tiempo real, la cual se ha diseñado dar soporte a redes evolutivas con un alto grado de configurabilidad, con un esquema de comunicación de spikes AER (Address Event Representation) jerárquico.

Se parte de los fundamentos de SNAVA para sentar las bases de HEENS. Esta es una arquitectura totalmente rediseñada que permite interconectar varios chips en una topología de anillo (Fig. 4.13) en un esquema de comunicación híbrido Master/Slave y Point to Point. El Chip Master (MC) toma el control de la red para:

1.- Configurar el anillo y la aplicación neuronal en todos los nodos.

2.- Dar soporte a la evolución de la red controlando la reconfiguración on-line dinámica de cada nodo.

Durante la ejecución de la aplicación neuronal, el MC actúa como un nodo slave más en la red. Por otro lado, cada nodo slave, en adelante denominado Chip Neuromórifico (NC), se encarga de procesar el algoritmo neuronal en un arreglo de multiprocesadores 2D tipo SIMD.

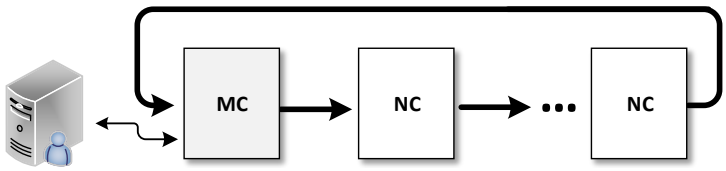


Figura 3.1: Arquitectura HEENS conformada por un Master Chip (MC) y n Chips Neuromórficos (NCs) conectados en anillo

La presentación de la arquitectura HEENS se ha dividido en dos partes: Multiprocesador HEENS (HEENS-MP) y el Modelo de Comunicación AER-SRT. La primera parte se detalla en este capítulo e incluye el diseño modificado del secuenciador y del arreglo de multiprocesadores con características funcionales añadidas. El Modelo de Comunicación se trata en el siguiente capítulo que además incluye la explicación detallada del MC.

3.1.1. Fases de operación de HEENS

HEENS basa su secuencia de procesamiento emulando el comportamiento biológico de las neuronas, para lo cual divide su procesamiento en cuatro fases de operación:

- Inicialización **(IPh)**
- Configuración **(CPh)**
- Ejecución **(EPh)**
- Evolución **(EvPh)**
- Distribución **(DPh)**

Como se ilustra en la Fig. 3.2 el procesamiento de cada fase se realiza de forma secuencial, tal que el siguiente estado no inicia hasta que la anterior finalice su operación. En el estado inicial **(Idle)** la señal de Rst coloca los registros en sus valores por defecto.

- **Fase de Inicialización (IPh):** Para configurar el anillo que conforma la plataforma multichip, se requiere identificar cada nodo (ID), así como el tamaño del anillo (Ring Size). Ésta fase es la encargada de asignar dinámicamente dichos parámetros a los NCs.
- **Fase de Configuración (CPh):** Administra el envío de los datos de configuración de la aplicación neuronal a procesarse en cada uno de los NCs de la red que incluye el algoritmo neuronal, parámetros sinápticos y neuronales, así como el mapeo de las conexiones sinápticas locales y globales. Esta fase finaliza una vez todos los NCs se encuentren configurados.
- **Fase de Ejecución (EPh):** Esta fase tiene correspondencia biológica con el soma. Aquí se procesa el algoritmo neuronal mediante el cálculo y actualización de las variables de estado. Cada neurona

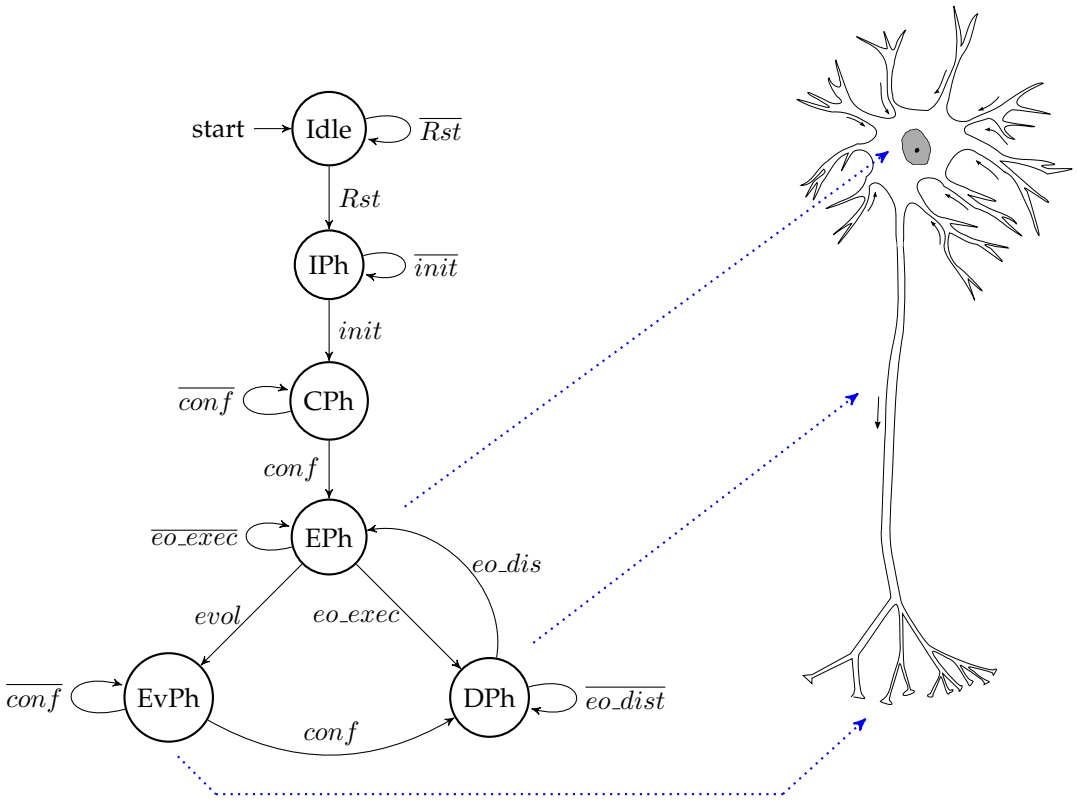


Figura 3.2: Fases de procesamiento de HEENS

utiliza parámetros individuales para su procesamiento. El inicio y fin de esta fase es marcado por una señal interna de control denominada eo_exec.

- **Fase de Distribución (DPh):** Se emula la propagación de neurotransmisores por las sinapsis mediante un broadcast de eventos de spikes transmitidos por el bus de comunicación serial. Los spikes obtenidos en la fase de ejecución son propagados y entregados a las neuronas destino localizados en el mismo o en diferentes NCs.

- **Fase de Evolución (EvPh):** Al final de cada ciclo de ejecución se verifica si se ha recibido una orden de evolución. Dado el caso, la actividad neuronal de el/los NCs seleccionados se ajustará ó evolucionará en función de la información recibida desde el MC.

Se utiliza como unidad de tiempo los ciclos de emulación. Estos corresponden al procesamiento de la actividad neuronal y a la distribución de spikes que se da en las EPh y DPh, una vez que las IPh y CPh finalizan como se ilustra en la Fig. 3.2.

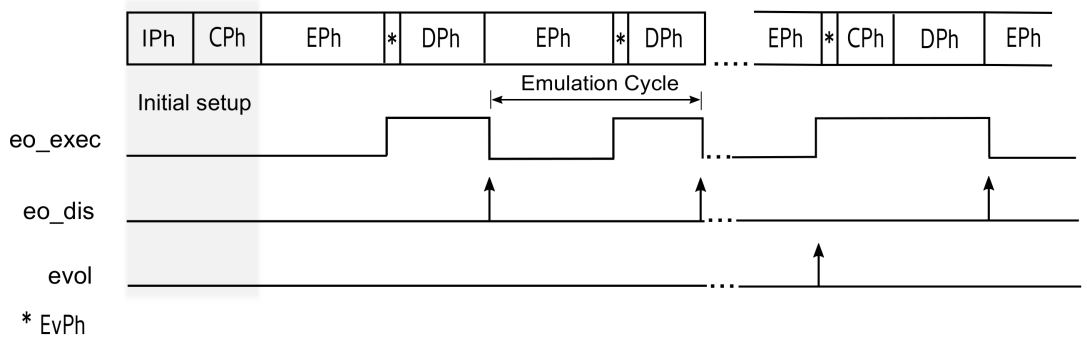


Figura 3.3: Fases de operación de HEENS

3.2. Multiprocesador HEENS (HEENS-MP)

El HEENS-MP ilustrado en la Fig. 3.4 corresponde al NC de la red neuronal. Este realiza el procesamiento de la actividad neuronal programada por el usuario.

HEENS-MP utiliza un esquema de cómputo SIMD con una única unidad de control para conseguir paralelismo a nivel de datos en el arreglo de PEs. Esta técnica es idónea para la implementación de redes neuronales debido a que reduce costos de área y ofrece un alto rendimiento computacional. No obstante, la ejecución de saltos condicionales es limitada debido a que los Elementos Procesadores (PEs) ejecutan las mismas instrucciones pero con datos y resultados diferentes. En este caso se introducen estados de no operación para poder sincronizar operaciones [1] [2].

Los principales bloques que lo conforman y que se describen a continuación son:

- Buses de comunicación
- Arreglo de PEs
- Control Unit
- AER-SRT Controller

3.2.1. Buses de comunicación

Como se observa en la Fig. 3.4, los buses de dirección y datos (HEENS_addr, HEENS_data) que permiten el flujo de información al arreglo se encuentran multiplexados entre los que transmiten paquetes de configuración (pkg_add, pkg_data) y los que entregan información de ejecución correspondientes a los opcodes (data_seq) despachados por el Sequencer y eventos de spikes (spike_in) entregados por el AER-SRT Controller. La selección de éstos está dada por la señal interna *config* que se activa en función de la fase que se está procesando. La información de configuración es destinada para las siguientes memorias:

- Instruction

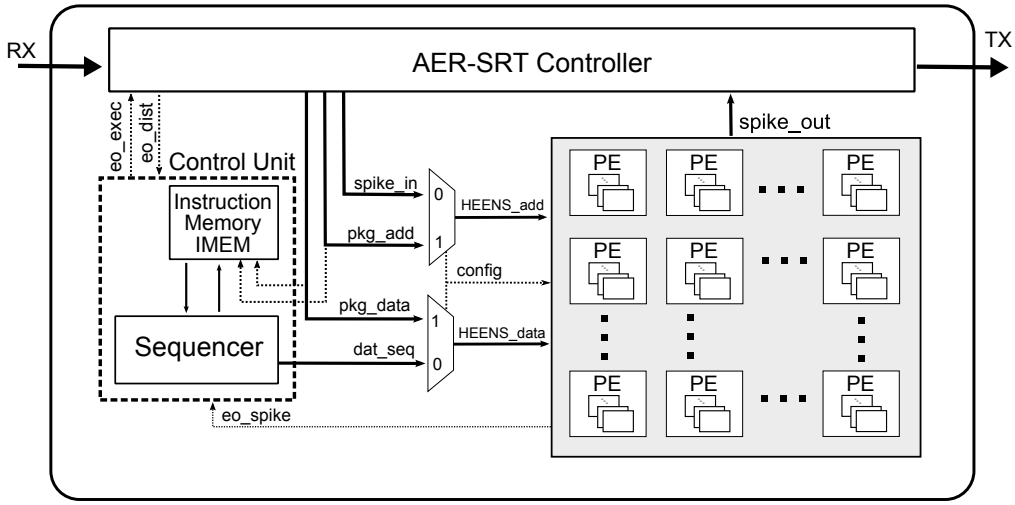


Figura 3.4: Diagrama de bloques del HEENS-MP

- Local y Global /PE.
- Synaptic/Neural (SNRAM)/PE

En el arreglo cada PE es identificado por su posición de fila y columna que permite establecer el destino de la entrega de datos. Es así que, la palabra de configuración compuesta por los campos de dirección y datos, utiliza el mapeo mostrado en las Tablas 3.1 y Tabla 3.2 para determinar el PE y la memoria a los cuales debe acceder para su respectiva inicialización o reconfiguración.

Tabla 3.1: Mapeo de la palabra de configuración - campo de dirección

Selection		**Address**									
		b31-b26	**b25-b23**			**b22-b20**		**b19-b10**		**b9-b0**	
Chip ID		x	0	1	0	x					
Instruction Memory		x	0	1	1	x				address	
Codification Memory	**ID**	x	1	0	0	x	0	x	ID	PE row	PE col
	row/col						1	row	col		
Conversion Memory	**ID**	x	1	0	1	x	0	x	conv id	PE row	PE col
	row/col						1	x	conv r/c		
Local Memory		x	1	1	0	Virt		row	col	PE row	PE col
Synaptic / Neural Memory		x	1	1	1	x		address		PE row	PE col

Durante el procesamiento neuronal, cada PE recibe y entrega paquetes de dirección (spike_in, spike_out) al AER-SRT Controller. Estos paquetes corresponden a eventos de spike que identifican la posición de la neurona que ha disparado en el arreglo de cada NC. Su formato se muestra en la Fig. 3.5.

Tabla 3.2: Mapeo de la palabra de configuración - campo de datos

Selection		Data			
		b31-b16	**b15-b7**	**b6-b5**	**b4-b0**
Chip ID		x		ID	
Instruction Memory		x	Data		
Codification Memory	**ID**	x			Data
	row/col				
Conversion Memory	**ID**	Data			
	row/col	Data			
Local Memory		x		Data	
Synaptic/Neural Memory		Data			

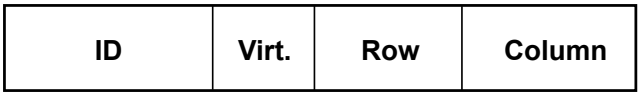


Figura 3.5: Formato de evento de dirección de spike. El ID identifica el NC, y el resto de parámetros: virtualización, row y column corresponden al nivel virtual asignado y a la posición del PE en el arreglo.

3.2.2. Arreglo de Elementos Procesadores

Como se observa previamente en la Fig. 3.4 este es un arreglo 2D de PEs. Su tamaño es parametrizable en función del números de filas y columnas del arreglo y del nivel virtual de los PEs. Estos últimos son los encargados de procesar la actividad neuronal de una o más neuronas multiplexando en el tiempo su funcionamiento.

Debido al carácter jerárquico y modular de HEENS, los PEs procesan dos tipos de spikes: locales y globales. Se consideran locales aquellos generados en el mismo NC, y globales a los provenientes de otros NC ó del MC.

Elemento Procesador (PE)

Es una unidad de ejecución tipo SIMD, encargada de procesar las instrucciones que describen el algoritmo neuronal. Como se muestra en la Fig 3.6, está constituido por:

- ALU: Soporta operaciones lógicas y aritméticas de 16 bits con punto fijo. La multiplicación es implementada con DSPs. Las banderas de estado de Acarreo (C) y Cero (Z) son accesibles para el usuario.
- Virtualización (VIRT): Los PEs multiplexan en el tiempo su procesamiento para emular más de una neurona por ciclo de ejecución. Se emula una neurona por nivel virtual. Como se muestra en la Fig. 3.7 se puede definir n niveles de virtualización pipeline más un nivel principal (VIRT=0). El nivel principal está conformado por neuronas tipo HUB que permiten la comunicación entre

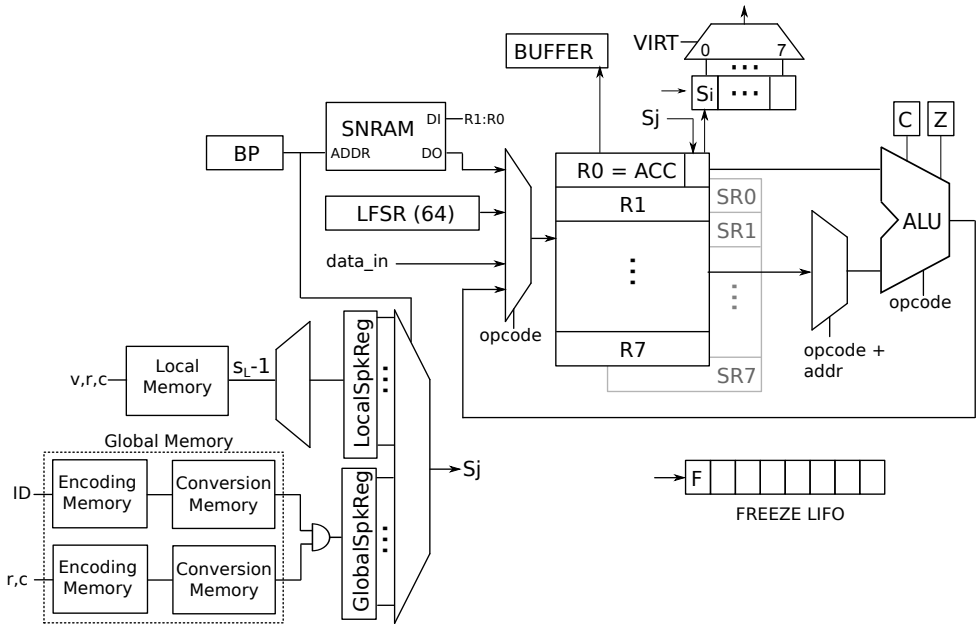


Figura 3.6: Processing Element

neuronas de diferentes NCs a través de los spikes globales, como entre neuronas del mismo NC por medio de spikes locales.

Cabe mencionar que el tiempo de ejecución escala con el número de niveles definido en el arreglo.

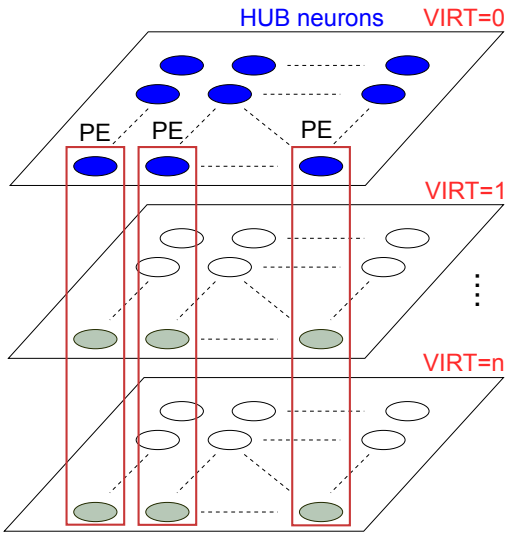


Figura 3.7: Virtualización del arreglo de PEs

- Banco de Registros: Se dispone de 1 banco de 8 registros de propósito general (R0 -R7) que interactúa directamente con la ALU a través del acumulador R0. Además, cada PE dispone de su propio banco registros de denominados de sombra (SR0-SR7) usado para extender el espacio de almacenamiento de las variables neuronales o sinápticas. El intercambio de información entre bancos se puede realizar en ambas direcciones, y siempre a través de sus registros pares, es decir R0 ¡-¿SR0, R1 ¡-¿SR1, etc. El acceso a los bancos de sombra es gestionado por el Sequencer.

- Synaptic/Neural Memory (SNRAM): Bloque de memoria de datos encargada de almacenar parámetros neuronales, sinápticos, banco de registros de sombra y las semillas de los LFSR de las neuronas procesadas por el PE en cada nivel virtual. Se utiliza la estrategia de coalescencia para acelerar la lectura/escritura en memoria enviando 2 operandos en un ciclo de reloj.

- Memorias de conexiones locales y globales: Estas están conformadas por un bloque de memoria local, junto con un novel esquema de memoria asociativa que permite la decodificación de conexiones globales. A continuación se describen:

  - *Local Memory:* Aquí se modela la interconectividad entre neuronas locales. Esta procesa spikes con el mismo ID del NC al que pertenece (spikes locales). Las sinapsis asociadas a cada neurona local se decodifican a partir de las direcciones de spikes post-sinápticos (nivel de virtualización, fila y columna). El número de total de bits/PE de la memoria local es:

    $$\#bits = (2^{v+r+c}).log_2(s_L - 1) \tag{3.1}$$

    donde s_L corresponde a la máxima cantidad de sinapsis soportada por los PEs del arreglo. Los campos v, r, c representan el ancho en bits de los campos de virtualización, fila y columna del evento de dirección de spike.

    Se reduce en una unidad la codificación de sinapsis locales debido a que se ha asignado arbitrariamente el código 0 para casos de no coincidencia. El registro LocalSpkReg contiene el resultado de la decodificación, que corresponde a un set en la posición de la sinapsis donde exista una correspondencia entre pares.

    El número máximo de sinapsis locales es equivalente al numero de filas por número de columnas del arreglo, las cuales se distribuyen entre los niveles virtuales asignados al PE.

  - *Esquema de Memoria Asociativa - Global Memory:* Este bloque modela la interconexión entre neuronas de diferentes NCs. Es así que cada PE dispone de $s_G - 1$ sinapsis globales procesadas en el nivel principal (Virt=0) para emular un conexionado jerárquico caracterizado por una escasa conectividad entre clusters. Para reducir el consumo de memoria requerido para la decodificación de spikes globales se aplica el esquema de memoria asociativa mostrado en la Fig. 3.8.

    Tomando en cuenta que al menos s_G NCs producirán spikes globales, su ID se codifica en $log_2 s_G$ bits. Los campos de row y column (r,c) son tratados de la misma manera. Ambas salidas se decodifican en $s_G - 1$ bits por separado debido a que varios spikes pueden tener el

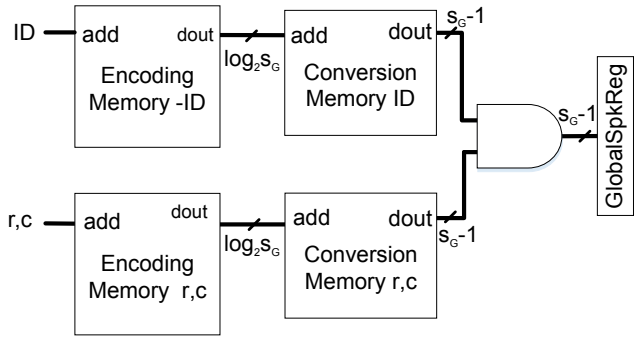


Figura 3.8: Decodificación de spikes globales

mismo ID con diferentes valores en los campos de row y column, así como el caso contrario. Para obtener el dato válido se utiliza una compuerta AND en la salida. Además, expandir la salida a un registro tipo one-hot de $s_G - 1$ bits es necesario para obtener el GlobalSpkReg que es procesado posteriormente para leer su contenido en la ejecución del algoritmo neuronal. El número total de bits/PE utilizados para esta aproximación es:

$$\#bits = (2^{ID} + 2^{r+c}).log_2 s_G + 2s_G.(s_G - 1) \tag{3.2}$$

Esta técnica es más eficiente respecto al número de BRAM empleados a diferencia de otras alternativas como la decodificación a través de registros que consumen mucha área de lógica programable. Así también la decodificación directa de la dirección de los spikes requeriría un consumo de memoria 1000 veces más grande que el actual diseño. El uso de memoria externa proporciona un espectro más amplio de memoria, no obstante sus tiempos de acceso son mayores comparado con la memoria on-chip utilizada en esta aproximación y requieren un mayor consumo de energía. En [3] se detalla esta técnica de decodificación de spikes y su comparación con otras alternativas.

- Generador aleatorio (LFSR): Se dispone de un registro de desplazamiento con retroalimentación lineal de 16 bits. La semilla de 64 bits se define por software con la instrucción SEED, y es almacenada en la memoria SNRAM. Este registro permite generar ruido no correlado a cada PE.

- La implementación de lazos con instrucciones condicionales están ligadas a las banderas de estado de la ALU: Carry y Zero (FREEZEC, FREEZENC, FREEZEZ, FREEZENZ). Éstas utilizan una LIFO (Last In First Out) para su ejecución. Se admiten 8 niveles de anidamiento.

  Cuando una condición de FREEZE se cumple, se apila (PUSH) un nivel en la LIFO, y a su vez se desactiva el banco de registros activos (R0-R7), de sombra (SR0-SR7) y las banderas de estado de la ALU. Como resultado, las instrucciones contenidas dentro del lazo no se ejecutarán. El fin del lazo condicional es determinado por la instrucción UNFREEZE que desapila (POP) un nivel en la LIFO. Cuando ésta se vacía, se activa nuevamente los bancos y las banderas de estado.

3.2.3. Control Unit

Este módulo previamente mostrado en la Fig. 3.4 se encarga de administrar el flujo completo de datos e instrucciones del arreglo de PEs por medio del Sequencer. Además, genera las señales de control correspondientes para sincronizar las operaciones con el controlador AER-SRT.

HEENS-MP es una arquitectura de tipo Harvard. Las instrucciones se leen desde un único bloque de memoria (IMEM). Cada PE tiene su propia memoria de datos (SNRAM) donde almacena los parámetros sinápticos y neuronales de cada neurona.

El Sequencer trabaja en función de máquina de estados para gestionar todas las operaciones. La palabra de instrucción enviada al arreglo contiene información de opcodes y otros parámetros como puntero de memoria de datos, dirección de los registros de propósito general fuente o destino o constantes dependiendo del tipo de instrucción. El set de instrucciones detallado en el Anexo I, está descrito en lenguaje ensamblador y se creó específicamente para ésta arquitectura. De forma general las 59 instrucciones que lo conforman se clasifican en operaciones de:

- **SEQUENCER**: Utilizadas para operaciones de control de salto incondicional y de interacción con el puntero de pila de la IMEM.
- **REGISTER**: operaciones que actúan sobre el banco de registros activos de propósito general.
- **MOVEMENT**: Operaciones de transferencia de datos entre registros activos y/o de sombra. Vuelcan el contenido de un registro en otro.
- **FLAGS**: Modifican las banderas de estado de la ALU.
- **ARITHMETIC**: Agrupa operaciones aritméticas soportadas por la ALU.
- **LOGIC**: Cubren las operaciones lógicas elementales.
- **CONDITIONAL**: Corresponden a operaciones de ejecución condicional mediante la evaluación de banderas de estado.
- **OTHERS:** Implementadas para funciones específicas en el procesamiento del algoritmo neuronal.

En la Tabla 3.3 se lista el set de instrucciones de HEENS de acuerdo a su categoría, se resalta tanto las que han sido modificadas respecto a la versión de SNAVA, como las nuevas instrucciones creadas para HEENS.

A continuación se realiza una descripción de las principales instrucciones:

- *FREEZEC*: Deshabilita los registros relacionados con la ALU cuando el bit de Carry es 1.
- *FREEZENC*: Deshabilita los registros relacionados con la ALU cuando el bit de Carry es 0.
- *FREEZEZ*: Deshabilita los registros relacionados con la ALU cuando el bit de Zero es 1.
- *FREEZENZ*: Deshabilita los registros relacionados con la ALU cuando el bit de Zero es 0.

Tabla 3.3: Clasificación del Set de Instrucciones. Este está conformado por algunas de instrucciones que se desarrollaron para la arquitectura SNAVA, las que se modificaron y las nuevas que se han añadido para HEENS.

LOGIC	ARITHMETIC	MOVEMENT	REGISTERS	CONDITIONAL	FLAGS	SEQ	OTHERS
AND ♢	INC ♢	LLFSR*	LDALL*	FREEZEC*	SETZ♢	NOP♢	LOADSP*
OR♢	DEC♢	MOVA♢	RST♢	FREEZENC*	SETC♢	LOOP*	STOREB♢
INV♢	ADD♢	MOVR	SET♢	FREEZEZ*	CLRZ♢	LOOPV**	STORESP*
XOR ♢	SUB♢	SWAPS♢	SHLN ♢	FREEZENZ*	CLRC♢	ENDL*	STOREPS*
	MUL*	MOVRS**	SHRN♢	UNFREEZE*		GOSUB**	LOADSN**
	MULS*	SEED**	RTL♢			RET♢	RANDON*
		MOVSR**	RTR♢			HALT♢	RANDOFF♢
			SHLAN**			SPKDIS♢	LOADBP**
			SHRAN**			READMP♢	SPMOV**
			BITSET**			RST_SEQ**	
			BITCLR**			LAYERV**	
						GOTO**	
						INCV**	
						READMPV**	

♢ Instrucciones desarrolladas para la arquitectura SNAVA.
* Instrucciones modificadas respecto a SNAVA.
** Nuevas instrucciones

- *UNFREEZEZ*: Finaliza el lazo condicional de cualquiera de los FREEZE.
- *SWAPS*: Intercambia el contenido del registro activo con el respectivo registro de sombra.
- *MOVRS*: Carga el contenido de un registro de sombra al correspondiente registro activo.
- *MOVSR*: Carga el contenido de un registro activo al correspondiente registro de sombra.
- *HALT*: Para la operación del sequencer.
- *RST_SEQ*: Resetea el sequencer.
- *RANDOM*: Habilita la operación del LFSR.
- *LAYERV*: Define el número de capas virtuales
- *LOOP*: Lazo neuronal para operaciones con virtualización. El cambio de nivel virtual lo gestiona el sequencer por cada iteración del lazo.
- *LOADBP*: Carga el puntero (BP) de memoria correspondiente a la neurona que está siendo procesada.
- *LOADSN*: Carga parámetros neuronales desde la SNRAM para que puedan ser procesados en el algoritmo. La dirección está dada por el puntero BP.
- *LOOPV*: Lazo sináptico, aquí se realiza la iteración de procesamiento de sinapsis correspondientes a cada nivel virtual.

- *LOADSP*: Carga parámetros sinápticos, incluyendo el spike pre-sinápticos de la neurona que se está procesando.
- *STORESP*: Almacena parámetros sinápticos de la neurona que está siendo procesada, e incrementa el BP para leer la próxima sinapsis.
- *STOREPS*: Escanea todos los PE del arreglo, para obtener los spikes post-sinápticos y generar la dirección del evento. Los eventos de spike son almacenados en una FIFO para su posterior distribución.
- *INCV*: Indica al sequencer que debe pasar a procesar el siguiente nivel virtual.
- *SPKDIS*: Para la operación del sequencer y activa la señal interna eo_exec que permite pasar a la siguiente fase de evolución.

El sequencer está implementado con un pipeline de 4 estados: captura, decodificación, ejecución, y escritura. Ésta es una técnica extensamente usada para reducir el número de ciclos de reloj requeridos para ejecutar un algoritmo. Su tasa de ejecución es de 1 instrucción por cada ciclo de reloj, excepto en aquellas que requieren operaciones compuestas para su procesamiento o saltos condicionales y/o incondicionales. Así como el juego de instrucciones ha sido significativamente modificado y el código VHDL del secuencia se ha depurado y optimizado, el pipeline conserva la misma estructura de SNAVA.

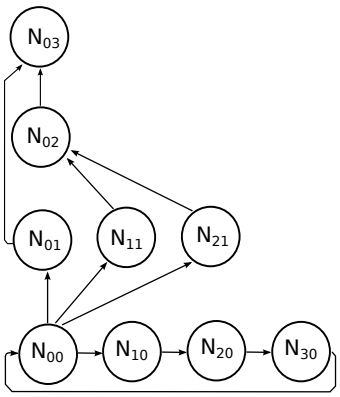
3.2.4. Controlador AER-SRT

Una vez se ha procesado el algoritmo en los PEs, se realiza un barrido por filas para obtener los S_i de cada PE. Estos corresponden a los spikes postsinápticos disparados por las neuronas cuando el voltaje de membrana ha superado el voltaje de threshold. Los spikes generados en cada ciclo de ejecución son codificados en eventos de dirección y almacenados en una FIFO en espera de ser transmitidos durante la DPh. Los detalles de su funcionamiento e implementación se discuten en el siguiente capítulo.

3.3. Flujo de diseño para la implementación de Aplicaciones Neuronales

Para realizar la implementación de una aplicación neuronal, el usuario debe generar el archivo de configuración que define la inicialización de todos los módulos de la arquitectura. Para esto se requiere la especificación de la topología de la red, así como los algoritmos neuronales y sinápticos que describen la actividad neuronal. En un archivo de netlist se crea la topología de la red conformado por la posición de las neuronas locales origen y destino en el arreglo por cada conexión sináptica. A cada sinapsis se asigna el peso correspondiente. El formato a seguir es el mostrado en la Fig 3.9.

El ejemplo corresponde a un oscilador formado por las cuatro neuronas interiores conectado a una combinación de neuronas con excitaciones e inhibiciones que realizan un filtrado paso banda. La inhibición corresponde a peso sináptico negativo.



Neuron source			Neuron destination			Weight
Virt	Row	Col	Row	Col	Synap	
0	0	0	0	1	1	65536000
0	0	0	1	1	1	32768000
0	0	0	2	1	1	32768000
0	0	1	0	3	1	65536000
0	1	1	0	2	1	32768000
0	2	1	0	2	2	32768000
0	0	2	0	3	2	-65536000
0	0	0	1	0	1	163840000
0	1	0	2	0	1	163840000
0	2	0	3	0	1	163840000
0	3	0	0	0	1	163840000

Figura 3.9: Netlist - Definición de la topología de la red

En el caso ilustrado todas las neuronas origen se encuentran en el nivel principal (Virt=0). Para la asignación de parámetros neuronales y/o otros requeridos por la aplicación neuronal a implementar, se tienen la flexibilidad de definir en un archivo de texto la dirección de memoria a partir de la cual estos se van a almacenar. Como se ilustra en la Fig. 3.10 cada columna corresponde a los parámetros asignados a cada neurona del arreglo. En este caso particular se ha definido los voltajes de restitución a partir de la dirección 0x3E3 y a partir de 0X3FD las semillas de los LFSR.

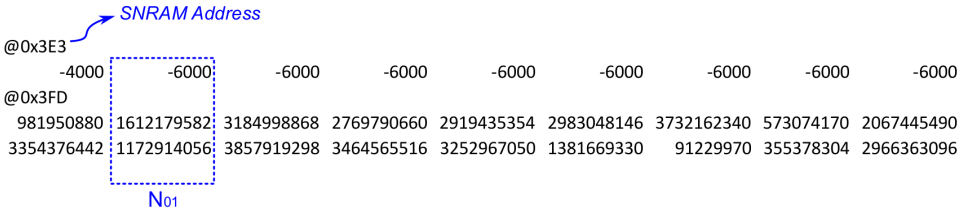


Figura 3.10: Netlist - Definición de la topología de la red

3.3.1. Especificación del Modelo Neuronal

El algoritmo neuronal se ejecuta en la EPh. Su descripción tiene la estructura mostrada en la Fig 3.11 y se divide en 4 secciones:

- **(a)** Es el bloque de declaraciones, aquí se realiza:
  - La declaración de constantes.
  - La asignación del número de sinapsis de cada nivel virtual (NV).El número total de sinapsis locales se distribuye entre todos los NV arbitrariamente, a elección del usuario.
  - Los parámetros neuronales, sinápticos y las semillas de los LSFRs están localizados en diferentes secciones de la memoria SNRAM. La dirección de origen de cada bloque se debe

declarar en esta sección. Ésta debe coincidir con la declarada en los archivos de netlist. Las direcciones de memoria son comunes para todos los PE.

- **(b)** Carga por software la definición del número de capas virtuales que se procesarán en el arreglo de PEs.
- **(c)** En el bucle de ejecución EXEC_LOOP se efectúa la resolución matemática del algoritmo para el cálculo del potencial de membrana.
- **(d)** En el bucle sináptico la instrucción LOOPV carga el puntero de la memoria SNRAM para acceder a las sinapsis asignadas a cada NV.

Terminado el cálculo del algoritmo la subrutina *DETECT_SPIKE* compara si el voltaje de membrana supera el threshold, de cumplirse la condición se produce un spike (S_i). Al final del bucle de ejecución la instrucción *SPKDIS* realiza el barrido del arreglo para obtener los spikes generados como resultado del procesamiento del algoritmo neuronal. Cuando concluye la EPh los spikes son entregados al Controlador AER-SRT para ser distribuidos por la red. El programa completo se puede ver en el Anexo II.

3.3.2. Generación de Archivos de Configuración

Para ensamblar, crear archivos de simulación (MIF) y configuración para los NCs, se utilizan 3 Scrips codificados en Phyton que automatizan estas operaciones. De acuerdo al esquema de la Fig 3.12, el *MIF Generator* crea los archivos de inicialización de las memorias: locales y SNRAMs de cada PE.

El *Assembler Generator* compila y genera el archivo de inicialización de la IMEM. El archivo de configuración (*Configuration File*) se obtiene ejecutando el *Config File Generator* con la información de los scripts anteriores.

La configuración es particular para cada NC. No obstante, el *Configuration File* contiene la información de todos los NCs de la red. En la CPh, el Master Chip es el encargado de enviar esta información a todos los chips de la red para la configuración de la aplicación neuronal en cada uno de ellos.

Por otro lado, el NC contiene la arquitectura HEENS-MP sintetizada e implementada en espera de los datos de configuración entregados por el MC. De acuerdo a la aplicación, se puede cambiar el tamaño del arreglo de PEs a partir de parámetros estáticos i.e número de filas y columnas del arreglo. De ser el caso de debe volver a sintetizar la arquitectura.

3.4. Implementación

La arquitectura del NC está descrita completamente en VHDL. Para la síntesis e implementación se utiliza la herramienta de Xilinx Vivado 2013.2. Para depuración on-chip se emplearon las herramientas integradas de Xilinx, Virtual I/O (VIO) e Integrated Logic Analizer (ILA). La simulación de toda la

```
define    virtual_layers    7          (a)   ; From 0 up to 7
define    lsynapses         80               ; 99 max Due to the local RAM encoding, synapse 0 cannot
                                             ; be used (corresponds to no synapse code)

.DATA
; Virtual layers
V0 = "0000001A"                              ; Number of assigned synapses to the main layer
V1 = "00000003"                              ; Number of assigned synapses to virtual layer 1
V2 = "00000009"                              ; Number of assigned synapses to virtual layer 2
V3 = "00000002"                              ; Number of assigned synapses to virtual layer 3
V4 = "0000000A"                              ; Number of assigned synapses to virtual layer 4
V5 = "00000002"                              ; Number of assigned synapses to virtual layer 5
V6 = "0000000F"                              ; Number of assigned synapses to virtual layer 6
V7 = "00000005"                              ; Number of assigned synapses to virtual layer 7
VLAYERS="00000007"                           ; Number of virtual layers.
; Neural and Synaptic RAM addresses
SYN_ADDR="00000000"                          ; First address of Synaptic parameters in SNRAM.
NEU_ADDR="000003E3"                          ; First address of Neural parameters in SNRAM (995)
SEEDH_ADDR = "000003FD"                      ; Address of noise seed in SNRAM
SEEDL_ADDR = "000003FE"

.MAIN
LAYERV    virtual_layers           (b)   ; Init sequencer vlayers. It is 0 for non-virtual operation
LDALL     ACC, VLAYERS                   ; Load defined virtual layers to PE array
SPMOV     0                              ; VIRT <= ACC

.EXEC_LOOP                               ; Execution loop
    LOOP      virtual_layers             ; Neuron loop for virtual operation
        GOSUB     LOAD_NEURON      (c)
        GOSUB     MEMBRANE_DECAY         ; Calculate membrane potential decay
        GOSUB     ADD_NOISE
        LOADBP    SYN_ADDR               ; Initial position for addresses
        LOOPV     V0               (d)   ; Synaptic loop. Reads number of current-layer
            GOSUB     SYNAPSE_CALC       ;  synapses
        ENDL
        GOSUB     DETECT_SPIKE           ; Compare and eventually spike
        GOSUB     STORE_NEURON
        INCV
    ENDL
SPKDIS                                   ; Distribute spikes
GOTO    EXEC_LOOP                        ; Execution loop
```

Figura 3.11: Estructura principal del programa - Código ensamblador

plataforma se realiza en QuestaSim. Los scripts referidos en la sección anterior generan los archivos de soporte para ésta tarea.

En cada NC los buses y los módulos de instanciación de hardware se crean en función del número de filas y columnas del arreglo. El tamaño máximo dimensionable es de 31x31 sujeto a la disponibilidad de recursos de área en la FPGA. En nuestro caso particular se utiliza una FPGA Kintex 7 XC7K325T, la cual soporta un arreglo de hasta 12x12 PEs. Con lo cual se consigue emular hasta 1152 neuronas con 25,334 conexiones sinápticas programables por chip (144 locales y 32 globales por PE), utilizando 7 niveles de virtualización más un nivel principal (Virt=0).

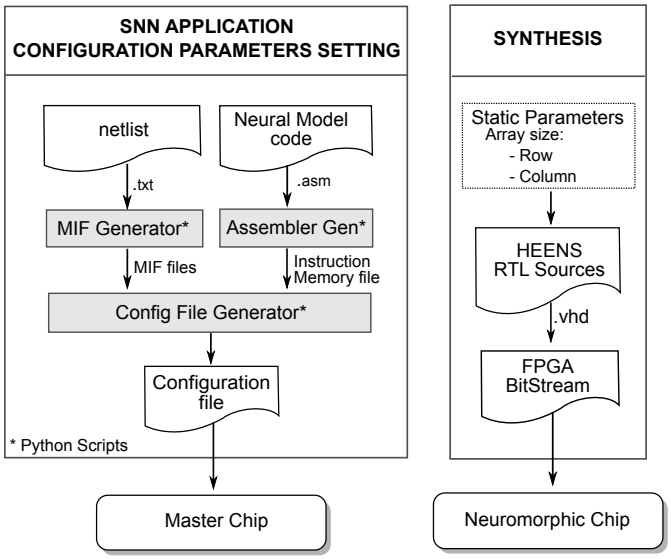


Figura 3.12: Flujo de diseño para una aplicación SNN

3.4.1. Consumo de área

En la Tabla 3.4 se lista la estimación de consumo de recursos para el arreglo de 12x12 de las tres instancias principales de HEENS-MP.

Tabla 3.4: Ocupación de HEENS-MP par un arreglo de 12x12 PEs

Lógica	Array PEs	Control Unit	AER-SRT Controller	Disponible
Slice Registers	70848 (17.4 %)	299 (0.1 %)	1684 (0.4 %)	407600
Slice LUTs	179920 (88.3 %)	347 (0.2 %)	1259 (0.6 %)	203800
RAM36/FIFO	432(97.1 %)	0.5(0.1 %)	3.5 (0.8 %)	445
DSP	144 (17.1 %)	0 (0 %)	0 (0 %)	840

Se puede observar que el arreglo de PEs es el que ocupa casi la totalidad de los recursos de RAM y LUTs de la FPGA limitando el crecimiento de la arquitectura. La recursos empleados por los otros módulos es mínima gracias a la utilización de un esquema SIMD. Es importante destacar que la utilización del array varía en proporción a su tamaño.

En la Tabla 3.5 se presenta los recursos consumidos por cada uno de los bloques funcionales que conforma un PE. Como se describió en la sección 3.2.2 éstos corresponden a la unidad aritmético-lógica (ALU), el bloque de registros (REG), el esquema de memoria asociativa para la decodificación de spikes locales y globales, la memoria sináptica/neuronal (AM), en Otros se catalogan los utilizados para la implementación del LFSR y del multiplexor de selección de datos, así como lógica usada por los registros de pipeline insertados en caminos críticos para evitar violaciones de tiempo.

Tabla 3.5: Consumo de recursos por PE.

Lógica	ALU/PE	REG/PE	AM+SNRAM /PE	Otros/PE	Total
Slice Registers	0 (0 %)	256 (53 %)	133 (27 %)	389 (21 %)	492
Slice LUTs	439 (36 %)	152 (13 %)	172 (14 %)	763 (37 %)	1213
RAM36/FIFO	0 (0 %)	0 (0 %)	3 (100 %)	0 (0 %)	3
DSP	1 (100 %)	0 (0 %)	0 (0 %)	0 (0 %)	1

Tabla 3.6: Consumo de potencia estimado de las principales instancias de HEENS-MP para un arreglo de 12x12

Instancia	Consumo de Potencia (W)
Array PEs	2.82 (82.3 %)
Control Unit	0.007 (<0.2 %)
AER-ST Controller	0.246 (7.18 %)
GTX	0.228 (6.65 %)

Se puede observar en la gráfica que la ALU y la lógica del mux y el LFSR son los más representativos en el consumo de LUTs. Así también, los bloques de memoria del PE son otro factor limitante, ya que las 3 BRAMs empleadas por AM y SNRAM se escalan con el número de PEs del arreglo, resultando en una ocupación del 97.1 % de los recursos de la FPGA para el tamaño máximo soportado. Cabe resaltar que tanto la virtualización como la complejidad del modelo neuronal implementado no influyen en el consumo de área de la FPGA. Esta es una de las características destacables de HEENS.

3.4.2. Consumo de potencia

La potencia estimada del NC para un arreglo de 12x12 PEs es de 3.45W. En la Tabla 3.6 se muestra el consumo de las instancias principales obtenido con la herramienta XPower Analizer de Xilinx.

De acuerdo a los datos, es evidente que el arreglo de PEs es el bloque que requiere más energía. Considerando que cada PE tiene un consumo medio de 0.0196 W, en la Tabla 3.7 se muestra la proporción de gasto de cada una de las instancias que lo conforman.

Tabla 3.7: Consumo de potencia estimado de un PE

Instancia /PE	Consumo de Potencia (W)
ALU	0.003 (15.3 %)
REG	0.002 (10.2 %)
AM+SNRAM	0.01 (51.02 %)
Otros	0.004 (20.4 %)

De los datos obtenidos se observa que los bloques que conforman el esquema de Memoria Asociativa y la SNRAM son los que tienen mayor consumo dinámico el cual es derivado principalmente por el uso de las BRAM. Esta es una penalización que se tiene que pagar a cambio de la flexibilidad de tener

conexiones y parámetros dinámicamente programables.

3.5. Contribuciones en HEENS respecto a la arquitectura SNAVA previamente reportada

Para efectos de comparación con la arquitectura SNAVA [4], versión anterior de HEENS, en la Fig.3.13 se muestran los resultados de la implementación de un NC de 10x10 (tamaño máximo soportado por SNAVA). En SNAVA el alto consumo de LUTs respecto al resto limita el número de neuronas a 200 y sinapsis a 100 por neurona. Su porcentaje de utilización principalmente corresponde a la demanda de recursos de cada PE. En cada PE dicho consumo se divide entre lo empleado por la ALU y por la lógica que definen las conexiones sinápticas. El uso de lógica tiene la ventaja de su bajo consumo de potencia. Sin embargo la conectividad es fija y cualquier cambio requiere volver a sintetizar la arquitectura restando gran flexibilidad al sistema. Como resultado, con HEENS-MP se consigue 5.76 veces más neuronas y 2.56 veces más conexiones sinápticas que SNAVA. Además, si se considera que SNAVA no posee un modelo de comunicación jerárquico, y se toma en cuenta únicamente conexiones locales, HEENS-MP podría sintetizar hasta 1352 neuronas, es decir un factor de 6.76 veces más que SNAVA.

El número de DSPs se mantienen constante debido a que no ha cambiado la forma de implementar los multiplicadores. En SNAVA el número de LUTs es variable, dependiendo del conexionado y dado el caso de un sistema grande podría fallar la síntesis, en HEENS-MP la ocupación es determinista.

HEENS-MP, por otro lado ha reducido el consumo de LUTs un 19 % respecto a SNAVA. Este todavía se mantiene alto no obstante, en este caso se debe principalmente a lo ocupado por la ALU/PE. Se debe considerar que se ha añadido nuevas instrucciones que inciden directamente en la ocupación de la ALU y el secuenciador aumentando la lógica combinacional de estas instancias. La principal diferencia está en el número de bloques de RAM (BRAM) destinados a almacenar las conexiones neuronales locales y globales de cada PE. Es así que el consumo de LUTs y BRAMs son ahora los factores limitantes de la nueva arquitectura como se mencionó en la sección anterior. Se descata que se ha conseguido mayor flexibilidad, programabilidad de conexionamiento, y un balance en el uso de recursos de la FPGA obteniendo máximo partido de ellas.

Tal como se ha indicado, una aplicación en SNAVA, una vez sintetizada no puede cambiar sus conexiones sinápticas, éstas se mantienen fijas debido a que se implementan utilizando recursos de LUTs en la FPGA. Además que el tiempo de síntesis crece linealmente con el número de conexiones sinápticas, reportando hasta 17 horas para arreglos de 10x10 con 100 sinapsis por PE. En el caso de HEENS-MP las conexiones sinápticas son programables y se implementan en bloques de memoria RAM on chip. Las sinapsis programables dan soporte a aplicaciones evolutivas al permitir un cambio dinámico de la topología sin necesidad de re-sintetizar la descripción VHDL. Así también el tiempo de síntesis se reduce en un factor de 34 veces el tiempo de SNAVA como se reporta en la Tabla 3.8.

Para las condiciones mencionadas se obtuvo el reporte de consumo de potencia para ambas arquitecturas. Los resultados se presentan en la Fig 3.15 y Fig. 3.14. Los datos obtenidos reflejan

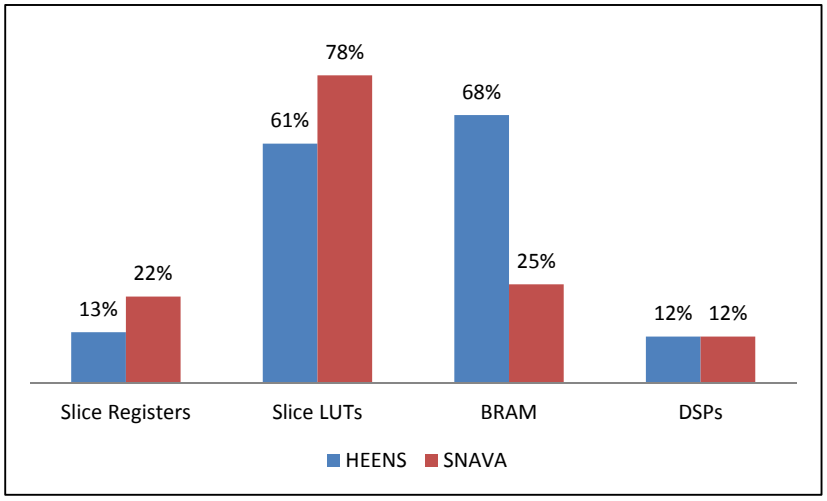


Figura 3.13: Comparación de recursos entre SNAVA utilizando 2 niveles virtuales y HEENS-MP con un nivel de virtualización (El consumo de recursos de HEENS no varía en función de los niveles virtuales)

Tabla 3.8: Comparación entre SNAVA y HEENS-MP para un arreglo de 10x10 PE

Lógica	SNAVA	HEENS-MP
Consumo de Potencia	1.39 W	2.563 W
Tiempo de síntesis	17 h 35 min	10 min
Tiempo de implementación	39 min	27 min

precisamente el alto consumo de registros en SNAVA que independientemente de su menor consumo de potencia, limitan completamente su desempeño. Por contrapartida, HEENS-MP consume un 84.3 % más de potencia que SNAVA, siendo las BRAMs el recurso que más contribuye al consumo dinámico. Aquí se presenta un compromiso entre flexibilidad de la arquitectura (programabilidad) y el consumo de energía. Es importante mencionar que las cifras obtenidas representan estimaciones de consumo de potencia con poco fiabilidad. Además, las BRAMs sinápticas sólo se usan para decodificar spikes, una fracción de tiempo muy corta del ciclo. Entre otras diferencias destacables tenemos que HEENS-MP soporta un esquema de comunicación jerárquico mediante la diferenciación de conexiones locales y globales para reproducir configuraciones estructurales inter-neuronales e inter-regionales. Además, permite modelar un rango programable de retardos axonales que imitan características biológicas temporales de propagación a lo largo del córtex. La contribución de información temporal está fuertemente relacionada con los procesos de aprendizaje [5], por lo cual la implementación de estos permitirá emular aplicaciones con mayor realismo biológico.

Se cuenta con mayor flexibilidad para determinar los espacios de memoria donde se almacenan los datos del procesamiento neuronal y sináptico en cada PE (memoria SNRAM). Su direccionamiento se programa por software en lugar de usar posiciones fijas, con la ventaja de que futuras modificaciones a la arquitectura serán más sencillas y conllevarán menos cambios.

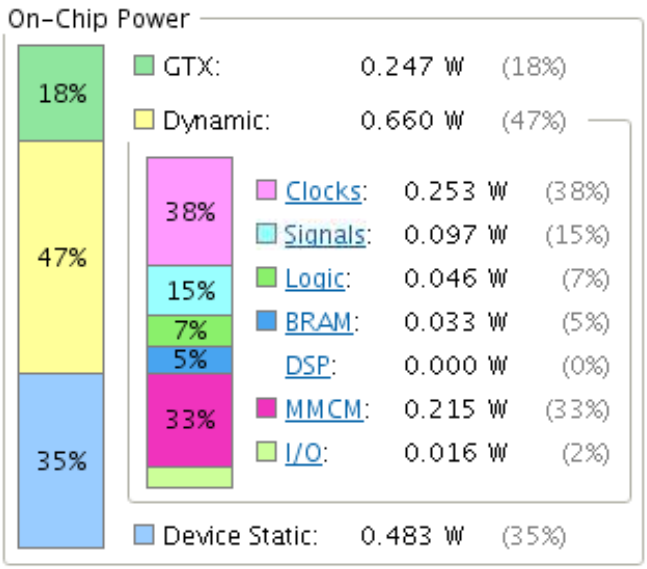


Figura 3.14: Consumo estimado de potencia en SNAVA. Obtenido con la herramienta de XPower Analizer de Xilinx

Se añadieron las instrucciones BITSET, BITCLR, SHLAN, SHRAN para facilitar y simplificar la programación al usuario. Realizar estas operaciones con SNAVA requiere de varias instrucciones además se puede necesitar la misma operación varias veces en el programa. La operación MUL (multiplicación con signo) se modificó para que presente su resultado de 32 bits en dos registros ACC y R1 en lugar de uno como anteriormente estaba implementado. Esto permite obtener mayor precisión para el cálculo de los algoritmos. Estos cambios permiten ahorrar ciclos de ejecución ya que el secuenciador tendrá que gestionar menos instrucciones en el programa.

Además se añadió la instrucción LOADBP para controlar el puntero de la SNRAM con lo cual es posible extraer los datos de cualquier posición de memoria. Una de las principales ventajas es la lectura de los parámetros sinápticos y neuronales para cada nivel virtual. El sequencer se encarga de controlar los saltos entre niveles, y por medio de esta instrucción se apunta al bloque de datos correspondiente. Esto permite por un lado, tener la flexibilidad de escoger el número de sinapsis por nivel virtual sin tener la limitación del mismo número de sinapsis para todos los niveles virtuales como es el caso de SNAVA; y por otro eliminar la necesidad de un banco de registros de sombra por nivel virtual como anteriormente se requería. Las instrucciones LOOPV, LAYERV, INCV se crearon para manejar esta nueva forma de implementar virtualización en el arreglo de PEs.

Los niveles de virtualización que el usuario decida asignar no influye en el consumo de recursos de la FPGA. Por el contrario en SNAVA la virtualización se implementa de la misma forma que las conexiones sinápticas, es decir en base de lógica programable. Esto implica que el consumo de LUTs crece también en función de los niveles virtuales.

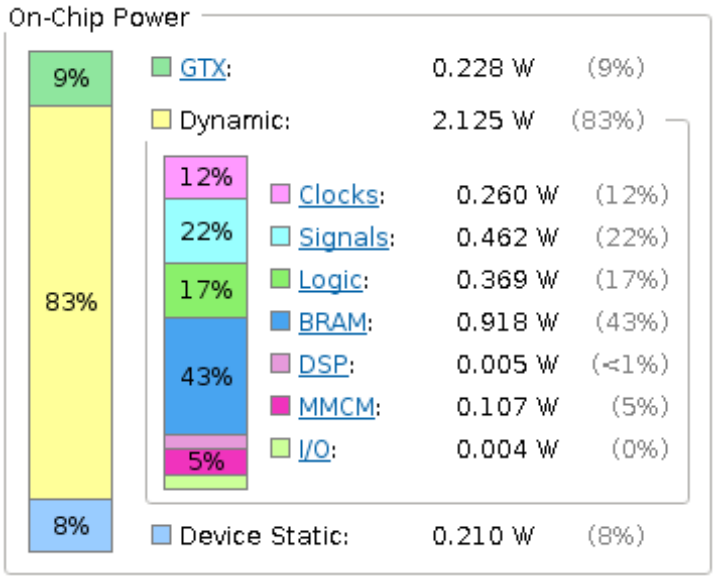


Figura 3.15: Consumo estimado de potencia en HEENS-MP. Obtenido con la herramienta de XPower Analizer de Xilinx

3.6. Conclusiones

EL multiprocesador de HEENS es introducido en este capítulo. Las características que lo distinguen son su alto grado de programabilidad , uso eficiente y balanceo de recursos. Su esquema de procesamiento SIMD permite lograr un procesamiento paralelo masivo de los PEs para la ejecución de cualquier algoritmo neuronal o sináptico, con una resolución limitada por el ancho de su franja temporal.

Su set de instrucciones y los programas desarrollados para definir la configuración de HEENS-MP a partir de simples archivos de texto (net-list) proporciona a los usuarios una plataforma que permite desarrollar aplicaciones neuronales optimizando el tiempo, sin necesitar de tener avanzados conocimientos en hardware, ni esperar por largos tiempos de síntesis. Una vez que se ha definido el tamaño del arreglo, la configuración de la aplicación neuronal se realiza a nivel de escritura de datos en memoria, para lo cual no se requiere re-sintetizar la arquitectura.

El esquema de memoria con dos niveles de jerarquía locales y globales, habilita la comunicación inter e intra modular que además, da soporte al escalamiento de la arquitectura a través de su modelo de comunicación. El uso de memorias on-chip permite rápidos tiempos de acceso y menor consumo de potencia en relación con otras arquitecturas que utilizan memoria externa.

Un aspecto a resaltar es que el número de conexiones sinápticas está limitado por el tamaño de la memoria, no obstante el esquema de memoria asociativa utilizado permite minimizar la cantidad de memoria por PE. Así también, se da soporte para reconfiguración de la red neuronal en su modo de evolución, al permitir realizar cambios dinámicos on-line de cualquier parámetro de la red.

Referencias

[1] J. Madrenas y J. M. Moreno, "Strategies in SIMD computing for complex neural bioinspired applications", *Proceedings - 2009 NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2009*, págs. 376-381, 2009. DOI: 10.1109/AHS.2009.31.

[2] R. Brette y D. F. M. Goodman, "Simulating spiking neural networks on GPU.", *Network (Bristol, England)*, vol. 23, n.º 4, págs. 167-82, 2012, ISSN: 1361-6536. DOI: 10.3109/0954898X.2012.730170. dirección: http://www.ncbi.nlm.nih.gov/pubmed/23067314.

[3] M. Zapata y J. Madrenas, "Compact associative memory for aer spike decoding in fpga-based evolvable snn emulation", *International Conference on Artificial Neural Networks, ICANN*, págs. 399-407, 2016. DOI: 10.1007/978-3-319-44778-0_47.

[4] G. Sanchez, "Efficient multiprocessing architectures for spiking neural network emulation based on configurable devices", Tesis doct., Universitat Potecnica de Catalunya, 2014.

[5] P. S. Churchland y T. J. Sejnowski, "The computational brain", 1992.

Capítulo 4

Modelo de Comunicación AER-SRT

En este capítulo se presenta el Modelo de Comunicación AER-SRT (Address Event Representation over Synchronous Serial Ring Topology) utilizado para establecer la comunicación Multi-Chip de la plataforma HEENS. En las siguientes secciones se presenta los paquetes de señalización utilizados en el protocolo AER-SRT. La estructura del MC y sus funciones se detallan en la Sección 4.2. En la Sección 4.3 se estudia el AER-SRT implementado en los Chips Neuromórficos y se remarcan las diferencias respecto al del MC. Finalmente se concluye con las características de implementación del modelo de comunicación de HEENS.

4.1. Protocolo AER-SRT

Tal como se ha indicado en la sección 1.6.1, el bus AER se utiliza para transmitir eventos neuronales de forma eficiente, codificando la dirección de la neurona presináptica. El bus AER original [1] es paralelo y asume que hay un único transmisor y múltiples receptores, lo que impide claramente la escalabilidad al aumentar la capacidad de las líneas cuando se incorporan nuevos receptores al bus. Además, la arquitectura HEENS requiere que cada nodo del bus sea transmisor y receptor. A diferencia del AER original, el bus propuesto como modelo de comunicación de HEENS utiliza el protocolo síncrono basado en paquetes AER-SRT [2] para permitir la comunicación de eventos AER en una plataforma SNN multi-chip conectada en anillo, funcionando en pipeline de forma síncrona, lo que permite una gran escalabilidad y eficiencia de transmisión al emplear conexiones serie de alta velocidad. Los eventos AER son asignados a slots temporales por lo cual, su resolución se limita por el ancho de los mismos. Esto además le confiere la ventaja de ser un esquema libre de colisiones.

Así también, como se puede observar en la Fig. 4.1 **??**, la conexión entre nodos es punto a punto y unidireccional mediante enlaces seriales de alta velocidad. Se utiliza el protocolo de Xilinx Aurora para la serialización y deserialización de los paquetes que circulan por el anillo. Las operaciones de transmisión y recepción se realizan a través de 2 controladores: el Z_AER SRT implementado en el MC y el AER-SRT del NC. Dichos controladores, (detallados en las siguientes secciones) están conformados por: un módulo transmisor, un receptor y las respectivas FIFO, utilizadas para almacenar y enviar datos

entre diferentes dominios de reloj.

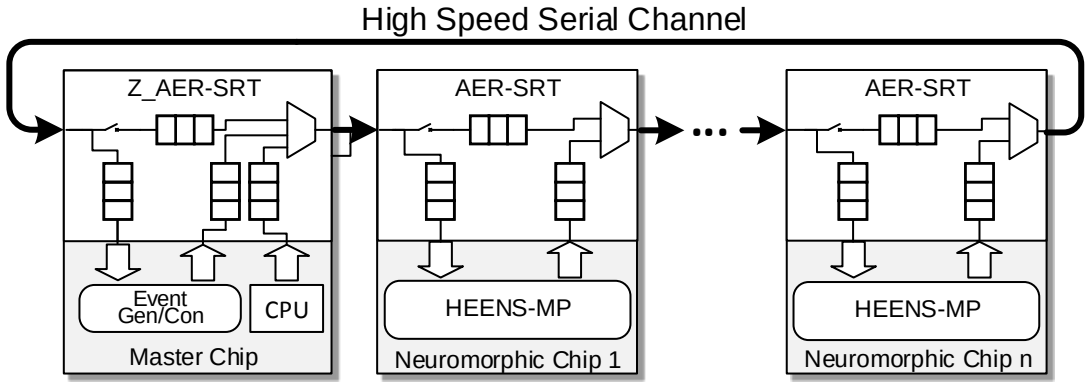


Figura 4.1: Modelo de comunicación AER-SRT

Una limitación de los sistemas multi-FPGA, entre ellos SNAVA, es que cada FPGA se debe configurar independientemente. En caso de configuraciones estáticas, aunque engorroso puede ser viable, pero cuando se desea que la red evolucione, se debe reconfigurar en tiempo real, por lo que se hace imprescindible que todos los nodos sean accesibles de forma centralizada. Con este propósito se ha generalizado el uso del bus AER, que se emplea no sólo para transmitir spikes sinó que se realizan funciones de inicializaición, control y reconfiguración. El Master Chip desempeña un rol principal en la red, ya que es el encargado de controlar varias fases de funcionamiento del protocolo, como son: la inicialización del anillo, la configuración de la aplicación neuronal en los NCs, y la evolución on-line dinámica de cualquier nodo. La estructura del Z_AER-SRT es muy similar a la presentada en los NCs, no obstante incorpora funcionalidades distintas que le permiten gestionar las tareas antes mencionadas.

4.1.1. Paquetes de señalización del protocolo AER-SRT

El protocolo de AER-SRT diferencia entre dos tipos de información: datos y control identificados por el bit más significativo ($D/\bar{C}$) de cada paquete de 16 bits. Los paquetes de datos encapsulan los eventos de spikes generados en un NC en fase de ejecución (EPh). A su vez los paquetes de control listados en la Tabla 4.1, son utilizados para la sincronización de las fases en las que interviene la transmisión de paquetes por el bus AER.

Cada una de las fases de inicialización, configuración y evolución, maneja su propia trama de datos delimitada por sus respectivos paquetes de señalización. Estas permiten al AER-SRT sincronizar sus operaciones e identificar el tipo de dato que llegan por el bus de comunicación. Cabe mencionar que cuando HEENS-MP se encuentra procesando el algoritmo neuronal es decir en EPh, por el bus circulan paquetes IDLE con el fin de mantener el enlace activo.

Tabla 4.1: Paquetes de control del protocolo AER-SRT

Paquetes de Control	Descripción
IDLE	Mantiene activo el enlace
INIT EOINIT	Fase de inicialización
CONF EOCONF	Fase de configuración y evolución
EVOL	Fase de evolución
SYNC START FINISH	Fase de distribución

Trama de Inicialización (Fase IPh)

AER-SRT emplea dos parámetros para configurar el anillo en cada NC: Ring Size e ID. El Ring Size define el número de nodos en el sistema, y el ID es usado para identificar el origen de cada evento que circula por el enlace. El MC se encarga de configurar dichos parámetros en todos los nodos antes de iniciar la ejecución de la aplicación neuronal.

Como se introdujo en el capítulo anterior, en la IPh se realiza la asignación dinámica de ID y Ring Size en cada NC para configurar el anillo. Estos son considerados como paquetes de datos y son transmitidos por el MC con sus respectivos paquetes de señalización como se muestra en la Fig. 4.2.

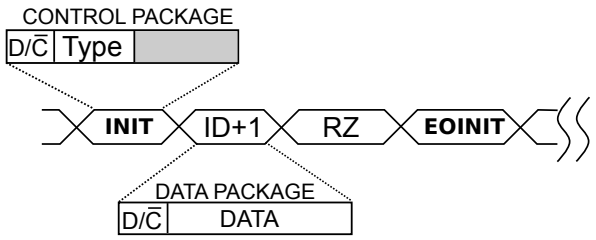


Figura 4.2: Trama de inicialización del anillo enviada únicamente por el MC

Se parte de la premisa de que el ID del MC es siempre 1. Así, cuando un NC recibe una trama de inicialización, este se configura con los datos de ID+1 y Ring Size recibidos. A su vez antes de que cada NC retransmita la trama, se suma uno al ID (ID+1) para la configuración del siguiente nodo. Se repite el mismo proceso hasta configurar el anillo completo. El MC reconoce que todos los chips se han inicializado cuando recibe de vuelta el paquete EOINIT.

Trama de Configuración (Fases CPh y EvPh)

Esta trama (ver Fig 4.3), utilizada en las fases de configuración y evolución es enviada por el MC. Cumple doble función debido a que puede contener datos que configuran la aplicación neuronal, así como los que se envían para la reconfiguración de la red. En ambos casos los delimitadores son los mismos.

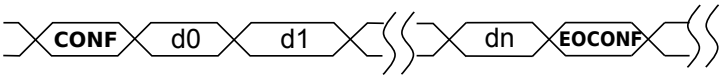


Figura 4.3: Trama de configuración enviada por el MC

- *Datos de Configuración*

  La configuración de todos los NCs se envía en una sola trama. Esta se exporta desde un único archivo formado por palabras de configuración (direcciones y datos) usados para la inicialización de las memorias en cada NC del anillo. Cabe señalar que se envía información a las 4 memorias (Locales, Sinap/Neuronal, Codificación y Conversión) de cada PE independientemente. El número de PE depende del tamaño del arreglo en cada HEENS-MP. Para el mapeo de datos el archivo de configuración incluye información del ID destino (DESTINATION ID), así como la posición del PE en el arreglo en el campo de dirección de la palabra de configuración. El formato utilizado se muestra en la Fig. 4.4. En las sección 3.2.1 se muestran las tablas de mapeo utilizadas para la codificación del archivo de configuración.

  Como se observa el inicio de cada bloque de datos está delimitado por la palabra de configuración que contiene el ID IDENTIFIER y ID del NC destino. El ID IDENTIFIER es un código único de dirección asignado para identificar los ID destino. Si este último corresponde al ID del Master significa que los datos se destinan a todos los NCs (Configuración en Modo Común). Esta característica es particularmente útil e.g. si más de un NC ejecutan el mismo algoritmo. En este caso la información de la memoria de instrucciones se podría enviar en Modo Común.

- *Datos de Aprendizaje*

  Se utiliza el paquete de control EVOL para informar a cada NC el estado ON/OFF (activado o desactivado) de la EvPh. Dicho paquete es insertado por el MC en cada ciclo de emulación circulando por todo el anillo hasta regresar a su origen. Cuando el MC activa la EvPH, este inserta un campo de señalización en el paquete EVOL que notifica que la EvPh está ON y que la próxima trama enviada por el MC corresponderá a una de configuración. En este caso todos los NCs de la red entran en modo de espera hasta recibir dicha trama. Cuando los datos arriban, se verifica su destino a través del ID IDENTIFIER y el DESTINATION ID. Paralelamente los paquetes se retransmiten al siguiente nodo.

Al igual que la trama de inicialización, el MC detecta que ha terminado la configuración/reconfiguración cuando recibe de vuelta el paquete EOCONF. Este es retransmitido a medida

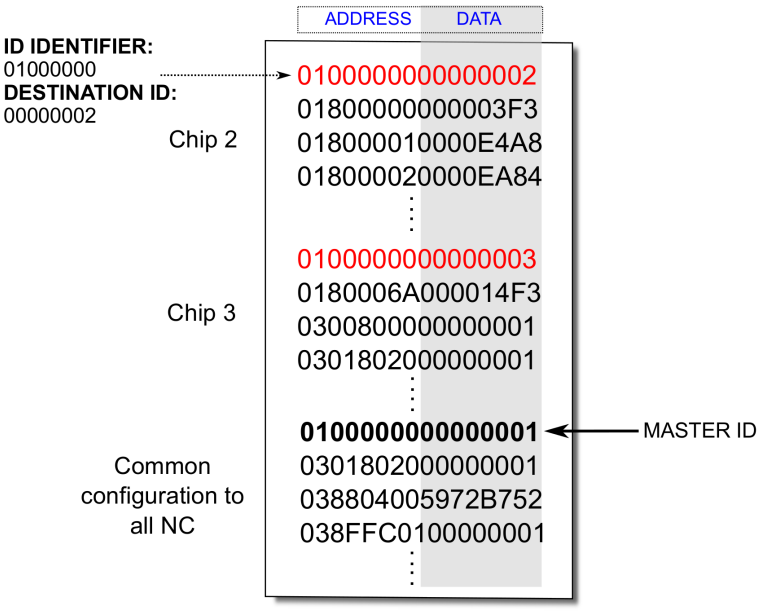


Figura 4.4: Formato de archivo de configuración

que se propaga por todos los nodos hasta llegar al origen, donde es finalmente descartado.

Trama de Distribución (Fase DPh)

Cuando el anillo se encuentra inicializado y configurado, el Sequencer del NC recibe la orden de empezar el procesamiento de la aplicación neuronal procediendo a la ejecución del algoritmo sináptico/neuronal en el arreglo de PEs. Debido a que cada NC de la red termina su EPh en diferente tiempo dependiendo del algoritmo procesado, se deben sincronizar los nodos para asegurar que todos hayan finalizado su ejecución y estén listos para el volcado de datos en el canal, antes de iniciar la transmisión.

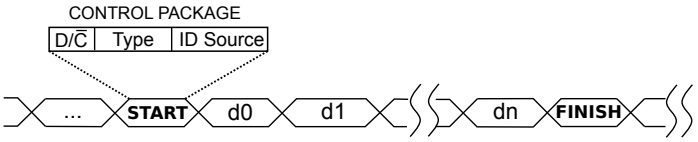


Figura 4.5: Trama de distribución enviada por cada NC

Es así que cuando un NC inicia la fase de distribución, este envía un paquete SYNC indicando que está listo para transmitir y, posteriormente propaga los SYNC que recibe de los otros nodos. Cuando se detecta que el número de SYNC recibidos es igual al tamaño del anillo (Ring Size), significa que los nodos se han sincronizado y puede empezar la distribución de eventos AER, mismos que se encapsulan en la trama mostrada en la Fig 4.5.

La trama tiene como cabecera un paquete START, utilizado para identificar el ID del NC donde se originaron los spikes. De esta forma se reduce el payload (carga útil) y la latencia del mensaje. Así, sólo se transmite el ID al inicio de cada trama de distribución, ya que este campo es común para todos los spikes de su respectivo NC. Los datos correspondientes a los eventos de spike son enviados secuencialmente, y circulan por todo el anillo hasta regresar al NC origen donde fueron generados, para posteriormente ser eliminados. De esta manera, cada vez que se recibe un paquete START, este se compara con el ID del NC correspondiente. Si existe coincidencia, significa que los datos han terminado de circular por el anillo, por lo cual son descartados.

Al final de la trama de distribución se envía un paquete FINISH. Este se procesa de la misma forma que los paquetes SYNC, es decir si el número de FINISH recibidos es igual al Ring Size, los nodos sabrán que termina la fase de distribución.

4.2. Master Chip

En la arquitectura HEENS, el MC gestiona las IPh, CPh y EvPh actuando como Master, despachando los datos correspondientes junto a su respectivos paquetes de señalización a los NCs. El resto del tiempo trabaja en modo Punto a Punto (Point to Point, P2P) distribuyendo eventos de spikes en la red neuronal. Como se observa en la Fig. 4.6, está conformado por:

- CPU Core
- PS_PL Interface
- Spike Generator/Consumer
- Z_AER SRT Controller

Mediante, las señales de control se verifica el inicio y finalización de las fases correspondientes.

4.2.1. CPU Core

Lee el archivo de configuración almacenado en una memoria externa. Este archivo contiene los parámetros de configuración de la aplicación neuronal que se procesa en cada NC. A través de una interface GUI, el usuario es el encargado de iniciar la transmisión de los datos de configuración, mismos que son depositados en una FIFO (CONFIG_FIFO) para su posterior transmisión por el bus AER.

4.2.2. PS_PL Interface

Lee los datos del CPU Core, sin embargo debido a que este trabaja con un protocolo diferente al utilizado por el resto de módulos del sistema, los datos de configuración extraídos de la memoria externa deben convertirse al tipo de dato establecido por el protocolo AER-SRT para que puedan ser transmitidos por el bus.

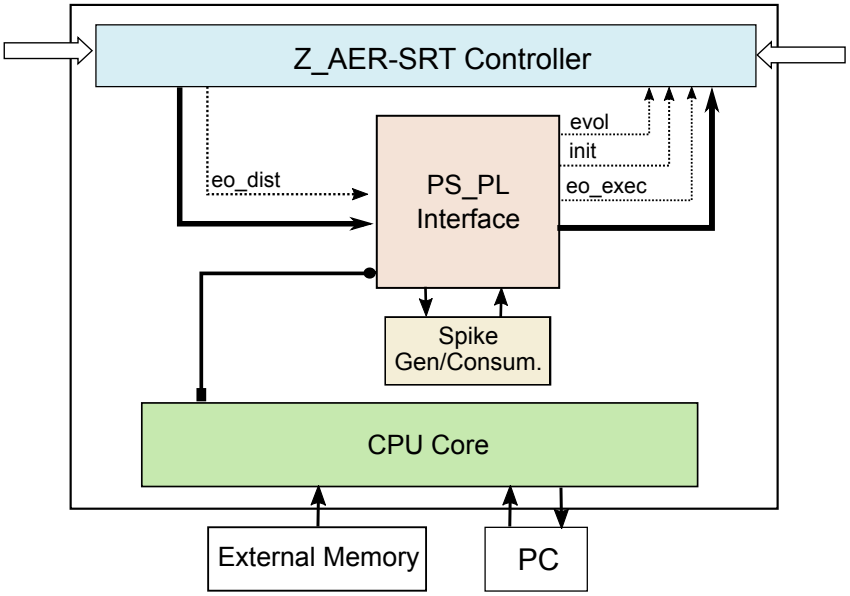


Figura 4.6: Estructura del Master Chip

Así también, el PS_PL Interface administra el envío de datos de: inicialización del anillo, configuración y los generados por el Spike GenConsumer por medio de las señales de control, para que sean entregados en sus respectivas fases.

Para dotar de flexibilidad al MC, se emplea un dispositivo ZYNQ en la placa de evaluación Picozed. Los dispositivos ZYNQ incorporan una parte de lógica programable (PL), un procesador ARM de doble núcleo (PS) y diversos periféricos configurables. El PS simplifica y hace versátil la comunicación del sistema con el exterior y también da soporte de cálculo a las operacioens de evolución que el usuario desea programar en función del estado de la red neuronal, ya que mediante el bus AER, el MC visualiza todos los spikes que se generan. La PL actúa de interfaz con el anillo, implementando el controlador Z_AER-SRT

4.2.3. Spike Generator/Consumer

Se trata de un bloque que genera o recibe spikes de forma programable sin necesidad de implementar el multiproesador. Es utilizado para la depuración del tráfico de entrada y salida, así como para insertar spikes a la red neuronal y transmitirlos a los NCs.

4.2.4. Z_AER SRT Controller

Está conformado por los módulos mostrados en la figura Fig. 4.7, de los cuales las principales instancias son el Z_AER RX y el Z_AER TX. Este último habilita los estados que permiten el control de las operaciones llevadas a cabo por el Master. A continuación se describen sus funciones.

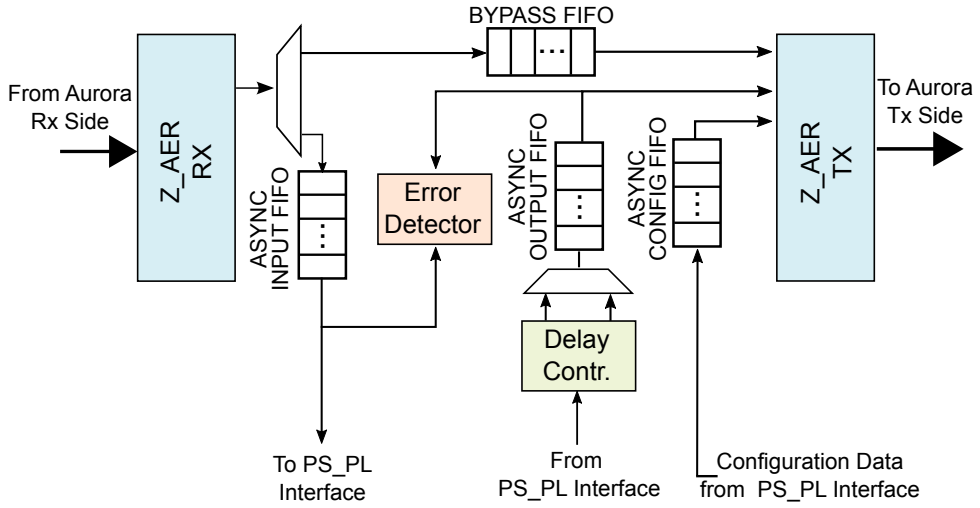


Figura 4.7: Z_AER SRT Controller

- Z_AER RX: Se encarga de leer y procesar el tráfico recibido desde el Aurora Rx Side para detectar los paquetes AER de datos y control en función de los cuales se realizará el enrutamiento de los datos a las FIFOs respectivas.

  En este módulo también se detecta la finalización de las fases de IPh y CPh mediante la detección de los paquetes EOINIT Y EOCONF. De igual manera se detecta el inicio y finalización de la sincronización de los nodos para la distribución de spikes, mediante los paquetes SYNC y FINISH respectivamente. Mediante contadores se verifica si el número de SYNC/FINISH recibidos es igual al Ring Size, de ser el caso se activan/desactivan las señales de control correspondientes utilizadas por el resto de módulos para habilitar sus operaciones.

  Cuando se recibe una trama de distribución con un bloque de datos AER, se extrae el ID del paquete de control START para añadir este campo a la dirección AER y enviarla a la INPUT FIFO. Al mismo tiempo se verifica si el ID recibido es igual al ID del MC. Si la condición es verdadera significa que los paquetes son propios, es decir, han terminado de circular por el anillo y han regresado al origen. En este caso no son escritos en la BYPASS FIFO para ser retransmitidos. La misma condición se aplica para los paquetes de control.

- Z_AER TX: Se encarga de generar las tramas AER que serán volcadas al AURORA Tx Side dependiendo del estado del Tx Controller que se muestra en la Fig. 4.8.

  Las tramas de inicialización, configuración y distribución con sus respectivos paquetes de señalización se generan y transmiten en este módulo. Aquí se procesan los siguientes estados:

  1. Se envía la trama de inicialización cuando se detecta que el enlace de Aurora está activo y listo para transmitir.
  2. Cuando se recibe la notificación que la IPh ha terminado, se transmite la trama de

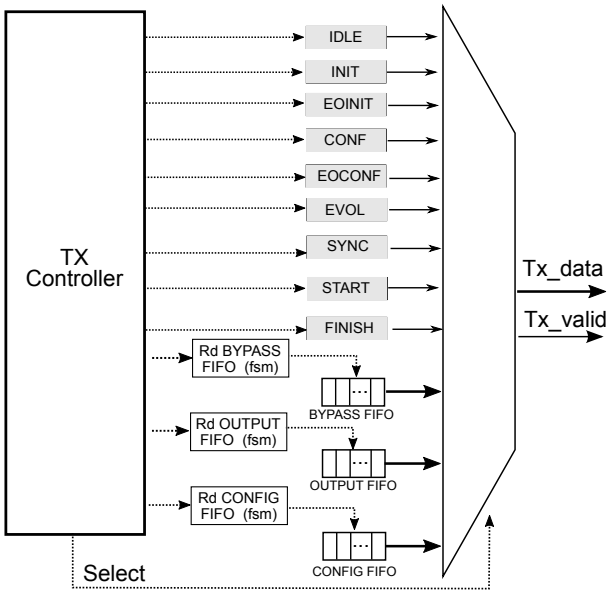


Figura 4.8: Módulo Z_AER TX

configuración para configurar todos los nodos. Los datos se leen de la CONFIG FIFO. El MC permanece en estado de IDLE hasta recibir la notificación que la CPh ha finalizado antes de pasar al siguiente estado.

3. Se transmite un paquete EVOL para notificar el estado ON/OFF de la fase de evolución al anillo. El MC inserta la señalización correspondiente en este paquete antes de ser enviado. Si el MC activa la EvPh se envía la trama de configuración.
4. Se transmite un paquete SYNC para propósitos de sincronización de nodos.
5. Se transmite el contenido de la BYPASS FIFO que almacena los paquetes SYNC recibidos de los otros NCs.
6. Una vez recibida la notificación de anillo sincronizado, se envía la trama de distribución. Los datos se leen de la OUTPUT FIFO.
7. Se transmite el contenido de la BYPASS FIFO que almacena los eventos de spikes recibidos de los otros NCs.
8. Se coloca en espera hasta detectar que la distribución de eventos ha finalizado e iniciar la EPh.
9. Una vez termina la EPh se vuelve a repetir el proceso anterior desde el punto 3.

Los receptores respectivos (Z_AER RX y AER RX) son los encargados de contar los paquetes SYNC para detectar cuando se ha completado la sincronización del anillo. En los estados de espera, se envía paquetes IDLE para mantener activo el enlace de Aurora.

La reconfiguración que se realiza en la fase de evolución cuando su estado es ON, no requiere que los nodos estén sincronizados, pero si que hayan finalizado su fase de ejecución (EPh).

El resto de instancias cumplen las mismas funciones que en el AER-SRT Controller, que se explica a continuación.

4.3. Chip Neuromórfico - AER-SRT Controller

Los NCs siempre actúan como esclavos en la red. Se comunican a través del AER-SRT Controller cuya estructura se muestra en la Fig. 4.9 y se detalla a continuación:

4.3.1. OUTPUT e INPUT FIFO Asíncronas

Estas permiten la comunicación entre los dos dominios de reloj que utiliza el hardware. La OUTPUT FIFO se encarga de almacenar temporalmente los eventos de spikes que están listos para ser transmitidos por el canal, es decir aquellos con un retardo axonal igual a cero. De forma similar, la INPUT FIFO almacena los eventos de spike post-sinápticos provenientes de todos los nodos conectados al anillo incluyendo los generados en el mismo chip. Los paquetes de control no se almacenan, únicamente se procesan y retransmiten. Así también, si la información de llegada corresponde a una trama de configuración se verifica el ID para determinar correspondencia. Si no se valida, los datos son encaminados al siguiente nodo sin ingresar a la INPUT FIFO.

4.3.2. AER RX

Al igual que el Z_AER RX, escucha los paquetes recibidos desde el bus AER. Se comprueba si corresponden a paquetes de señalización para determinar cómo deben ser procesados los datos entrantes dependiendo del tipo de trama. En el bus de comunicación, los paquetes circulan por todo el anillo antes de ser descartados, por lo cual una vez son procesados se escriben en la BYPASS FIFO para ser retransmitidos al siguiente nodo.

Este módulo trata los datos recibidos dependiendo de la fase en que se encuentre el NC:

- *En Fase de Inicialización (IPh)*

  1. Los datos recibidos en esta fase (ID+1 y Ring Size) se procesan y se escriben en la BYPASS FIFO para ser retransmitidos.

- *En Fase de Configuración y Evolución (CPh, EvPh)*

  1. Procesa los paquetes de configuración en un pipeline de 2 etapas para formar dirección y dato de la palabra de configuración.
  2. Verifica si la dirección corresponde a un código de ID_IDENTIFIER. Si es el caso, se procede a comprobar coincidencias entre el ID del NC y el recibido como dato. Así también, la

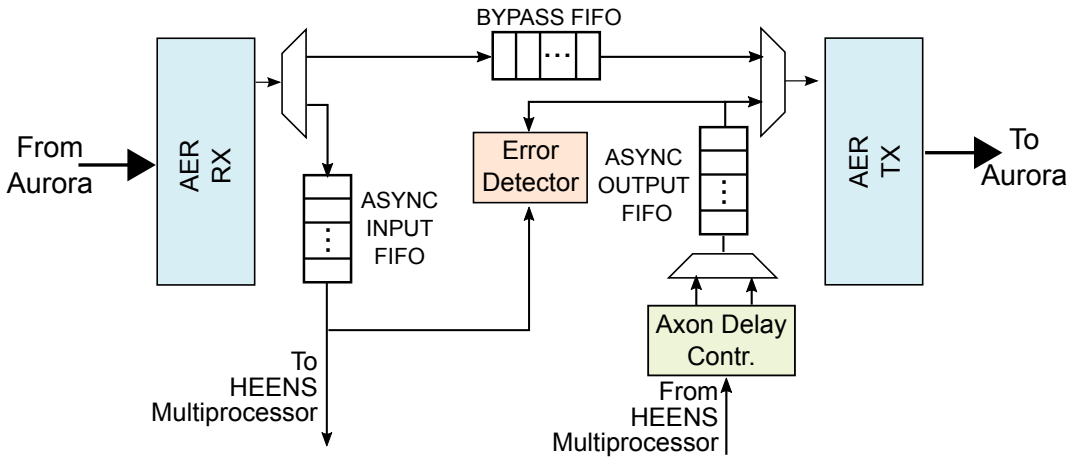


Figura 4.9: AER-SRT Controller

condición se dará como válida si se recibe el ID del MC. Este caso corresponde al Modo de Configuración Común.

3. Si la validación es verdadera, se envía el bloque de configuración al multiprocesador. Caso contrario se continúa comprobando el arribo del ID_IDENTIFIER hasta que termina la CPh.

- *En Fase de Distribución (DPh)*

1. La trama de distribución solamente se propaga en la DPh. Los datos recibidos de esta trama son tratados al igual que el Z_AER RX incluida la sincronización. Los datos AER se envían a la INPUT FIFO para ser posteriormente procesados por el HEENS-MP. Si se comprueba a través del ID que fueron generados por la misma tarjeta, es decir han circulado por todo el anillo no se retransmiten finalizando su circulación por el anillo.
2. Cuando se detecta el paquete EVOL, se comprueba su estado para determinar si existe orden de evolución (EvPh). Dado el caso, el siguiente bloque de datos que se reciba desde el Aurora Rx Side corresponderá a una trama de configuración por lo que serán procesados como tales. En cualquier condición, el paquete EVOL siempre se retransmite al inicio de la DPh, al siguiente nodo para informar su estado.

4.3.3. AER TX

Al igual que el Z_AER TX, se encarga de gestionar el envío de tramas al Aurora Tx Side. Sin embargo, como se muestra en la Fig. 4.10 gestiona menos paquetes de control debido a que se limita a retransmitir la información de configuración enviada por el MASTER. De acuerdo a las fases de operación se HEENS-MP se realizan los siguientes acciones:

- *Fase de Inicialización, Configuración:* Lee el contenido de la BYPASS FIFO para retransmitir tramas de inicialización y configuración. Éstas deben circular por todo el anillo antes de ser descartadas.

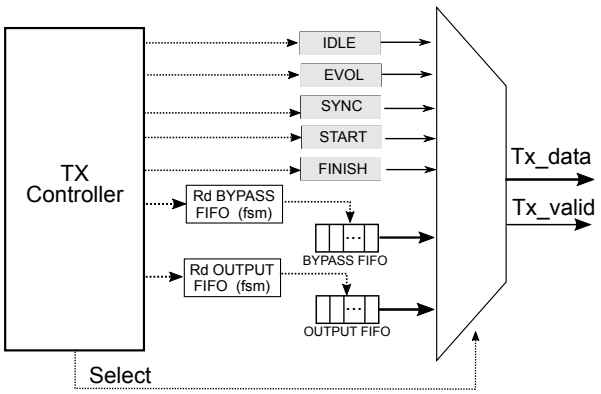


Figura 4.10: Módulo AER TX

- *Fase de Evolución:* Retransmite el paquete EVOL, y la trama de configuración que lo acompaña cuando el MC inicia la reconfiguración de la red.
- *Fase de Distribución:* Una vez sincronizados los NC, se envía la trama de distribución. Se sigue el mismo procedimiento que el Z_AER TX en este caso.

4.3.4. Error Detector

Debido a que el protocolo AER-SRT ha sido diseñado para que los datos circulen por el anillo hasta regresar a su origen, la detección de pérdida o corrupción de información es directa y relativamente simple.

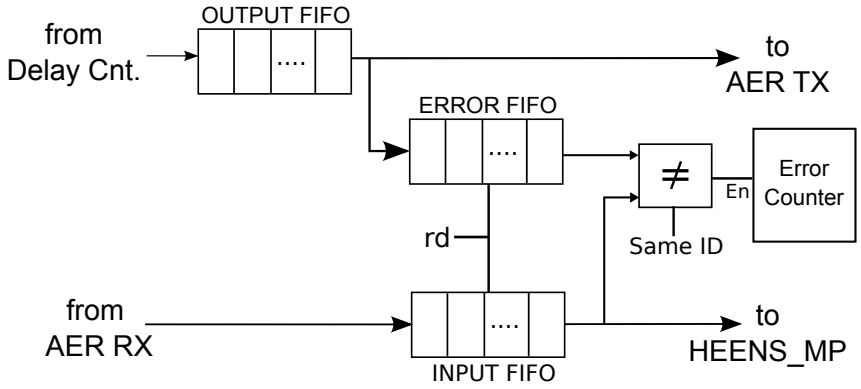


Figura 4.11: Error Detector

Como se observa en la Fig. 4.11, en este módulo se dispone de una ERROR FIFO que almacena una copia del contenido de la OUTPUT FIFO. De esta manera, cuando el AER RX detecta que el mensaje de datos ha regresado al origen, los datos recibidos son comparados con los almacenados en la ERROR

FIFO, determinando así la pérdida o corrupción de datos en el proceso de transmisión. El Error Detector trabaja paralelamente al resto de instancias, con lo cual no añade latencia al proceso de transmisión. La detección de errores se realiza de la misma forma en el Master Chiip.

4.3.5. Axonal Delay Controller

Este módulo permite emular retardos axonales. Es decir que, el spike no se transmite inmediatamente, sino que alcanzará su destino después que hayan transcurrido un cierto retardo que en términos biológicos corresponden a la longitud del axón de la conexión neuronal.

Para esto, la operación del controlador tiene tiene lugar en las EPh y DPh. En la Fig. 4.12 se representa su estructura. Su diseño consiste principalmente de una memoria (RAM Time Delay) y una FIFO (DELAY FIFO). La RAM Time Delay almacena los valores de retardos ((td)) en la dirección del spike asociado. Se reserva el código de 0 en memoria, para el caso en que los spikes tengan un $(td = 0)$. La DELAY FIFO actúa como un buffer circular que se procesa en cada ciclo de emulación, es decir que, por cada uno de éstos se consumirá una unidad de tiempo de td.

Es así que, cuando HEENS-MP termina de ejecutar el algoritmo neuronal, los eventos de spikes post-sinápticos Si_{nm} resultantes son enviados a la RAM Time Delay para determinar su valor de retardo. Si este es diferente de cero, se escribe la dirección del spike junto con el valor del td en la DELAY FIFO. Si no existe ninguna asociación en la memoria, los Si_{nm} se escriben directamente en la OUTPUT FIFO en espera de ser distribuidos por el enlace.

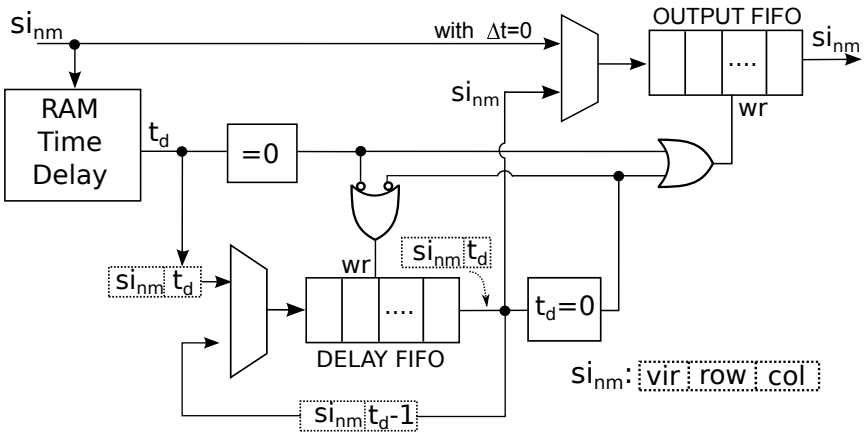


Figura 4.12: Delay Controller

En la DPh, el AER TX lee la OUTPUT FIFO, y vuelca su contenido en el Aurora Tx Side. Una vez la OUTPUT FIFO se ha vaciado, se inicia el procesamiento en la DELAY FIFO. Este consiste en verificar el valor del t_d de cada uno de los spikes almacenados. Si $t_d \neq 0$ se decrementa en 1 su valor y se guarda nuevamente en la DELAY FIFO. Lo contrario representa que el Si_{nm} está listo para ser transmitido y debe ser enviado a la OUTPUT FIFO para ser transmitido en el siguiente ciclo de emulación. El procesamiento del controlador termina cuando se ha examinado todos los spikes de la DELAY FIFO.

El ancho de la palabra de la RAM Time Delay y del DELAY FIFO determina el retardo máximo que se puede programar.

4.4. Implementación

La plataforma multi-chip HEENS que interconecta el MC con n NCs se presenta en la Fig. 4.13. El número máximo de nodos que se pueden interconectar en el anillo incluyendo el MC es de 128. Es decir, se tiene la capacidad de crear una red de hasta 127 NCs. Este valor está dado por el tamaño del registro RingSize de 7 bits. Los NCs están implementados en Xilinx Kintex 7 325T y el MC en una tarjeta Avnet PicoZed System On Module (SOM) Z030 de altas prestaciones de la familia Zynq. A su vez la PicoZed FMC Carrier Card [3] acoplada a la Z030 da soporte como tarjeta de desarrollo para el diseño de aplicaciones. Ésta dispone de los módulos de potencia, control de reset por hardware, y periféricos que dan accesibilidad a los pines de I\O de la Z030 [4].

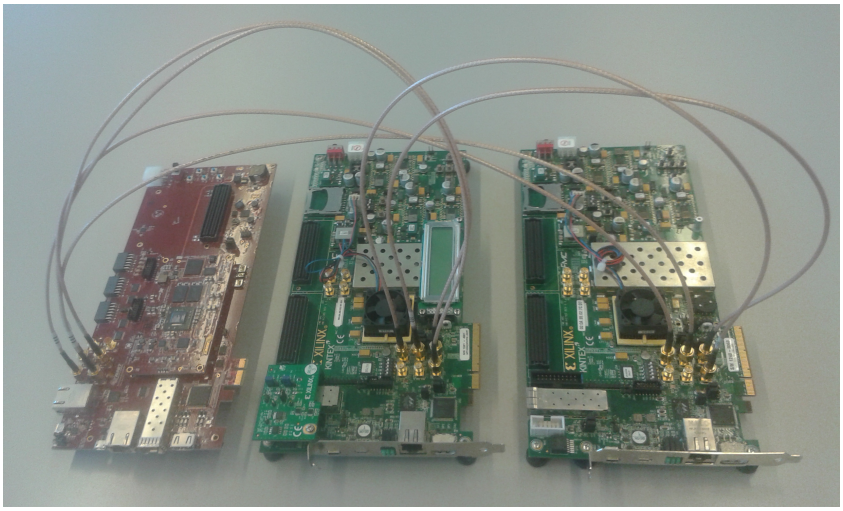


Figura 4.13: Implementación de la Arquitectura HEENS en un anillo de 2 NCs. Master Chip (roja): Picozed Z030 , Chip Neuromórficos (verde): Kintex 7 XC7325T

Tal como se ha comentado, las tarjetas tipo Zynq están constituidas por dos secciones: sistema de procesamiento (PS) y lógica programable (PL). El ZYNQ PS lo conforma un dual-core ARM CORTEX-A9 junto con un controlador de memoria DRAM y varios periféricos. El PL extiende la funcionalidad y velocidad de las aplicaciones, incluye bloques lógicos configurables (CLBs), bloques RAM (BRAM), multi-gigabit transeivers dependiendo del tipo tarjeta, ADC programables, etc. Ambos PS y PL están conectados por interfaces AXI [5] que permiten la comunicación entre ambos subsistemas.

En este caso particular, el código que corre en el PS del MC es una aplicación de software que interactúa con el hardware en modo Standalone (sin OS) implementada con Xilinx SDK. Este modo contiene librerías de soporte para lenguaje C estándar, funciones matemáticas, manejo de interrupciones, stack de TCP/IP y MFS (Memory File System). Las operaciones implementadas son

la lectura de archivos desde una MicroSD y la conexión serial con el PC. Para la inicialización y las operaciones de lectura con la MicroSD desde el PS, se utiliza el driver XSDPS que permite la interacción entre controladores (SD-PS) mediante el protocolo SDIO. La librería FAT-FS permite configurar el acceso y recuperación de los datos del archivo de configuración que posteriormente serán transmitidos al PL en paquetes de 32 bits.

Para la interacción con el usuario se utiliza el periférico UART (Universal Asynchronous Receiver/Transmitter) para establecer una conexión serie con el PC. Cuando el anillo se encuentra inicializado, el sistema espera a que el usuario inicie el proceso de configuración. El poder controlar cuándo se envía la trama de configuración a los NCs facilita el proceso de depuración tanto en el MC como en el resto de nodos del anillo.

En el PL se implementa el Z_AER-SRT Controller junto con el resto de módulos que conforman la parte lógica del Master Chip.

4.4.1. Gigabit Transceivers y Aurora Core

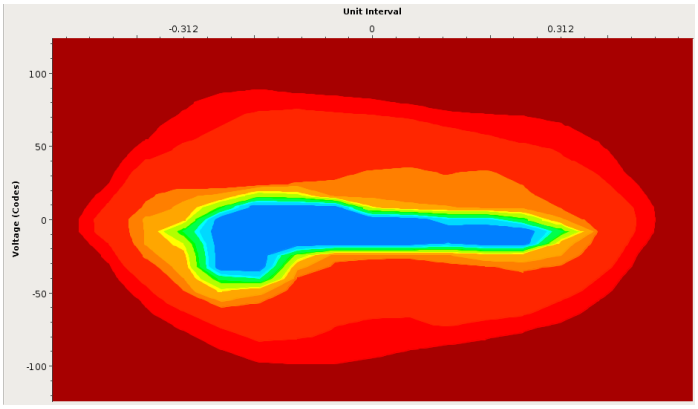
Para la recepción y transmisión de datos entre chips se utilizan Gigabit Serial I/O Transceivers. Estos tienen la ventaja de maximizar el flujo de datos alcanzando velocidades desde 500 Mb/s hasta los 12.5 Gb/s para la familia GTX. Poseen un consumo eficiente de potencia y recursos que además están fuertemente integrados a la lógica programable de la FPGA [6].

Su función es recoger datos paralelos generados a una f_{clk1}, serializarlos a una f_{clk2} para ser transmitidos en un stream de bits por un enlace serial hasta el lado receptor donde se deserializan para ser tratados nuevamente como datos paralelos a f_{clk1}

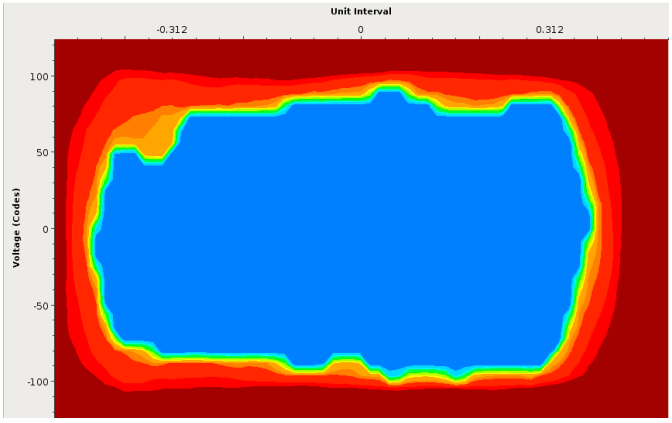
Los transceivers están localizados en Quads equivalentes a un grupo de 4 transceivers. En el caso de la Z030[4] se dispone de un solo GTX Quad que distingue a cada transceiver por sus coordenadas X0Y0, X0Y1, X0Y2, X0Y3. El Quad requiere de un reloj diferencial de referencia independiente del reloj que controla el resto de la lógica programable. Para esta tesis se utiliza el X0Y1 que corresponde al GTX vinculado a los conectores SMA de la Carrier Card. Estos son compatibles con los usados en los NC implementados en Kintex 7K325T.

Para analizar las tasas de transmisión y el comportamiento de los conectores y cables de conexión de los GTX de la Z030, se utilizó el IP IBERT junto con la herramienta de Vivado Serial I/O Analyzer para obtener el Diagrama de Ojo del X0Y0 en tiempo real. En la Fig 4.14(a) y 4.14(b) se muestran los resultados obtenidos a diferentes tasas de transmisión.

La Fig. 4.14(a) presenta un $BER = 5x10^{-1}$ (obtenido en modo Loopback), que indica una tasa de error considerablemente alta e interferencia entre símbolos debido a que el ojo tiende a cerrarse verticalmente. Como conclusión a una tasa de transmisión de 3.25 Gbps no se tiene un enlace chip a chip confiable. Es importante mencionar que los cables blindados de cobre utilizados para esta prueba son los mismos que se usaron para transmitir con GTX a velocidades de 3.25 Gbps en las tarjetas Kintex 7K325T, por lo cual se descartó que los problemas de degradación se deban al medio de transmisión. Se confirmó con los fabricantes que las Carrier Card presentan problemas de integridad de señal, por lo que se tuvo que reducir la tasa de transmisión a 1 Gbps.



(a) Tasa de transmisión: 3.125 Gbps



(b) Tasa de transmisión: 1 Gbps

Figura 4.14: Diagramas de ojo de X0Y0 - SMA Picozed Z030 a diferentes tasas de transmisión, obtenidos con IBERT Core

La Fig. 4.14(b) muestra una baja tasa de errores ($BER = 1{,}9x10^{-12}$) y buena calidad de señal acorde a los requerimientos del protocolo Aurora (ver Fig. 4.15) implementado como interfaz de los GTX.

Aurora es un protocolo abierto de capa de enlace de datos de Xilinx[7] utilizado para comunicar dispositivos o transceivers a través de una o múltiples líneas. Brinda características de alta escalabilidad, baja latencia y bajo consumo de área. Al ser un protocolo independiente puede ser implementado sobre otros protocolos facilitando la configuración de determinadas funciones. Al ser usado con GTX trabaja a la máxima velocidad de línea soportada por los respectivos transceivers. Por otro lado, se encarga de gestionar tareas automáticamente como encapsulación de datos, iniciación del canal, compensación de reloj para mantener el enlace activo, etc. En lo concerniente a este trabajo, se realiza el encapsulamiento de tráfico en paquetes de 16 bits y, de acuerdo a los resultados obtenidos para asegurar un enlace confiable se transmite a una tasa de 1 Gbps debido principalmente a las restricciones impuestas por la Carrier Card como se ha mencionado. El GTX utiliza un reloj de 250 MHz.

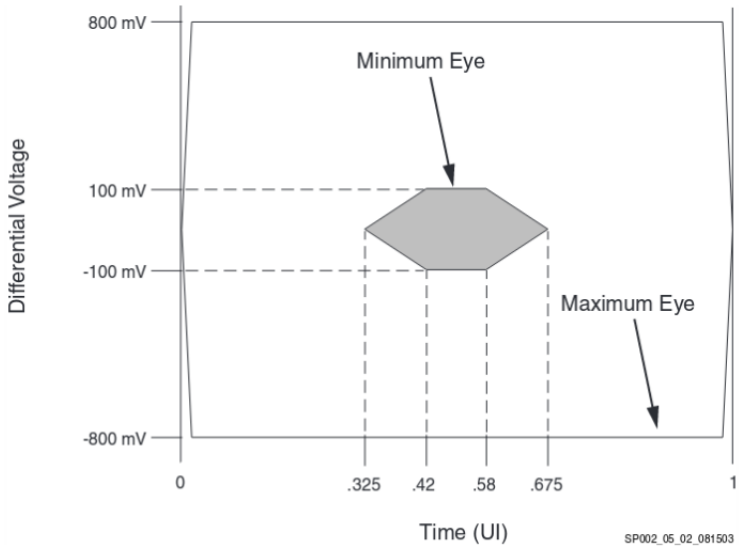


Figura 4.15: Requerimientos del Diagrama de Ojo para asegurar una óptima operación del Protocolo Aurora.

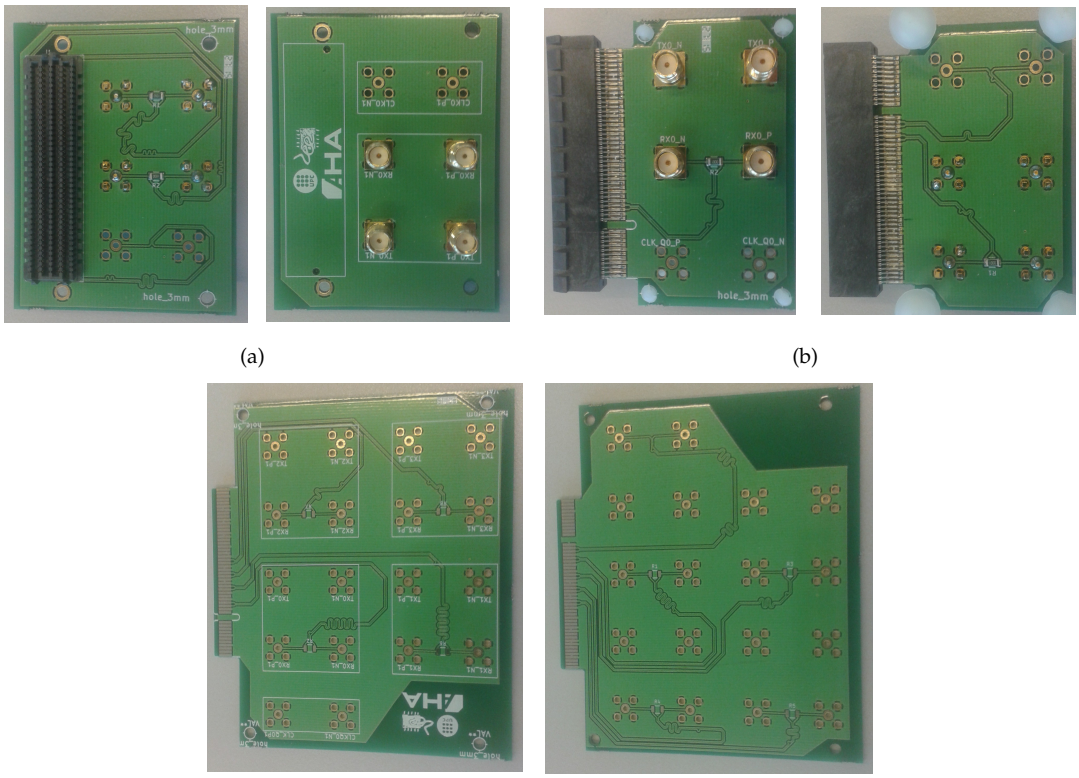
Para extender el número de puertos de interconexión a través de los transceivers disponibles en los puertos FMC, y PCI Express en ambas tarjetas Kintex y Picozed, se construyeron las tarjetas SMA mostradas en la Fig. 4.16, las cuales quedan disponibles para futuras implementaciones.

4.4.2. Consumo de área

En la Tabla 4.2 se presenta el consumo de área del AER-SRT y Z_AER-SRT Controller sobre las dos tarjetas Kintex y Picozed respectivamente. Los datos reportados muestran una eficiencia de uso de hardware mostrando un consumo de área muy bajo $< 1\,\%$ para la Kintex, y $< 7\,\%$ en el caso de la Picozed, dejando el resto de lógica disponible para el implementar el HEENS-MP en las Kintex, u otras funciones en el caso de la Picozed.

Tabla 4.2: Reporte de utilización del AER-SRT y del Z_AER-SRT Controller

Lógica	**Picozed Z030**		**Kintex XC7K325T**	
	Utilización	**Disponible**	**Utilización**	**Disponible**
Slice Registers	6,678 (4.25 %)	157,200	1,684 (0.41 %)	407,600
Slice LUTs	5,306 (6.75 %)	78,600	1,259 (0.61 %)	203,800
RAM36/FIFO	7 (2.64 %)	265	3.5 (0.78 %)	445
GT Channels	1 (25 %)	4	1 (6.25 %)	16



(a) (b)

(c)

Figura 4.16: a) Adaptador SMA para conector FMC-HPC. (b) Adaptador SMA para conector PCI-Express para Kintex 7 y Picozed. (c) Adaptar SMA para conector PCI-Express en Kintex 7, se dispone de 8 transceiver en el PCI-Express

4.4.3. Consumo de Potencia

El consumo de potencia del MC está directamente relacionado con el tipo de plataforma utilizada para la implementación. Sin embargo, en el caso de las FPGAs basadas en SoC como la Picozed presentan una contribución adicional debido al consumo del CPU ARM Core como se puede observar en la estimación de consumo mostrado en la Fig.4.17. De hecho este es el elemento que más consume en este caso. Adicionalmente, el consumo de la Carrier Card que incorpora varios periféricos añade una contribución adicional no relacionada con el diseño implementado que no se ha tomado en cuenta en esta estimación.

Es así que, el valor de potencia real, tiene una débil dependencia con la lógica implementada. Utilizando las herramientas de Vivado 2013.2 se obtuvo que el chip Z030 disipa 0.451 W en estado de Idle y 1.460 W cuando está procesando.

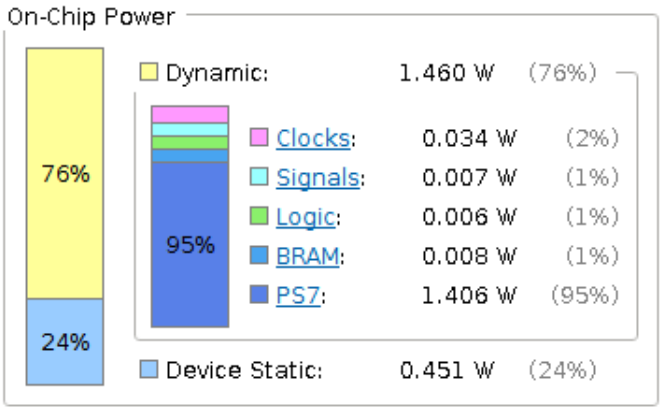


Figura 4.17: Consumo de Potencia Estimada del MC, obtenido con Vivado 2013.2

4.4.4. Resultados de Retardo

El valor máximo de retardo axonal que puede ser asignado es de 31 ciclos de emulación, que está en concordancia con los retardos biológicos presentes en las neuronas. El retardo típico en el cortex está en el rango de $0{,}1\ ms$ a $44\ ms$ [8]. No obstante implementar retardos más largos puede ser fácilmente adaptado y equivale a cambiar el ancho de palabra de la Memoria y la DELAY FIFO. Si se considera tiempo real, con una resolución de 1ms, HEENS es capaz de introducir retardos axonales hasta de $31\ ms$.

4.5. Conclusiones

Debido a que la propagación de spikes se realiza de forma serial y es función del número de sinapsis para redes de larga escala, la distribución de spikes puede llegar a representar el factor dominante en el costo computacional de la arquitectura y además se convierte en el principal cuello de botella debido al tráfico generado en el anillo que necesita ser procesado serialmente en cada chip. No obstante el uso de una comunicación en pipeline a altas tasas de transmisión con mínimo overhead de paquetes de control permite obtener aplicaciones neuronales ejecutándose en tiempo real, como se analizará en el siguiente capítulo.

Una de las ventajas del AER-SRT que brinda flexibilidad y adaptabilidad al modelo, es que el canal de comunicación que interconecta los nodos del anillo se está reutilizando para enviar además de eventos de spike, información de inicialización y configuración por medio del MC. Así también que independientemente del número de NCs, se requiere sólo un PC para controlar el envío de dicha información, además de soportar la emulación de características espacio temporales presentes en las neuronas, individualmente programables.

La utilización de una tarjeta SoC Zynq que integra un sistema de procesamiento dual Core ARM de altas prestaciones con Lógica Programable ofrece grandes ventajas al ser un sistema con un amplio

set de periféricos programables interconectados en silicio. Aunque en ésta tesis se utiliza el ARM sólo para despachar la información de configuración y evolución; la lógica del Modelo de Comunicación está completamente adaptada a la plataforma Zynq. Es así, que añadir funcionalidades o trabajar con otros periféricos consistirá mayormente en un diseño de software. Actualmente se está trabajando en una aplicación de adquisición de datos vía Ethernet para el monitoreo de los eventos de spikes que circulan por el anillo.

Finalmente, el modelo de comunicación implementado en HEENS permite obtener un balance entre escalabilidad y eficiencia debido a las altas tasas de transmisión que compensan en gran medida la latencia introducida por el pipeline del anillo. Además, debido a su flexibilidad es una solución muy conveniente para establecer la comunicación entre chips, así como para añadir el soporte de redes evolutivas, retardos de transmisión y re-configurabilidad dinámica propuesto en HEENS.

Referencias

[1] M. Mahowald, "VLSI Analogs of Neuronal Visual Processing: What does the retina know about a Synthesis of Form and Function", Tesis doct., California Institute of Technology y Pasadena, 1992.

[2] T. Dorta, M. Zapata, J. Madrenas y G. S??nchez, "AER-SRT: Scalable spike distribution by means of synchronous serial ring topology address event representation", *Neurocomputing*, vol. 171, págs. 1684-1690, 2016, ISSN: 18728286. DOI: 10.1016/j.neucom.2015.07.080. dirección: http://dx.doi.org/10.1016/j.neucom.2015.07.080.

[3] AVNET, "PicoZed FMC Carrier Card Hardware User Guide", págs. 1-48, sep. de 2015. dirección: http://zedboard.org/sites/default/files/documentations/PZCC-FMC%7B%5C_%7DUG%7B%5C_%7D1.1.pdf.

[4] ——, "PicoZed 7Z015/7Z030 SOM Hardware User Guide", vol. 1.6, sep. de 2014. dirección: http://zedboard.org/sites/default/files/documentations/PicoZed%207015%7B%5C_%7D7030_User's%7B%5C_%7DGuide%7B%5C_%7Dv1.6.pdf.

[5] ARM, "AMBA AXI and ACE Protocol Specification", feb. de 2013. dirección: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0022e/index.html.

[6] Xilinx, "7 Series FPGAs GTX / GTH Transceivers", vol. 476, págs. 1-412, 2012. dirección: http://www.xilinx.com/support/documentation/user%7B%5C_%7Dguides/ug476%7B%5C_%7D7Series%7B%5C_%7DTransceivers.pdf.

[7] ——, "Aurora 8b/10 Protocol Specification", vol. SP002, oct. de 2014. dirección: %7Bhttps://www.xilinx.com/support/documentation/ip%7B%5C_%7Ddocumentation/aurora_8b10b%7B%5C_%7Dprotocol_spec%7B%5C_%7Dsp002.pdf%7D.

[8] H. A. Swadlow, "Physiological properties of individual cerebral axons studied in vivo for as long as one year.", *J. Neurophysiol.*, págs. 1346-1362, 1995.

Capítulo 5

Test para Verificación y Caracterización de HEENS

En este capítulo se presenta un test para verificación de la arquitectura HEENS, tanto del Multiprocesador como del modelo de comunicación AER-SRT. Posteriormente se presenta la caracterización de las fases de operación de HEENS, así como el análisis de su escalabilidad dentro de la frontera de tiempo real. Por último se realiza la discusión respecto a los resultados obtenidos y la comparación con otras arquitecturas.

5.1. Conectividad Jerárquica y Reconfiguración on-line

En esta sección se presenta una red neuronal que permite demostrar la flexibilidad y la funcionalidad del multiprocesador HEENS y de su modelo de comunicación jerárquico para ejecutar las tareas de inicialización y configuración de la aplicación, así como de la distribución de spikes locales y globales en la red. Además, se demuestra la reconfiguración on-line de parámetros neuronales y sinápticos que modifica el comportamiento de la actividad neuronal.

En esta aplicación se ejecutan dos casos de prueba de concepto, la topología de configuración inicial con la que arranca el sistema, y la segunda que es el resultado de la reconfiguración de la red.

En el primer caso, la red neuronal configurada está compuesta por dos módulos de 10 neuronas con procesamiento local, es decir no existe comunicación entre las dos agrupaciones como se muestra en la Fig. 5.1. Para la implementación se ha configurado un anillo de 3 nodos, 2 NCs (uno por módulo) con un arreglo de 4x4 PEs, más el MC. Todas las neuronas de la red ejecutan el mismo algoritmo neuronal LIF dado por la siguiente ecuación:

$$V(t+1) = V_r + (V(t) - V_r).k_m + \sum_i \omega_i(t) S_i(t) + B \tag{5.1}$$

Condiciones de reset:

$$V(t) > V_{th} \qquad \Rightarrow \qquad V(t) = V_r \tag{5.2}$$

Donde :

- $V(t)$: Voltaje de membrana.
- $V_{th} = -55\ mV$: Voltaje de threshold (común para todas las neuronas).
- $V_r = -70\ mV$: Voltaje de restitución (común para todas las neuronas).
- S_i: Spikes post-sinápticos .
- $k_m = e^{-\frac{1}{\tau_m}}$: Constante de decaimiento ($\tau_m = 10\ ms$).
- $\omega_i(t)$: Peso sináptico excitatorio o inhibitorio
- B : ruido de fondo.

Todas las conexiones sinápticas son excitatorias y locales con un peso de $\omega = 25\ mV$. En ambos módulos se ha definido las condiciones iniciales de $V(t = 0) = -60\ mV$ para todas las neuronas excepto para la N_{00}. Esta última tiene un $V(t = 0) = -40\ mV$ para provocar el disparo de un spike que corresponde al estímulo de entrada que excita el resto de neuronas de cada módulo. Todas las neuronas poseen un ruido de fondo generado a partir de los LFSR de cada PE y valores aleatorios de semilla enviados en el proceso de configuración. El programa implementado se reporta en el Anexo III.

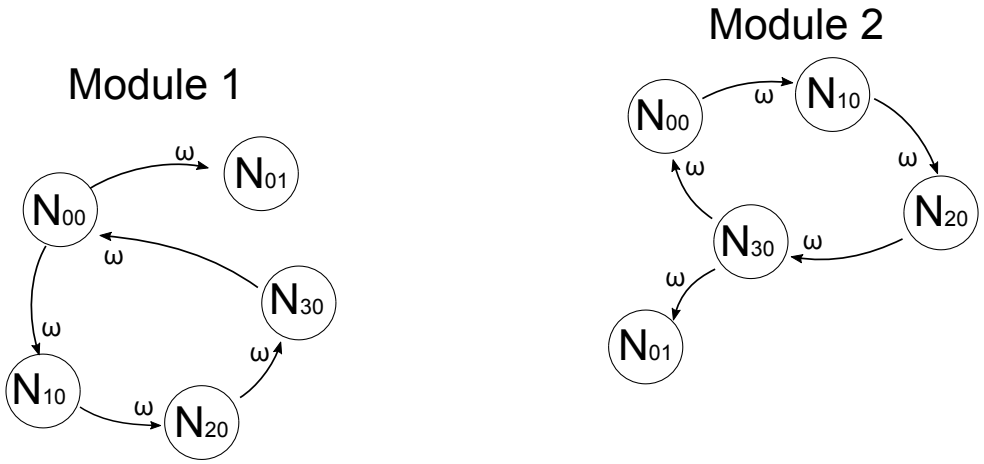


Figura 5.1: Topología de la red neuronal modular

5.1.1. Especificación del Archivo de Configuración

En la fase de configuración el MC envía en un solo bloque los datos que contiene la especificación del algoritmo neuronal, así como los parámetros sinápticos y neuronales que inicializan las memorias

de cada PE. Los scripts referidos en la sección 3.3.2 son utilizados para generar dicho archivo en base a dos net-list que contienen los parámetros sinápticos/neuronales y el mapeo de la red.

En este caso, debido a que todas las neuronas ejecutan el mismo algoritmo, la configuración de la IMEM de la Unidad de Control (Control Unit) es la misma para los dos NCs, por lo que los datos correspondientes se transmiten en *Modo de Configuración Común*. El resto de parámetros se envían de acuerdo al ID del NC al cual pertenecen. El tamaño del archivo que configura los NC para este caso particular es de 2.5 kB (311 palabras de 64 bits).

5.1.2. Reconfiguración on-line

Una vez que el MC ha inicializado el anillo y configurado los 2 NCs, se inicia el procesamiento neuronal. Por cada ciclo de emulación se ejecuta el algoritmo neuronal y los spikes generados son distribuidos localmente. Cada PE comprueba si debe procesar o no los spikes postsináptico recibidos mediante su memoria local.

Ya en ejecución, la reconfiguración on-line se produce cuando el MC activa la fase de evolución y envía los nuevos parámetros que en este caso modificará la estructura y el comportamiento de la aplicación. En la Fig. 5.2 se muestra las modificaciones realizadas a la red original por medio de la reconfiguración online que incluye 2 nuevas conexiones sinápticas globales que permite la comunicación entre módulos a través de las neuronas hub marcadas en azul; 2 conexiones locales entre N_{00} - N_{01} del módulo 1 y N_{01} - N_{30} del módulo 2. Así también, se modificaron pesos sinápticos (incluyendo sinapsis inhibitorias). En el gráfico se puede observar los nuevos valores.

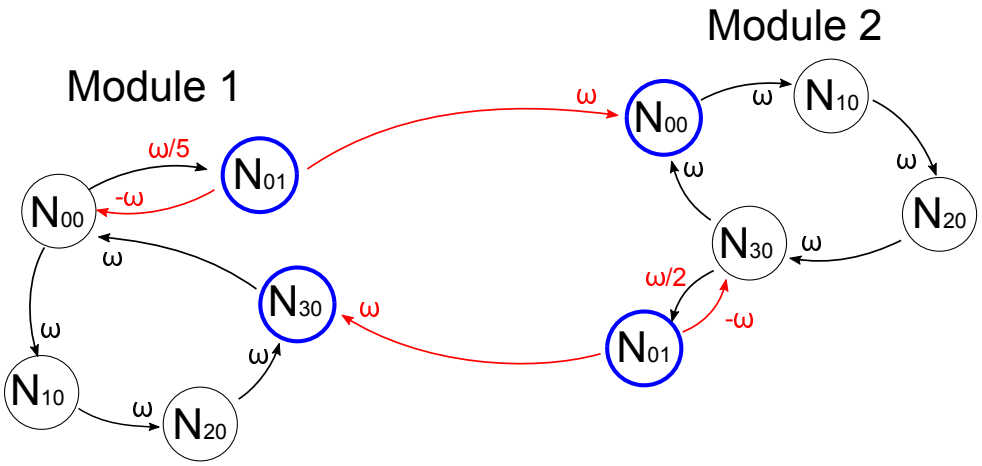


Figura 5.2: Topología de la red neuronal modular después de la reconfiguración (evolución)

La nueva configuración contiene 23 palabras que permiten modificar los parámetros de las SNRAM de las 5 neuronas sujetas a cambios. Así también, dado que la decodificación de spikes locales y globales es diferente, en el bloque de datos enviados por el MC se incluye la configuración de las memorias globales (Encoding, Conversion) para las neuronas N_{00} y N_{30} de los módulos 1 y 2 respectivamente.

Tabla 5.1: Correspondencia de índices de neuronas en el Raster Plot

Módulo 1		**Módulo 2**	
Neurona	**Índice**	**Neurona**	**Índice**
N_{00}	1	N_{00}	6
N_{10}	2	N_{10}	7
N_{20}	3	N_{20}	8
N_{30}	4	N_{30}	9
N_{01}	5	N_{01}	10

5.1.3. Resultados

En el gráfico de la Fig. 5.3 se ilustra el raster plot de la actividad neuronal de la red para 180 ciclos de emulación. En la Tabla 5.1 se indica el índice que le corresponde a cada neurona en el raster plot. En los primeros 53 ciclos se puede observar la actividad de los spikes post-sinápticos en la topología inicial para los 2 NCs. En ambos casos los spikes se transmiten de una neurona a otra de forma cíclica sin que exista comunicación entre módulos.

Cuando los NCs reciben datos de reconfiguración (ciclo 53), los spikes del ciclo de emulación anterior no se pierden. Éstos se quedan en espera de la finalización de la EvPh para distribuirse por la red, y procesarse conforme a la nueva topología definida. En los ciclos 53 y 60 se observa precisamente como la red se va adaptando a la nueva configuración hasta alcanzar un comportamiento estable.

A partir del ciclo 60, en el caso del módulo 1, los spikes se siguen transmitiendo de una neurona a otra hasta cuando se dispara la neurona 5. Ésta inhibe la 1 con lo cual se no se genera ningún estímulo que continue excitando el resto de neuronas del módulo y la actividad eventualmente desaparece. El mismo comportamiento se produce en el módulo 2 entre las neuronas 10 y 9. El detalle importante es que N_{01} es inhibitoria para las neuronas locales, y excitatoria para las globales en ambos módulos. Cuando dispara, además de inhibir la transmisión de spikes locales, excita el otro módulo con lo cual se inicia nuevamente la propagación de spikes en el NC contrario como se observa en el raster plot.

Las neuronas 5 y 10 corresponden a la N_{01} de cada NC. Se observa la interacción entre ambos clusters a través de los spikes globales proporcionando el estímulo de entrada que habilita su actividad neuronal. Las flechas rojas del raster plot marcan la acción de las dos sinapsis globales en la red. Las dinámicas independientes de los dos osciladores separados inicialmente quedan ahora acopladas y se produce una única oscilación de menor frecuencia. En la Fig. 5.4 se ilustra como varía el voltaje de membrana de la neurona 5. Podemos observar como varía su comportamiento a partir del ciclo 53 cuando se produce la reconfiguración de la red.

5.2. Análisis de las Características temporales de HEENS

En esta sección se presenta la caracterización de las fases de funcionamiento de HEENS. Se obtienen las funciones de tiempo que permiten estimar los ciclos consumidos y duración de cada fase.

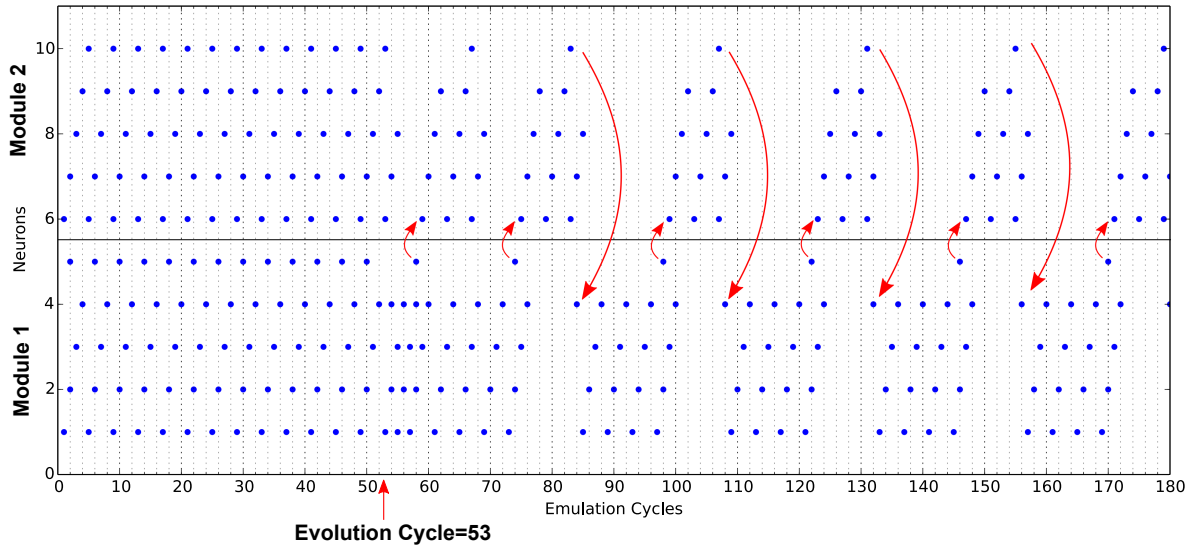


Figura 5.3: Raster plot de la actividad neuronal de la red.

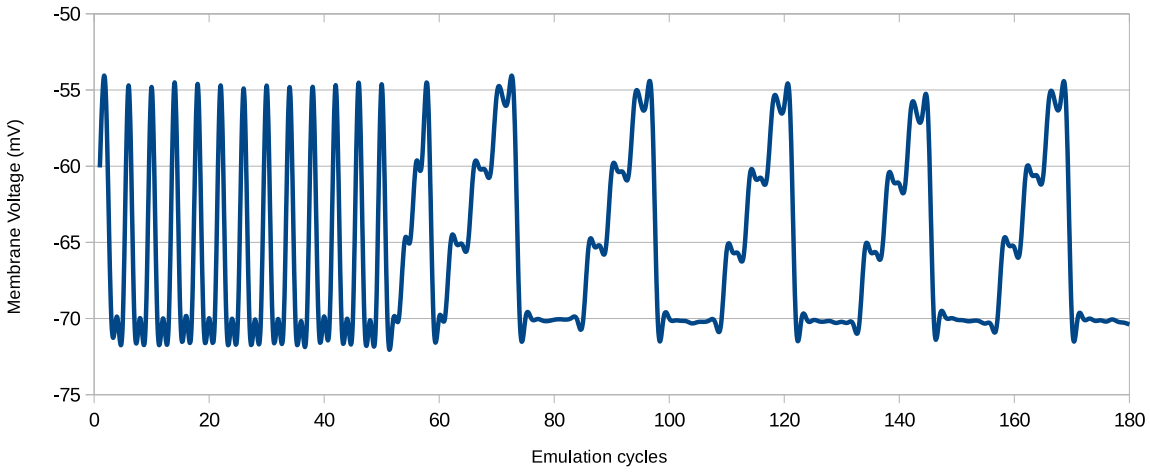


Figura 5.4: Voltaje de membrana de N_{01} (neurona 5) localizada en el Módulo 1

Para las pruebas realizadas, se simuló un anillo que interconecta un Master con hasta 5 NCs. Los NCs se configuraron con un arreglo de 10x10 PE's con un conexionado de loopback, es decir con una única conexión por neurona, realimentada consigo misma (ver Fig. 5.5). El multiprocesador trabaja a una frecuencia de f_{CLK1}. El algoritmo utilizado es el mismo LIF de la sección anterior. Con esta configuración se genera un tráfico de 100 spikes por ciclo de emulación. Los dos controladores AER-SRT y Z_AER-SRT trabajan a una frecuencia de f_{CLK2} para realizar la distribución de eventos por el canal. En particular, f_{CLK2} está dada por la tasa de transmisión de los transceivers GTX utilizados para

la comunicación serial de alta velocidad.

Figura 5.5: Conexionado loopback

5.3. Fase de Inicialización

En la Fig. 5.6 se ilustra el número de ciclos que el MC consume para inicializar los parámetros de ID y RING SIZE en los nodos de la red. Como se mencionó en el capítulo anterior, dichos parámetros son necesarios para identificar cada nodo y sincronizar la transmisión de eventos entre chips. Para la medición se consideró el periodo desde que el MC inicia las transacciones para enviar ambos parámetros, hasta que recibe la confirmación a través de los paquetes de control que el ID y RING SIZE han terminado de circular por el anillo inicializando los NCs.

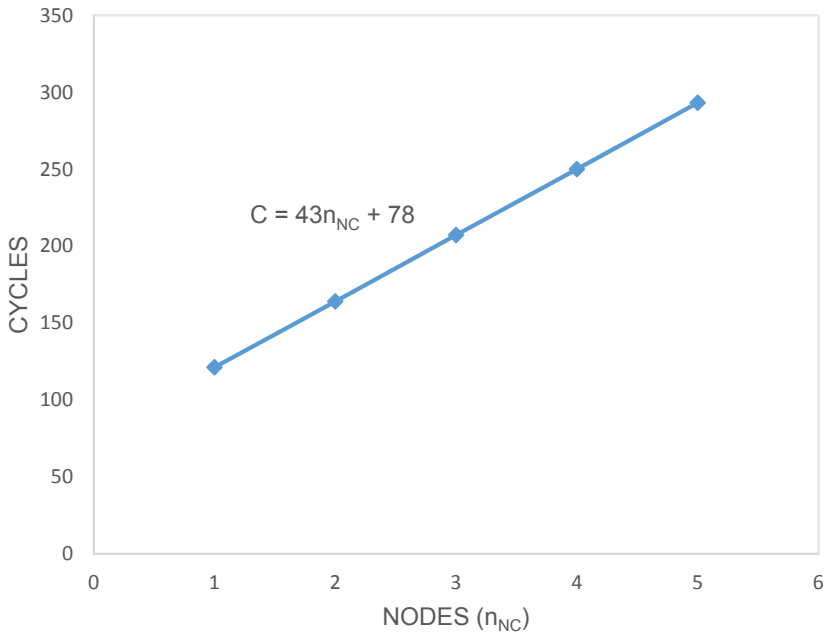


Figura 5.6: Caracterización del la Fase de Inicialización (IPh)

De los resultados obtenidos al ir añadiendo al anillo 1 NC cada vez, se puede observar en la gráfica de la Fig 5.6 que existe una dependencia lineal respecto al número de NCs. De acuerdo a los resultados se requiere en promedio 43 ciclos por NC más un offset λ_{IPh} de 78 ciclos en promedio. Aunque en primera instancia se puede suponer que el offset es relativamente alto, se debe tener en cuenta que el MC también actúa como un nodo más, por lo cual introduce una latencia adicional media de 43 ciclos.

Generalizando, en la Ec. 5.3 se presenta la caracterización temporal de esta fase que corresponde al tiempo de inicialización en función del número de NCs (n_{NC}) interconectados en la red.

$$t_{IPh} = \frac{1}{f_{CLK2}}.(43.n_{NC} + \lambda_{IPh}) \tag{5.3}$$

Para la implementación realiza con 5 NCs y una $f_{CLK2} = 50Mhz$ se obtiene un $t_{IPh} = 5{,}86\ us$

5.4. Fase de Configuración

Para la caracterización de esta fase se utilizó el archivo de configuración de 2,488 bytes del diseño presentado en la sección anterior. Por simplicidad el DESTINATION ID de la configuración corresponde al del MC, con lo cual los datos serán procesados por todos los NCs (Configuración en Modo Común). Esta fase inicia inmediatamente después de terminar la inicialización. En la Fig. 5.7 se ilustra el número de ciclos empleados para que un bloque de paquetes de configuración recorra el anillo completo.

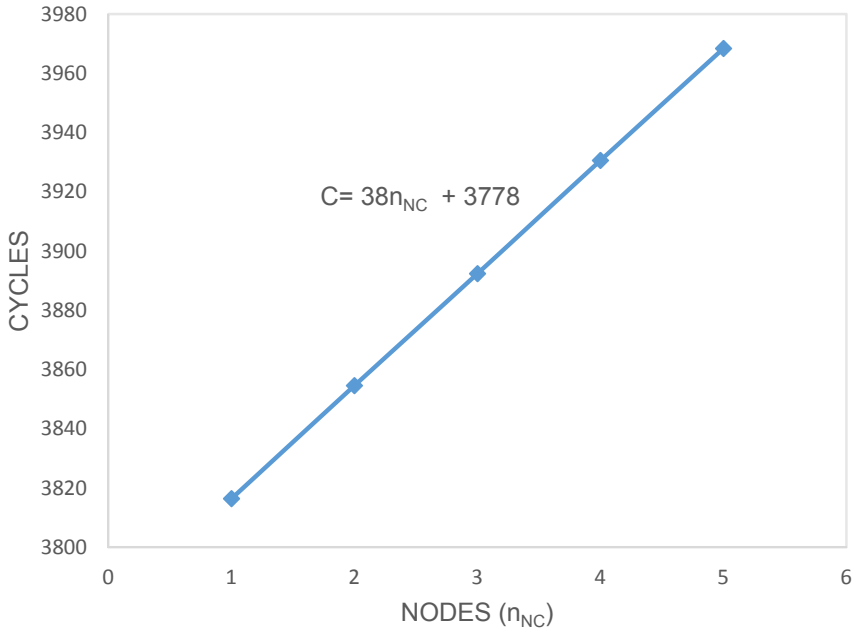


Figura 5.7: Caracterización de la Fase de Configuración (CPh)

Se debe destacar que si la configuración fuera diferente en cada NC, la única diferencia sería el tamaño del archivo; dado que a medida que los paquetes llegan a cada nodo se procesan y se retransmiten paralelamente.

Como se puede observar en la gráfica y al igual que el caso anterior, el número de ciclos requeridos para completar esta fase muestra una dependencia lineal respecto al tamaño del anillo y en este caso

del archivo de configuración más un offset. La Ec. 5.4 corresponde al tiempo requerido para configurar la red neuronal.

$$t_{CPh} = \frac{1}{f_{CLK2}} \left(38 n_{NC} + 3{,}75 \frac{f_{CLK2}}{f_{CLK1}} . (FileSize) + \lambda_{CPh} \right) \tag{5.4}$$

Los 38 ciclos por NC equivalen a la latencia introducida por cada nodo. El tamaño del archivo (FileSize) está dado en bytes y el factor de 3.75 responde a los ciclos empleados por el MC para leer la memoria externa (donde se almacena el archivo de configuración) y escribir los datos en la FIFO_CONF. Esta es asíncrona y se utiliza para permitir el flujo de datos entre ambos dominios. El procesador de la Picozed entrega paquetes de 32 bits cada 12 ciclos (2 por palabra de configuración), mientras que Aurora transmite paquetes de 16 bits (15 bits de datos más 1 bit de control). La latencia introducida por la lectura de la memoria externa se minimiza debido a que los ciclos entre cada lectura se utilizan para dividir los paquetes recibidos por el procesador y colocarlos en el canal de transmisión. El offset de $\lambda_{CPh} = 46\ ciclos$ se debe principalmente al MC como se explicó anteriormente.

Para la implementación realiza con 5 NCs, $f_{CLK1} = 125\ Mhz$, $f_{CLK2} = 50\ Mhz$ y un FileSize=2,488 bytes, se obtiene un $t_{CPh} = 79{,}36\ us$.

El tamaño máximo del archivo de configuración a máxima ocupación, en el que un NC con un arreglo de 12x12 PEs necesita configurar todas sus memorias al 100 % de su capacidad (ver Tabla 5.2) sería de 4MB.

Tabla 5.2: Capacidad máxima de las memorias por NC

Memoria	KB
Instrucción	8
SinapNeuronal/PE	8
Local /PE	16
Codificación/PE	4
Conversión/PE	0.5

5.5. Fase de Ejecución

Dado que HEENS es una arquitectura de propósito general, se puede programar cualquier algoritmo neuronal de tipo spiking. HEENS procesa las neuronas en paralelo y las sinapsis en serie. Además, si el arreglo se encuentra virtualizado, el secuenciador empezará a procesar las neuronas del nivel 0 con sus respectivas sinapsis y luego pasará al siguiente nivel repitiendo la misma operación, y así sucesivamente. Como se explicó en el capítulo anterior solo en el nivel principal 0 dispone de sinapsis globales y locales, el resto de niveles procesa sinapsis locales.

Una propiedad muy interesante de la arquitectura es que el usuario puede asignar libremente un número de sinapsis locales arbitrario a cada nivel de virtualización, siempre que la sumo no exceda el total de sinapsis disponible por el PE.

Tabla 5.3: Consumo de ciclos del algoritmo neuronal por subrutina

Descripción	Ciclos	Caso particular
Subrutinas Neuronales		
Carga de parámetros neuronales	LN.(n_V)	13
Cálculo del voltaje de membrana	MV.(n_V)	19
Ruido sináptico	SN.(n_V)	22
Detección de spikes	SD.(n_V)	20
Almacenamiento de voltaje de membrana	SV.(n_V)	10
Subrutinas Sinápticas		
Carga de puntero con las dirección de las sinapsis locales	LLP.(n_V)	5
Bucle local	FLS	12
Cálculo de potencial post-sináptico local	LW.(S_{L_V})	17x2
Carga del puntero con la dirección de las sinapsis globales	LGP	5
Cálculo de potencial post-sináptico global	GW.(S_G)	17x2
Virtualización		
Lectura del nivel virtual actual	VL.(n_V)	5
Incremento del nivel virtual	NVL.(n_V)	1

n_V: es el número de niveles virtuales asignados al arreglo de PEs.
S_{L_V}: número de sinapsis locales por cada nivel virtual.
S_G: número de sinapsis globales en el arreglo.

En la Tabla 5.3 se lista las tareas o subrutinas básicas que se deben realizar para cualquier algoritmo. El número de ciclos de cada una dependerá de la complejidad computacional del mismo, así como del número de niveles virtuales n_V y de la cantidad de sinapsis asignadas a cada nivel.

En la Ec 5.5 se muestra el tiempo de duración de la fase de ejecución en función de los parámetros descritos anteriormente. El offset λ_{EPh} corresponde a cualquier otra subrutina adicional que se requiera implementar.

$$\begin{aligned} t_{EPh} = & \frac{1}{f_{CLK1}}[(LN + MV + SN + SD + SV + LLP + VL + NVL).n_V + \\ & + (\sum_{V=0}^{n_V} LW(S_{L_V})) + GW(S_G) + FLS + LGP + \lambda_{EPh}] \end{aligned} \tag{5.5}$$

En la última columna de la tabla 5.3, se muestra los ciclos utilizados para la aplicación de la sección 5.1, con lo cual se obtiene que el algoritmo está siendo ejecutado en 1.44 us.

5.6. Fase de Evolución

En cada ciclo de emulación, la notificación del estado de evolución (activado/desactivado) realizada a través del paquete de control EVOL, añade un overhead prácticamente despreciable de

1 ciclo/NC. Dicho paquete se envía junto al de sincronización (SYNC), por lo que la latencia del anillo se refleja en la siguiente fase de distribución.

En su defecto cuando el MC activa la evolución y envía posteriormente una trama de configuración, el tiempo requerido para completar esta fase está dado por la Ec. 5.4 y dependerá del tamaño del archivo que para este caso contendrá únicamente los nuevos datos de reconfiguración.

Para la reconfiguración del test de la sección 5.1 se emplearon 184 bytes (23 palabras de configuración) con las mismas frecuencias mencionadas anteriormente, con lo cual el tiempo empleado para reconfigurar los 2 NCs fue de $t_{EvPh} = 8\ us$.

5.7. Fase de Distribución

Como se explicó anteriormente esta fase está compuesta de tres etapas: sincronización del anillo, transmisión de eventos y la decodificación de spikes tanto locales como globales.

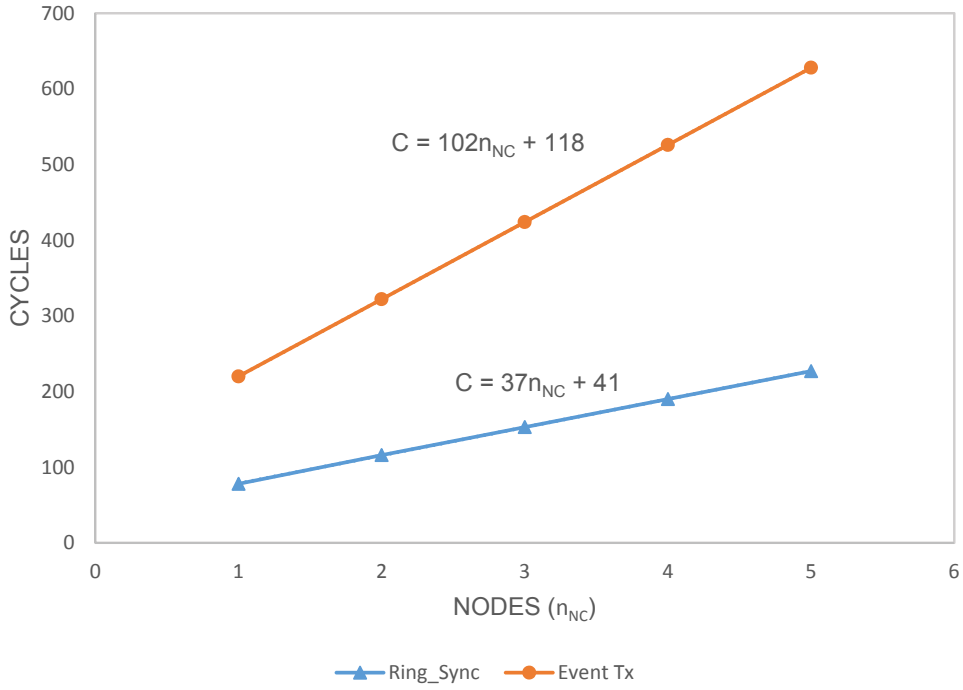


Figura 5.8: Caracterización de la sincronización de NCs y transmisión de eventos de spike

En la Fig. 5.8 se muestran los ciclos empleados para las dos primeras en función del número de NCs. Como se puede observar, la función de sincronización (azul) crece más lento que la transmisión, empleando 37 ciclos/NC más un overhead. Además, su duración es independiente del número de spikes transmitidos por los nodos.

Por otro lado, para medir la transmisión de evento se ha configurado todos los nodos incluido el MC para que generen un tráfico 100 spikes/nodo a ser distribuido por el canal. El MC está diseñado para

insertar eventos de spikes a la red. Como se observa en la gráfica, la función de transmisión muestra que el número de ciclos consumidos es proporcional al tráfico de spikes/NC, con un offset de 118 ciclos en promedio. Este valor obtenido de la regresión lineal corresponde a los 100 spikes generados por el MC más el correspondiente overhead.

Con respecto a la última etapa de recepción de spikes, corresponde a los ciclos empleados (1 ciclo/evento) para la lectura de la INPUT_FIFO que permite el flujo de datos entre los dos dominios de reloj utilizados en la arquitectura. Los datos de lectura se envían a las AM para su respectiva decodificación.

En la Ec. 5.6 se muestra el tiempo requerido para esta fase, resultado de añadir las contribuciones de las 3 etapas.

$$t_{DPh} = \frac{1}{f_{CLK2}} \left(39n_{NC} + S_T + \frac{f_{CLK2}}{f_{CLK1}}.S_T + \lambda_{DPh} \right) \tag{5.6}$$

Siendo:

$$S_T = \sum_{i=1}^{n_{NC}} S_i + S_{MC} \tag{5.7}$$

El primer término es proporcional al número de NCs y se debe a la latencia del anillo introducida por Aurora. S_T representa el número de spikes que circulan por el canal, que corresponde a la suma de los eventos generados en cada NC (S_i) más los insertados por el MC (S_{MC}), cuando se requiera actividad externa. λ_{DPh} de aproximadamente 59 ciclos responde en gran parte al costo del MC en el anillo, así también a los ciclos utilizados para monitorear la secuencia de control del protocolo AER. Además, cada 10,000 ciclos Aurora detiene la transmisión para realizar operaciones de alineamiento y compensación de reloj lo cual influye también influye en el valor de λ_{DPh}. Estas operaciones aseguran que el enlace permanecerá activo y que se mantendrá la sincronización entre el transmisión y el receptor de Aurora.

5.7.1. Eficiencia del Canal de Comunicación

Para determinar la eficiencia del canal de comunicación se consideran las etapas de sincronización y transmisión de eventos de la fase de distribución ya es aquí donde interviene el modelo de comunicación para controlar la transmisión de datos por el canal. Dado el caso, a una frecuencia de f_{CLK2} se ocupa:

$$\#cycles = 39n_{NC} + \sum_{i=1}^{n_{NC}} S_i + \lambda_{DPh} \tag{5.8}$$

para realizar la distribución de eventos (no se considera el eventos externos) en cada ciclo de emulación. El valor de λ_{DPh} es el mismo obtenido anteriormente.

En el compromiso entre el tamaño del anillo y el payload de cada NC, la mejor condición en términos de un mejor desempeño y eficiencia es tener red con pocos nodos interconectados y con alta actividad de spikes, debido a que la latencia es proporcional al tamaño del anillo. Así tambien, mientras mayor es el payload, el porcentaje de overhead será menor, con lo que el throughput del canal

Tabla 5.4: Ciclos de ejecución en función de la frecuencia de trabajo

f_{CLK1} (MHz)	$\#Cycles_{IPh}$
125	62,500
200	100,000
250	125,000
300	150,000

se incrementará. Por ejemplo si consideramos el caso de un anillo de 5 NCs cada uno generando 1,000 eventos por ciclo de emulación, se utilizarían 5,254 ciclos de reloj, con lo que se tendría un overhead de apenas el 5 %. En su defecto si se transmitieran 200 eventos/ciclo de emulación en el mismo anillo el overhead sería del 25.4 %.

5.8. Escalabilidad y Tiempo Real

Para asegurar la operación de HEENS dentro de la ventana de tiempo real, se considera que el $\Delta t_{emul} <= 1\ ms$. Como se ha citado anteriormente, un ciclo de emulación en HEENS corresponde a:

$$\Delta t_{emul} = t_{EPh} + t_{DPh} \tag{5.9}$$

En la ecuación no se toma en cuenta el t_{EvPh} debido a que es prácticamente despreciable durante la operación normal de la red neuronal. Por facilidad se asume que el Δt_{emul} se reparte por igual entre las fases de ejecución y distribución, $0{,}5\ ms$ para EPh y $0{,}5\ ms$ para DPh.

Con esta condición y dado que la EPh depende del número de ciclos utilizados para procesar el algoritmo neuronal tal como se estableció en la Ec. 5.5, se obtiene la siguiente relación:

$$\#cycles_{EPh} = 0{,}0005 f_{CLK1} \tag{5.10}$$

Donde, $\#cycles_{EPh}$ corresponde al máximo número de ciclos disponibles para procesar la actividad neuronal, a una frecuencia de operación f_{CLK1} dentro de la frontera de tiempo real. En la Tabla 5.4 se muestra esta relación para diferentes frecuencias.

El primer caso mostrado, corresponde a las condiciones de implementación de HEENS con una frecuencia de 125 MHz. Para efectos de comparación, en la aplicación presentada en la sección 5.1 los NC emplean 180 ciclos (1.44 us), es decir están trabajando 347 veces por debajo del límite temporal impuesto de 0.5 ms. A su vez, si se considera la máxima actividad del NC, es decir un arreglo de 12x12 PE, con 144 sinapsis locales y 32 globales y todos los niveles de virtualización activados; el algoritmo se ejecutaría en 30.15 us, esto es 16.58 veces por debajo del límite impuesto.

En el otro extremo, para la fase de distribución como se analizó anteriormente depende del:

- Número de NC en el anillo.
- Frecuencia del multiprocesador f_{CLK1}.

Tabla 5.5: Tiempo de distribución para un tráfico de 1,152 eventos/cycle

f_{CLK1} (MHz)	Tasa de Tx (Gbps)	t_{DPh} (ms)			
		RZ=15	RZ=35	RZ=75	RZ=90
125	1	0.497	1.157	2.479	2.974
200	1	0.445	1.036	2.220	2.663
250	1	0.428	0.996	2.133	2.560
300	1	0.416	0.969	2.076	2.491
125	6.6	0.193	0.449	0.962	1.154
200	6.6	0.141	0.328	0.703	0.843
250	6.6	0.123	0.288	0.616	0.740
300	6.6	0.112	0.261	0.559	0.671
125	12.5	0.167	0.389	0.834	1.001
200	12.5	0.115	0.268	0.575	0.690
250	12.5	0.098	0.228	0.489	0.586
300	12.5	0.086	0.201	0.431	0.517
125	16.5	0.160	0.373	0.800	0.959
200	16.5	0.108	0.252	0.540	0.648
250	16.5	0.091	0.212	0.454	0.545
300	16.5	0.079	0.185	0.396	0.476

RZ número de NCs en el anillo.
Las tasas de transmisión reportadas, son las disponibles actualmente para los GTX y GTH soportados por AURORA para diferentes familias de FPGA.

- Tasa de transmisión impuesta por los transceivers seriales.

En la Tabla 5.5 se lista el t_{DPh} obtenido para diferentes condiciones de frecuencia y tamaño de anillos (RZ), tomando como referencia una máxima actividad de 1,152 eventos por ciclo de emulación en todos los NC.

Los datos reportados muestra como el sistema puede escalar variando cada parámetro. En el caso de la implementación de HEENS (125 Mhz y 1 Gps) se puede conectar hasta 15 nodos asegurando una operación en tiempo real, con un overhead del 3.7 % y un throughput de 16.6 Mevent/s. Sin embargo, utilizando tarjetas FPGA de mejores prestaciones, se podría obtener un throughput de 100.1 Mevent/s con un overhead del 3.4 % utilizando transceivers a 16.5 Gbps en un anillo de 90 NCs, y con el multiprocesador trabajando a una frecuencia de 300 Mhz. Los datos presentados podrían escalar de mejor forma para una actividad de spikes mayor a la considerada de 1,152 eventos/cycle (12x12 PEs).

Es importante resaltar que estas condiciones están dadas utilizando una sola línea de transmisión punto a punto. En el caso de habilitar más lineas el throughput escalaría en proporción.

5.9. Consumo de Potencia

El consumo de potencia de HEENS es equivalente a la del MC más la generada por cada NC del anillo. De acuerdo a los datos obtenidos un NC de 12x12 presenta una potencia estimada de 3.3 W y el MC de 2.9 W. Si consideramos la máxima escalabilidad obtenida bajo las condiciones del hardware utilizado, es decir 15 NC interconectados en el anillo, tenemos que la arquitectura HEENS exhibe un consumo estimado de 52.4 W.

5.10. Resumen de Resultados

En la Tabla 5.6 se lista el resumen de las principales características de HEENS.

Tabla 5.6: Resumen de las principales características de HEENS

Arquitectura HEENS	
Dispositivo	FPGA - Kintex XC7K325T , Picozed Z030
Neuronas/ Core	1152
# Cores	15
Sinapsis/Neurona	144
Modelo neuronal	Programable
Resolución	1 ms
Expandibilidad	Anillo
Conectividad	Jerárquica, local (conectividad completa), global (nivel principal)
Potencia	3.3 W por FPGA
Throughput	16.6 Mevent/s
Reconfigurabilidad	Si
Configurabilidad	Si (sobre bus AER síncrono)
Retardos Axonales	hasta 31 ms

5.11. Discusión

En esta sección se realiza la discusión sobre algunos de los aspectos más relevantes de la arquitectura HEENS en relación a otras propuestas presentadas en el estado del arte. Al respecto HEENS ofrece una plataforma de propósito general con un set propio de instrucciones en la cual el usuario tiene la libertad de definir varias características como la precisión de las variables de estado, posiciones de memoria, número de neuronas y sinapsis locales y globales por chip, etc. Todo esto con un procesamiento paralelo masivo de tipo SIMD que asegura una operación en tiempo real a diferencia de aproximaciones por software caracterizadas por su alta programabilidad pero que distan por mucho de la frontera temporal biológica.

Pasando a las arquitecturas hardware, tenemos las basadas en ASIC como Neurogrid [1], FACET [2] [3], TrueNorth [4], IFAT [5] [6] que poseen una muy alta densidad de neuronas y sinapsis como común denominador, así como también están limitadas a algoritmos neuronales específicos. Esto les permite optimizar sus diseños pero cuando requieren migrar de tecnología para implementar nuevas o mejores funcionalidades deben rediseñar todo el sistema.

Como se observa en la Tabla 5.7 aproximaciones basadas en FPGA como Thomas & Luk [7] y Pani [8] presenta características similares a HEENS en cuanto al número de neuronas y sinapsis. Sin embargo, no son escalables ni permiten reconfiguración on-line y su diseño se restringe a un solo tipo de neurona. Si tomamos como referencia el uso de memoria on-chip utilizado en HEENS para el mapeo de sinapsis, podemos compararnos respecto a Pani [8]. En esta aproximación, un punto remarcable es que alcanzan mayor número de sinapsis por neurona con un consumo de memoria muy parecido al nuestro. Sin embargo, a pesar de utilizar aritmética de punto fijo como el caso de HEENS, en Pani el número de DSP utilizados es 4 veces mayor, con 2.57 veces más consumo de potencia. Así también su precisión es fija lo cual limita su rango dinámico, mientras que en HEENS el usuario define la precisión de cada variable en un rango de $[-32767, +32767]$ fijando su propio factor de escala.

La plataforma multi-chip de NeuroFlow [9] también basada en FPGA tiene un alto consumo de potencia de 20-30 W en comparación con los 3.3 W de HEENS, aunque el número de neuronas soportada en cada caso no es comparable. A diferencia de HEENS, NeuroFlow no permite una conectividad completa, sino que está limitada a una función de distribución gausiana. La gran densidad de neuronas alcanzada se debe a que por cada aplicación se automatiza por software el rediseño completo de la arquitectura, optimizando el consumo de recursos a un costo de altos tiempos de síntesis. No soporta una reconfiguración dinámica, cualquier cambio requiere volver a rediseñar todo el sistema, siendo poco práctico para el desarrollo de aplicaciones en las que frecuentemente se realiza varias pruebas antes de obtener el resultado esperado. Con HEENS la configuración de la aplicación es sencilla y cualquier modificación de parámetros implica apenas la preparación de un nuevo archivo de configuración. Otro aspecto a resaltar en Neuroflow es su limitada escalabilidad ya que solo puede conectar hasta 6 tarjetas, con HEENS el límite es 127 y podría extenderse aún más realizando pequeñas modificaciones.

La arquitectura SpiNNaker [10] es una de las más destacadas en el estado del arte y un referente de comparación. Como se muestra en la Tabla 5.7 comparte con HEENS la característica de ser una plataforma de propósito general en las que el usuario puede programar el algoritmo neuronal y su modo de reconfigurabilidad. A su vez, SpiNNaker emplea procesadores ARM y routers asíncronos genéricos con procesamiento manejado por eventos usando comunicación AER para la comunicación entre chips. Maneja un esquema localmente síncrono y globalmente asíncrono donde cada procesador es independiente con su propia señal de reloj siguiendo un esquema MIMD (Múltiples Instrucciones, Múltiples Datos). Posee 2 tipos de memoria local y global de 64 k y 1 GB respectivamente y carece de mecanismos que gestionen condiciones de alto tráfico. En el caso de HEENS, se utiliza un modelo de comunicación y procesadores a medida que optimizan el consumo de recurso y potencia. De igual manera el tipo de procesamiento SIMD utilizado en HEENS se beneficia de una simple sincronización con un diseño menos costoso en cuando a recursos. Además, no exhibe problemas de congestión

Tabla 5.7: Comparación de HEENS con otras plataformas hardwdare SNN.

Plataforma	Dispositivo	Neuronas/ Core	# Cores	Modelo	Sinapsis/ Neurona	Resol.	Reconf.	Potencia/ Core
Thomas y Luk (2009)	FPGA	1,024	1	IZ	1,024	10 us	No	-
SpiNNaker (2013)	ARM	100	18	Prog	1,000	1 ms	Yes	1 W
NeuroFlow (2016)	FPGA	98,304	6	Prog	1,000-10,000	1 ms	No	30-40 W
Pani (2017)	FPGA	1,440	1	IZ	256	0,1 ms	No	8.5 W
HEENS (2017)	FPGA	1,152	15	Prog	144	1 ms	Yes	3.3 W

debido a que su procesamiento se basa en slots temporales que asegura una operación en tiempo real. En contraste, arquitecturas digitales como SpiNNaker que basan su procesamiento en eventos, no aseguran una operación en una escala temporal biológica, en donde además no es posible modelar fuentes de ruido con precisión ya que esto requiere un continuo procesamiento temporal. Téngase en cuenta que el ruido ha sido reportado como un fenómeno fundamental en el funcionamiento de los sistemas neuronales. Las transacciones con la memoria externa dada en varias aproximaciones hardware para poblar y actualizar tablas de enrutamiento contribuyen a la latencia y al consumo de energía.

5.12. Conclusiones

La limitación en ancho de banda del bus que establece la comunicación P2P entre chips, es solventada por medio de los transceivers seriales de alta velocidad que utilizan señalamiento diferencial, soportando un ancho de banda de 1Gbps. Para una operación en tiempo real, el número de spikes distribuidos por el bus AER está en el orden de 10^5.

Así también, el tiempo que consume la fase de inicialización y de configuración no afectan el desempeño de la arquitectura ya que se producen una sola vez al inicio de la puesta en marcha del sistema.

La latencia en la transmisión tiene un componente fijo introducido por el MC de entre 30 a 40 ciclos dependiendo de la fase que se esté procesando. El resto es introducido por la lógica de control del protocolo AER-SRT, así como por el protocolo Aurora que añade un overhead por cada trama enviada por el canal.

Los ciclos que se emplean en la sincronización de los nodos en la fase de distribución no son significativos considerando condiciones de alta actividad de spikes, dicha condición se cumple especialmente para redes neuronales de gran escala como es el caso de HEENS. Las condiciones de transmisión más idóneas son precisamente un anillo con pocos nodos con alta actividad de spikes.

El throughput obtenido es en promedio de 1 spike por ciclo de reloj con una latencia aproximada

de 39 ciclos por NC. Con el esquema utilizado, el modelo de comunicación AER-SRT nos permite transferir gran cantidad de eventos en un canal libre de colisiones dentro de la franja de tiempo real establecida de 0.5 ms. Se puede alcanzar mayores tasas de transmisión incrementando la frecuencia de reloj del multiprocesador y la velocidad de los transceivers.

El bus AER en anillo permite que el MC pueda monitorizar todos los spikes del sistema, pudiendo almacenar el funcionamiento de la red o representar en raster plots su dinámica en tiempo real.

Referencias

[1] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla y K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations", *Proceedings of the IEEE*, vol. 102, n.º 5, págs. 699-716, 2014, ISSN: 00189219. DOI: 10.1109/JPROC.2014.2313565.

[2] M. Ehrlich, C. Mayr, H. Eisenreich, S. Henker, A. Srowig, A. Grübl, J. Schemmel y R. Schüffny, "Wafer-scale {VLSI} implementations of pulse coupled neural networks", *Proceedings of 4th IEEE International Multi-Conference on Systems, Signals & Devices SSD07, electronic publication, Abstract on page 409 of Proc.*, 2007.

[3] J. Schemmel, J. Fieres y K. Meier, "Wafer-scale integration of analog neural networks", *Proceedings of the International Joint Conference on Neural Networks*, págs. 431-438, 2008, ISSN: 1098-7576. DOI: 10.1109/IJCNN.2008.4633828.

[4] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar y D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface", *Science*, vol. 345, n.º 6197, págs. 668-673, 2014, ISSN: 0036-8075. DOI: 10.1126/science.1254642.

[5] J. Park, T. Yu, S. Joshi, C. Maier y G. Cauwenberghs, "Hierarchical Address Event Routing for Reconfigurable Large-Scale Neuromorphic Systems", *IEEE Transactions on Neural Networks and Learning Systems*, págs. 1-15, 2016, ISSN: 21622388. DOI: 10.1109/TNNLS.2016.2572164.

[6] J. Vogelstein, U. Mallik y col., "Dynamically Reconfigurable Silicon Array of Spiking Neurons With Conductance Based Synapses", *IEEE Transactions on Neural Network*, 2007.

[7] D. B. Thomas y W. Luk, "FPGA accelerated simulation of biologically plausible spiking neural networks", *Proceedings - IEEE Symposium on Field Programmable Custom Computing Machines, FCCM 2009*, págs. 45-52, 2009. DOI: 10.1109/FCCM.2009.46.

[8] D. Pani, P. Meloni, G. Tuveri, F. Palumbo, P. Massobrio y L. Raffo, "An FPGA Platform for Real-Time Simulation of Spiking Neuronal Networks", *Frontiers in Neuroscience*, vol. 11, n.º February, 2017, ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00090.

[9] K. Cheung, S. R. Schultz y W. Luk, "NeuroFlow: A general purpose spiking neural network simulation platform using customizable processors", *Frontiers in Neuroscience*, vol. 9, págs. 1-15, 2016, ISSN: 1662453X. DOI: 10.3389/fnins.2015.00516.

[10] A. D. Rast, X. Jin, F. Galluppi, L. A. Plana, C. Patterson y S. Furber, "Scalable event-driven native parallel processing: the spinnaker neuromimetic system", CF '10, págs. 21-30, 2010. DOI: 10.1145/1787275.1787279. dirección: http://doi.acm.org/10.1145/1787275.1787279.

Capítulo 6

Conclusiones

En este último capítulo se describe los resultados más relevantes y las conclusiones obtenidas a lo largo del desarrollo de esta tesis. También se puntualiza posibles trabajos futuros y líneas de investigación que serán de utilidad no solamente para la arquitectura HEENS, sino también para el amplio espectro de arquitecturas hardware de SNN

6.1. Conclusiones

En los últimos años ha surgido un enorme interés por desarrollar aplicaciones de ingeniería basadas en inteligencia artificial en general y neuronales en particular que permitan superar las limitaciones de los sistemas de computación clásicos. Estos últimos no parecen adecuados para resolver problemas de percepción que los seres vivos realizan de forma altamente optimizada. Por otro lado, los grandes avances en neurociencia permiten plantear la modelación de redes neuronales bioinspiradas con aplicaciones tecnológicas pero también de utilidad para el proceso en el conocimiento del cerebro.

La arquitectura HEENS presentada en esta tesis pretende contribuir a la existente necesidad de contar con sistemas biológicamente plausibles capaces de emular redes de gran escala para estudiar la dinámica del sistema nervioso. Siendo uno de sus principales objetivos, el que permitan contrastar resultados con datos de imágenes del cerebro que potencialmente pueda utilizarse para el desarrollo de tratamientos médicos o aplicaciones tecnológicas.

En este sentido, la aproximación computacional de HEENS imita la configuración del cerebro en su modelo de red distribuida de procesamiento de información. En el diseño se tomó en cuenta los principios de organización a gran escala del córtex incluyendo propiedades de ”small world”, modularidad y alta eficiencia en la comunicación. A nivel evolutivo, la ventaja de dividir la red en módulos representa una alta especialización y minimización de los costos de conexión, y en nuestro caso la consecuentemente reducción el tamaño de las tablas de enrutamiento. Así también se maximiza el valor adaptativo de la red y se mejora su robustez ante perturbaciones originadas en otros módulos.

La modularidad y escalabilidad de HEENS es introducida a través de su modelo de comunicación. Mediante la conexión multi-chip en una topología de anillo se consigue lograr una excelente

escalabilidad y baja latencia para condiciones de alto tráfico, asegurando la operación del sistema dentro de la escala considerada como tiempo real. La latencia escala linealmente con el tamaño del anillo. Así también la incorporación de NC en la red es simple con baja complejidad a nivel de hardware, proporcionando una arquitectura eficiente y a su vez portable.

Además, la estructura jerárquica de HEENS diferencia entre spikes locales y globales, lo cual permite eventualmente reducir el tráfico por la red y como consecuencia mejorar la escalabilidad. Así mismo, permite de optimizar el consumo de recursos de memoria utilizados para la decodificación de spikes, favoreciendo la densidad neuronal por chip.

La eficiencia en el consumo de recursos también está dada en la reutilización del canal de comunicación para la configuración de los NC del anillo. Esto permite eliminar la necesidad de una interfaz de comunicación adicional en cada chip destinada para este propósito con lo que se puede explotar la lógica disponible para emular más neuronas, optimizando el uso de recursos por chip. La función de reconfiguración dinámica que da soporte para la implementación de aplicaciones evolutivas también aprovecha el mismo enlace.

La incorporación del MC para la comunicación entre la plataforma de emulación y el entorno externo introduce una latencia adicional a la red. No obstante, al ser implementado con una FPGA de la familia Xilinx Zynq proporciona las ventajas de un PSoC (Programmable System on Chip). Este tipo de dispositivos al disponer de un procesador integrado, controladores de memoria y un set muy amplio de periféricos integrado con lógica programable, se adecua perfectamente a los requerimientos de HEENS, tanto para las funciones de configuración, como para expandirse a muchas otras aplicaciones.

Por otro lado, HEENS soporta un grado muy fino de programabilidad a nivel del modelo neuronal a implementar, que la distingue de muchas otras aproximaciones. Así también, las herramientas de software desarrolladas permiten agilizar la puesta en marcha de una nueva aplicación, sin que se necesiten conocimientos específicos de diseño de hardware. Estos aspectos benefician a los usuarios finales al contar con una plataforma de modelamiento flexible, accesible y de fácil configuración para la emulación de SNN de gran escala en tiempo real.

Así también, el presente sistema es eficiente implementado redes spiking de gran tamaño en el orden de 10^5 neuronas usando conexiones punto a punto. Sin embargo, la limitación en este caso es impuesta por el tamaño de la FIFOs. Y un uso eficiente de memoria on-chip consecuencia del esquema de memoria asociativa utilizado, que además se refleja en un bajo consumo de potencia/chip en relación a otras aproximaciones que utilizan DRAM.

Finalmente, puesto que HEENS está basada en FPGA, puede sin mucho esfuerzo mejorar sus prestaciones migrando a circuitos integrados de aplicación específica (ASIC). De hecho, HEENS está diseñado con esta concepción, ya que su diseño paramétrico permite ajustar el tamaño del arreglo de acuerdo a la consideración del usuario dentro de los límites dados por las restricciones de hardware.

Por todos los puntos mencionados, la arquitectura HEENS ofrece una solución prometedora a la emulación de SNN, que contribuye al estado del arte con su conjunto de prestaciones en una sola plataforma.

6.2. Trabajo futuro

Se plantean como líneas futuras de investigación la extensión de los niveles de jerarquía del modelo de comunicación para mejorar su expandibilidad, migrando de una topología tipo anillo a una tipo malla 2D, haciendo uso de las tarjetas SMA ya desarrolladas que se muestran en la sección 4.4.1.

Aumentar la tasa de transmisión de los transceivers para la comunicación inter-chip, la cual actualmente se encuentra limitada por defectos de fábrica en la Carrier Card de la Picozed usada en la implementación del MC.

Para mejorar las prestaciones por chip de HEENS, se puede incrementar el número de sinapsis por neurona combinando el uso de memoria on-chip con SRAM. Estas son de rápido acceso y presentan un consumo eficiente de potencia a diferencia de las DRAM usadas en otras aproximaciones, además de contribuir al consumo de energía.

Se puede incorporar fácilmente retardos dendríticos en los NC, con lo cual se podría añadir la característica de controlar la distancia entre neuronas en un mismo módulo.

A nivel de software se puede desarrollar una API (Application Programming Interface) que integre las tareas de mapeo, configuración, monitoreo de la actividad de la red e inyección de eventos externos de sensores AER , etc., que interactúe con HEENS por medio de la Picozed a través de una sola una interface de comunicación como por ejemplo Ethernet.

Otra línea abierta es el desarrollo de aplicaciones que incorporen sensores neuromórficos para obtener sistemas de procesamiento sensorial artificial con capacidades cognitivas.

Anexo I

Set de Instrucciones del Secuenciador

Instruction	Group	Format	Opcode	Hex	Flags*	En**	Function
NOP	SEQ	NOP	000000	00			No operation
LDALL	REGISTERS	LDALL reg ****	000001	01	Z***	/F	reg <= DMEM (from sequencer)
LLFSR	MOVEMENT	LLFSR	000010	02	Z	/F	ACC <= LFSR(15:0)
LOADSP	LOADSP	LOADSP	000011	03		/F	R1 & ACC(15:1) <= BRAM(BP,31:1); ACC(0) <= spike_register(BP(3:0))
STOREB	STOREB	STOREB	000100	04			EXT_BUFFER <= ACC
STORESP	STORESP	STORESP	000101	05		/F	BRAM(BP) <= R1 & ACC; BP <= BP + 1
STOREPS	STOREPS	STOREPS	000110	06		/F	AER_FIFO <= ACC(0) (post-synaptic Si)
RST	REGISTERS	RST reg	000111	07	Z***	/F	reg <= “0000”
SET	REGISTERS	SET reg	001000	08	Z***	/F	reg <= “FFFF”
SHLN	REGISTERS	SHLN n	001001	09	C,Z	/F	ACC <= ACC << n, (1 <= n <= 8), (n = number of positions)
SHRN	REGISTERS	SHRN n	001010	0A	C,Z	/F	ACC <= ACC >> n, (1 <= n <= 8), (n = number of positions)
RTL	REGISTERS	RTL	001011	0B	C,Z	/F	ACC <= ACC <<, carry = ACC(msb) Rotate Accumulator Left
RTR	REGISTERS	RTR	001100	0C	C,Z	/F	ACC <= ACC >>, carry = ACC(lsb) Rotate Accumulator Right
INC	ARITHMETIC	INC	001101	0D	C,Z	/F	ACC <= ACC + 1
DEC	ARITHMETIC	DEC	001110	0E	C,Z	/F	ACC <= ACC - 1
LOADSN	LOADSN	LOADSN	001111	0F	C,Z	/F	R1 & ACC <= BRAM(BP)
ADD	ARITHMETIC	ADD reg	010000	10	C,Z	/F	ACC <= ACC + reg (Saturated addition)
SUB	ARITHMETIC	SUB reg	010001	11	C,Z	/F	ACC <= ACC – reg (Saturated subtraction)
MUL	ARITHMETIC	MUL reg	010010	12	Z	/F	ACC & R1 <= ACC * reg (Signed product)
MULS	ARITHMETIC	MULS reg	010011	13	Z	/F	ACC <= ACC * reg (Most significant word signed product)
AND	LOGIC	AND reg	010100	14	Z	/F	ACC <= ACC AND reg
OR	LOGIC	OR reg	010101	15	Z	/F	ACC <= ACC OR reg
INV	LOGIC	INV reg	010110	16	Z	/F	ACC <= INV reg
XOR	LOGIC	XOR reg	010111	17	Z	/F	ACC <= ACC XOR reg
MOVA	MOVEMENT	MOVA reg	011000	18	Z	/F	ACC <= reg
MOVR	MOVEMENT	MOVR reg	011001	19		/F	reg <= ACC
SWAPS	MOVEMENT	SWAPS reg	011010	1A	Z***	/F	reg <=> shadow_reg (Swap register)
MOVRS	MOVEMENT	MOVRS reg	011011	1B	Z***	/F	reg <= shadow_reg
LOOP	SEQ	LOOP n	011100	1C			Push LOOP_BUFFER(n-1);Push PC_BUFFER(PC+1)
LOOPV	SEQ	LOOPV ****	011101	1D			Push LOOP_BUFFER(DMEM-1);Push PC_BUFFER(PC+1)
ENDL	SEQ	ENDL	011110	1E			If LOOP_BUFFER = 0 then pop LOOP_BUFFER; pop PC_BUFFER; else LOOP_BUFFER <= LOOP_BUFFER – 1; PC <= PC_BUFFER

Instruction	Group	Format	Opcode	Hex	Flags*	En**	Function
GOSUB	SEQ	GOSUB addr	011111	1F			PC <= addr; Push PC_BUFFER(PC+1)
RET	SEQ	RET	100000	20			PC <= PC_BUFFER
FREEZEC	CONDITIONAL	FREEZEC	100001	21	F		if C=1 then F <= 1; push F_BUFFER(1)
FREEZENC	CONDITIONAL	FREEZENC	100010	22	F		if C=0 then F <= 1; push F_BUFFER(1)
FREEZEZ	CONDITIONAL	FREEZEZ	100011	23	F		if Z=1 then F <= 1; push F_BUFFER(1)
FREEZENZ	CONDITIONAL	FREEZENZ	100100	24	F		if Z=0 then F <= 1; push F_BUFFER(1)
UNFREEZE	CONDITIONAL	UNFREEZE	100101	25	F		F <= pop F_BUFFER
HALT	SEQ	HALT	100110	26			INT<=1;sequencer halted until external input signal INT_ACK=1
SETZ	FLAGS	SETZ	100111	27	Z	/F	Z <= 1
SETC	FLAGS	SETC	101000	28	C	/F	Sets the carry flags C <= 1
CLRZ	FLAGS	CLRZ	101001	29	Z	/F	Clears the zero flags Z <= 0
CLRC	FLAGS	CLRC	101010	2A	C	/F	Clears the zero flags C <= 0
RANDON	RAND	RANDON	101011	2B		/F	random_en <= 1; LFSR becomes source register for LLFSR
SEED	MOVEMENT	SEED	101100	2C		/F	LFSR(63:32) <= LFSR(31:0) <= R1 & ACC
RANDOFF	RAND	RANDOFF	101101	2D		/F	random_en <= 0; LFSR_STEP <=0; LFSR disabled
SPKDIS	SEQ	SPKDIS	101110	2E			eo_exec <= 1, Stops the sequencer and stores spikes until input signal cam_en <= 0 (from AER control unit)
READMP	SEQ	READMP addr	101111	2F			DMEM <= BRAM(address)
RST_SEQ	SEQ	RST_SEQ	110000	30			Jumps to RESET state
LAYERV	SEQ	LAYERV n	110010	32			SEQ_VIRT <= n; defines number of virtual layers (currently 0 <= n <= 7)
GOTO	SEQ	GOTO addr	110011	33			PC <= addr
SHLAN	REGISTERS	SHLAN n	110100	34	C,Z	/F	ACC <= ACC << n, (1 <= n <= 8), Arithmetic shift
SHRAN	REGISTERS	SHRAN n	110101	35	C,Z	/F	ACC <= ACC >> n, (1 <= n <= 8), Arithmetic shift
LOADBP	LOADBP	LOADBP ****	110110	36		/F	BP <= DMEM Loads PE BRAM pointer.
BITSET	REGISTERS	BITSET n	110111	37	Z	/F	ACC(n) <= 1
BITCLR	REGISTERS	BITCLR n	111000	38	Z	/F	ACC(n) <= 0
SPMOV	SPMOV	SPMOV n	111001	39		/F	Special MOVE. n = 0: VIRT <= ACC;
INCV	SEQ	INCV	111010	3A			SEQ_VIRT <= SEQ_VIRT + 1
READMPV	SEQ	READMPV addr	111011	3B			DMEM <= BRAM(address + SEQ_VIRT)
MOVSR	MOVEMENT	MOVSR reg	111100	3C		/F	shadow_reg <= reg

*Flags If the given instruction can change the indicated flag

** En

F: Frozen flag. /F= not(F) means unfrozen and the indicated instructions become enabled

*** Z can change only if ACC is set or reset (not in case of other registers)

**** See macros

MACROS:

LDALL		LDALL reg, const	reg <= DMEM(const) (from sequencer)
Elementary instructions:		NOP	
		READMP const	
		LDALL reg	

LOOPV		LOOPV vp	Push LOOP_BUFFER(DMEM(vp)-1);Push PC_BUFFER(PC+1)
Elementary instructions:		NOP	
		READMPV vp	
		LOOPV	

LOADBP		LOADBP bp	BP <= DMEM(bp) Loads PE BRAM pointer.
Elementary instructions:		NOP	
		READMP bp	
		LOADBP	

Anexo II

Programa Ensamblador 1

```
; Network definitions

define virtual_layers 7  ; From 0 up to 7
define gsynapses  0      ; Up to 32 global synapses
define lsynapses 99      ; 99  Due to the local RAM encoding, synapse 0 cannot be used
                         ;(corresponds to no synapse code)
.DATA

; Virtual layers

V0 = "00000002"        ; Number of assigned synapses (s-1) to the main layer 100 sinapsis
V1 = "00000002"        ; Number of assigned synapses (s-1) to virtual layer 1
V2 = "00000002"        ; Number of assigned synapses (s-1) to virtual layer 2
V3 = "00000002"        ; Number of assigned synapses (s-1) to virtual layer 3
V4 = "00000002"        ; Number of assigned synapses (s-1) to virtual layer 4
V5 = "00000002"        ; Number of assigned synapses (s-1) to virtual layer 5
V6 = "00000002"        ; Number of assigned synapses (s-1) to virtual layer 6
V7 = "00000002"        ; Number of assigned synapses (s-1) to virtual layer 7
VLAYERS="00000000"     ; Number of virtual layers (n-1).

; Membrane potential parameters common to all neurons
VREST= "FFFFE4A8"      ; Resting potential -70 mV = -7000 in tens of of uV
VTHRES="FFFFEA84"      ; Threshold voltage -55 mV = -5500
VDEPOL="FFFFE0C0"      ; Depolarization voltage -80 mV = -8000
VACT = "00001771"      ; Action potential +10 mV = +1000

; Synapse parameters common to all neurons come here
; Neural and Synaptic RAM addresses

SYN_ADDR="00000000"     ; First address of Synaptic parameters in SNRAM.
NEU_ADDR="000003E3"     ; First address of Neural parameters in SNRAM (995)
SEEDH_ADDR = "000003FD" ; Address of noise seed in SNRAM
SEEDL_ADDR = "000003FE"
```

```
; General constants
THAU_MEM="00007EE0"     ; Membrane time constant decay (inverse value). To be tuned
NOISE_MSK="0000001F"    ; Noise mask. To be tuned
INIT_VAL ="FFFFE890"   ; Vmem initiated at -60 mV, 10 mV above rest potential

.CODE
;
GOTO MAIN                     ; Jump to main program
;
; ***************************** PROCEDURES BEGIN ****************************
;
.RANDOM_INIT                  ; Uses R0 and R1
        LOADBP SEEDH_ADDR
        LOADSN
        SEED                  ; High seed
        LOADBP SEEDL_ADDR
        LOADSN
        SEED                  ; Low seed
RET
;
.LOAD_NEURON                  ; Uses R0, R1, R2 and R3
        READMPV NEU_ADDR      ; Address of real neuron + virt (valid also for non-virtual)
        LOADBP                ; SNRAM pointer to currently processed neuron
        LOADSN                ; Load Neural parameters from SNRAM to R1 & ACC
        MOVR R2               ; Move Vmem from ACC to R2
RET
;
.MEMBRANE_DECAY               ; Uses R0, R4
        MOVA R2
        LDALL R4 VREST
        SUB R4
        LDALL R1, THAU_MEM
        MULS R1               ; Calculate decay
        SHLAN 1
        ADD R4
        MOVR R2               ; Back to R2 where membrane potential is stored
RET
;
.ADD_NOISE ; Uses R0, R2 and R5
        RANDON                ; LFSR ON
        LLFSR ; Noise to ACC
        MOVR R5
        LDALL ACC, NOISE_MSK
        AND R5
        SHRN 1
        RANDOFF               ; LFSR OFF. Arbitrarily here
        FREEZENC
            MOVR R5
            RST ACC
            SUB R5            ; Generate signed noise without the negative bias of two's
complement
        UNFREEZE
        MOVSR ACC            ;    TO MONITOR THE NOISE
        ADD R2  ; Add to Vmem
        MOVR R2 ; Back to R2
RET
;
```

```
.SYNAPSE_CALC
    LOADSP                    ; Load Synaptic parameters and spike to R1 & ACC
    SHRN 1                    ; Move spike to flag C
    FREEZENC
         MOVA R1              ; Synaptic parameter to ACC
        ADD R2
        MOVR R2               ; Save Neural parameter in R2
    UNFREEZE
    RST ACC
    STORESP                   ; Stores synaptic parameter and increases BP for next synapse
processing
RET
;
.DETECT_SPIKE                 ; Uses R0 and R2
       LDALL ACC, VTHRES
       SUB R2                 ; Compare Vth - Vmem
       SHLN 1                 ;subtraction sign to C flag
       RST ACC
       FREEZENC               ; If positive, spike
              SET ACC
              LDALL R2 VREST  ; Vmem to resting potential
       UNFREEZE
       STOREPS                ; Push spikes
RET
;
.STORE_NEURON                 ; uses R0 and R1
       MOVA R2                ; Move Vmem from R2 to ACC
       READMPV NEU_ADDR       ; Address of real neuron + virt (valid also for non-virtual)
       LOADBP                 ; SNRAM pointer to currently processed neuron
       STORESP                ; Store Vmem to SNRAM
RET
;
; *************************** PROCEDURES END ******************************

; *************************** MAIN PROGRAMME BEGIN ************************

.MAIN
; Virtual operation init
LAYERV virtual_layers         ; Init sequencer vlayers. It is 0 for non-virtual operation
LDALL ACC, VLAYERS            ; Load defined virtual layers to PE array
SPMOV 0                       ; VIRT <= ACC

; Initial instructions
GOSUB RANDOM_INIT             ; For noise initialization
.EXEC_LOOP                    ; Execution loop

LOOP virtual_layers ; Neuron loop for virtual operation
    NOP   ;to prevent pipeline error
    GOSUB LOAD_NEURON
    GOSUB MEMBRANE_DECAY      ; Calculate membrane potential decay
    GOSUB ADD_NOISE
    LOADBP SYN_ADDR           ; Initial position for addresses
           LOOPV V0           ; synaptic loop. Reads number of current-layer synapses
            NOP               ;to prevent pipeline error
            GOSUB SYNAPSE_CALC
        ENDL
    ; Compare and eventually spike
    GOSUB DETECT_SPIKE
    GOSUB STORE_NEURON
    INCV
    ENDL
NOP                           ; Empty pipeline wait NOPs
NOP
NOP
SPKDIS                        ; Distribute spikes
GOTO EXEC_LOOP                ; Execution loop
```

Anexo III

Programa Ensamblador 2

```
; Network definitions
define virtual_layers 0       ; From 0 up to 7
define gsynapses      1       ; Up to 32 global synapses
define lsynapses      15      ;99  Due to the local RAM encoding, synapse 0 cannot be used ;
                              (corresponds to no synapse code)
.DATA

; Virtual layers

V0 = "00000001"        ; Number of assigned synapses (s-1) to the main layer
V1 = "00000001"        ; Number of assigned synapses (s-1) to virtual layer 1
V2 = "00000001"        ; Number of assigned synapses (s-1) to virtual layer 2
V3 = "00000001"        ; Number of assigned synapses (s-1) to virtual layer 3
V4 = "00000001"        ; Number of assigned synapses (s-1) to virtual layer 4
V5 = "00000001"        ; Number of assigned synapses (s-1) to virtual layer 5
V6 = "00000001"        ; Number of assigned synapses (s-1) to virtual layer 6
V7 = "00000001"        ; Number of assigned synapses (s-1) to virtual layer 7
VLAYERS="00000000"     ; Number of virtual layers (n-1).

; Membrane potential parameters common to all neurons

VREST ="FFFFE4A8"      ; Resting potential -70 mV = -7000 in tens of of uV
VTHRES="FFFFEA84"      ; Threshold voltage -55 mV = -5500
VDEPOL="FFFFE0C0"      ; Depolarization voltage -80 mV = -8000
VACT  ="00001771"      ; Action potential +10 mV = +1000

; Synapse parameters common to all neurons come here
; Neural and Synaptic RAM addresses

SYN_ADDR="00000000"           ; First address of Synaptic parameters in SNRAM.
GSYN_ADDR="0000000F"          ; First address of Global Synaptic parameters in SNRAM.
NEU_ADDR="000003E3"           ; First address of Neural parameters in SNRAM (995)
SEEDH_ADDR = "000003FD"       ; Address of noise seed in SNRAM
SEEDL_ADDR = "000003FE"
```

```
; General constants

THAU_MEM ="00007EE0"          ; Membrane time constant decay (inverse value). To be tuned
NOISE_MSK="0000001F"          ; Noise mask. To be tuned
INIT_VAL ="FFFFE890"          ; Vmem initiated at -60 mV, 10 mV above rest potential

.CODE
;
GOTO MAIN      ; Jump to main program
;
; *************************** PROCEDURES BEGIN ***************************

.RANDOM_INIT                  ; Uses R0 and R1
        LOADBP SEEDH_ADDR
        LOADSN
        SEED                  ; High seed
        LOADBP SEEDL_ADDR
        LOADSN
        SEED                  ; Low seed
RET

.LOAD_NEURON                  ; Uses R0, R1, R2 and R3
        READMPV NEU_ADDR      ; Address of real neuron + virt (valid also for non-virtual)
        LOADBP                ; SNRAM pointer to currently processed neuron
        LOADSN                ; Load Neural parameters from SNRAM to R1 & ACC
        MOVR R2               ; Move Vmem from ACC to R2
RET

.MEMBRANE_DECAY               ; Uses R0, R4
        MOVA R2
        LDALL R4 VREST
        SUB R4
        LDALL R1, THAU_MEM
        MULS R1               ; Calculate decay
        SHLAN 1
        ADD R4
        MOVR R2               ; Back to R2 where membrane potential is stored
RET

.ADD_NOISE                    ; Uses R0, R2 and R5
        RANDON                 ; LFSR ON
        LLFSR ; Noise to ACC
        MOVR R5
        LDALL ACC, NOISE_MSK
        AND R5
        SHRN 1
        RANDOFF               ; LFSR OFF. Arbitrarily here
        FREEZENC
            MOVR R5
            RST ACC
            SUB R5     ; Generate signed noise without the negative bias of two's complement
        UNFREEZE
         MOVSR ACC            ; TO MONITOR THE NOISE
        ADD R2                ; Add to Vmem
        MOVR R2               ; Back to R2
RET
```

```
.SYNAPSE_CALC
    LOADSP             ; Load Synaptic parameters and spike to R1 & ACC
    SHRN 1             ; Move spike to flag C
    FREEZENC
         MOVA R1       ; Synaptic parameter to ACC
        ADD R2
        MOVR R2        ; Save Neural parameter in R2
    UNFREEZE
    RST ACC
    STORESP    ; Stores synaptic parameter and increases BP for next synapse processing
RET

.DETECT_SPIKE                  ; Uses R0 and R2
        LDALL ACC, VTHRES
        SUB R2                 ; Compare Vth - Vmem
        SHLN 1                 ;subtraction sign to C flag
        RST ACC
        FREEZENC               ; If positive, spike
                SET ACC
                LDALL R2 VREST ; Vmem to resting potential
        UNFREEZE
        STOREPS                ; Push spikes
RET
;
.STORE_NEURON ; uses R0 and R1
        MOVA R2                ; Move Vmem from R2 to ACC
        READMPV NEU_ADDR        ; Address of real neuron + virt (valid also for non-virtual)
        LOADBP                  ; SNRAM pointer to currently processed neuron
        STORESP                 ; Store Vmem to SNRAM
RET
;
; **************************** PROCEDURES END ******************************
; **************************** MAIN PROGRAMME BEGIN ************************
.MAIN
;
; Virtual operation init
LAYERV virtual_layers          ; Init sequencer vlayers. It is 0 for non-virtual operation
LDALL ACC, VLAYERS             ; Load defined virtual layers to PE array
SPMOV    0                     ; VIRT <= ACC

; Initial instructions
GOSUB RANDOM_INIT              ; For noise initialization

.EXEC_LOOP ; Execution loop

; LAYER 0 NEURON
; Global synapses (layer 0)
GOSUB LOAD_NEURON
GOSUB MEMBRANE_DECAY           ; Calculate membrane potential decay
GOSUB ADD_NOISE
LOADBP GSYN_ADDR
LOOP gsynapses
    NOP
    GOSUB SYNAPSE_CALC
ENDL
LOADBP SYN_ADDR                ; Initial position for addresses
LOOPV V0                       ; synaptic loop. Reads number of current-layer synapses
    NOP                        ;to prevent pipeline error
    GOSUB SYNAPSE_CALC
ENDL
GOSUB DETECT_SPIKE
GOSUB STORE_NEURON

; End of global synapses
```

```
; IF VIRTUAL NEURONS then loop
LDALL ACC, VLAYERS
FREEZENZ
    GOTO FINISH1
UNFREEZE
;LAYER 1 to V-1 NEURONS
INCV
LOOP virtual_layers           ; Neuron loop for virtual operation
    NOP
    GOSUB LOAD_NEURON
    GOSUB MEMBRANE_DECAY       ; Calculate membrane potential decay
    GOSUB ADD_NOISE
    LOADBP SYN_ADDR            ; Initial position for addresses
    LOOPV V0                   ; synaptic loop.  Reads number of current-layer synapses
            NOP                       ; to prevent pipeline error
            GOSUB SYNAPSE_CALC
     ENDL
    ; Compare and eventually spike
    GOSUB DETECT_SPIKE
    GOSUB STORE_NEURON
    INCV
    ENDL
.FINISH
GOTO FINISH2
.FINISH1
UNFREEZE
.FINISH2
NOP                           ; Empty pipeline wait NOPs
NOP
NOP
SPKDIS                        ; Distribute spikes
GOTO EXEC_LOOP                ; Execution loop
```