Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Desarrollo y validación de un método para la visualización de resultados en la implementación del algoritmo de simulación de redes neuronales

Informe de Proyecto de Graduación para optar por el título de

Ingeniero en Electrónica con el grado académico de Licenciatura

Daniel Zamora Umaña

Cartago, Noviembre, 2017

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Proyecto de Graduación
Tribunal Evaluador

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

MSc.Ing. Carlos Salazar García
Profesor Lector

Ing. Javier Pérez Rodríguez
Profesor Lector

Dr.Ing. Alfonso Chacón Rodríguez
Profesor Asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica.

Cartago, 14 de noviembre de 2017

Declaratoria de autenticidad

Declaro que el presente Proyecto de Graduación ha sido realizado, en su totalidad, por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado material bibliográfico, he procedido a indicar las fuentes mediante referencias bibliográficas correspondientes.

En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Daniel Zamora Umaña

Cartago, 14 de noviembre de 2017
Céd: 115600917

Resumen

En el presente informe se expone el proceso de diseño, implementación, principales resultados y conclusiones de un entorno de simulación de redes neuronales biológicamente precisas sobre una plataforma de múltiples dispositivos FPGA.

El modelo utilizado para realizar la emulación cerebral es el Hodgkin-Huxley extendido como referencia, implementado en hardware en un dispositivo FPGA. Sin embargo, este no poseía forma de comunicación con otros dispositivos. Para implementar esta interfaz de comunicación se exponen y comparan dos métodos, el primero construido a partir de sockets y una aplicación bare-metal mínima generada con Vivado SDK. El segundo método es manejado por una implementación de Message Passing Interface (MPI) sobre una distribución de Ubuntu mínima en el ARM de la ZYNQ.

La aplicación se encuentra controlada por un host remoto que se trata de una computadora personal, este le da soporte a una página web la cual programa la simulación deseada y realiza la visualización de los resultados que se aprecian en gráficas generadas por Python tanto individual como en conjunto.

Palabras clave: red neuronal, socket, MPI, página web.

Abstract

This report exposes the design and implementation of multi-FPGA based brain simulations platform. Also, it includes the main results and conclusions for this process.

The model used to emulate the brain is the extended Hodgkin-Huxley model. It was implemented in hardware by FPGA devices. At first instance, this implementation did not have a communication's method with any other device. To solve it, it was implemented a communication interface by comparing two differents methods: the first one, a socket- based implementation over a bare-metal application. The second one, a MPI's implementation over a Ubuntu distribution running in the ZYNQ.

The application runs on a remote host, which works through a personal computer, which contains a web page that allows you to configure a new simulation and displays the results through Python-generated graphics.

Keywords: neural network, socket, MPI, web page.

a mis queridos padres

Agradecimientos

A mi familia, por todo el apoyo y soporte brindado a lo largo no solo de la ejecución de este proyecto, sino de toda la vida universitaria incluyendo sus altos y bajos.

A los profesores Alfonso Chacón y Carlos Salazar, por toda la paciencia y la guía brindadas durante mi paso como asistente del Laboratorio de Diseño de Circuitos Integrados y por supuesto, durante la ejecución del presente proyecto de graduación.

Daniel Zamora Umaña

Cartago, 14 de noviembre de 2017

Índice general

Índice de figuras

Índice de tablas

Lista de símbolos y abreviaciones

Abreviaciones

ARM ®	Siglas del inglés Advanced RISC Machine
BSP	Siglas del inglés Board Support Package
CSS	Siglas del inglés Cascading Style Sheets
DCN	Siglas del inglés Deep Cerebellar Nuclei
FPGA	Siglas del inglés Field programmable gate array
HDL	Siglas del inglés Hardware Description Language
IDE	Siglas del inglés Integrated Design Environment
IO	Siglas del inglés Inferior Olivary
ION	Siglas del inglés Inferior Olivary Nucleus
IP core	Del inglés Intellectual Property Core
IP	Del inglés Internet Protocol, no debe confundirse con IP core.
LAN	Siglas del inglés Local Area Network
MAC	Siglas del inglés Media Access Control
MPI	Siglas del inglés Message Passing Interface
PC	Siglas del inglés Purkinje Cell Layer
PHP	Siglas del inglés Hipertext Preprocessor
PL	Siglas del inglés Programmable Logic
PS	Siglas del inglés Processing System
SD	Siglas del inglés Secure Digital
SDK	Siglas del inglés Software Development Kit
SoC	Del inglés System on Chip
TCP	Siglas del inglés Transmission Control Protocol
USB	Siglas del inglés Universal Serial Bus

Capítulo 1

Introducción

1.1 Entorno del Problema

Los laboratorios DCILab (Laboratorio de Diseño de Circuitos Integrados) y HPCLab (Laboratorio de Computación de Alto Desempeño, o High Performance Computation Lab), son laboratorios de la Escuela de Ingeniería Electrónica que dedican su área a la integración de circuitos microelectrónicos, y a aplicaciones de arquitecturas de sistemas de alta integración y desempeño, respectivamente. Dentro de esta última área, estos laboratorios trabajan en conjunto en un proyecto llamado *"Implementación Multi-FPGA de modelos artificiales del cerebro"*, el cual tiene como finalidad realizar simulaciones de una red neuronal usando un modelo biológicamente preciso para emular regiones del cerebro y el comportamiento ante determinados estímulos.

Este proyecto se trabaja en conjunto con el Centro Médico Erasmus de Rotterdam, Holanda, en colaboración con su *Erasmus BrainProject*, el cual tiene como finalidad brindar herramientas para la investigación médica y científica del cerebro, así como brindar soluciones a problemas específicos relacionados con neurociencia.

En el cerebro, el olivar inferior(Inferior Olivary, IO por el resto del documento) según [3] es una de las partes más densas del cerebro y se encarga del manejo y control de las actividades senso-motoras del cuerpo humano. Biológicamente, según [4], el comportamiento de cada neurona se ve afectado por la actividad eléctrica en ella. Este es producto de la combinación de cambios externos en el potencial de la membrana y la concentración interna de los componentes químicos involucrados en esta actividad: calcio, potasio y sodio. Para generar una salida, la neurona recibe una corriente de entrada desde el núcleo cerebral y a partir de la misma, se genera una corriente de salida de acuerdo a la reacción electroquímica que genera en su interior. Además de la corriente de entrada, la neurona ve afectado su comportamiento de acuerdo a las neuronas que se encuentran a su alrededor, por lo que podemos ver una red neuronal como una interconexión entre todas los miembros que la componen. Cada celda está constituida por tres partes: soma, dendrita y axón; la dendrita es la etapa de entrada que conecta el cuerpo de la neurona con su vecindad; el soma es el núcleo donde se produce la reacción electroquímica a partir de las entradas; el axón es la etapa de salida, la cual se conecta con la vecindad.

Inicialmente se tiene una representación en software de la red neuronal por medio de un código C llamado Modelo Núcleo IO, en el cual se encuentran programadas las funciones necesarias para emular el comportamiento biológico de cada neurona que constituye la red. La complejidad del algoritmo representado en el código C hace que este requiera de un mayor tiempo de ejecución si se ejecuta una simulación en software por medio de un CPU, comparado con una implementación en hardware por medio de FPGA, como se observa en la figura 1.1. Dado el alto rendimiento que permite alcanzar la implementación en FPGA en cuanto a tiempo de ejecución cuando la red neuronal crece, se justifica la elección de este tipo de implementación como la deseada para la ejecución del proyecto.

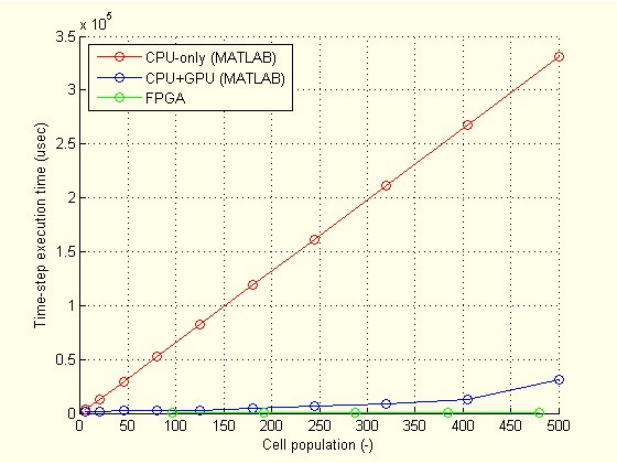


Figura 1.1: Gráfica comparación en tiempo de ejecución para diferentes plataformas. Imagen tomada de [1]

El desarrollo de este tipo de aplicaciones sumado al gran avance tecnológico es de gran importancia para el campo de la medicina, ya que solo en este caso particular, los neurocientíficos tendrán una herramienta útil que les permitirá ampliar el conocimiento e investigación del funcionamiento del cerebro, además de abrir camino hacia el desarrollo de aplicaciones que permitan mejorar la calidad de vida de las personas.

1.2 Definición del problema

La red neuronal implementada no poseía forma de visualización de resultados de una simulación y el único producto de esta era un archivo de texto que contenía los valores de tensión de Axón para cada neurona simulada por cada paso de simulación. Esto producía poca facilidad para observar el comportamiento tanto individual como del conjunto de neuronas simuladas. Además, la forma de representar los datos no era práctica para los neurocientíficos, ya que ellos requerían una representación gráfica de los resultados de las simulaciones que facilitara la realización de conclusiones a partir de los resultados. Además, no existía una plataforma que permitiera al neurocientífico colocar un estímulo de entrada para la red neuronal que se simularía, sino que la única forma de estimular la red era mediante valores por defecto. Por tanto, para que la red neuronal tenga un uso real y práctico, se debía hacer flexible el estímulo de acuerdo a la necesidad del neurocientífico.

Para implementar una plataforma de visualización de datos, se debió tener en cuenta que la red neuronal podía ser integrada por uno o varios dispositivos y que la comunicación entre estos dispositivos y la plataforma no existía. Era necesario implementar una interfaz de comunicación flexible para la red neuronal, que permitiera la comunicación de los dispositivos o FPGA con la plataforma de visualización por medio del puerto Ethernet. Además, se requería que esta interfaz de comunicación se ejecutara en el menor tiempo posible, de manera que su latencia no afectara en gran medida el tiempo de ejecución de la simulación general.

1.3 Síntesis del problema

La faltante de un método de visualización de datos para la simulación de una red neuronal implementada en múltiples FPGAs.

1.4 Enfoque de solución

Como solución al problema de visualización de datos se creó una página web. Esto permite a los neurocientíficos tener una herramienta de simulación de redes neuronales fácil de usar, accesible desde cualquier punto con conexión a internet y de configuración flexible al permitir variar parámetros como: tamaño de red y pasos de simulación a realizar. Además, la página web permite la visualización de la respuesta de cada neurona simulada para facilitar la obtención de conclusiones a partir de los resultados de la simulación. Para ello se crearon gráficas del comportamiento individual y grupal de las neuronas simuladas.

Como parte importante de la solución se implementó una interfaz de comunicación que permite comunicar el servidor web con cada FPGA conectada a la red de simulación. Cada FPGA cuenta con una red neuronal de tamaño variable y se comunica con el exterior por medio del puerto Ethernet. La interfaz es flexible en cuanto al número de dispositivos que desea utilizar el usuario para crear la simulación, en donde este puede llegar a utilizar hasta cuatro de ellos. Además la comunicación es de tipo dúplex.

Para implementar la interfaz de comunicación, se definió un diagrama de flujo de datos que se muestra en la figura 1.2, en el cual ordena primeramente una etapa de inicialización en donde se lee un archivo de texto con la inicialización que el usuario desee, luego se ejecutan los pasos de simulación necesarios para completar la simulación. En cada paso se deben enviar los valores de tensión de dendrita de todas las celdas que componen la red y la corriente de excitación para cada celda que se encuentra localmente en cada FPGA. Finalmente se guardan los resultados de tensión de axón de cada celda que compone la red en un archivo de texto con el fin de mostrarlos luego en interfaz de visualización.

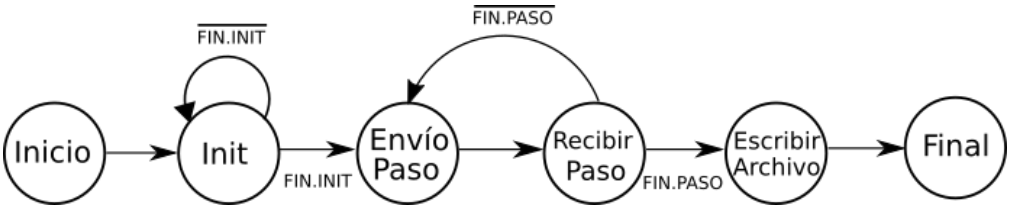


Figura 1.2: Diagrama de flujo para el protocolo de comunicación entre el servidor y los dispositivos periféricos. Imagen propia.

1.4.1 Requerimientos de la interfaz de comunicación

- Flexibilidad: el sistema debe permitir al usuario variar los parámetros que definen la simulación de acuerdo a sus necesidades.
- Rendimiento: la interfaz de comunicación debe ejecutarse en el menor tiempo posible para no afectar latencia general del sistema en gran medida, además de mostrar un comportamiento estable (sin errores o eventos aleatorios).
- Precisión: se desea que la interfaz de comunicación no interfiera en la precisión de los datos que son transmitidos.

1.5 Meta

Obtener una plataforma de simulación de redes neuronales flexible y accesible desde cualquier punto con acceso a internet, que permita a futuro la ejecución de modelos cerebrales biológicamente precisos que ayuden con la investigación y desarrollo de nuevas tecnologías en el área de la neurociencia.

1.6 Objetivos

Objetivo principal

Construir un entorno de simulación de redes neuronales sobre múltiples FPGAs utilizando un servidor a nivel local.

Objetivos específicos

- Desarrollar una estrategia de comunicación entre el servidor web y múltiples Zynq.
- Describir el software necesario para la comunicación del Zynq con el servidor utilizando Vivado SDK.
- Implementar el servidor y una página web amigable con el usuario que permita la simulación de redes neuronales biológicas.

1.7 Estructura del documento

Este documento realiza una comparación de interfaces de comunicación construidas a partir distintos ambientes de desarrollo. En el capítulo 2 se presenta el fundamento teórico necesario para el desarrollo del proyecto. En el capítulo 3 se explica el desarrollo de la interfaz de comunicación utilizando Vivado SDK como plataforma para crear una aplicación para ejecutar redes neuronales. El capítulo 4 presenta el desarrollo de una interfaz de comunicación para múltiples Zedboards utilizando sistema operativo y el estándar MPI. El capítulo 5 presenta la construcción del servidor Web que servirá como interfaz de visualización de datos. Finalmente, el capítulo 6 presenta las conclusiones y recomendaciones a las que se llegó luego de la ejecución de este proyecto.

Capítulo 2

Marco teórico

2.1 Plataforma embebida Zynq Zedboard

Zedboard es una plataforma de desarrollo de bajo costo basado en un Zynq-7000 all programmable SoC de Xilinx. Dados los diferentes recursos que contiene, esta plataforma permite el desarrollo de proyectos en diferentes áreas como: procesamiento de audio y vídeo, control automático, aceleradores de software y desarrollo de aplicaciones sobre sistemas operativos Linux.

La facilidad de desarrollo de diferentes proyectos la provee el núcleo Zynq-7000 SoC, formado por un procesador ARM de doble núcleo A9 con periféricos integrados (PS) y un bloque de lógica programable (PL) basado en la familia Artix-7 con 85 mil celdas de lógica programable. Además, esta plataforma contiene diversos puertos de comunicación como USB-JTag para programación, puerto Ethernet de 100M/1Gb, entrada para tarjeta SD y puerto USB-UART entre otros. Para mayor información sobre esta plataforma y sus características, se puede consultar su guía de usuario en [5].

2.2 Herramientas de Xilinx

Xilinx Design Suite HLx Edition es una plataforma de software que integra varias herramientas con el fin de permitir al usuario el desarrollo de hardware para distintos SoC. A continuación se realiza una breve explicación de algunas de sus herramientas utilizadas en este proyecto.

2.2.1 Vivado IDE

Vivado IDE es una herramienta que provee un entorno de programación en forma gráfica y también manejable vía TCL. Dentro de este entorno se encuentra el IP Integrator que permite la instanciación y conexión de *IP Cores* de forma gráfica. Estos *IP Cores* son en la mayoría reconfigurables a fin de adaptar su funcionalidad a los requerimientos del usuario. El uso del IP Integrator facilita la creación de proyectos para distintos SoC sin

la necesidad de escribir cada bloque desde cero usando un lenguaje HDL. Puede hallarse más información en [6].

2.2.2 Vivado SDK

Vivado SDK es un entorno de desarrollo para aplicaciones dirigidas a plataformas embebidas soportadas por Xilinx. Vivado SDK permite utilizar el hardware diseñado en el IDE. Para crear y correr aplicaciones sobre la plataforma embebida, Vivado SDK crea un *bsp*, el cual es una colección de bibliotecas y drivers que forman la capa más baja del software embebido y que permiten a la aplicación interactuar con el hardware.

Cada aplicación se crea bajo el directorio Proyecto de Aplicación, que contiene varios archivos como: los archivos C/C++ que describen la aplicación, el ejecutable `.elf` que será el archivo que corre sobre la plataforma, además de un *link* hacia el bsp, ya que sin este la aplicación no sería funcional. Las aplicaciones aquí creadas permiten al usuario evaluar su hardware de forma rápida sin necesidad de utilizar un sistema operativo, sino corriendo la misma en bare-metal. Mayor información acerca de este entorno se puede encontrar en [7].

2.3 Descripción del modelo ION

El modelo ION utilizado en este proyecto se basa sobre la parte del cerebro llamada olivar inferior. El modelo ION se representa por medio de un código programado en C, que separa cada neurona en tres partes fundamentales: soma, dendrita y axón. A su vez, este modelo crea una red neuronal por medio de la función *Gap Junctions*, que modela la interconexión de todas las neuronas en la red, y como se dice en [3], define el comportamiento y sincronización de las neuronas que componen la región IO. La figura 2.1 muestra la estructura del código C y cómo interactúan las funciones que lo componen.

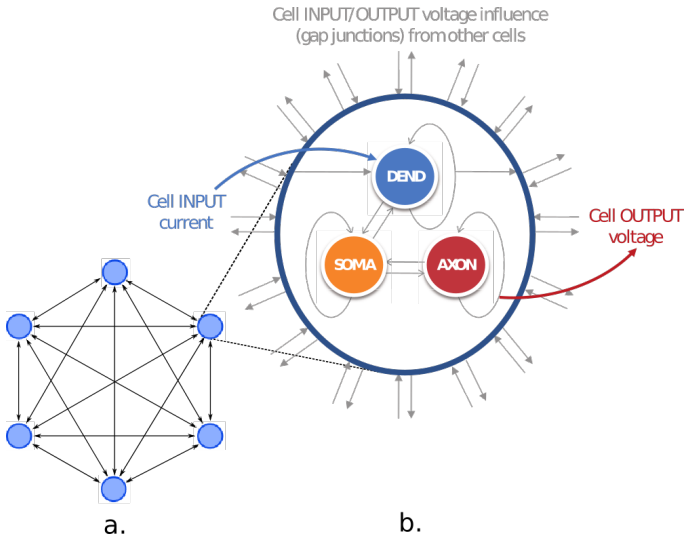


Figura 2.1: Figura de a) Esquema de interconexión de las celdas y b) Partes de la celda. Imagen tomada de [2]

A fin de mejorar el rendimiento de red neuronal cuando se realizan simulaciones de esta, el código C fue trasladado a hardware usando síntesis de alto nivel y llevado a Vivado como un bloque *IP core*, procedimiento que puede encontrase en [8]. Por tanto se tomará este bloque *IP core* como ya realizado y se prestará atención únicamente a sus puertos de entrada y salida.

Antes de ver en detalle los puertos del bloque *IP core* para el modelo ION, es preciso revisar el flujo del ejecución del mismo. Antes de cada simulación, la red neuronal debe inicializarse. Una vez se cargan los datos de inicialización, toda la red recibe un estímulo además de los valores de tensión de dendrita de las celdas para el estado previo. Estos valores son utilizados en la función *Gap Junctions*. Una vez terminado esto, se calcula el valor de salida de la red neuronal, compuesta por la tensión axón y dendrita, el primero como salida de la red neuronal y el segundo para ser utilizado en el siguiente paso de simulación.

2.3.1 Puertos de comunicación del bloque de simulación neuronal

Para el funcionamiento y comunicación del bloque de simulación neuronal es importante comprender los puertos de comunicación disponibles. La figura 2.2 presenta el diagrama del bloque de simulación neuronal junto a sus puertos tanto de entrada como salida.

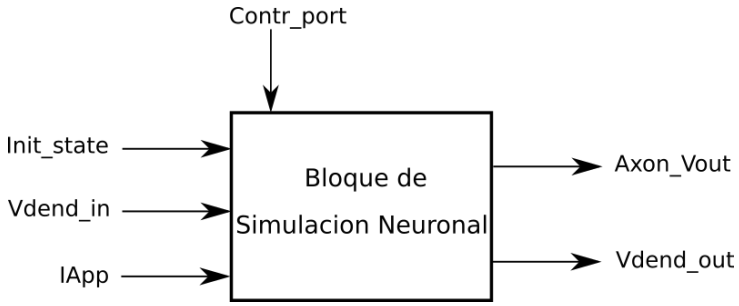


Figura 2.2: Diagrama de entradas y salidas para el bloque IP core del modelo ION. Imagen propia.

Para la figura 2.2, se presentan los puertos de comunicación los cuales se detallan de la manera:

- **Init_state:** puerto utilizado para leer la información referente a la inicialización de las celdas que componen la red neuronal. Al inicio de cada simulación, la totalidad de la red neuronal debe ser inicializada mediante una secuencia de parámetros que se detallarán más adelante.
- **Vdend_in:** se utiliza para leer los datos correspondientes a la tensión de dendrita de las celdas que componen la red neuronal. Estos datos corresponden al estado anterior de la red neuronal y cada celda deberá tener acceso a estos datos.
- **IApp:** puerto utilizado para leer el estímulo de la red neuronal en cada paso de simulación. Cada dato representa biológicamente una corriente de estímulo que puede cambiar de celda a celda, y será un dato nuevo cada paso de simulación.
- **Axon_Vout:** utilizado para exportar los datos de tensión de Axón correspondientes a la salida de cada celda en el actual paso de simulación. Estos datos se deben almacenar concatenados a los datos de cada paso de simulación realizado, ya que representan el comportamiento general de la red neuronal para el período simulado.
- **Vdend_out:** puerto por el cual se exportan los datos de tensión de dendrita correspondientes al actual paso de simulación y que serán utilizados como entrada para el siguiente paso. Estos datos deben ser almacenados temporalmente hasta el próximo paso de simulación.
- **Contr_port:** puerto utilizado para el control de ejecución del hardware. Con este puerto se puede controlar el inicio de ejecución de la red, verificar cuando la red ha terminado la ejecución de un paso de simulación o incluso hacer reset de la misma.

2.3.2 Parámetros de inicialización

Cada vez que se desea ejecutar una simulación, la totalidad de las celdas que componen la red neuronal deben ser inicializadas. Para ello se debe pasar a cada una de las celdas 19 diferentes parámetros que describen el comportamiento de cada celda durante la

simulación. Los parámetros de inicialización son valores flotantes y presentan un ordenamiento como el que sigue. Este orden es importante durante la inicialización a fin de no obtener errores en esta etapa. La tabla 2.1 presenta los parámetros de inicialización y su significado conocido.

Tabla 2.1: Parámetros de Inicialización de la ION

Parámetro	Significado
dend_V_dend	Tensión de dendrita
dend_Calcium_r	Calcio umbral alto
dend_Potassium_s	Potasio dependiente del calcio
dend_Hcurrent_q	Corriente H de dendrita
dend_Ca2Plus	Concentración de calcio
dend_I_CaH	Corriente de calcio umbral alto
soma_g_CaL	Calcio bajo umbral
soma_V_soma	Tensión de soma
soma_Sodium_m	Sodio activación
soma_Sodium_h	Sodio de inactivación
soma_Potassium_n	Potasio componente tardío
soma_Potassium_p	Potasio neocórnico
soma_Potassium_x_s	Potasio dependiente de tensión
soma_Calcium_k	Calcio de activación
soma_Calcium_l	Umbral bajo para calcio
axon_V_axon	Tensión de axón
axon_Sodium_m_a	Sodio de activación
axon_Sodium_h_a	Sodio de inactivación
axon_Potassium_x_a	Potasio transitorio

2.4 Estándar de comunicación Ethernet

Este es un estándar de comunicación de tipo *full-duplex* que permite la integración de varios equipos en una sola red usando conexión cableada. Según [9], este estándar apareció en los años 80 por la necesidad que existía en las empresas de conectar varias computadoras y comunicarlas entre ellas. Actualmente el Ethernet funciona bajo el estándar IEEE 802.3 y cuenta con velocidades de 10Mbps, 100Mbps, 1000Mbps y 10Gbps.

Para comunicar dos equipos por medio de Ethernet, existen varios supuestos que deben cumplirse: cada equipo debe poseer una dirección IP y dirección MAC únicas, además de estar ambas máquinas en una misma red. Una vez se establece una comunicación entre dos equipos, la información viaja en paquetes los cuales mantienen un formato como el mostrado en la figura 2.3, en donde cada paquete transportará un mínimo de 46 Bytes y no más de 1500 Bytes. Según explica [9], cada paquete se compone de siete bytes de `Preámbulo` seguido de un byte llamado `inicio de frama(SOF)`, los cuales siempre son constantes y permiten sincronización entre los extremos. Las direcciones `Destino`

y `Fuente` identifican ambos equipos en los extremos de la comunicación mientras que `Etiqueta 802.1` es un espacio opcional utilizado cuando se consideran crear subredes o LANs virtuales. El espacio `extensión` indica el tamaño de la carga útil, mientras que los datos enviados o carga útil se encuentran en el espacio `Datos`. Finalmente `FCS` permite mayor sincronización entre ambos extremos de la comunicación.

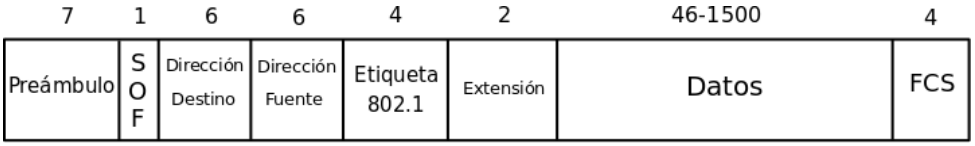


Figura 2.3: Formato de un paquete Ethernet(especificados en número de Bytes). Imagen propia.

2.5 Servidor HTTP Apache

El Apache Server Project es una herramienta de código abierto creada para proveer servicios HTTP por medio de un servidor fácil, eficiente y seguro sobre sistemas operativos como Linux o Windows. En el esquema de funcionamiento del servidor, el equipo anfitrión recibe cada petición de un cliente por el puerto 80 típicamente, decodifica la información, lee los archivos del sitio web encontrados dentro del directorio al que Apache apunta y devuelve el código de este sitio para que el *web browser* del cliente lo interprete y construya el sitio web del lado del usuario. Esta herramienta y su documentación se pueden encontrar en su sitio *web* en [10].

2.5.1 PHP 5

Este es un lenguaje de programación diseñado para el desarrollo de sitios *web* dinámicos. Este tipo de lenguaje ayuda al programador a realizar tareas del lado del servidor, mismas que son imposibles de realizar con otros lenguajes como Java. Dentro de las tareas que se pueden realizar se encuentra el manejo de archivos, creación de gráficas e incluso ser usada como interfaz de línea de comandos para realizar llamadas a sistema sobre la máquina anfitrión. Esta herramienta y su documentación se pueden encontrar en su sitio *web* en [11].

2.5.2 CSS

CSS es un lenguaje usado para describir el estilo de los objetos que se presentan en un sitio *web*. Este lenguaje permite además mayor ordenamiento del código al separar el contenido del sitio *web* y su presentación (tipografías, colores, estilos), ya que este último se puede colocar en un archivo separado con extensión `.css`.

2.6 Biblioteca Matplotlib

Matplotlib [12] es una biblioteca disponible para Python que permite crear gráficos de forma sencilla e interactiva sobre diferentes plataformas. Esta biblioteca provee un ambiente al programador similar al obtenido en Matlab pero de forma gratuita. Esta biblioteca puede ser utilizado en scripts de Python, sobre el shell estándar o sobre IPython, embebido en aplicaciones o sobre aplicaciones en *web servers*. Además permite agregar funcionalidades en 3D usando la clase `mplot3d`, además de otros juegos de herramientas que amplían la utilidad de la biblioteca dependiendo del uso que se le quiera dar.

2.7 MPI

MPI es una herramienta que permite el procesamiento en paralelo de tareas entre varias máquinas por medio de procesos concurrentes. Según [13], MPI tiene como objetivo cumplir una serie de objetivos como lo son:

- Crear una plataforma estándar de comunicación práctica, flexible y eficiente para el procesamiento concurrente.
- Permitir memoria compartida entre procesos.
- Permite la implementación de aplicaciones en ambientes heterogéneos.
- Permite la ejecución de diferentes tareas en paralelo dentro de una aplicación.

MPI funciona basado en grupos. Un grupo está formado por un conjunto de procesos los cuales pueden comunicarse entre sí por medio de un comunicador común, en donde dentro de este comunicador cada proceso se identifica por medio de un rango. Basados en este principio, es posible ejecutar diferentes tareas usando diferentes grupos para ejecutarlas. Además, la aplicación puede crear grupos en donde existan diferentes procesos o un mismo proceso perteneciente a diferentes grupos. Esta característica se puede observar en la figura 2.4.

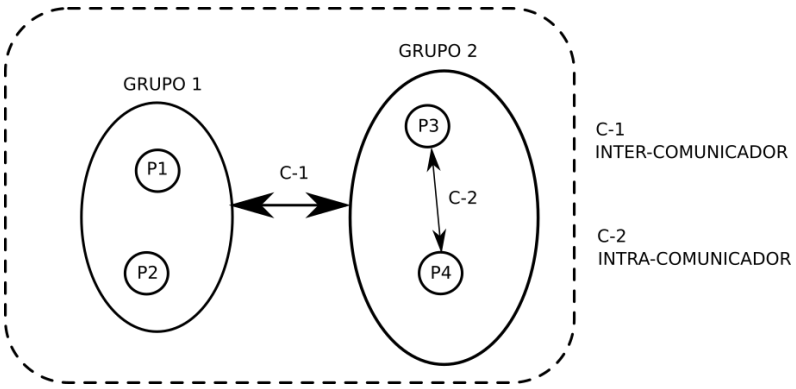


Figura 2.4: Formación de grupos de procesos, en donde se crean dos grupos y se ejemplifica el uso de inter e intra comunicadores. Imagen propia.

Los grupos pueden utilizar funciones básicas de comunicación entre un proceso y otro, o pueden utilizar funciones colectivas, en donde todos los procesos dentro de un grupo

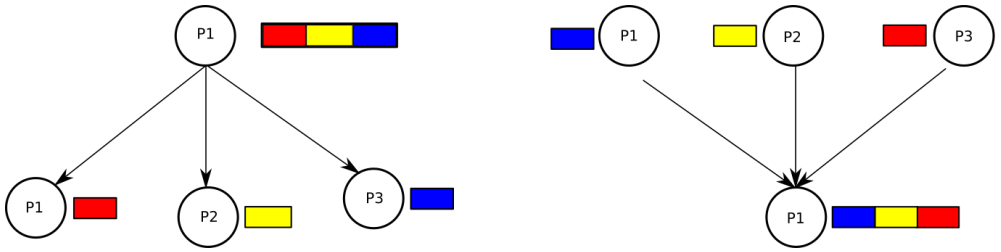
se comunican entre sí. Además como se observa en la figura 2.4, dos grupos pueden comunicarse entre sí por medio de intercomunicadores. Así, según [13], un comunicador para procesos de un mismo grupo es llamado `intra-comunicador` y uno que habilita la comunicación entre dos grupos es llamado `inter-comunicador`.

2.7.1 Principales rutinas de MPI

MPI utiliza un conjunto de instrucciones para pasar los mensajes entre cada proceso que se involucra. Las rutinas básicas de MPI son llamadas `MPI_Send` y `MPI_Recv`, las cuales se utilizan para transferir un dato entre dos procesos. Además, la rutina `MPI_Barrier` se utiliza para colocar puntos de control en la aplicación, en donde cada proceso que llegue al punto de control detiene su ejecución hasta que todos los demás procesos lleguen a tal punto, una vez sucede esto todos los procesos continúan su ejecución. Además, la rutina `MPI_WTime` se utiliza para calcular el tiempo de ejecución entre dos puntos de la aplicación. Estas rutinas se pueden estudiar más a fondo en [14].

MPI_Scatter y MPI_Gather

Estas rutinas son de tipo colectivas, ya que cada proceso dentro de un grupo se ve involucrado en la ejecución de estas rutinas. Según [14], la función `MPI_Scatter`, esparce (de ahí su nombre) un arreglo de datos transmitiéndolo desde un proceso raíz o principal hacia todos los demás procesos que conforman el grupo donde se ejecuta esta rutina. Cabe destacar que el arreglo se reparte en porciones de igual tamaño para todos los procesos incluyendo el proceso raíz como se muestra en la figura 2.5a.



(a) Rutina MPI_Scatter (b) Rutina MPI_Gather

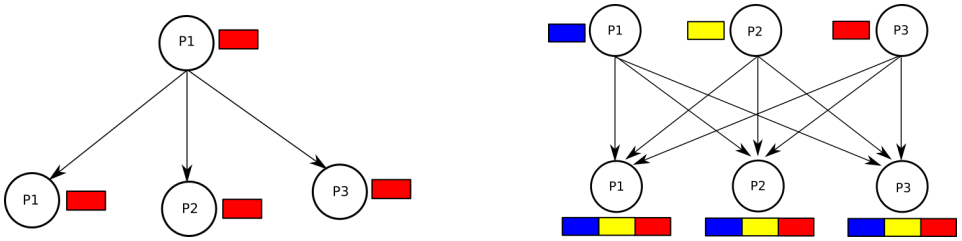
Figura 2.5: Rutinas colectivas: Scatter (esparcir) y Gather (recoger). Imagen propia.

La rutina `MPI_Gather`, realiza la función inversa al `MPI_Scatter`, pues cada proceso transmite una porción de datos hacia el nodo raíz o principal, y cada porción de datos se organiza en el nodo raíz que los recoge (de ahí su nombre) de acuerdo al rango del proceso que transmite. Esta rutina se muestra en la figura 2.5b.

MPI_Broadcast y MPI_Allgather

La rutina `MPI_Broadcast` según [14] permite emitir un paquete de datos desde un proceso que se seleccione y hacia todos los demás procesos que componen el grupo. Este compor-

tamiento se muestra en la figura 2.6a. Mientras que la rutina `MPI_Allgather` es de tipo colectiva y funciona como combinación entre las rutinas `MPI_Broadcast` y `MPI_Gather`, en donde cada proceso dentro de un grupo transmite una porción de datos hacia todos los demás procesos del grupo, de manera que al finalizar esta rutina, todos los procesos poseen los mismos datos. El funcionamiento de esta rutina se muestra en la figura 2.6b.



(a) Rutina de MPI_Broadcast. (b) Rutina de MPI_Allgather

Figura 2.6: Rutinas colectivas de MPI_Broadcast (emisión) y MPI_Allgather (recoger todos). Imagen propia.

2.8 Distribución Linaro de Ubuntu

Ubuntu Linaro es una distribución de Linux construida para dispositivos de la familia ARM, basada en la distribución de Ubuntu. Ofrece un sistema operativo ligero caracterizado por un bajo consumo de recursos y con herramientas integradas como por ejemplo compilador para lenguaje C. Un sistema operativo según lo presentado en [9], consta de varios archivos: el archivo llamado *Bootloader* encargado de encontrar la imagen del *kernel*, copiarla a la memoria y transferir el control del sistema a esta imagen. Este archivo contiene además la descripción del hardware que se va a programar en la FPGA. El archivo que contiene el *kernel* del sistema operativo es llamado *kernel image*(núcleo de imagen), mientras que el archivo que contiene la información acerca de dispositivos y periféricos que requiera utilizar el sistema operativo es llamado `árbol de dispositivos`. Finalmente el `Sistema de archivos` se define en [9] como quien contiene la distribución de archivos que componen el sistema operativo, como bibliotecas, scripts y aplicaciones.

En un sistema embebido, el `Sistema de archivos` contiene directorios como los siguientes:

- /bin : contiene los programas que son accesibles por el usuario.
- /dev : contiene los archivos para acceder a los dispositivos y periféricos.
- /etc : contiene lo referente a configuración general del sistema.
- /sbin : contiene los programas que son ejecutados por el sistema.
- /home : contiene todo tipo de archivos generados por el usuario.

Para construir estos archivos se utiliza compilación cruzada, es decir, se utiliza una computadora de uso general para compilar la imagen del sistema operativo que se desea instalar, en este caso, sobre una Zedboard. Ello pues la computadora de uso general posee mayores recursos para realizar esta tarea compleja comparada con una compilación sobre la Zedboard. Para realizar esta compilación cruzada se requiere de una serie de herramien-

tas vinculadas entre sí (serie comúnmente llamada toolchain, o cadena de herramientas), que de acuerdo a [9], es la encargada de proporcionar el ensamblador, el compilador y el enlazador, junto con una serie de otras utilidades necesarias para desarrollar un sistema operativo Linux. El *toolchain* utilizado en este proyecto fue *CodeSourcery* la cual se puede adquirir en [15] junto a información acerca de la misma.

2.9 Procesos e hilos

Una aplicación en un sistema operativo como Linux se compone inicialmente de un proceso que se encarga de proveer los recursos necesarios para ejecutar esta aplicación. Un proceso no comparte memoria con otros procesos y pueden existir varios procesos dentro de una misma aplicación, ejecutando diferentes tareas. Los hilos según [9] son múltiples instancias de ejecución dentro de un proceso que comparten el mismo espacio de memoria y descriptores para el acceso a los recursos del sistema. En Linux, los hilos se crean dentro de un proceso a través de la llamada al sistema, aunque en la mayoría de los casos los programadores usan la biblioteca `pthread` para crear y gestionar los hilos. Estos fomentan el paralelismo además de los recursos compartidos dentro de la ejecución de una aplicación.

Según [9], cada proceso puede crear varios hilos de ejecución usando la llamada sistema `taskOpen()`, requiriendo esta función parámetros como:

- Nombre.
- Prioridad: valor entre 0 y 255, donde 0 es la máxima prioridad.
- Base y tamaño de la pila.
- Puntero a proceso al cual es asignado.

Capítulo 3

Desarrollo de una interfaz de comunicación para simulaciones utilizando múltiples Zynq

Esta sección se dedica al desarrollo de una interfaz de comunicación que permita la interacción entre un nodo principal y múltiples nodos periféricos en una red local usando Ethernet. El nodo principal es una computadora de uso general mientras que cada nodo periférico se representa por el núcleo Zynq de una Zedboard. Para esta sección, cada Zedboard no cuenta con un sistema operativo, sino solo ejecuta una aplicación creada con la herramienta Vivado SDK.

3.1 Análisis de requerimientos

Para crear esta interfaz se partió del hecho que la herramienta Vivado SDK provee una plantilla para el desarrollo de una aplicación C que permite a la Zedboard establecer comunicación usando protocolo TCP con otro dispositivo al configurar la tarjeta como servidor (o sea, que espera por enlaces desde un cliente). Dada la característica anterior, la computadora se encarga de manejar el flujo de la simulación, conectándose como cliente a cada Zedboard cuando se requiere transmitir datos o ejecutar un paso de simulación. El esquema general de conexión se muestra en la figura 3.1.

Cada simulación puede variar su tamaño y duración basada en tres variables: `Tamaño de red total` representa la cantidad total de neuronas que se quiere crear para la simulación, `número de Zedboards` que representa la cantidad de Zedboards sobre los cuales se reparte la red neuronal y `tamaño de red local` que representa la cantidad de neuronas por Zedboard. Este último parámetro es producto de la división de los dos primeros, por lo que un requerimiento del diseño se encuentra en que la cantidad de neuronas total debe ser divisible entre las Zedboards que se desean utilizar.

Además, un requerimiento que se tomó en cuenta para esta sección fue el orden de transmisión de los datos a razón que todos los dispositivos involucrados en la simulación reciban

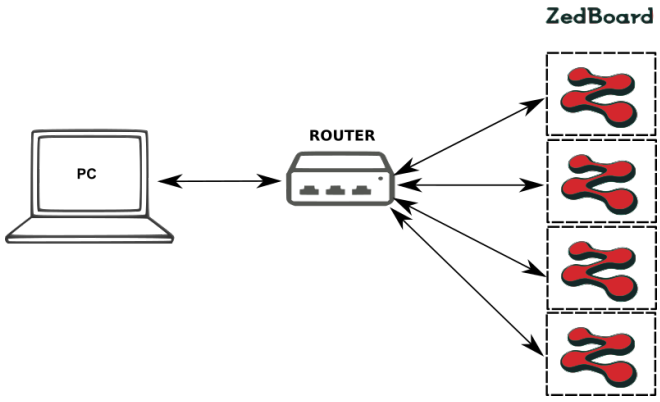


Figura 3.1: Diagrama de conexión para una configuración cliente servidor, donde la computadora funciona como cliente y cada Zedboard, como servidor. Imagen propia.

los datos necesarios para ejecutar una simulación. Este orden es abordado en las siguientes secciones de acuerdo a la etapa de simulación en la que se realice la transmisión de datos.

Para los valores de inicialización y estímulo de la red neuronal, se definió que estos se reciben del usuario por medio de un archivo de texto, el cual contiene primeros los valores de inicialización suficientes para cada neurona a simular, seguido del estímulo para cada paso de simulación.

3.2 Detalle de Solución desde el nodo principal

El nodo principal cuenta con un script en lenguaje Python que se ejecutará cada vez que se desee realizar una simulación de la red neuronal. Este script funciona bajo el diagrama de estados de la figura 3.2.

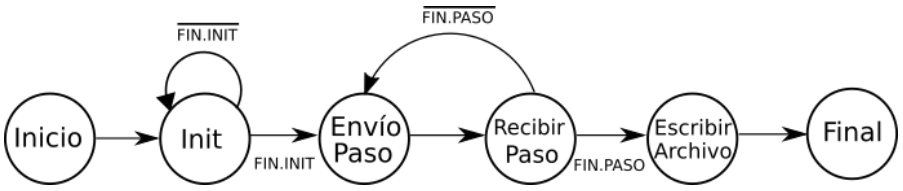


Figura 3.2: Diagrama de estados para el script Python, que se encarga de ejecutar la simulación de redes neuronales y recolectar sus resultados en la computadora. Imagen propia.

3.2.1 Inicio

En la etapa llamada `Inicio`, el script se ejecuta y se inicializan las variables globales y arreglos necesarios para ejecutar la simulación. En cuanto a las variables globales, estas toman su valor de los argumentos que son pasados al script cuando se ejecuta en el nodo principal. La tabla 3.1 muestra las variables y su significado dentro del entorno de la

aplicación, además de una descripción de los principales arreglos utilizados.

Tabla 3.1: Variables que definen las características de la simulación y arreglos que contienen los datos involucrados en la ejecución de la simulación para el script de Python.

	Variables	Descripción
Variables	`N_size`	Tamaño total de la red neuronal que se desea simular.
	`STEP`	Cantidad total de pasos de simulación que se desean realizar .
	`FPGA`	Número de dispositivos (Zedboard) que se desea utilizar en la simulación.
Arreglos	`cell_array`	Contiene el total de neuronas a simular.
	`cellout`	Arreglo que contendrá los valores de tensión de axón para la salida.
	`vdend`	Arreglo que contendrá los valores de tensión de dendrita necesarios en cada paso de simulación.
	`fpga_devices`	Contiene los datos de cada Zedboard que se utilice en la simulación.

En el arreglo `fpga_devices`, la información referente a cada Zedboard se guarda utilizando una clase que contiene parámetros como dirección IP, puerto de comunicación, tamaño de red local y una etiqueta como identificador único para cada Zedboard.

El arreglo `cell_array` contiene la información referente a cada neurona, en donde estas se crean a partir de una clase que contiene los parámetros de inicialización en el orden definido en la tabla 2.1 y para los valores de estímulo, se utiliza un método para inicializar un arreglo local en cada neurona que contendrá estos valores de acuerdo al número de pasos deseado. En la figura 3.3 se muestra la estructura de cada neurona en el arreglo `cell_array`.

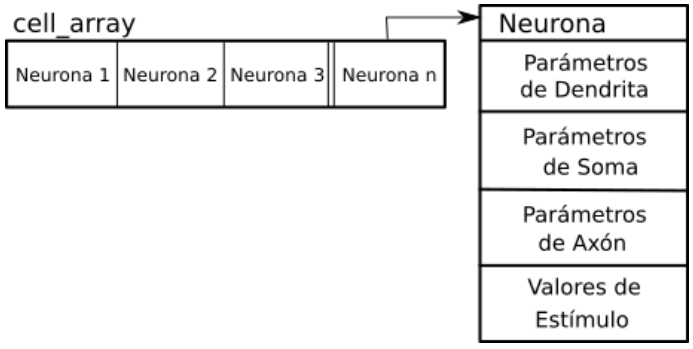


Figura 3.3: Estructura de datos para cada neurona simulada. Imagen propia.

Los valores de los parámetros de inicialización para cada neurona se leen desde el archivo de inicialización que el usuario carga al servidor *web*. Este proceso de lectura carga los datos desde el archivo hacia cada neurona que se encuentra en el arreglo `cell_array`, para luego utilizar estos datos durante la simulación sin la necesidad de acceder repetidamente al archivo de texto.

3.2.2 Inicialización de hardware

En esta etapa se da la inicialización de las neuronas que se encuentran en cada Zedboard con los datos que fueron cargados en el arreglo `cell_array`, además de enviar los valores de tamaño de la red total y el tamaño de la red local. Los mensajes que se envían a cada Zedboard mantienen un formato como el mostrado en la figura 3.4.

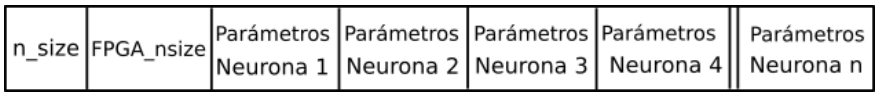


Figura 3.4: Estructura del mensaje enviado a cada Zedboard como inicialización. Imagen propia.

Una vez construido el mensaje de inicialización, este se envía a cada Zedboard en un enlace que consta de tres etapas. Primeramente, se establece un enlace de comunicación entre el nodo principal y cada Zedboard usando un *socket*. Seguidamente, el nodo principal transmite los datos de inicialización a cada Zedboard. En la etapa final, cada Zedboard transmite hacia el nodo principal una bandera que indica que se han recibido correctamente los datos de inicialización. Para realizar esta tarea en el menor tiempo posible y mejorar el rendimiento de la simulación se implementaron hilos de ejecución para realizar cada enlace en paralelo.

Para implementar los hilos se utilizó la biblioteca `Thread` de Python, que crea una rutina por cada Zedboard en la que se toma el mensaje que va a ser enviado y la dirección IP relacionada a cada dispositivo. Los *sockets* corren simultáneamente como se observa en la figura 3.5.

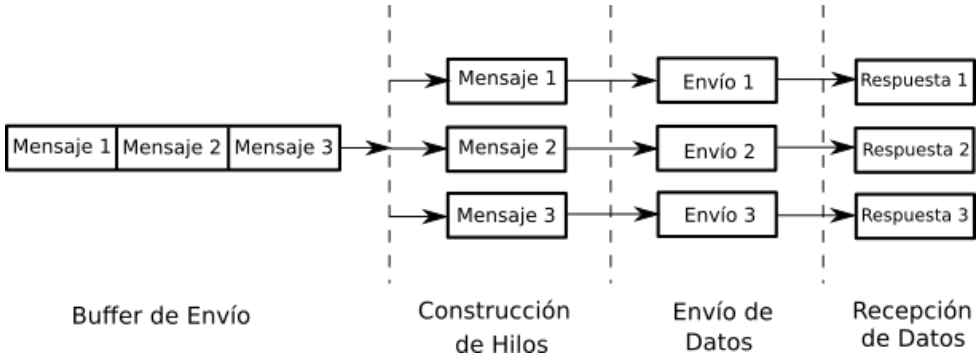


Figura 3.5: Etapas de la inicialización por *sockets*. Imagen propia.

3.2.3 Generación de pasos

La generación de pasos se da en dos etapas por cada paso de simulación: envío del estímulo a cada neurona y recepción de la salida de estas a partir del estímulo. Para generar cada paso se utilizó el mismo esquema de comunicación que en la inicialización del hardware, variando el mensaje que se quiere transmitir y recibir. En este caso el mensaje que se envía a cada Zedboard es mostrado en la figura 3.6.

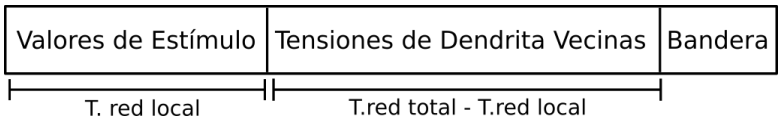


Figura 3.6: Estructura del mensaje enviado a cada Zedboard para un paso de simulación. Imagen propia.

Además, como se observa en la figura anterior, al final de cada cadena de mensaje se adjunta una bandera para informar a cada Zedboard si los datos que está recibiendo corresponden al último paso de simulación que se desea ejecutar. Las etapas de la comunicación para cada paso de simulación se muestran en la figura 3.7, y a diferencia de la inicialización, al final de cada conexión se toma los datos de respuesta de cada Zedboard que contienen las tensiones de axón y dendrita, y se guardan en los arreglos `cellout` y `vdend` respectivamente.

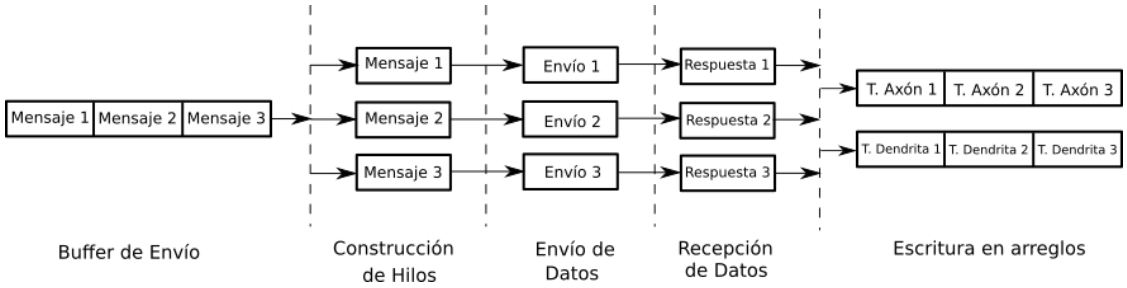


Figura 3.7: Etapas de la comunicación Servidor-Zedboard para cada paso de simulación. Imagen propia.

3.2.4 Escritura de archivo

Una vez se han realizado todos los pasos de simulación, los valores cargados en el arreglo *cellout* son guardados en un archivo de texto para que estos puedan ser utilizados posteriormente y visualizar la respuesta de la red neuronal a un estímulo. Para mayor facilidad al momento de leer este archivo de texto, los valores se ordenan por columnas en donde cada columna representa la salida de una neurona.

3.3 Detalle de Solución desde el SoC Zynq

Como parte de la interfaz de comunicación se debió crear una aplicación *bare-metal* utilizando Vivado SDK que permita la ejecución de redes neuronales. Esta aplicación consta de dos partes: la comunicación del núcleo Zynq con el nodo principal, por donde se transmiten los datos necesarios para la simulación, y la ejecución de la red neuronal alojada en la FPGA a partir de los datos recibidos. Esta sección solo abarca la parte de comunicación ya que la ejecución de la red neuronal es parte de un proyecto ejecutado en [8]. La arquitectura utilizada para ejecutar redes neuronales sobre la Zedboard se muestra en la figura 3.8, en la cual la red neuronal se sitúa sobre la FPGA y se comunica con el PS por

dos vías: los datos de inicialización y control se transmiten directamente por medio del puerto `AXI-Lite` mientras que los datos de estímulo se trasmiten desde la memoria hacia la red neuronal utilizando interfaz DMA por protocolo `AXI4`.

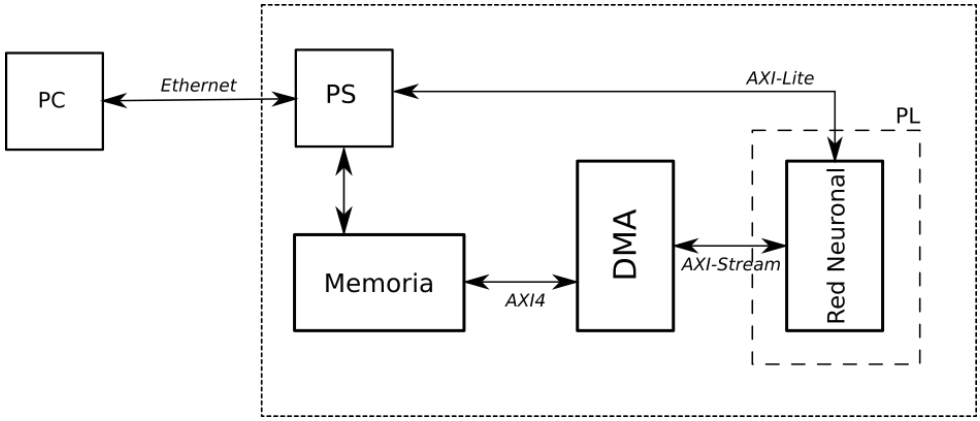


Figura 3.8: Diagrama de bloques de la arquitectura utilizada para estimular la red neuronal. Los datos de estímulo son transmitidos a la red neuronal usando un controlador DMA, mientras que los datos de inicialización, se escriben directamente hacia la red por protocolo AXI-Lite. Imagen propia.

3.3.1 IP y MAC

Un paso importante en el desarrollo de la aplicación es la definición de la dirección IP y la dirección MAC para cada tarjeta. Estas direcciones deben ser distintas en cada dispositivo que se conecta a la red cableada a fin de que el servidor web pueda comunicarse con cada Zedboard. Para ello se utiliza el parámetro `Board_Enum`, el cual se varía para cada Zedboard cambiando así la última cifra de las direcciones IP y MAC mediante las variables `IP4_ADDR` y `mac_ethernet_address` presentes en el código de la aplicación.

3.3.2 Flujo de programa

El flujo que sigue la aplicación durante una simulación es mostrado en la figura 3.9 es complementario al que corre sobre la computadora. Cabe destacar que a diferencia del anterior, este se construyó para que se ejecute indefinidamente luego de que la Zynq es programada desde el Vivado SDK.

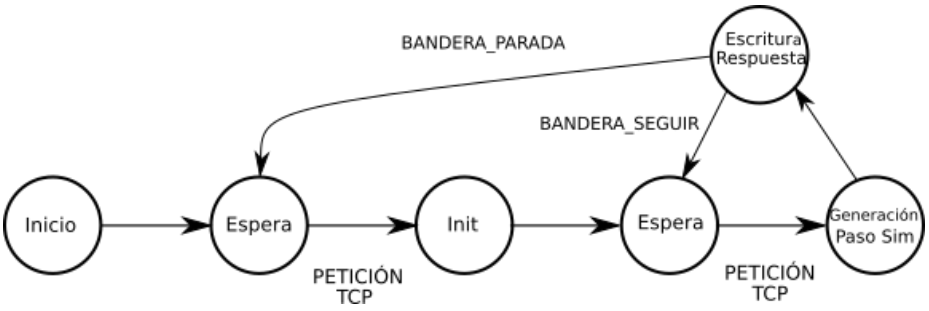


Figura 3.9: Diagrama de estados para la aplicación sobre la Zynq, la cual se encarga de establecer comunicación con el nodo principal y ejecutar la red neuronal con los datos recibidos. Imagen propia.

Inicio

Esta etapa se encarga de crear el servidor TCP fijando una dirección IP y MAC con las cuales la Zedboard se identificará dentro de la red a la cual se encuentra conectada, y en donde cada dispositivo tendrá estas direcciones distintas y únicas variando el valor de `Board_Enum`. Esta variable y otras, además de los arreglos utilizados durante la simulación son mostrados en la tabla 3.2.

Tabla 3.2: Variables y arreglos en la aplicación que ejecuta simulaciones usando la red neuronal.

Tipo	Variables	Descripción
Variable	`ion_n_size`	Tamaño de red total.
	`mux_factor`	Tamaño de la red local.
	`Board_Enum`	Valor identificador de la Zynq en la red a la que se encuentra conectada.
Arreglos	`initState`	Arreglo para los parámetros de inicialización.
	`Iapp`	Arreglo para los datos de estímulo de la red local.
	`Cellout`	Arreglo para la salida de la red local.

Inicialización

Una vez configurada la comunicación en la Zynq, la aplicación espera hasta que exista un primer enlace por parte del nodo principal. Una vez este se establece, la aplicación entra en una etapa de inicialización en donde se reciben los datos de tamaño de red a simular además de los parámetros de inicialización para la red local en el orden mostrado en 3.10.

Una vez recibidos los parámetros de cada neurona, estos son escritos hacia el hardware alojado en el PL usando protocolo `AXI Lite`. Esta etapa se finaliza respondiendo hacia el servidor una bandera de confirmación a fin de informar al servidor que los datos fueron recibidos correctamente.

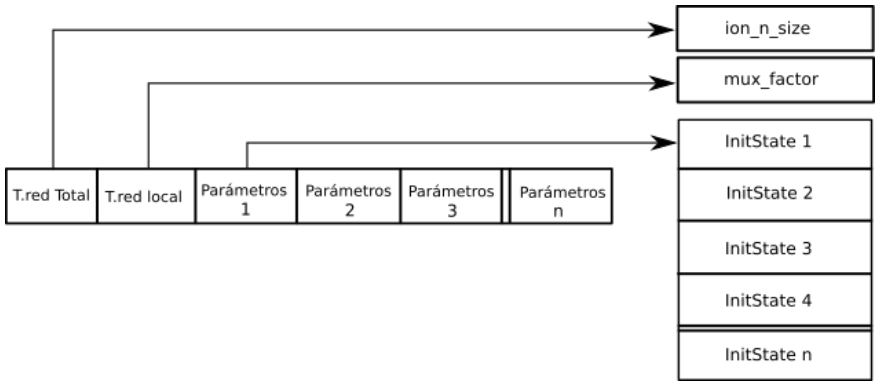


Figura 3.10: Carga de datos de Inicialización de la Zynq. Imagen propia.

Generación de pasos de simulación

Una vez realizada la etapa de inicialización, la aplicación espera un nuevo enlace TCP, con el cual se reciben en una primera fase los datos de estímulo para la red neuronal que contiene la FPGA, los cuales presentan el contenido y ordenamiento mostrado en la figura 3.11 para la recepción. Estos datos son guardados en el arreglo nombrado como `Iapp` visto en la tabla 3.2. Estos datos se escriben hacia la red neuronal usando protocolo `AXI4 Stream` para la comunicación entre el PS-PL y viceversa. La bandera al final del mensaje indica si el paso que se va a ejecutar es el último de la simulación.

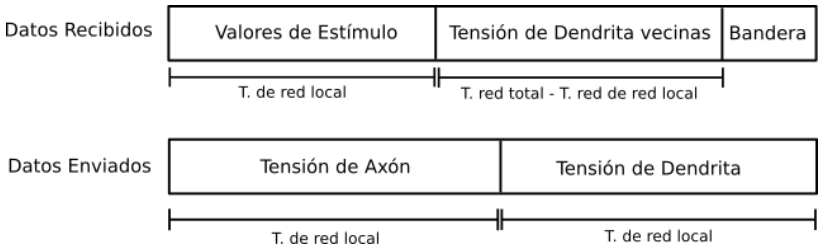


Figura 3.11: Estructura de datos enviados y recibidos en la aplicación. Imagen propia.

En la segunda fase de generación de pasos de simulación, el arreglo nombrado `Cellout` que contiene la salida de la red neuronal correspondiente al paso de simulación ejecutado se escribe hacia el servidor usando la función `tcp_write`. Esta función recibe un arreglo con la estructura mostrada en la figura 3.11 para el envío y hace la escritura hacia el cliente. Por último, se revisa la bandera recibida con la cual se determina si la simulación continua y el flujo del programa se devuelve a la primera fase de generación de pasos de simulación o salta a la etapa de espera para una nueva simulación.

3.4 Evaluación de resultados

La evaluación de resultados para la implementación de la interfaz de comunicación usando `sockets` se realizó en dos etapas: medición de latencia de cada componente por paso de simulación y verificación del porcentaje de error de las transacciones entre la computadora

y múltiples Zedboards. Cabe destacar que para esta sección, se utilizó una red neuronal implementada en [8] a fin de obtener resultados basados en un ambiente de simulación completo.

3.4.1 Latencia de la comunicación

La medición de latencia de comunicación se realizó utilizando un temporizador en Python y el IP core de Vivado `AXI Timer`, en donde se varía tanto la cantidad de Zedboards conectadas como el tamaño de la red utilizada. Además, la cantidad de pasos de simulación realizados fue un total de 500, ya que se considera suficientes para calcular el promedio de latencia por paso de simulación. Los resultados de latencia para diferentes configuraciones se muestran en la tabla A.1 en los apéndices.

En esta sección, no se tomaron en cuenta los resultados para una Zedboard, ya que al tratarse de una implementación para múltiples FPGA, no procede evaluar este caso. Durante la etapa de resultados, se observó que la ejecución de la aplicación creada en Vivado SDK algunas veces se detenía sin mostrar información alguna acerca de errores o excepciones ocurridas durante la ejecución. Este comportamiento aleatorio fue reportado al soporte de la empresa Xilinx sin obtener respuesta hasta la fecha de redacción de este informe.

La figura 3.12 muestra la latencia promedio del entorno de simulación en cada paso de simulación. Los datos mostrados fueron tomados de la tabla A.1.

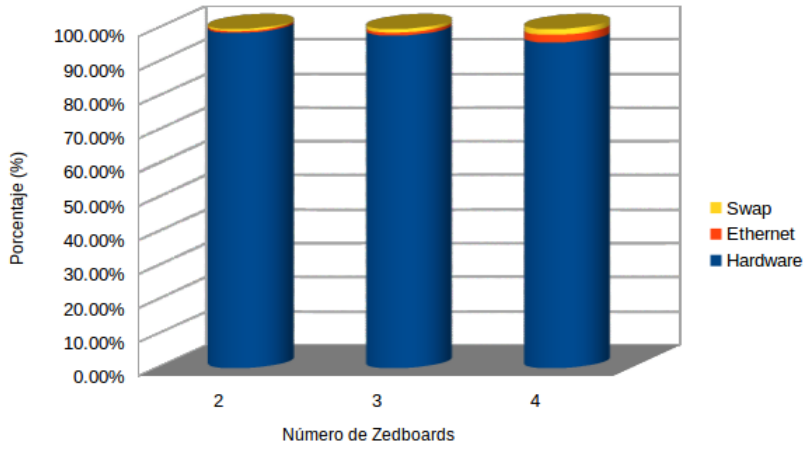


Figura 3.12: Latencia general para la implementación usando sockets. La figura muestra los resultados al variar la cantidad de Zedboards utilizadas para una red de tamaño 4000 neuronas. La latencia asociada a *swap* se relaciona al tiempo necesario para el ordenamiento de los datos que van a ser enviados y recibidos. Imagen propia.

La figura 3.12 permite determinar que el cuello de botella para esta implementación de redes neuronales lo representa la ejecución del hardware para la red. Esta implementación puede demorar hasta un 98% del tiempo total de ejecución cuando se analiza el peor de los casos (dos Zedboards y 4000 neuronas). Mientras que para las otras compo-

nentes, comunicación y *swap*(intercambios), el ritmo de crecimiento obtenido no ofrece muestras de tendencia a ocupar el puesto como cuello de botella en la ejecución para esta implementación.

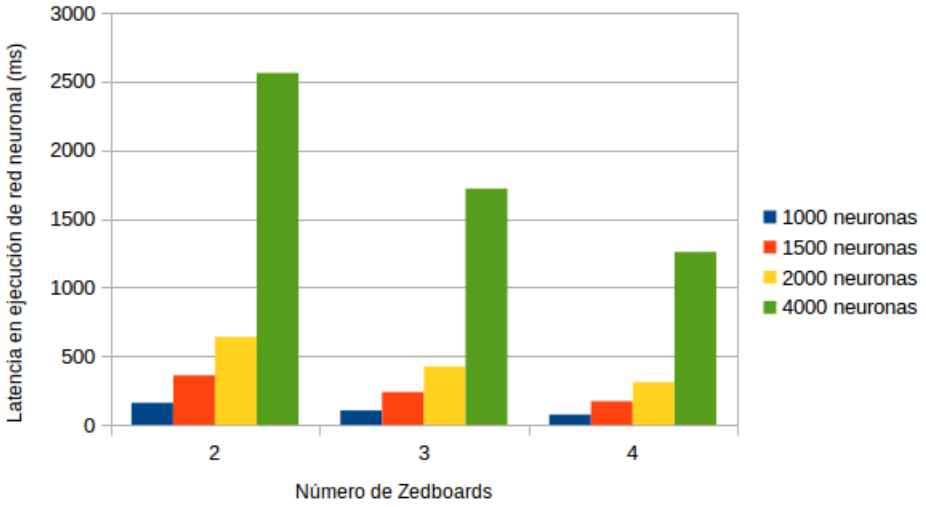


Figura 3.13: Latencia registrada para la ejecución del hardware de la red neuronal en una implementación con múltiples Zedboards y tamaños de red de 1000, 1500, 2000 y 4000 neuronas. Imagen propia.

El comportamiento del hardware por separado es mostrado en la figura 3.13, que permite analizar el comportamiento del tiempo de ejecución para el hardware ante los tamaños de red más relevantes. Si bien la respuesta del hardware presenta un comportamiento cuadrático al incremento de neuronas simuladas, a nivel de escalabilidad, los resultados mostrados en la figura 3.13 muestran una disminución clara en el tiempo de ejecución del hardware ante un mismo tamaño de red y aumentando el número de nodos que ejecutan la simulación. Una discusión más a profundo del comportamiento del hardware para la red neuronal se discute en [8].

La tendencia lineal para la comunicación también mostrada en la figura 3.14 es esperada para un tipo de comunicación como la implementada, ya que al aumentar el tamaño de la red neuronal, el incremento en la cantidad de datos a transmitir es proporcional a este primer aumento.

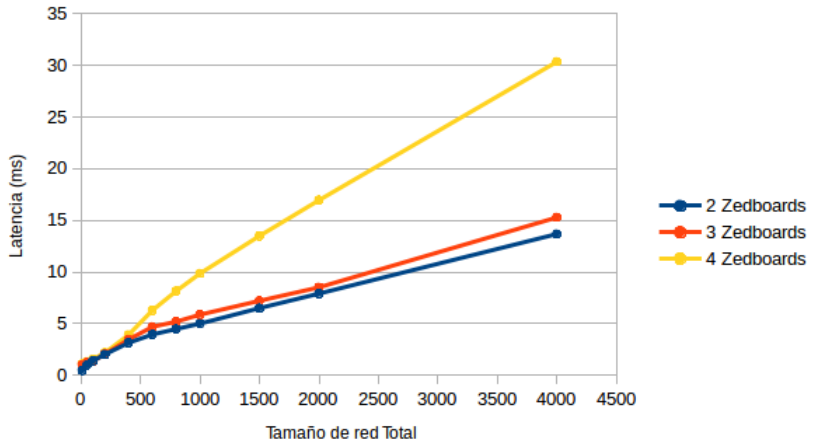


Figura 3.14: Latencia relacionada a la comunicación usando `sockets` en donde se varía el número total de Zedboards utilizadas para diferentes tamaños de red neuronal . Imagen propia.

En la figura 3.14 debe anotarse que cuando se utilizan cuatro Zedboards, el crecimiento en latencia es mayor al mostrado con implementaciones de dos y tres Zedboards, esto a partir de las 400 neuronas simuladas, ya que antes de este valor, la comunicación alcanzaba la velocidad máxima, el limite del cual ya no es posible disminuir dadas las características propias del protocolo utilizado, y aunque existe diferencia entre los datos que es proporcional a la variación del número de Zedboards, la baja cantidad de datos que se transmiten hace que la latencia varíe poco entre implementaciones.

Para el despegue en la tendencia lineal cuando se utiliza cuatro Zedboards observado en la figura 3.14, esta claramente no coincide con el crecimiento mostrado en las dos restantes implementaciones. Para esta inconsistencia se planteó como primera potencial causa, la hipótesis de que este comportamiento se debiera al uso de múltiples hilos para controlar la comunicación, y que esto repercutía en la arquitectura propia de la computadora que se utilizó como nodo principal. Sin embargo, se realizó una segunda medición de latencia utilizando una computadora con mayores recursos como nodo principal, obteniendo resultados muy similares a los mostrados en la tabla A.1, por lo que se descartó al nodo principal como fuente de estos resultados.

Otra hipótesis, es que la fuente de error esté relacionada con el uso de la Zedboard, ya que la implementación de la red neuronal consume gran cantidad de recursos y estos pueden volverse demasiado escasos al crecer el número de datos que hay que almacenar en la Zedboard, pero dado el tiempo disponible para realizar este proyecto, no fue posible explorar más a fondo esta hipótesis.

Sin embargo, los resultados para latencia presentados en la figura 3.14 muestran como resultado un tiempo por paso de 30.348571 ms para cuatro Zedboards con 1000 neuronas cada una, lo que resulta hasta trece veces menor al tiempo establecido como máximo para la comunicación dentro de los objetivos de este proyecto el cual era 400 ms, cumpliendo parcialmente el primer objetivo para el presente proyecto, y el cual se terminará de discutir en el siguiente capítulo.

3.4.2 Precisión en las transacciones

Para medir la precisión en las transacciones se realizó una simulación para el tamaño de red de 1000 neuronas empleando cuatro Zedboards durante 50000 pasos de simulación, todo esto con una red neuronal alojada en la FPGA de cada Zedboard. Para los valores de referencia, se utilizaron los producidos por la cosimulación en la herramienta de Vivado durante la síntesis de la red neuronal en hardware, ver [8] para más detalles.

Para evaluar la precisión, la comparación se realizó calculando el porcentaje de error relativo existente entre los datos experimentales y los de referencia usando la fórmula descrita en (3.1).

$$e_r = \frac{ref - exp}{ref} x100 \tag{3.1}$$

Esta comparación se hizo sobre una hipótesis, en la cual se establece que el porcentaje de error relativo obtenido no debe ser mayor al 5%. Luego de realizar la comparación entre ambos conjuntos de datos, no se encontró evidencia que la hipótesis fallara.

Capítulo 4

Ejecución de redes neuronales utilizando un sistema operativo

Hasta ahora la implementación de la red neuronal se ha ejecutado mediante la utilización de un kernel reducido que solo cuenta con la aplicación desarrollada sobre el entorno de Vivado SDK. Este capítulo realiza un primer acercamiento a la ejecución de redes neuronales utilizando un entorno de simulación más robusto al agregar un sistema operativo que permite la incorporación de herramientas que facilitan tareas como manejo de datos y comunicación con mayor eficiencia entre los diferentes nodos que componen el entorno de simulación.

El esquema de conexión utilizado para este capítulo es similar al mostrado en la figura 3.1, pero a diferencia de la implementación anterior, la comunicación directa entre Zedboards está habilitada para este capítulo. Además, en esta nueva implementación, aunque todos los dispositivos mantienen plena conexión por medio del `switch`, la computadora solo se comunica con la Zedboard que funciona como nodo principal, transmitiendo a esta los parámetros de la simulación además del archivo de inicialización, mientras que la simulación propiamente se ejecuta entre las Zedboards conectadas y los resultados se escriben en un archivo de texto que luego es devuelto hacia la computadora.

4.1 Sistema operativo en la Zedboard

Gracias al núcleo Zynq-7000 SoC de la Zedboard, es posible instalar sobre este un sistema operativo reducido que se ejecute sobre la arquitectura ARM. Para la implementación en este capítulo se utilizó la distribución Linaro, basada en Ubuntu. Esta distribución de Linux para sistemas embebidos se caracteriza por el bajo consumo de recursos que requiere para su ejecución, además de poseer herramientas necesarias para la creación de aplicaciones como lo son compiladores para lenguaje C y Python. La justificación para la elección de este sistema operativo viene del trabajo realizado en [16], ya que gracias a este trabajo previo se demostró que esta distribución puede ejecutarse sobre la Zedboard y que además, se pueden ejecutar aplicaciones muy similares a las que este proyecto requería

alcanzar.

El sistema operativo que se instaló sobre la Zedboard tiene como base el kernel de Linux que provee la herramienta `Linux Digilent Tools` versión cuatro, la cual se puede encontrar en [17] junto a mayor información de la misma. El sistema de archivos utilizado para este proyecto fue la distribución de Linaro versión *developer* 2015, la cual se puede encontrar en [18]. Además, el árbol de dispositivos se debió modificar para agregar el hardware correspondiente a la red neuronal; sin embargo, este tema no se abordó durante la realización del presente proyecto, sino que se trabajó en otro proyecto y del cual se puede encontrar mayor información en [8], en el cual se explica el proceso de creación del árbol de dispositivos, además del driver necesario para la utilización de la red neuronal desde el sistema operativo.

4.2 Aplicación en MPI

MPI se utilizó como herramienta para generar procesamiento en paralelo durante la simulación de redes neuronales. Cabe mencionar que la aplicación creada para utilizar MPI debe existir en todos los dispositivos que vayan a participar en la simulación de redes neuronales. Para esta implementación, se uso la distribución `MPICH` versión 3.2 encontrada en [14].

MPI permite dividir la aplicación creada en dos partes: proceso principal y procesos esclavos. Para realizar esto de la mejor manera, se crearon dos grupos dentro de la aplicación, un grupo para el proceso principal que únicamente que se encarga del manejo de datos que intervienen en la simulación y la comunicación con una computadora, y un grupo en donde se encuentran los procesos esclavos que se encargan de ejecutar la red neuronal de acuerdo a los datos que reciben desde el proceso principal.

Es necesario construir un intercomunicador mediante la función `MPI_Intercomm_create` para estos grupos. Las rutinas `MPI_Scatter` y `MPI_Gather` se modifican de manera que la transferencia de datos se realiza entre el proceso principal y los esclavos, sin que este primero se quede con porción de los datos que transfiere, contrario al comportamiento intra-grupal en donde el proceso principal si obtiene una porción de datos.

Una característica de la aplicación es que durante la generación de pasos de simulación, los nodos esclavos almacenan localmente la tensión de axón evitando transferencias constantes hacia el nodo principal. Como la Zedboard posee memoria limitada, esta se puede ver comprometida cuando se exportan gran cantidad de datos hacia el nodo principal para que este los escriba en archivo de texto, esto cuando se maneja simulaciones extensas. Por ello se implementó un hilo de ejecución que permite realizar escrituras en un archivo de texto de los resultados parciales de la simulación siguiendo el flujo mostrado en la figura 4.1. El hilo corre solo sobre el nodo principal y ejecuta una tarea de escritura cada vez que la `bandera de escritura` es activada.

Esta bandera se activa desde el proceso principal, que según el flujo de la figura 4.1, primero se divide el trabajo de cálculo de tensión de axón de la red entre los nodos esclavos

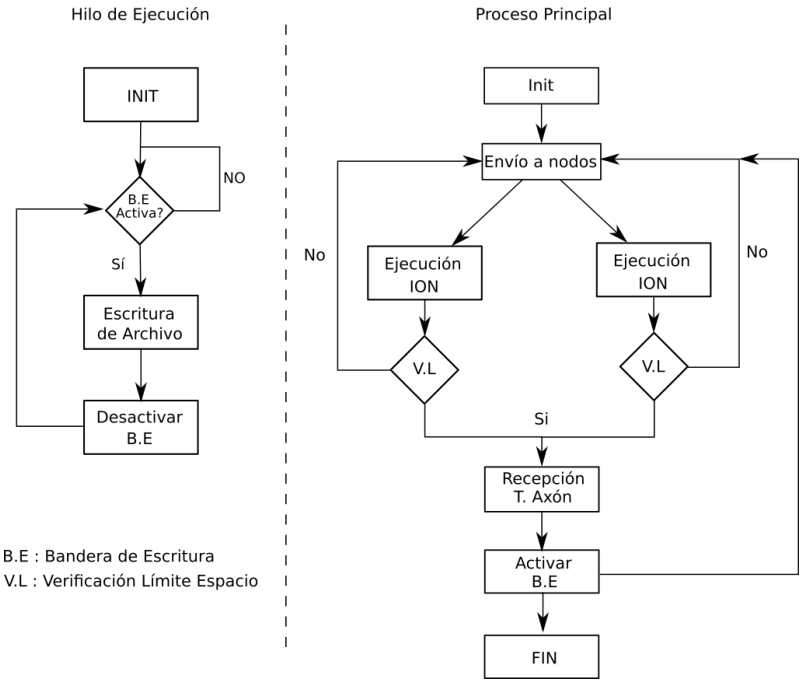


Figura 4.1: Diagrama de flujo para el proceso principal y el hilo de ejecución que corren sobre el nodo principal. Estos procesos se ejecutan en paralelo. Imagen propia.

que participen en la simulación. Posterior a esto, se verifica si los datos almacenados superan el límite de memoria establecido, el cual es de 50 MBytes. Este valor se fijó dado que la cantidad de datos que se pueden almacenar antes de realizar una escritura es tal que no interrumpe simulaciones cortas ni permite desbordamiento en memoria cuando se realizan simulaciones extensas.

Si se alcanza el límite, las tensiones de axón se exportan hacia el nodo principal y se levanta la `bandera de escritura`. Mientras el proceso principal continua con la ejecución de pasos, el hilo escribe los datos ya calculados en el archivo de texto. Esta operación ayuda a mejorar el tiempo de ejecución ya que el proceso principal no es detenido mientras la escritura se realiza. El arreglo donde se guardan las tensiones de axón en el nodo principal además de otros arreglos que participan en la aplicación son mostrados en la tabla 4.1.

Tabla 4.1: Arreglos creados para cada proceso que participa en la simulación.

Proceso	Arreglo	Descripción
Principal	`Init_data_Array`	Contiene los parámetros de inicialización de todas las neuronas.
	`Step_data_Array`	Contiene el estímulo en cada paso de simulación para todas las neuronas.
	`Final_Vaxon_Array`	Arreglo que contendrá los valores de tensión de axón resultado de la simulación realizada.
Esclavos	`Sub_init_Array`	contiene los parámetros de inicialización de las neuronas locales en cada proceso.
	`iApp_Array`	Contiene los valores de estímulo de las neuronas locales.
	`Vend_Array`	Contiene los valores de tensión de dendrita de todas las neuronas simuladas.
	`Sub_vend_Array`	Contiene los valores de tensión de dendrita locales de cada neurona en el proceso.
	`Sub_vaxon_Array`	Contiene la tensión de axón de las neuronas locales producto de la ejecución de un paso de simulación.

Para la ejecución de una simulación se requiere de tres archivos los cuales contienen los datos de la simulación que se va a realizar y que se describen como sigue:

- **config**: contiene los parámetros de la simulación como tamaño de red neuronal, cantidad de pasos de simulación y cantidad de Zedboards a utilizar.
- **hostfile**: Contiene la dirección IP de cada dispositivo disponible para la ejecución de redes neuronales, en donde la primer dirección en el archivo es la correspondiente al dispositivo donde se ejecuta el proceso principal.
- **data.txt**: Contiene los parámetros de inicialización de cada neurona que se desee simular además del estímulo correspondiente a cada paso de simulación.

4.2.1 Proceso principal

La figura 4.2 muestra el diagrama de estados creado para el proceso principal. La primera etapa, `Init`, se inicializan los arreglos necesarios para este proceso descritos en la tabla 4.1. Además, realiza la lectura del archivo `config` el cual contiene los parámetros de la simulación que se desea realizar. Estos datos se cargan en memoria del proceso principal para luego ser enviados a cada proceso esclavo de manera que cada proceso tenga conocimiento de las dimensiones de la simulación. Este envío se realiza por medio de la rutina `MPI_Send`. Seguidamente se realiza la lectura del archivo de inicialización que contiene los parámetros para cada neurona que se desee simular, almacenando estos datos en el arreglo `Init_data_Array`. Una vez obtenidos los datos, estos se envían a cada proceso esclavo en la etapa llamada `Envío de parámetros init`. Durante esta etapa el arreglo `Init_data_Array` es repartido por medio de la rutina `MPI_Scatter`.

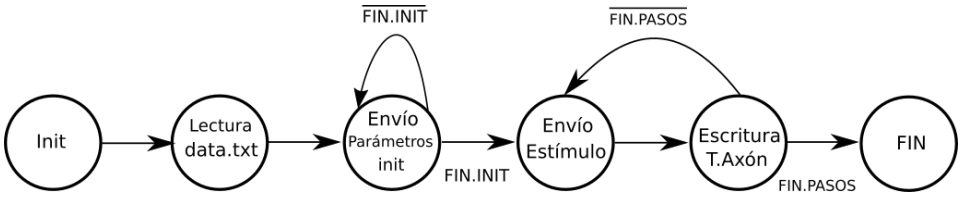


Figura 4.2: Diagrama de estados para el proceso principal. Imagen propia.

En la etapa `Envío de Estímulo` el proceso principal utiliza los datos de estímulo leídos desde el archivo de inicialización para enviarlos a cada proceso esclavo como parte de la ejecución de los pasos de simulación. Para esta etapa también se utiliza la rutina `MPI_Scatter` para enviar los datos que se alojan en el arreglo `Step_data_Array`. Si se alcanza el límite de 50MB, la etapa de `Escritura T.Axón` utiliza la rutina `MPI_Gather` para almacenar las tensiones de axón de los procesos esclavos en el arreglo `Final_Vaxon_Array` y luego escribirlos en un archivo de texto por medio de un hilo de ejecución, activando este último al levantar la `bandera de escritura`. Mientras no se alcance el límite, cada proceso esclavo almacena los resultados de la simulación localmente.

Si durante la ejecución de pasos no se alcanza el límite de memoria, los datos son escritos al archivo de texto en la etapa `Escritura T.Axón` durante el último paso de simulación a fin de no perder estos.

4.2.2 Procesos esclavos

Para los procesos esclavos, el diagrama de estados se muestra en la figura 4.3. La primer etapa de cada proceso, `init`, inicializa los arreglos necesarios durante la simulación y mostrados en la tabla 4.1, además de recibir los parámetros que definen la simulación, como tamaño de red local y global. Estos datos son recibidos desde el proceso principal utilizando la rutina `MPI_Recv`. Seguido a esto, en la etapa de `Recibir Parámetros init` cada proceso esclavo recibe su porción del arreglo `Init_data_Array` desde el proceso principal. Estos datos son producto de la ejecución de la rutina `MPI_Scatter` y los datos son almacenados en el arreglo `Sub_init_Array` para luego moverlos hacia el PL usando comunicación `AXI Lite`, así se inicializa la red neuronal.

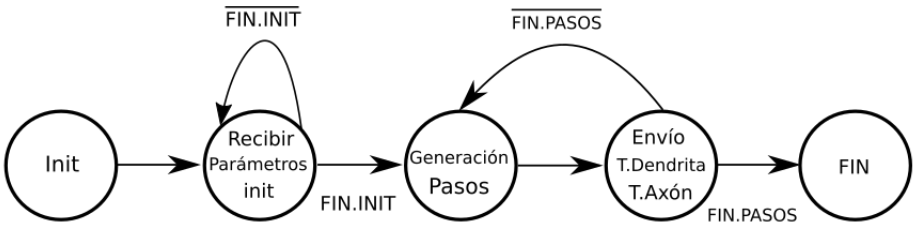


Figura 4.3: Diagrama de estados para procesos esclavos. Imagen propia.

En la etapa de `Generación de Pasos` cada proceso esclavo recibe el estímulo para las neuronas de su red local por medio de la rutina `MPI_Scatter`, almacenando los datos en el arreglo `iApp_Array`. Esta etapa utiliza el protocolo `AXI4 Stream` para la transmisión

de datos entre PS y la red neuronal alojada en el PL. Al ejecutar el paso de simulación, se realiza el envío de la tensión de dendrita de la red neuronal local en cada proceso esclavo hacia todos los demás por medio de la rutina `MPI_Allgather` para utilizar estos valores en el siguiente paso de simulación. Esto supone una ventaja ya que al utilizar la rutina `MPI_Allgather` para la transferencia de los datos de tensión de dendrita en cada paso de simulación, esta se ejecuta solo entre los procesos esclavos, evitando así las trasferencias hacia el proceso principal y mejorando la latencia del sistema. Además, si se alcanza el límite establecido en la sección del proceso principal, en esta etapa se envía la tensión de axón de las neuronas locales hacia el proceso principal por medio de la rutina `MPI_Gather`, en donde se transfieren los datos hacia el arreglo del proceso principal `Final_vaxon_Array` desde el arreglo `Sub_vaxon_Array`. De no superar el límite, este último arreglo almacena los datos de forma local y continua la ejecución de pasos hasta que el límite sea alcanzado o la simulación finalice.

4.3 Ejecución de la aplicación desarrollada

La aplicación desarrollada se ejecuta desde una computadora de uso general gracias a un conjunto de scripts escritos en bash. Estos scripts realizan envío y recepción de datos por medio de protocolo TCP entre la computadora y la Zedboard que funciona como nodo principal. Para ejecutar la aplicación se requiere de dos scripts corriendo en ambos extremos de la comunicación. El script que se ejecuta sobre la computadora se encarga de enviar los parámetros de la simulación en el archivo `config` además del archivo de inicialización; luego, envía una bandera que ejecuta la simulación y espera por el archivo que contiene la salida de la red neuronal.

El script que corre sobre la Zedboard recibe los archivos necesarios para la simulación (`config` e `Inicialización`) y espera la bandera que le indica que debe ejecutar una simulación. Recibida esta bandera, una nueva simulación se desarrolla con los parámetros recibidos y al terminarse, el script envía el archivo que contiene la salida de la simulación hacia la computadora.

4.4 Evaluación de resultados

La evaluación para este capítulo se realizó de forma similar al mostrado en el capítulo 3, es decir midiendo tanto latencia como precisión para esta interfaz de comunicación. Durante la medición de latencia, la aplicación desarrollada utilizando MPI no generó problemas relacionados con estabilidad o errores de ejecución, mismos que sí se presentaron con la versión de Vivado SDK, que detenía la ejecución aleatoriamente sin mostrar información acerca de las causas.

4.4.1 Latencia en la comunicación

Los resultados para la latencia de comunicación son presentados en la tabla A.2 ubicada en los apéndices, en donde utilizó el dispositivo `Axi_Timer` y la rutina `MPI_Timer` para medir la latencia por software relacionada con cada componente del ambiente de simulación. Para el nodo principal se utilizó una Zedboard la cual solo ejecuta las tareas de manejo de datos y no ejecuta redes neuronales. Aunque MPI permite a esta Zedboard actuar como nodo principal y a su vez ejecutar una red neuronal, la carga de trabajo sobre esta Zedboard introduce retardos no deseados en la ejecución de simulaciones.

Dado que el nodo principal solo ejecuta manejo de datos, se pensó inicialmente en implementarlo sobre una plataforma con mayores recursos. Sin embargo, no fue posible ejecutar aplicaciones usando MPI entre una computadora y Zedboards, ya que el enlace nunca se estableció. Una hipótesis que se maneja de lo sucedido es que la diferencia entre arquitecturas no permite a MPI funcionar correctamente, ya sea por diferencias en el orden de byte o velocidad de ejecución. Dado el tiempo disponible para ejecutar este proyecto, no se exploró más a fondo este problema.

La figura 4.4 muestra los resultados para latencia del entorno de simulación completo. Cabe destacar que esta implementación no tiene rubro de *swap* ya que la tarea de ordenamiento de datos de envío y recepción la ejecuta cada rutina colectiva en MPI, por tanto, este tiempo se toma en cuenta para la comunicación.

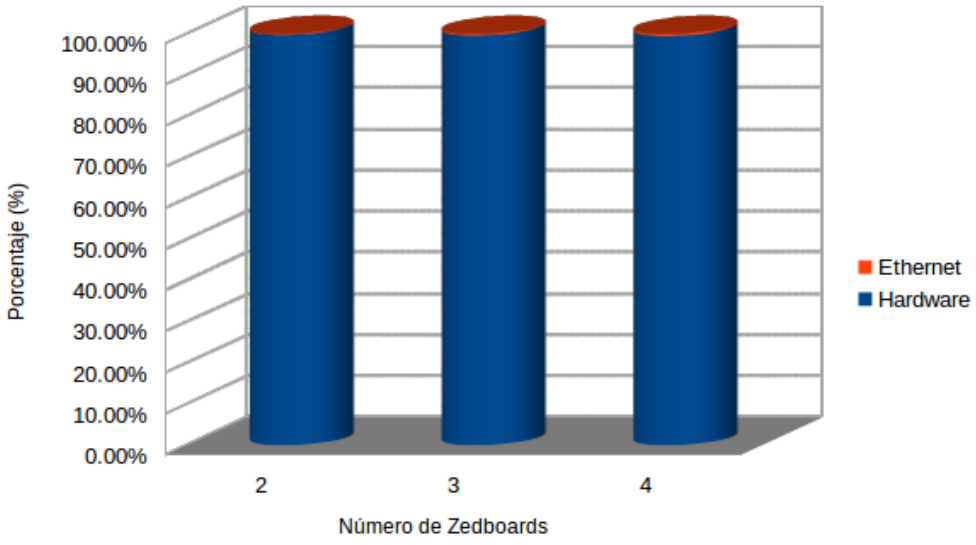


Figura 4.4: Resultados de latencia general para la implementación en MPI de redes neuronales. Estos resultados utilizan una red neuronal de 4000 neuronas y se varía la cantidad de Zedboards utilizadas. Imagen propia.

Los datos en la figura 4.4 evidencian que el cuello de botella para esta implementación sigue siendo el hardware sintetizado para la red neuronal, y esta implementación completa cerca de la totalidad del tiempo de ejecución por paso de simulación. Con respecto a la interfaz de comunicación, al estudiar los resultados de latencia que se muestran en la figura 4.5, el incremento en el tiempo de ejecución para un mismo tamaño de red es proporcional

al aumento de la cantidad de Zedboards utilizadas. Este es un comportamiento propio de la cantidad de enlaces que se deben realizar para comunicar un mayor número de Zedboards. Si contrastamos con la versión en sockets, cabe apreciarse que no existe el despegue de la tendencia lineal que observamos en la figura 3.14. Ello sirve para apuntalar la hipótesis de que existe ya sea un error de codificación o algún problema de manejo de recursos erróneos en aquella implementación.

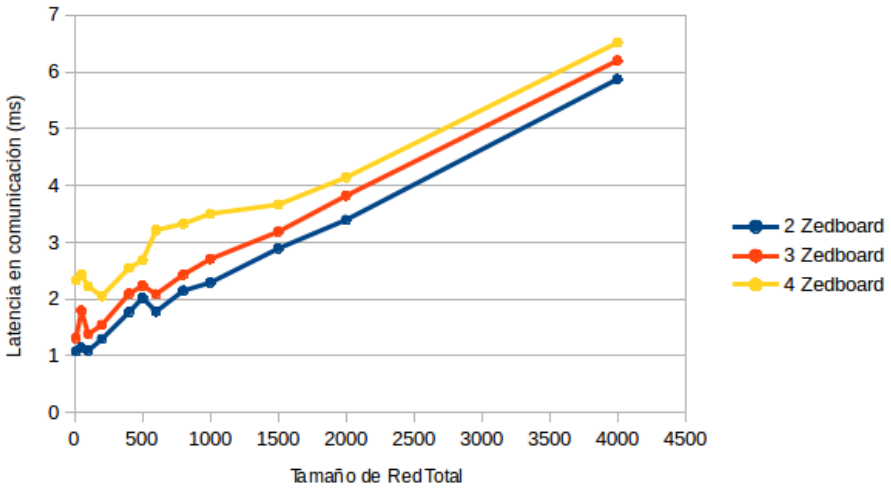


Figura 4.5: Resultados de latencia en comunicación para la aplicación usando MPI. Los resultados se presentan para múltiples Zedboards utilizadas y con distintos tamaños de red. Imagen propia.

Además, para evaluar con mayor detalle ambas implementaciones, se realizó una comparación a nivel de tiempo de ejecución de hardware y comunicación entre dos interfaces de comunicación. Para la latencia referente al hardware utilizado, los resultados se muestran en la figura 4.4, los cuales determinan que la implementación usando MPI presenta una ligera disminución en la eficiencia con respecto a la implementación en *bare-metal*. Esto es esperable puesto que esta implementación cuenta con un sistema operativo que introduce cierto retardo en el trasiego de datos entre el hardware alojado en la FPGA y la aplicación.

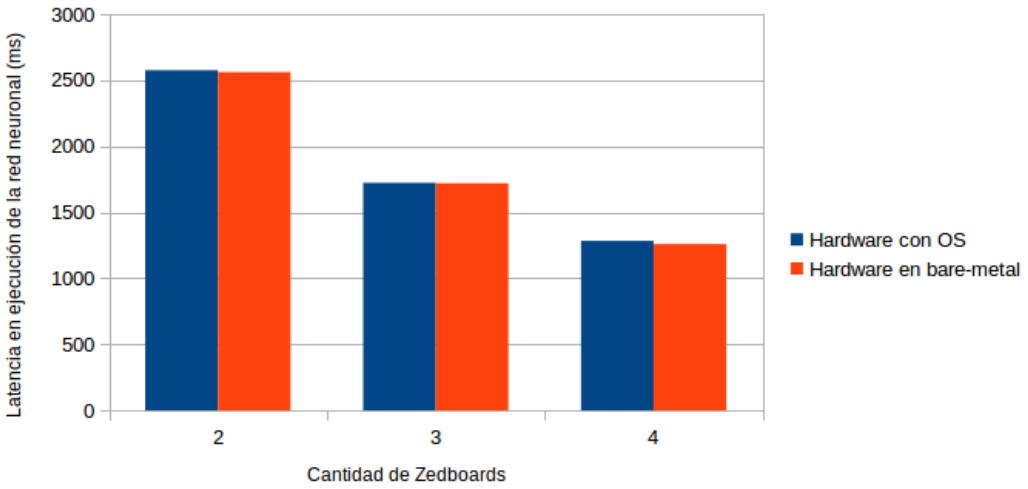


Figura 4.6: Gráfico comparativo para latencia del hardware utilizado en ambas implementaciones. Para esta ilustración se utilizó una red neuronal de 4000 neuronas y se ofrece resultados variando la cantidad de Zedboards. Imagen propia.

Además, se realizó una comparación entre las dos interfaces de comunicación, cuyos resultados se muestran en la figura 4.7 para una implementación usando cuatro Zedboards. Gracias a esta comparación se puede determinar que la interfaz utilizando MPI representa una forma más eficiente de comunicación para una plataforma con múltiples FPGAs con respecto a la versión *bare-metal* generada con Vivado SDK. La razón de mayor peso es el protocolo utilizado, MPI, que es una herramienta robusta creada y validada para tareas como la que se requería ejecutar en este proyecto, que permite el intercambio de datos entre nodos con una implementación dentro de la aplicación mucho más sencilla y enmascarada en pocas funciones. Además, la existencia de un sistema operativo sobre la Zedboard brinda características como mejor manejo de memoria o calendarización de tareas, incluyendo también manejo de errores, mismas que carecen en la implementación *bare-metal*.

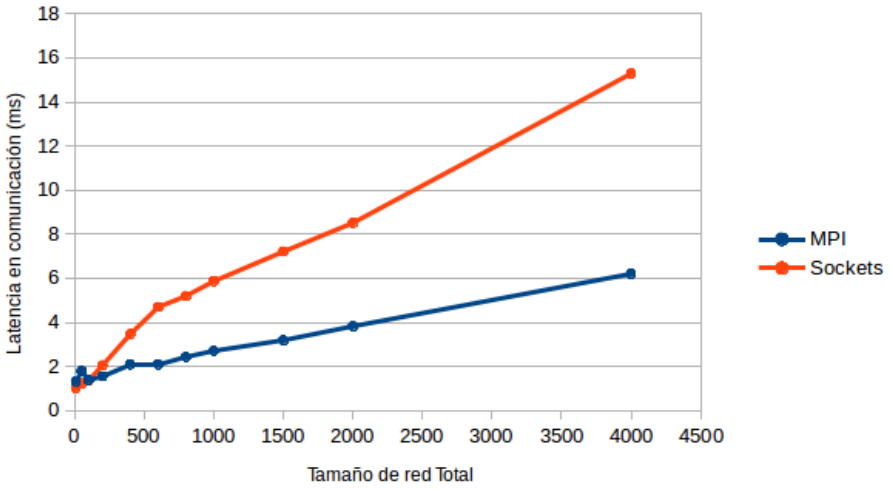


Figura 4.7: Gráfica comparativa de los tiempos de ejecución para dos métodos de comunicación usando tres Zedboards. Imagen propia.

Por tanto, la figura muestra una comparación entre rendimiento total de ambas implementaciones para redes con un número alto de neuronas simuladas usando tres Zedboards. Si se analiza esta imagen, la ventaja de la aplicación usando MPI con respecto a la versión *bare-metal* no es mayor puesto que la implementación del hardware con sistema operativo mostró más bajo rendimiento como se discutió en la figura 4.6. Sin embargo, esto no afectó en gran medida la mejora en rendimiento alcanzada al utilizar el estándar MPI y permite concluir que se puede alcanzar mejores resultados mediante la implementación de una aplicación usando sistema operativo y estándar MPI.

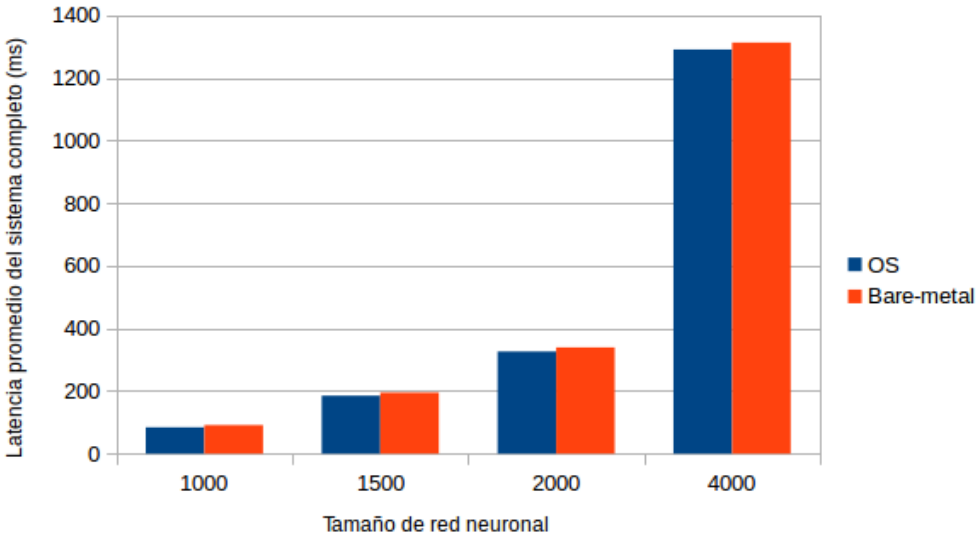


Figura 4.8: Gráfica comparativa de los tiempos de ejecución en ambos entornos de simulación. Este tiempo toma en cuenta tanto latencia de comunicación como tiempo de ejecución del hardware para el caso cuando se utilizan tres Zedboards y se varía el tamaño de la red neuronal en 1000, 1500, 2000 y 4000 neuronas. Imagen propia.

Además, como se denota en las tablas A.1 y A.2, cada interfaz de comunicación presenta una latencia inferior a la establecida como máxima dentro de los objetivos de este proyecto, que tenía un valor de 400 ms, por lo que se concluye que fue cumplido el primer objetivo del presente proyecto.

4.4.2 Precisión en las transacciones

Para evaluar la precisión del sistema, se ejecutó una simulación de redes neuronales usando cuatro Zedboards como nodos esclavos y un nodo principal implementado con una tarjeta del mismo tipo. Esta simulación constaba de 1000 neuronas en cada Zedboard con una duración de 50000 pasos. Para los valores de referencia, también se utilizaron los producidos por la cosimulación en la herramienta de Vivado durante la síntesis de la red neuronal en hardware como en el capítulo anterior.

La precisión de los datos transferidos se calculó mediante el porcentaje de error relativo usando la ecuación (3.1), y la hipótesis para este caso es la misma que la utilizada en el capítulo anterior, es decir, que el porcentaje de error relativo obtenido no debe ser mayor al 5%. Los resultados mostrados en la figura 4.9 muestran que esta hipótesis no se cumple para los datos obtenidos de la simulación.

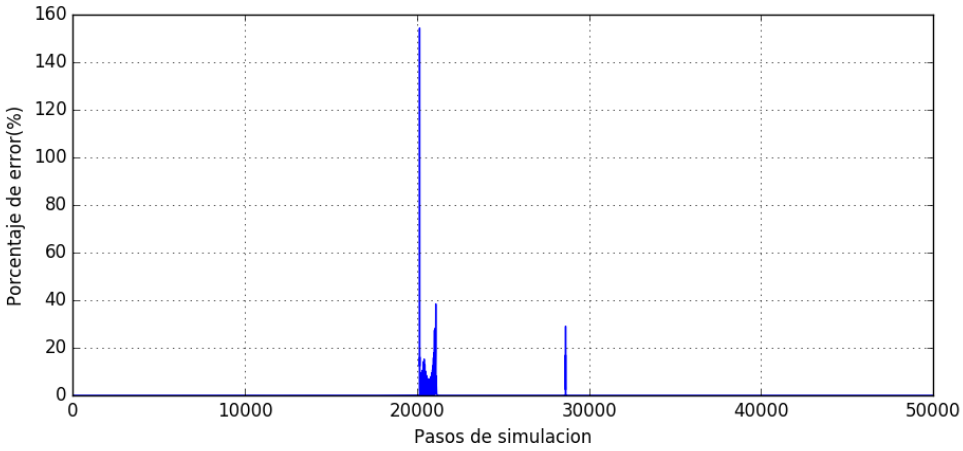


Figura 4.9: Resultados para el porcentaje de error en las transacciones usando 3 Zedboards con 1000 neuronas cada una, para un total de 50000 pasos de simulación. Imagen propia.

Dado este porcentaje de error no compatible con los valores esperados de acuerdo a la hipótesis, se modificó la aplicación creada para las Zedboards de forma que no se ejecute la red neuronal sino solo la interfaz de comunicación MPI, disminuyendo las posibles fuentes del error presentado. Esta aplicación se encarga de enviar un vector de datos desde la computadora al nodo principal y este ejecuta la simulación devolviendo para este caso los mismos datos que recibe desde la computadora, de manera que se pueda evidenciar pérdidas o errores en la transmisión de datos durante la aplicación.

Luego de realizar la comparación entre los datos producto de la prueba anterior, no se encontró evidencia que la hipótesis planteada inicialmente fallara, por lo que se determinó

que para esta implementación de redes neuronales, existe un error de precisión asociado al hardware sintetizado para la red neuronal, y del cual existe un análisis más a profundo realizado en [8]. Con respecto a la implementación de la interfaz de comunicación, esta no compromete la integridad de los datos para las pruebas realizadas; por tanto se concluye ambas interfaces implementadas cumplen con el segundo objetivo planteado para este proyecto.

Capítulo 5

Desarrollo del método de visualización basado en un servidor *web*

Una página *web* ofrece al científico el acceso a simulaciones de redes neuronales desde cualquier lugar con acceso a internet, sin necesidad de instalar software adicional. Para implementar todas las funcionalidades de la página *web* se utilizó el lenguaje PHP, que permite ejecutar comandos y aplicaciones sobre el servidor transparentes para el usuario, mejorando la experiencia del mismo.

5.1 Diseño y requerimientos de la página *web*

El diseño de la página *web* concentró todos los elementos de mayor relevancia dentro de la página principal, a fin de facilitar el uso y navegación del usuario sobre esta. Un requisito del diseño es incorporar la característica de flexibilidad en la página *web*. Por ello, se definieron las variables mostradas en la tabla 5.1, que le permiten al usuario personalizar una simulación de acuerdo a sus necesidades.

Tabla 5.1: Variables que modifican las características de la simulación a visualizar en la página *web*.

Variable	Descripción	Tipo
`Inicialización`	Contiene la inicialización de la red neuronal más el estímulo	Archivo Texto
`Tamaño de red Global`	Representa el total de neuronas a simular	Numérico
`Número de pasos`	Representa la cantidad total de pasos de simulación que se desean realizar	Numérico
`Dispositivos FPGA`	Representa la cantidad de dispositivos (Zedboard) que se desea utilizar	Numérico

Además, se deben mostrar los resultados de la simulación de forma gráfica, por lo que se definió utilizar tres diferentes gráficas para visualizar los datos, definidos en la tabla 5.2. Como estas gráficas representan el comportamiento eléctrico de las neuronas simuladas, es pertinente que estas contengan además de escalas numéricas, una escala con color que permita analizar visualmente y de forma rápida los resultados.

Tabla 5.2: Tipos de gráficas utilizadas en el sitio *web*.

Variable	Descripción
`Gráfica neurona Individual`	Permite visualizar la respuesta individual de cada neurona
`Gráfica de Red neuronal 2D`	Representación 2D de la red neuronal y su comportamiento
`Gráfica de Red neuronal 3D`	Representación 3D de la red neuronal y su comportamiento

Deben además implementarse opciones para descargar y cargar los resultados de una simulación del sitio *web*. Con ello se le ofrece al usuario la posibilidad de guardar en su computadora los datos de simulación y visualizarlos cuando desee simplemente cargando estos al sitio *web*. Además, se debe existir un indicador de cuantos dispositivos (Zedboards) están disponible para realizar la simulación. Finalmente, se definió que para este proyecto se implementará un servidor *web* de tipo local, es decir, solo se puede tener acceso al mismo si el usuario se encuentra en la misma red en donde se encuentra el servidor *web*.

5.2 Detalle de Solución

Para crear el servidor *web* se utilizó la herramienta Apache, que crea un directorio y asocia este a la página *web*. En este directorio se encuentran los archivos que forman la página *web* y como se trata de un servidor de tipo local, basta con escribir sobre un buscador *web* la dirección IP del computador sobre el cual corre el servidor *web* para accederlo. La capa interna de la página *web* está conformada por varios archivos que constituyen su funcionamiento. Estos archivos se muestran en la tabla 5.3 junto a una descripción de sus funciones.

Tabla 5.3: Archivos que componen la página *web*.

Archivo	Descripción
`index.php`	Contiene la estructura principal de la página *web*
`ethernet.py`	Se encarga ejecutar la simulación de acuerdo a los parámetros seleccionados por el usuario
`Axongraphic.py`	Realiza la gráfica de la respuesta individual para una neurona
`Rastergraphic.py`	Realiza la gráfica para el conjunto de neuronas simuladas
`Volumetricgraphic.py`	Realiza la gráfica para el conjunto de neuronas simuladas con una representación 3D
`styles.css`	Contiene los estilos y fuentes

Estos archivos integrados construyen la página *web* que se muestra en la figura 5.1, que está formada por tres partes principales: una barra de funciones principales, un menú lateral para configurar la simulación que se desea realizar y un área de gráficas, donde se encuentran las gráficas de representación de resultados de simulación y una representación para dispositivos conectados.

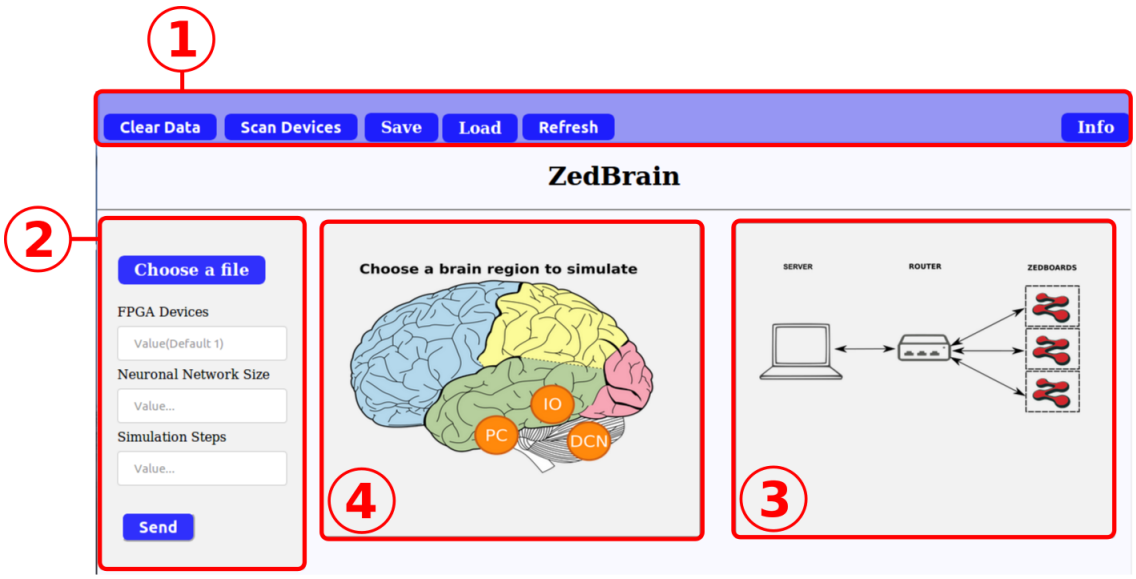


Figura 5.1: Página principal de la página *web*, en donde 1 destaca la barra de funciones principal, 2 la barra lateral para configuración de las características de una simulación, 3 presenta el número de Zedboards conectadas a la red local y 4 presenta las regiones del cerebro que se pueden simular. Imagen propia.

5.2.1 Barra de funciones principal

La barra principal, numerada como 1 en la figura 5.1 se encarga de ejecutar funciones como: carga o descarga de resultados, escaneo de Zedboards disponibles y actualización

de gráficas. El botón de `Clear Data` tiene como función limpiar las diferentes gráficas que se muestran en la página *web*, llevándolas a un valor por defecto de cero. El botón de `Scan Devices` permite al usuario realizar un escaneo de las Zedboards que se encuentran disponibles para utilizar en la simulación; este botón relaciona su función con una representación gráfica de dispositivos conectados y su funcionamiento se verá en detalle más adelante.

El botón de `Save` permite al usuario descargar los resultados de una simulación realizada previamente. Esta operación se lleva a cabo mediante la función `download='objetivo'` del lenguaje HTML. El botón de `Load` de clase archivo de entrada permite cargar un archivo que contiene los resultados de simulaciones previamente realizadas en caso que el usuario desee observar los resultados nuevamente. El botón de `Refresh` permite generar gráficas a partir de los datos que se encuentran en el archivo `dataG.txt` en dos casos; el primero cuando se realiza una nueva simulación, y el segundo caso cuando se carga un archivo a la página *web*. Finalmente, el botón `info` despliega información general acerca de la página *web*.

5.2.2 Menú lateral

El menú lateral, numerado en la figura 5.1 como 2, es la sección que permite configurar una nueva simulación. En este menú el usuario puede variar los parámetros establecidos en la tabla 5.1.

Con el botón `Choose a file` el usuario puede cargar un archivo de inicialización para la red neuronal que desea simular. Las opciones `FPGA Devices`, `Neural Network Size` y `Simulation Steps` son variables que se encargan de capturar los datos introducidos por el usuario.

Finalmente el botón `Send` permite iniciar una nueva simulación. Al ser presionado, primeramente se verifica que el usuario haya cargado un archivo de inicialización, además de que los valores para las variables de configuración de la simulación sean válidos y la cantidad de dispositivos seleccionados no sea mayor a cuatro, ya que para efectos de este proyecto, se cuenta con un máximo de cuatro Zedboards conectadas. Luego de la verificación, se realiza una llamada a sistema para ejecutar la simulación dependiendo del tipo de implementación, ya que para el caso del capítulo 3 se ejecuta un script en Python y para el capitulo 4, un script en `bash`.

5.2.3 Función de escaneo de red

Como se mencionó, el botón de `Scan Devices` tiene asociado un recurso gráfico para informar al usuario la cantidad de Zedboards disponibles, mostrado en la figura 5.1 como 3. Su funcionamiento está basado en una llamada a sistema que ejecuta el comando de Linux `nmap` junto a la dirección IP base de la red local a la que se encuentran tanto el servidor *web* como las Zedboards. Este comando tiene como salida una tira de caracteres con información acerca de la red local, por lo que realizando operaciones sobre la tira se

puede extraer el número de Zedboards disponibles.

5.2.4 Mensajes de Alerta

La página *web* muestra diferentes mensajes que ayudan al usuario a darse cuenta cuando una acción se ejecuta; por ejemplo, cuando se ha cargado correctamente un archivo al sitio *web* o cuando una simulación ha terminado. Para realizar esto se utilizó la función `alert()` del lenguaje Javascript, que muestra el mensaje contenido en una ventana emergente.

5.2.5 Selección de región del cerebro a simular

En la figura 5.1 rotulado como 4, se muestran las diferentes regiones del cerebro que se pueden simular por los distintos modelos matemáticos, a saber PC (Purkinje Cell Layer), DCN (Deep Cerebellar Nuclei) e IO, sin embargo para efectos de este proyecto solo existe la posibilidad de simulación del modelo IO. Para seleccionar una región, se utilizó la función $< map >$ de HTML, el cual define una región sobre la figura deseada y al dar click sobre ella, se selecciona esta región en la imagen.

5.2.6 Gráfica para la respuesta individual de cada neurona

El objetivo de esta gráfica es mostrar al usuario el comportamiento individual durante la simulación de una neurona, que puede ser cualquiera dentro del rango de uno hasta el tamaño de red total. Para realizar esto, el usuario debe seleccionar el número de neurona que desea visualizar en el recuadro bajo la etiqueta `Choose a neuron response to display`, seguido del botón de `Refresh` visto en la barra de funciones principal. Con ello se ejecuta script Python que genera la gráfica de respuesta individual para la neurona. Un ejemplo del resultado de este script es mostrado en la figura 5.2.

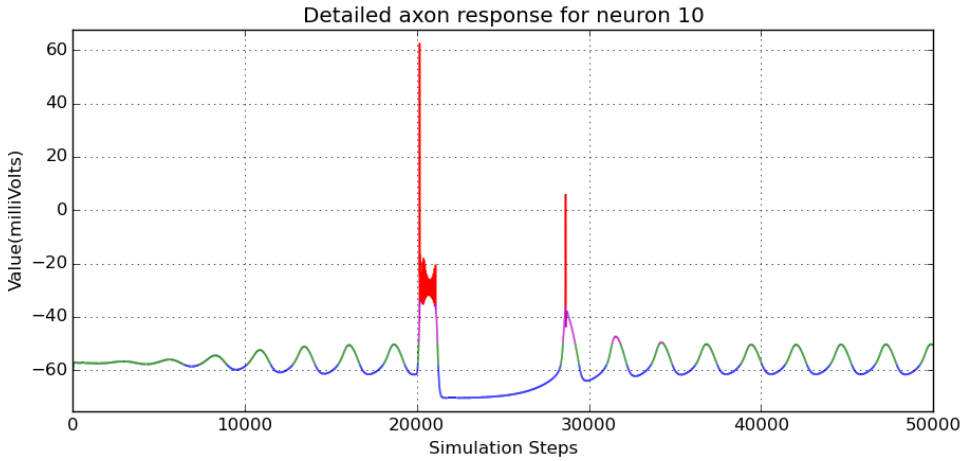


Figura 5.2: Gráfica para la respuesta individual de una neurona ante una simulación de 50000 pasos. Imagen propia.

Cada script mostrado en este documento utiliza la biblioteca Matplotlib. Este y los siguientes scripts reciben una bandera que indica el modo de construcción de gráfica, sea para limpiar la gráfica o generar una nueva a partir de los datos disponibles. Si la bandera es para generar una gráfica, esta se construye a partir de los valores de tensión de axón de una neurona seleccionada contra la cantidad de pasos simulados. Además, se asocia una escala de color de acuerdo al valor de tensión de axón para representar la intensidad eléctrica. El script finaliza exportando una imagen tipo png similar a la mostrada en 5.2 que luego será mostrada en la página *web*.

5.2.7 Gráfica trama del conjunto de neuronas simuladas

Consiste en una gráfica en donde se muestra el comportamiento eléctrico de la red neuronal en dos dimensiones, en donde el eje "x" contiene la cantidad de neuronas simuladas y el eje "y" representa la cantidad de pasos simulados. Cada línea a lo largo del eje "y" representa una neurona y la tensión de axón de cada neurona define la escala de color que se utiliza en estas líneas. Este script utiliza la función `imshow` para crear la gráfica de superficie a partir de los datos en el archivo `dataG.txt` y exporta una imagen para mostrarla en la página *web*. El resultado de este script se muestra en la figura 5.3, en donde se observa un patrón uniforme ya que todas las neuronas se comportan de la misma manera para este caso particular.

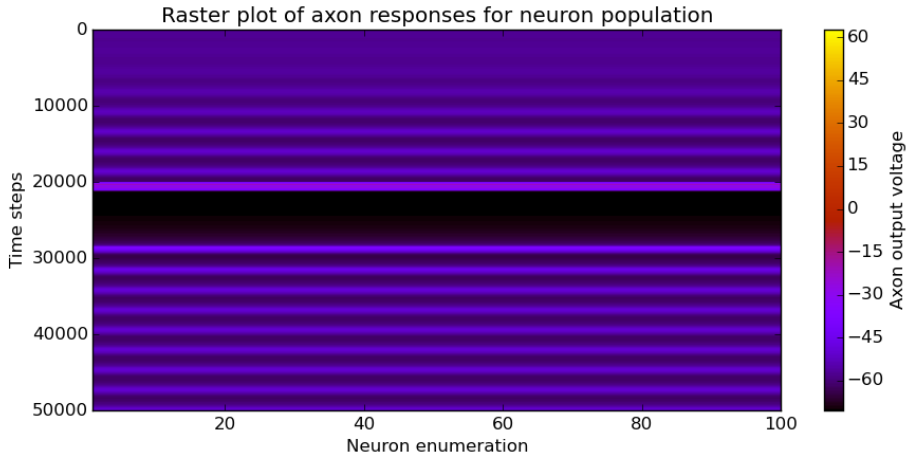


Figura 5.3: Gráfica de trama de la respuesta en conjunto de las neuronas ante una simulación con 50000 pasos. Imagen propia.

5.2.8 Gráfica 3D del conjunto de neuronas simuladas

En la gráfica formato 3D cada neurona se representa por medio de un punto dentro de un gráfico de dispersión que cambia de color de acuerdo a la intensidad de la respuesta obtenida en la simulación. Para obtener este efecto, se creó una imagen tipo GIF creada a su vez por múltiples imágenes en donde se varía el paso de simulación que se muestra entre cada imagen y por ende, la intensidad de color de cada neurona varía.

Esta implementación presenta una limitante la cual se relaciona a la cantidad de neuronas que puede representar, ya que al tratarse de una figura cúbica, la cantidad total de neuronas debe poder repartirse uniformemente en el volumen de este cubo a fin de evitar errores en la biblioteca. Para crear el GIF, se usó el comando `convert`. Un ejemplo del resultado que se obtiene al crear este tipo de gráfica se muestran en 5.4.

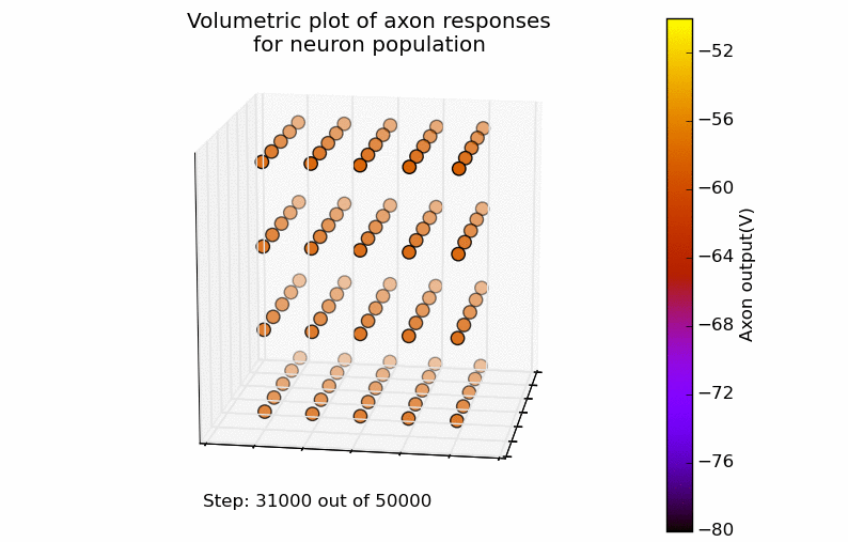


Figura 5.4: Gráfica 3D de la respuesta en conjunto de neuronas durante una simulación de 50000 pasos. Imagen propia.

5.3 Resultados

La página *web* tiene como resultado la interfaz mostrada en 5.5, en donde este sitio se puede acceder desde cualquier dispositivo conectado a la red local. Además se verificó su funcionalidad realizando las siguientes acciones sobre la misma:

- Se cargó un archivo de inicialización al servidor *web*.
- Se ejecutaron simulaciones con los parámetros establecidos en el menú lateral.
- La página *web* crea las diferentes gráficas cuando se presiona el botón de `refresh`.
- Se puede descargar y cargar archivos con resultados de simulación.
- La función de escanear la red detecta los cambios en la cantidad de Zedboards conectadas.

Además, toda la funcionalidad de esta página y del servidor *web* fue mostrada al equipo del Centro Médico Erasmus, los cuales aprobaron el diseño realizado luego de hacer algunas observaciones en cuanto a forma a fin de mejorar la experiencia del usuario. Con esto se afirma que el tercer objetivo del presente proyecto fue cumplido.

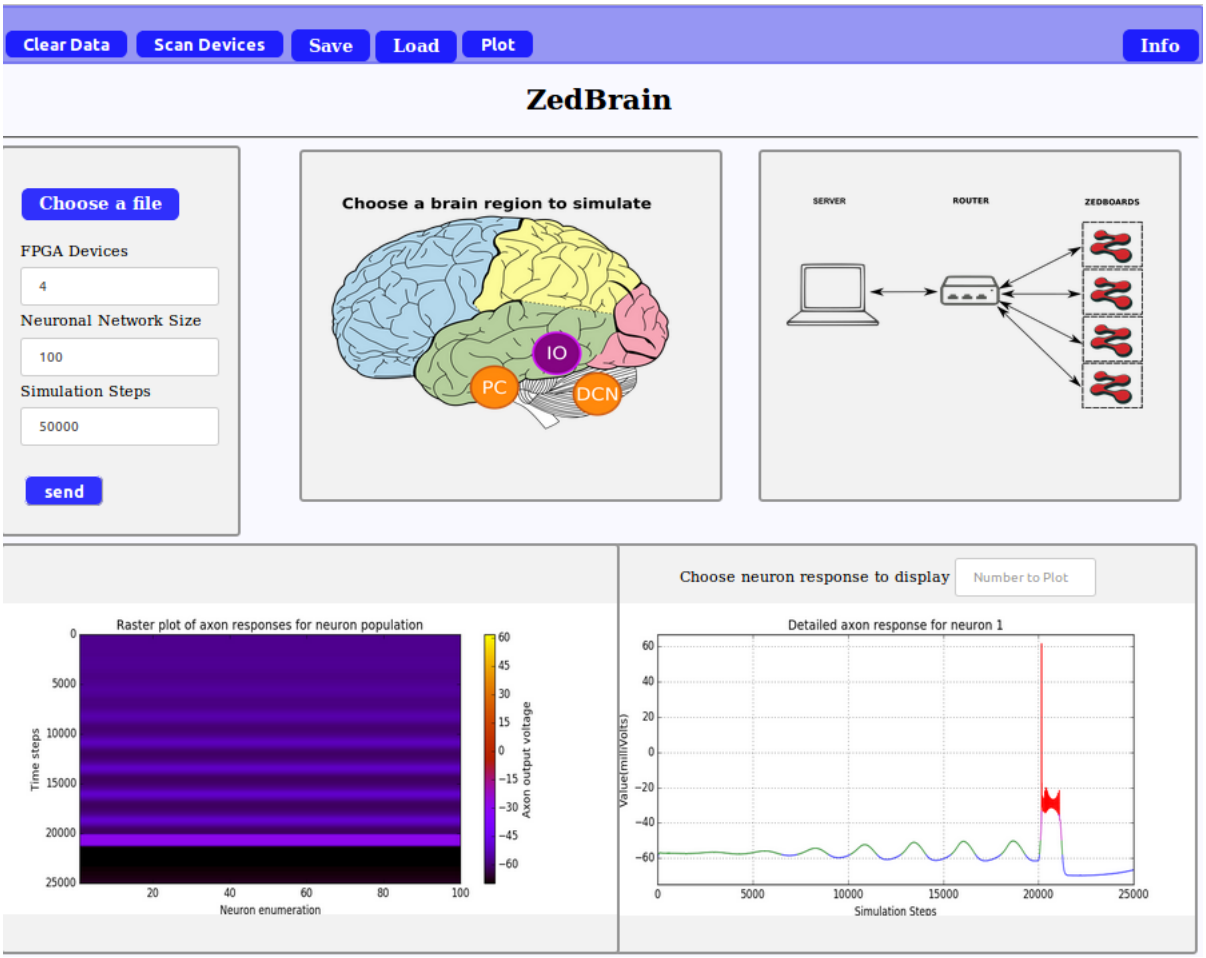


Figura 5.5: Página *web* obtenida como resultado. Se denotan los resultados en forma gráfica para una simulación realizada, un total de cuatro Zedboards conectadas a la red y los parámetros necesarios para ejecutar una nueva simulación en el menú lateral. Imagen propia.

Capítulo 6

Conclusiones y Recomendaciones

6.1 Conclusiones

Se desarrolló una interfaz de comunicación con el software de Vivado SDK que demuestra que es posible implementar simulaciones de redes neuronales basadas en el modelo ION utilizando múltiples FPGAs. En dicha implementación se ha comprobado que los resultados son fieles a la respuesta esperada de la red neuronal.

Fue posible crear una interfaz de comunicación basada en el estándar MPI y haciendo uso de un sistema operativo, dotando al entorno de simulación de mayor robustez en su implementación. Dados los requerimientos planteados para la interfaz de comunicación al inicio del documento, esta implementación ofreció mejor rendimiento general comparado con la versión en Vivado SDK, además de mejorar las características de flexibilidad que se ofrece al entorno de simulación en cuanto a cantidad de dispositivos que participan y cambios en los parámetros que manejan cada simulación.

Se implementó un método para visualización de resultados de simulaciones, en el cual es posible controlar simulaciones de redes neuronales biológicamente realistas y observar sus resultados de forma gráfica, además de adaptarse a las características de flexibilidad y fácil uso que mejoran la experiencia del usuario.

6.2 Recomendaciones para la interfaz de comunicación

Es importante mantener una versión estable del software de Xilinx, ya que se observó que durante la implementación de la interfaz de comunicación usando Vivado SDK, cambios entre versiones del software repercutían en errores de ejecución de la aplicación.

Es recomendable explorar puertos de comunicación alternativos que permitan velocidades de transmisión de datos mayores al estándar Ethernet. Por ejemplo la Zedboard incluye el puerto de alta velocidad FMC HPC con velocidades de hasta 5 GB, el cual no fue utilizado en este proyecto ya que no se incluía dentro de los alcances del mismo.

Realizar un estudio donde se evalúen otros protocolos de comunicación diferentes a MPI,

que permita medir el rendimiento de cada protocolo y determinar si la actual implementación usando MPI es la más eficiente para este tipo de aplicaciones.

Realizar un estudio para establecer las capacidades de ambas interfaces de comunicación en términos de máximo tamaño de red neuronal o máxima cantidad de pasos de simulación que se pueden disponer en una simulación sin comprometer los recursos de memoria disponibles para utilizar en el nodo principal.

Analizar el comportamiento obtenido al conectar un número mayor a 4 Zedboards al entorno de simulación. Analizar cómo afecta la latencia al utilizar un mayor número de dispositivos y si estas nuevas incorporaciones producen algún inconveniente en la ejecución de simulaciones con redes neuronales.

Estudiar el comportamiento del entorno de simulación cuando se conectan más de cuatro Zedboards a la red. Determinar si la tendencia lineal de crecimiento referente al aumento de dispositivos conectados se mantiene.

Explorar variantes del sistema operativo como Petalinux y medir el rendimiento además de establecer ventajas que estos ofrecen para la implementación de la interfaz de comunicación utilizando MPI. Este estudio permitirá evaluar si la versión utilizando Linaro es la más competente para la aplicación desarrollada.

Elaborar una métrica para evaluar la demanda de recursos de memoria ante determinados parámetros de red como tamaño de red y pasos de simulación.

Se recomienda estudiar más a fondo el comportamiento evidenciado en la interfaz de comunicación por sockets cuando se utilizan cuatro Zedboards a fin de hallar la posible fuente que provoca el despegue en la tendencia lineal, esto si se desea seguir utilizando esta interfaz de comunicación en futuras implementaciones.

6.3 Recomendaciones para el sitio Web

La utilidad de la página web puede aumentarse al hacerla accesible desde cualquier lugar con acceso a internet; solo se requiere adquirir un dominio web. Con ello no se requiere estar en la misma red que el servidor web y permite ejecutar simulaciones remotamente.

Es posible mejorar el rendimiento de la página web cuando construye las gráficas, ya que el actual método tarda un tiempo considerable en realizar esta tarea por la cantidad de datos que debe administrar cuando la simulación es muy extensa (miles de pasos). Se recomienda explorar métodos de visualización para gran cantidad de datos.

Bibliografía

[1] Erasmus Brain Project. (2016) Framework for high-detail and real-time brain simulations (brainframe). [Online]. Available: http://erasmusbrainproject.com/index.php/themes/brainframe

[2] Smaragdos.G, M. O, Ciobanu, Z. C, D. C, Sourdis.I, Strydis.C, and Rodopoulos.D, *Real-Time Olivary Neuron Simulations on Dataflow Computing Machines.* ISC, 2014.

[3] Smaragdos.G, Chatzikostantis.G, Nomikou.S, Sourdis.I, Strydis.C, and Rodopoulos.D, *Performance Analysis of Accelerated Biophysically-Meaningful Neuron Simulations.* ISPASS, 2016.

[4] Smaragdos.G, Isaza.S, V. Eijk.M, Sourdis.I, and Strydis.C, *FPGA-based biophysically-meaningful modeling of olivocerebellar neurons.* ACM/SIGDA international symposium on Field-programmable gate arrays, 2014.

[5] Xilinx, *Zedboard Hardware User's Guide*, 2nd ed., 2014.

[6] Xilinx., *Vivado Design Suite User Guide.* UG893 (v2016.4), 2016.

[7] Xilinx, *Xilinx Software Development Kit (SDK) User Guide.* UG1145 (v2016.4), 2016.

[8] Alfaro-Badilla.K, *Diseño de un acelerador de hardware para simulaciones de redes neuronales biológicamente precisas utilizando un sistema multi-FPGA.* Tesis de grado, Instituto Tecnológico de Costa Rica, 2017.

[9] Barry.P and Crowley.P, *Modern Embedded Computing.* Elsevier Inc, 2012.

[10] The Apache Software Foundation. (2017) The apache http server project. [Online]. Available: https://httpd.apache.org/

[11] The PHP Group. (2017) Php: Hipertext preprocessor. [Online]. Available: http://php.net/docs.php

[12] Matplotlib Development Team. (2016) Matplotlib: Python plotting. [Online]. Available: https://matplotlib.org/

[13] T. University of Tennessee, *MPI, Message Passing Interface.* MPI Forum(Version 3.1), 2015.

[14] MPICH.org. (2015) High-performance portable mpi. [Online]. Available: https://www.mpich.org/static/docs/v3.2/

[15] Mentor. (2013) Sourcery codebench,v4.8. [Online]. Available: https://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/

[16] Jimenez-Vargas.D, *Propuesta de una plataforma de computación paralela para la simulación de neuronas del núcleo olivar inferior*. Tesis de grado, Instituto Tecnológico de Costa Rica, 2016.

[17] Digilent Inc. (2015) Git repository,linux-digilent dev. [Online]. Available: https://github.com/DigilentInc/Linux-Digilent-Dev/tree/master/arch

[18] Linaro. (2016) Linaro releases version 15.07. [Online]. Available: https://www.linaro.org/downloads/

Apéndice A

Latencia en ejecución de simulaciones

Tabla A.1: Latencia registrada para comunicación usando Sockets TCP-IP.

	Tiempo de ejecución (ms)				
Tamaño de la red	Zedboards	Hardware	Ethernet	Swap	Tiempo total
10	2	0.047225	0.46894012	0.0510406	0.6020324
	3	0.029129	1.0209069	0.07123231	1.12126821
	4	0.043072	1.08730147	0.07142782	1.20180129
50	2	0.394393	0.9792561	0.1265466	1.2664127
	3	0.222453	1.2183532	0.20853638	1.64934258
	4	0.092233	1.31159347	0.17505884	1.57888531
100	2	1.563341	1.3947684	0.2884376	2.96895
	3	0.970249	1.3667203	0.28761744	2.62458674
	4	0.6573744	1.55077623	0.31655073	2.52470136
200	2	6.344214	2.0222622	0.6500089	9.0164851
	3	4.161462	2.0496769	0.68149561	6.89263451
	4	2.940817	2.17422351	0.67395448	5.78899499
400	2	25.554352	3.1475784	1.6026926	30.304623
	3	16.592612	3.4638204	1.53925299	21.59568539
	4	11.735323	3.91722984	1.60183191	17.25438475
600	2	57.670974	3.9495071	2.5515831	64.1720642
	3	37.900524	4.4921483	3.01948428	45.41215658
	4	26.401591	6.2864449	3.12477946	35.81281536
800	2	102.457738	4.4799768	3.5174047	110.4551195
	3	67.798011	5.19202739	4.05812382	77.04816221
	4	47.797682	8.1525197	4.64946269	60.59966439
1000	2	160.147073	5.0029852	4.3652129	169.5152711
	3	105.657198	5.8676068	5.16827583	116.69308063
	4	75.30285	9.8536563	5.78007936	90.93658566
1500	2	640.533826	6.502864	6.33001089	373.17029589
	3	428.072826	6.9122861	7.62425541	253.93592951
	4	310.744128	13.4892217	8.64931106	194.93275876
2000	2	640.533826	7.9065714	8.55237483	656.99277223
	3	425.072826	8.5150637	9.80051278	443.38840248
	4	310.744128	16.952466	11.44075512	339.13734912
4000	2	2562.09482	13.673941	16.2789315	2592.047693
	3	1720.317921	15.2842371	18.32245543	1753.924614
	4	1259.59032	30.348571	22.3475292	1312.283622

Tabla A.2: Latencia registrada para comunicación usando MPI.

	Tiempo de ejecución (ms)			
Tamaño de la red	Zedboards	Hardware	Ethernet	Tiempo total
10	2	0.124305	1.078745	1.20305
	3	0.120249	1.31347	1.433719
	4	0.121931	2.33142	1.453351
50	2	0.680463	1.143296	1.823759
	3	0.410533	1.795598	2.206131
	4	0.342186	2.432079	2.774265
100	2	1.910792	1.092257	3.003049
	3	1.589710	1.281685	2.871395
	4	1.003178	2.222434	3.225612
200	2	6.836039	1.295346	8.131385
	3	4.654584	1.54442	6.199004
	4	3.457996	2.054146	5.512142
400	2	26.417126	1.768146	28.185272
	3	17.538671	2.0939373	19.6326083
	4	13.239086	2.546402	15.785488
600	2	58.739006	1.780903	60.519909
	3	39.195454	1.887143	41.082597
	4	29.423758	3.216367	32.640125
800	2	103.891174	2.147516	106.03869
	3	69.470131	2.429847	71.899978
	4	51.999435	3.326634	55.326069
1000	2	161.848404	2.290542	164.138946
	3	107.722290	2.707164	110.429454
	4	80.982620	2.904631	83.887251
1500	2	362.746277	2.891905	365.638182
	3	241.906311	3.187225	245.093833
	4	181.431427	3.664727	185.096154
2000	2	643.616394	3.396603	647.012997
	3	429.555725	3.821398	433.377123
	4	321.888397	4.1441945	326.0325915
4000	2	2567.150146	5.873984	2573.02413
	3	1713.227173	6.200747	1719.42792
	4	1283.698364	6.515553	1290.213917