

Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica



Extensión de las capacidades de comunicación y memoria para sistema de modelado
de redes neuronales del olivo inferior

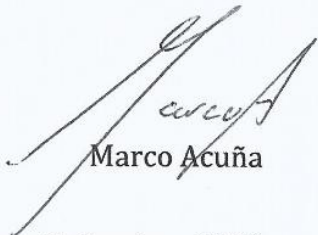
Informe de Proyecto de Graduación para optar por el título de Ingeniero en
Electrónica con el grado académico de Licenciatura

Marco Acuña

Cartago, Febrero, 2016

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

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.



Marco Acuña

Cartago, Setiembre, 2015

Ced: 1-1504-0701

INSTITUTO TECNOLOGICO DE COSTA RICA

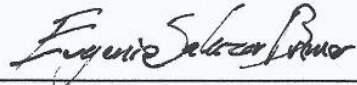
ESCUELA DE INGENIERIA ELECTRONICA

PROYECTO DE GRADUACIÓN

ACTA DE APROBACIÓN

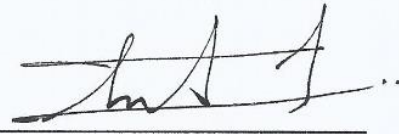
Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal



Ing. Eugenio Salazar Brenes

Profesor lector



Ing. Leonardo Sandoval Cascante

Profesor lector



Dr. 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, 8 de febrero, 2016

Resumen

En el presente informe se expone el proceso de diseño, implementación y principales resultados y conclusiones de un sistema anfitrión basado en procesador Microblaze para albergar un soft IP core de modelado de redes neuronales desarrollado en Vivado HLS por el Centro Médico Erasmo en Rotterdam, Holanda.

En primera instancia el módulo no contaba con hardware para funcionar con los puertos de comunicación como UART y Ethernet. Esto se abordó, diseñando e implementando un sistema host basado en un procesador Microblaze que integra soft IP cores de Ethernet, UART y DMA para manejar las tareas de carga y descarga de datos desde una computadora.

El protocolo Ethernet escogido para la implementación es comparado luego contra USB v2 y PCIe gen2 para determinar cuál protocolo es ideal para proponer una interfaz de comunicación capaz de manejar simulaciones neuronales en línea.

Palabras clave: soft IP core, Microblaze, USB v2, PCIe.

Abstract

This report expose the design and implementation process of a Microblaze based system to host a neural network modelling soft IP core developed using Vivado HLS designed by Erasmus Medical Center.

At first instance the module did not have hardware to drive the UART and Ethernet communication ports. This was solved, designing and implementing a Microblaze host system that integrates Ethernet, UART and DMA cores for loading and downloading data from a host computer.

The implemented ethernet interface is then compared against USB v2 and PCIe gen2 to determine which protocol should be used in an inline simulation capable system interface.

Keywords: soft IP core, Microblaze, USB v2, PCIe

A mi querida familia

Agradecimientos

A mi familia que siempre está en las buenas y en las malas. Por siempre darme las fuerzas para seguir.

Al Ing. Alfonso Chacón por la paciencia y la guía brindadas.

A Christos Strydis y Georgios Smaragdos por creer que este proyecto podía ser llevado a buen puerto.

Marco Vinicio Acuña Vargas

Índice General

Índice de figuras	iv
Índice de tablas	vi
Lista de abreviaciones	vii
1 Introducción	1
2 Meta y Objetivos	4
2.1 Meta.....	4
2.2 Objetivos.....	4
3 Procedimiento Metodológico.....	5
3.1 Estudio del arte	5
3.2 Aprendizaje de las herramientas	5
3.3 Propuesta inicial de diseño.....	5
3.4 Traslado del núcleo HLS al sistema propuesto.....	7
3.5 Evaluación del sistema prototipo.....	7
3.6 Propuesta y desarrollo de interfaces de comunicación.....	7
4 Estado del arte	8
4.1 Plataforma embebida Microblaze para soporte de un núcleo de simulación de redes neuronales	8
4.1.1 Sistemas en chip sobre FPGA.....	8
4.1.2 Sistemas en chip basados en procesador Microblaze.....	8
4.1.2.1 Controlador de memoria.....	9
4.1.2.2 Protocolo AXI para comunicación entre IP cores en ambiente Microblaze	10
4.1.2.3 Infraestructura de interconexión AXI	13
4.1.3 Consumo de área de IP cores	14
4.2 Herramientas de software de Xilinx para desarrollo de SoC.....	15
4.2.1.1 Vivado HLS.....	15

4.2.1.2	Vivado IDE	16
4.2.1.3	Kit de desarrollo de software de Xilinx	17
4.3	Protocolos de comunicación PCIe, USB y Ethernet	18
4.3.1	Capas de encapsulamiento de datos.....	18
4.3.2	Paquetes de datos.....	19
4.4	Núcleo de procesamiento neuronal	21
4.4.1	Arquitectura del núcleo de procesamiento neuronal.....	21
4.4.2	Representación a nivel de bloque del NSN	22
4.4.3	Funcionamiento del núcleo de procesamiento neuronal.....	23
5	Adaptación y evaluación del núcleo de simulación neuronal ..	27
5.1	Reducción	27
5.2	Evaluación.....	31
5.3	Modificaciones recomendadas para el NSN	34
6	Desarrollo de un sistema embebido para soportar el núcleo de simulación	37
6.1	Estrategia de diseño y propuesta	37
6.2	Análisis de requerimientos y restricciones.....	38
6.3	Detalle de la arquitectura de solución.....	39
6.3.1	Implementación de buses de datos	40
6.3.2	El procesador	42
6.3.3	Interfaz con la memoria	43
6.3.4	Interfaz de acceso directo a memoria del núcleo de simulación neuronal	45
6.3.5	Dispositivos de E/S.....	48
6.4	Evaluación del diseño y propuesta.....	51
6.4.1	Evaluación del acceso a memoria.....	52
6.4.2	Evaluación dispositivos de Entrada/Salida.....	53
6.4.2.1	Evaluación AXI UARLITE.....	53
6.4.2.2	Evaluación AXI ETHERNET LITE	53
6.4.3	Recomendaciones para la interfaz Ethernet.....	56
6.4.4	Evaluación de una simulación en el NSN.....	57
6.4.5	Propuesta para interfaz mejorada del NSN	62
7	Desarrollo de las interfaces de comunicación	63
7.1	Propuesta de red que integre varios NSN.....	64
7.2	Análisis de las especificaciones de los protocolos Ethernet, USB v2 y PCIe....	65
7.2.1	Arquitectura a nivel de sistema	66
7.2.2	Estructura de control.....	67
7.2.3	Tasa de transferencia de datos.....	68

7.2.4	Paquetes y eficiencia de los datos	69
7.3	Análisis de IP cores disponibles para implementación	71
7.4	Recomendaciones para interfaz de comunicación	73
8	Conclusiones.....	75
9	Recomendaciones	76
9.1	Optimización del sistema anfitrión y el NSN	76
9.1.1	Interfaces de comunicación de alta velocidad del sistema anfitrión.....	76
9.1.2	Arquitectura de control e interfaces del NSN	77
9.2	Metodología de aprendizaje en las herramientas.....	77
9.2.1	Documentación.....	77
9.2.2	Tarjeta de desarrollo	77
10	Bibliografía.....	78
11	Anexos.....	81
11.1	Tabla de configuraciones físicas del ip core controlador de memoria	81

Índice de Figuras

iv

Figura 1.1. Núcleo de simulación de redes neuronales de espigas.....	2
Figura 3.1. Diagrama de propuesta inicial de diseño.....	6
Figura 4.1. Arquitectura Xilinx 7 series memory interface solutions core.....	9
Figura 4.2. Estructura de una transacción de lectura AXI4.....	11
Figura 4.3. Estructura de una transacción de escritura AXI4.....	12
Figura 4.4. Proceso de transferencia de datos en interfaz AXI Stream.....	13
Figura 4.5. Diagrama de bloques internos de AXI Interconnect.....	13
Figura 4.6. Flujo de diseño para desarrollo de SoC.....	15
Figura 4.7. Entradas y productos de la función Export RTL.....	16
Figura 4.8. Estructura lógica de la aplicación de software.....	17
Figura 4.9. Estructura de capas PCIe y USB.....	18
Figura 4.10. Estructura de capas de protocolo 802.3.....	19
Figura 4.11. Estructura de paquetes de datos de PCIe, USB y IEEE 802.3.....	20
Figura 4.12. Organización lógica del NSN.....	22
Figura 4.13. Representación de nivel de bloque de NSN.....	22
Figura 4.14. Diagrama de eventos en el NSN.....	24
Figura 4.15. Estructura del arreglo inicialización.....	25
Figura 4.16. Estructura de la información de estado de las neuronas.....	25
Figura 4.17. Arreglo de valores de estímulo.....	25
Figura 4.18. Arreglos de voltaje dendrítico.....	26
Figura 5.1. Arquitectura de versión reducida del NSN.....	29
Figura 5.2. Voltaje de Axón de salida del NSN (reducido).....	30
Figura 5.3. Comparación de los voltajes de axón de la primera neurona de la versión original y reducida del NSN.....	30
Figura 5.4. Diagrama de tiempo de la operación del NSN.....	32
Figura 5.5. Modificaciones a nivel de bloque.....	35
Figura 5.6. Modificación propuesta para fases de simulación.....	36
Figura.6.1. Diagrama general de arquitectura implementada.....	40
Figura 6.2. Detalle de interfaces conectadas a bus AXI4 Lite.....	41
Figura 6.3. Detalle de interfaces conectadas a bus AXI4 Full.....	41
Figura 6.4. Sistema básico con procesador Microblaze.....	42
Figura 6.5 Representación de nivel de bloque de módulo DMA de lectura.....	45
Figura 6.6. Representación de nivel de bloque de módulo DMA de escritura.....	45
Figura 6.7. Interconexión de los módulos DMA con el NSN.....	46
Figura 6.8. Rutina de inicialización general.....	47
Figura 6.9. Mapa de memoria para los buffer de DMA.....	48
Figura 6.10. Representación de nivel de bloque del AXI UART Lite en el IPI.....	48

Figura 6.11. Interconexión de Ethernet PHY con FPGA.....	49
Figura 6.12. Diagrama de bloque de chip LAN 8720.....	49
Figura 6.13. Representación a nivel de bloque del AXI Ethernet Lite	50
Figura 6.14. Conexión de acoplador con AXI Ethernet Lite.....	50
Figura 6.15. Estrategia de prueba utilizada durante evaluación.....	52
Figura 6.16. Resultado de la prueba de memoria.....	53
Figura 6.17. Topología de prueba utilizada en la evaluación de la interfaz.....	54
Figura 6.18. Conexión de Integrated Logic Analyzer	57
Figura 6.19. Diagrama de flujo de programa para experimento	59
Figura 6.20. Resultado de simulación en FPGA.....	60
Figura 6.21. Resultados de simulación neuronal con precisión simple.....	60
Figura 6.22. Porcentaje de error entre implementación de hardware y versión C original del NSN.....	61
Figura 7.1. Red de multiples NSNs	65
Figura 7.2. Arquitecturas de sistemas PCIe y USB	67
Figura 7.3. Eficiencia de datos PCIe, USB y Ethernet	70

Índice de tablas

vi

Tabla 4.1. Utilización típica de protocolos AXI.....	10
Tabla 4.2. Descripción de los canales de comunicación de protocolos AXI	10
Tabla 4.3. Tipos de implementación de bus AXI	12
Tabla 4.4. Recursos de las FPGA de la serie 7 de Xilinx	14
Tabla 4.5. Descripción de señales de puerto ap_ctrl	23
Tabla 5.1. Resultados del proceso de síntesis de la versión original del NSN.....	27
Tabla 5.2. Consumo de una instancia de un multiplexor de tiempo	28
Tabla 5.3. Resultados del proceso de síntesis de la versión reducida del NSN.....	31
Tabla 6.1. Mapa de memoria del procesador MICROBLAZE del SoC anfitrión	43
Tabla 6.2. Soluciones recomendadas por Digilent para interfaz con memoria	43
Tabla 6.3. Parámetros de configuración de la interfaz AXI esclava del controlador de memoria	44
Tabla 6.4. Comandos de control para consola XMD	52
Tabla 6.5. Parámetros de configuración para prueba con interfaz Ethernet	54
Tabla 7.1. Tipos de interfaces y opciones de conectividad para tarjeta VC707.....	64
Tabla 7.2. Velocidad de transferencia de datos teóricos de PCIe, USB y Ethernet	68
Tabla 7.3. Campos tomados en cuenta para cálculo de eficiencia	70
Tabla 7.4. Utilización de recursos de área para IP cores de comunicación	72
Tabla 7.5. Ventajas de los protocolos seleccionados en los parámetros revisados	73

Lista de abreviaciones

E/S	Entrada/Salida
NSN	Núcleo de simulación neuronal
SoC	Del inglés System On Chip
IPI	Del inglés Intellectual Property Integrator
DDR SDRAM	Del inglés Double Data Rate Synchronous Dynamic Random Access Memory
FPGA	Del inglés Field programmable Gate Array
FIFO	Del inglés First In First Out

1 Introducción

La simulación del cerebro humano ha sido catalogada por la academia de ingenieros de los Estados Unidos como un gran reto de la ingeniería, ya que esta permite potenciar en gran medida el entendimiento que se tiene del comportamiento del cerebro. Además, la simulación en tiempo real del mismo significaría la capacidad de producir dispositivos que puedan ser utilizados en aplicaciones médicas [1], así como también se podrían llegar a producir prótesis e implantes para restaurar funciones cerebrales en diversos tipos de pacientes.

Los grandes avances en la neurociencia han conducido a un mejor entendimiento del cerebro, lo cual a su vez ha dado paso a la creación de modelos matemáticos que representan el cerebro con gran nivel de detalle. Las redes neuronales de espigas son un ejemplo de este tipo de modelos matemáticos, que aunados a los avances en ciencias de la computación han hecho posible implementar redes neuronales de mayor tamaño para simular el comportamiento del cerebro.

El objeto de estudio de la investigación a la cual este proyecto contribuyó es el núcleo inferior olivar del cerebelo. El equipo de ingenieros de Erasmus Brain Project decidió utilizar el modelo matemático Hodgkin-Huxley para modelar y simular dicho sistema a partir de redes neuronales de espigas. Este permite generar simulaciones que comprenden el estado general del sistema y los procesos biofísicos que suceden dentro de cada una de las neuronas. El cálculo de los pasos de una red neuronal generada a partir de este modelo es muy exigente. Izhikevich señala en [2] que el modelo Hodgkin-Huxley es de todos los modelos para redes neuronales de espigas, el que necesita mayor capacidad de cómputo. Esto se debe a que el estado futuro de cada una de las neuronas de la red depende del estado presente de todas las demás neuronas del sistema. Esto significa que el número de operaciones de punto flotante requerido para el cálculo del estado futuro de la red aumenta de forma exponencial con respecto al número de neuronas de la misma. Esto provoca que el tiempo de ejecución de la red neuronal también aumente con la misma tendencia cuando la red neuronal se implementa en una arquitectura de computación Von Neumann.

Es importante destacar que el sentido de contar con una simulación implementada por medio de un modelo con el grado de detalle que tiene el Hodgkin-Huxley es poder simular la mayor cantidad de neuronas posible en tiempo real [1]. La razón para esto es que idealmente las señales producidas en la simulación puedan ser conectadas con tejido vivo.

El equipo de Erasmus Brain Project analizó las exigencias de computación y la tendencia de aumento en el tiempo de ejecución con respecto al tamaño de red neuronal (número de neuronas) en una arquitectura Von Neumann. Este análisis determinó que

la mejor forma de construir un sistema que se acoplara a las necesidades de la aplicación era utilizar la tecnología FPGA.

El diagrama general del sistema de aceleración diseñado se puede observar en la figura 1.1. En este diagrama se observa la presencia de tres módulos principales, la red de módulos de procesamiento de neuronas (IO network), la BRAM del chip FPGA (Block Ram) y el control general. Los cálculos para la simulación de la red neuronal se llevan a cabo en paralelo en la red de módulos de procesamiento de neuronas la cual consta actualmente de ocho módulos idénticos. La BRAM del sistema es utilizada para el almacenamiento del estímulo de entrada al sistema y del estado de las neuronas. El control del flujo de ejecución de los módulos de procesamiento neuronal es llevado a cabo por el control general del sistema.

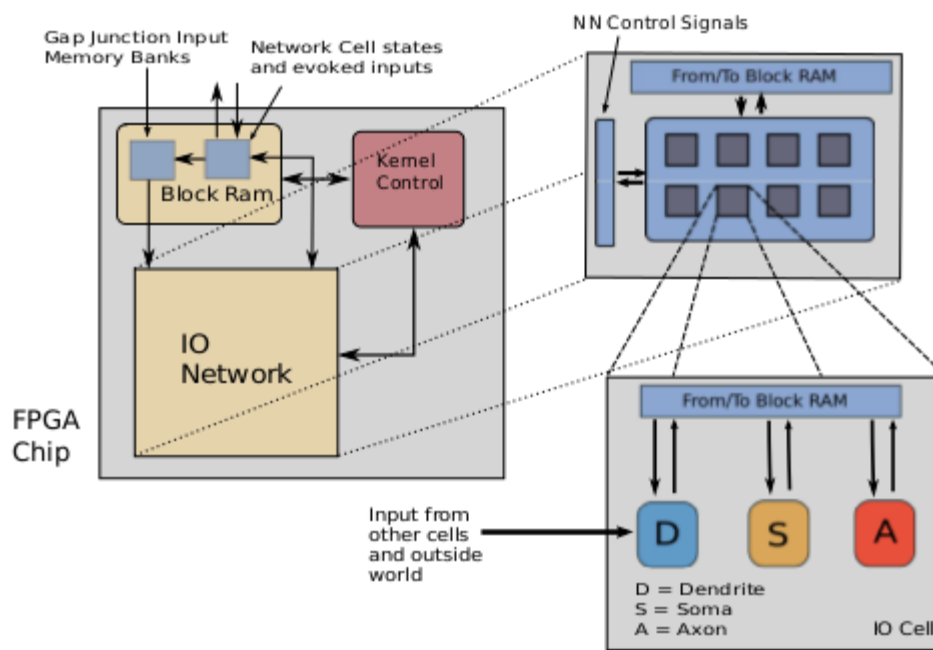


Figura 1.1. Núcleo de simulación de redes neuronales de espigas [1]

El sistema que se desarrolló fue validado por el equipo con la utilización de Questasim 10.1 y de esta forma se determinaron las ventajas que representaba este acelerador con respecto a un procesador convencional.

El núcleo de simulación sin embargo nunca fue probado en físico en una FPGA debido a que carecía de un sistema anfitrión que controlara el flujo de datos hacia sus interfaces y que además lo dotara de un medio de comunicación con una computadora anfitriona para que pudiera funcionar como un acelerador. Esto último, era necesario ya que los protocolos presentes en las interfaces del núcleo de simulación son de tipo AXI (Advanced Extensible Interface), el cual se utiliza para comunicaciones internas en un sistema en chip pero nunca para comunicaciones de chip a chip o de sistema a sistema, por lo cual no había un medio para introducir o sacar datos del mismo funcionando en un chip FPGA.

Como una analogía, el núcleo de simulación era como un cerebro sin cuerpo. Un cerebro para comunicarse con otro cerebro en otro cuerpo necesita de una boca que proyecte una voz y oídos para escuchar los mensajes que le envían. Así, este proyecto lo que busco fue darle un medio de comunicación al núcleo de simulación.

Este informe detalla el proceso de diseño e implementación de un sistema anfitrión desarrollado para albergar el núcleo de simulación. Este sistema hizo posible descargar el NSN en un chip FPGA, con el fin de poder verificarlo fuera de línea y proponer una interfaz de comunicación para que este pudiera funcionar como un verdadero acelerador de una computadora convencional anfitriona.

2 Meta y Objetivos

2.1 Meta

La razón primordial por la cual se desea extender el sistema con el que se cuenta actualmente es lograr sistemas de simulación cerebral que integren múltiples chips FPGA para poder así multiplicar varias veces el tamaño de las redes neuronales que se pueden simular y así potenciar el desarrollo de la neurociencia.

La aceleración de los procesos de simulación en este tipo de aplicaciones pretende alcanzar dispositivos electrónicos que se comporten de manera similar al tejido vivo y que puedan interactuar con su contraparte biológica para así trabajar en aplicaciones mediante las cuales sería posible hacer implantes a personas que han perdido la funcionalidad en diversas zonas del cerebro.

2.2 Objetivos

Implementar en un chip FPGA una versión completa del núcleo de simulación de hardware de la aplicación de modelado del olivo inferior dentro de un sistema empotrado basado en un procesador de núcleo suave.

Validar exhaustivamente la respuesta fuera de línea a estímulos vectoriales de una red neuronal del olivo inferior corriendo en un sistema anfitrión basado en FPGA.

Definir las características óptimas de la interfaz de comunicación y del diseño de las interfaces de datos NSN de modelado del núcleo inferior olivar para así maximizar la capacidad de procesamiento y transferencia de datos.

3 Procedimiento Metodológico

3.1 Estudio del arte

El desarrollo de este proyecto involucró el estudio de diversos temas relacionados con la implementación de sistemas embebidos en chips FPGA. En relación con este ámbito se revisó bibliografía relacionada con el diseño de sistemas con procesadores embebidos. Esto se complementó estudiando el proceso de diseño de sistemas a partir de soft IP cores. Lo cual, comprendió investigar los tipos de interfaces y protocolos presentes en las comunicaciones dentro de sistemas en chip.

Dado que los soft IP cores del catálogo del fabricante Xilinx se consideraron desde un principio como parte de la solución, estos se estudiaron mediante la utilización de sus respectivas hojas de datos, prestándose especial atención al consumo de recursos, a sus interfaces y a las recomendaciones de diseño provistas por el fabricante. Por último, se investigó el desempeño y escalabilidad de los protocolos USB, PCIe y Ethernet para que así fuera posible seleccionar una interfaz ideal para el NSN.

3.2 Aprendizaje de las herramientas

Las herramientas de software que se utilizaron en el transcurso de este proyecto fueron Vivado IDE, Vivado HLS y Xilinx SDK. Estos programas forman parte de la suite de diseño de Xilinx y cada uno tiene un propósito distinto en el desarrollo de sistemas en chip sobre FPGA. Al comienzo de este proyecto, el conocimiento sobre estas herramientas de software era muy limitado. Por esta razón se recabaron fuentes de información relacionadas a sus capacidades y limitaciones. Adicionalmente, se consultaron fuentes multimedia que sirvieran en el proceso de aprendizaje en el uso de las mismas.

3.3 Propuesta inicial de diseño

Los requerimientos de diseño solicitados por el equipo de Erasmus Brain Project se centraron primordialmente en dotar al núcleo de simulación de una interfaz de datos y de acceso a un volumen de memoria externo al chip FPGA. La implementación de una interfaz de datos tenía como objetivo que el núcleo de simulación pudiera consumir y exportar datos.

Idealmente se buscaba que dicha interfaz fuera compatible con algún protocolo de comunicación estándar. La implementación del acceso a memoria, se buscaba para poder registrar los resultados de simulaciones y potencialmente poder aumentar el

tamaño de la red que el núcleo era capaz de simular.

Es claro que estos requerimientos tenían que ser resueltos agregando módulos adicionales al núcleo de simulación. Sin embargo, debido a que este se encuentra en constante desarrollo, es esperable que se agreguen capacidades de simulación adicionales en el futuro. Estas modificaciones podrían implicar cambios en la estructura de control y manejo de los datos interno del núcleo.

Por esta razón era muy importante que la forma en la que se fueran a conectar todos los módulos del sistema permitiera encapsular funcionalidades. Esto quiere decir que el núcleo de simulación no tiene que conocer acerca de la lógica que los módulos de comunicación y acceso a memoria utilizan para acceder a los medios físicos y viceversa. Esta estrategia incluso permitiría hacer intercambiables los módulos para acceder a los medios de comunicación y de memoria.

Con estas consideraciones se planteó el sistema que se muestra en la figura 3.1. En esta se puede observar que se incorporó un procesador para actuar como control general del sistema. Además, se puede ver que en primera instancia la comunicación se propuso con Ethernet y que la comunicación lógica entre este módulo de comunicación y el núcleo de simulación se diera por medio de la memoria. Es importante destacar que para esta propuesta se tuvo como premisa utilizar los soft IP cores disponibles en el catálogo de Vivado IDE.

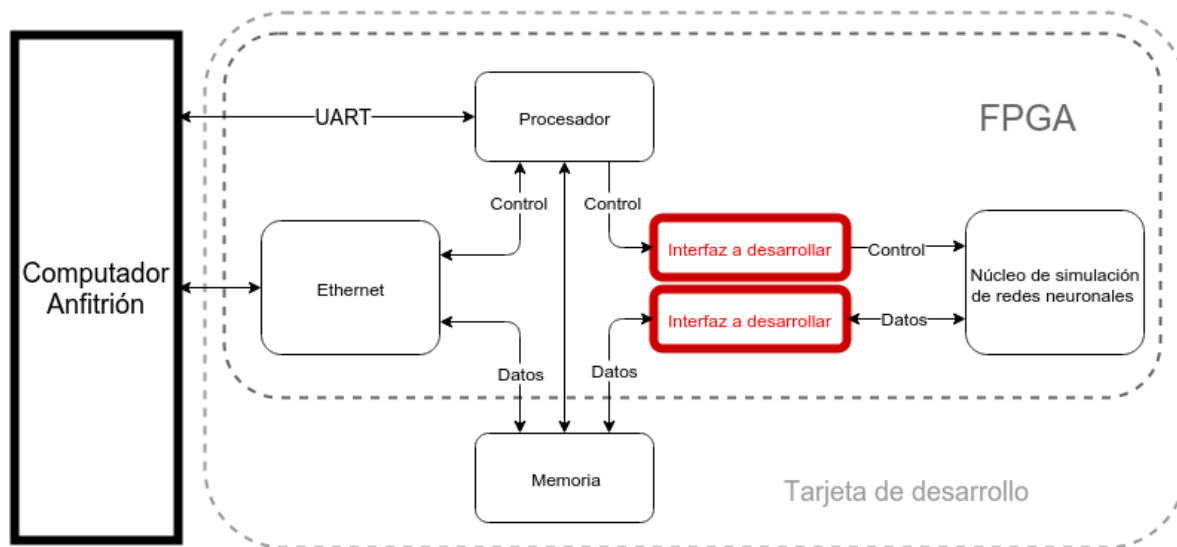


Figura 3.1. Diagrama de propuesta inicial de diseño

En la figura [3.1], también se muestra un puerto de comunicación UART entre el computador anfitrión y el procesador del sistema en la FPGA. Esta interfaz de datos se agregó para depuración del software del sistema embebido.

3.4 Traslado del núcleo HLS al sistema propuesto

El núcleo de simulación fue originalmente implementado utilizando la herramienta Vivado HLS, de esta forma se obtuvo un soft IP core a partir de un programa en lenguaje C. El modulo que fue entregado por parte de Erasmus Brain project consumía en principio muchos recursos para poder implementarse en la FPGA disponible y además sus puertos de control no cumplían con un protocolo de comunicación estándar para sistemas en chip. Por lo cual fue necesario utilizar Vivado HLS para modificar el núcleo con el fin de reducir su consumo de recursos y de que sus puertos de control fueran compatibles con el protocolo de comunicación que se utilizaría en el resto de las comunicaciones a lo interno del SoC.

3.5 Evaluación del sistema prototipo

La evaluación del sistema consistió en primera instancia comprobar el funcionamiento correcto de las interfaces de comunicación y la memoria. Luego, se verificó que el núcleo de simulación produjera los mismos datos de salida que el algoritmo original en lenguaje C ejecutándose en una computadora para los mismos parámetros de inicialización y de estímulo.

3.6 Propuesta y desarrollo de interfaces de comunicación

La interfaz de comunicación que se escogió para ser parte del sistema fue una interfaz de Ethernet. La misma se escogió por razones relacionadas a la posibilidad de implementarla de forma práctica, ya que para otras opciones de interfaz no se contaba con el hardware necesario para su implementación.

En este proyecto se realizó también una investigación bibliográfica para proponer una interfaz de comunicación ideal para el núcleo de simulación neuronal. Las interfaces que se analizaron en dicha investigación son PCIe, USB y Ethernet. Los criterios que se escogieron para el análisis fueron velocidad de transporte de datos y escalabilidad.

En el caso de este proyecto la escalabilidad significa cuantos sistemas como el desarrollado se pueden conectar en paralelo, ya que esto es algo que se buscaría hacer en un futuro para construir sistemas con más capacidad computacional. Por lo tanto también se investigó cómo se comporta la utilización del ancho de banda para control en las 3 interfaces.

4 Estado del arte

4.1 Plataforma embebida Microblaze para soporte de un núcleo de simulación de redes neuronales

4.1.1 Sistemas en chip sobre FPGA

Los circuitos integrados han sido desarrollados hasta un nivel tal de densidad de transistores que en la actualidad es posible encapsular sistemas que cuentan con procesadores, memorias y dispositivos E/S en un solo chip. Estos sistemas son conocidos como sistemas en chip o SoC (system on chip) y tienen grandes ventajas en cuanto a costo de fabricación, consumo de potencia y flexibilidad de diseño [3]. La flexibilidad de diseño significa que estos pueden integrar diversos protocolos de comunicación, así como también diversos tipos de procesamiento.

El desarrollo de un SoC comprende el diseño de hardware y software. Debido a la complejidad inherente al proceso de integración de una gran cantidad de transistores en un solo chip, es muy frecuente que el desarrollo del hardware consuma la mayor cantidad de la inversión de tiempo y recursos. Los cortos plazos de desarrollo del mercado actual requieren de la reutilización de piezas de diseño y su portabilidad. Por esta razón, surgen herramientas para desarrollo de SoCs en FPGA por medio de la utilización de soft IP cores . Los cuales se distribuyen como módulos sintetizables descritos en lenguajes HDL [9].

Estas herramientas aprovechan el hecho de que los soft IP cores se pueden adquirir como módulos funcionales completamente verificados, los cuales pueden ser utilizados por los ingenieros a nivel de bloque para reducir el tiempo de desarrollo de un sistema. Esta metodología también permite concentrar el esfuerzo de diseño en la arquitectura del sistema en chip, lo que resulta en un mayor valor agregado para el producto final.

La compañía líder en cuanto a chips y herramientas de desarrollo para FPGA es Xilinx Inc, la cual ofrece un paquete de software denominado Vivado Design Suite el cual tiene un flujo de diseño especializado para desarrollo de SoCs en FPGA. En este paquete de software se ofrece bajo licencia un catálogo de soft IP cores, en el que se encuentran todo tipo de soluciones de procesamiento, memoria e interfaces de entrada/salida.

4.1.2 Sistemas en chip basados en procesador Microblaze

La utilización de procesadores en los SoC es una constante, ya que estos requieren de software para ejecutar las tareas de procesamiento de datos para las que son diseñados. El IPI de Vivado ofrece el procesador Microblaze para cumplir con tal requerimiento. Este procesador tiene una arquitectura RISC, de tipo Harvard y sus registros internos son de 32 bits.

Este procesador se puede configurar de acuerdo a las necesidades que se tengan en la aplicación. La interfaz de usuario del IPI de Vivado permite configurarlo en 4 modos diferentes (Máximo rendimiento, área mínima, máxima frecuencia y configuración típica).

Las interfaces de datos de este procesador son mapeadas a memoria tanto para datos como para las instrucciones. El procesador consta de 2 conjuntos de interfaces denominadas periféricas y de cache. Las interfaces periféricas (datos e instrucciones) están implementadas como maestras de 32 bits que se pueden interconectar con periféricos por medio de inter conectores AXI. Las interfaces de cache se pueden implementar en varios anchos de bus, lo cual permite aumentar la velocidad de las transferencias y le permiten al procesador un acceso a la memoria de mayor rendimiento [11].

4.1.2.1 Controlador de memoria

La solución que ofrece Xilinx para que los IP cores con interfaces AXI4 se conecten con chips memoria externos al FPGA es el Xilinx 7 series memory interface solutions core. Este dispositivo tiene una estructura como la que se muestra en la figura 4.1. En esta se puede ver que este IP core está estructurado en bloques que sirven de interfaz entre el bus AXI4 y el chip de memoria.

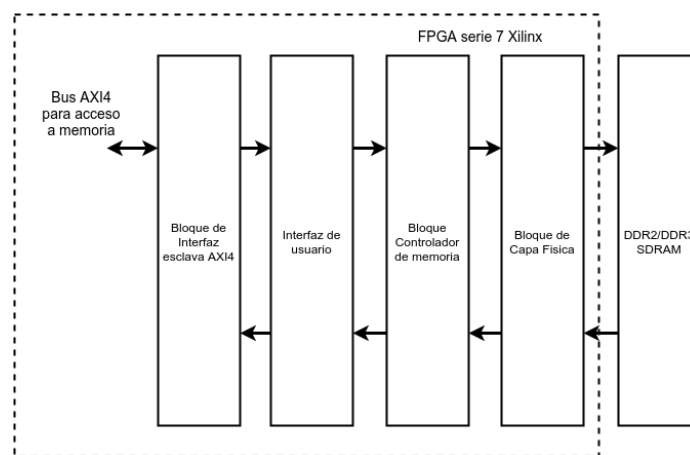


Figura 4.1. Arquitectura Xilinx 7 series memory interface solutions core

Los bloques que conforman esta interfaz se denominan bloque de interfaz esclava AXI4, interfaz de usuario, controlador de memoria y bloque de capa física. Estos bloques

funcionan en cascada para resolver las solicitudes de lectura y escritura que recibe el bloque de interfaz AXI4 Esclava. El bloque que implementa esta interfaz traduce las solicitudes AXI en solicitudes para la interfaz de usuario de la figura 4.1.

La interfaz de usuario presenta un espacio de direcciones uniforme a la lógica del usuario (en este caso dispositivos AXI). Esta, pre almacena la información de lectura y escritura y reordena los datos leídos desde el controlador de memoria que puedan estar en desorden.

El controlador de memoria es el bloque primario del IP core. Este recibe las solicitudes desde la interfaz de usuario y las almacena en fila. Además, esta etapa puede reordenar las solicitudes para maximizar el rendimiento de la memoria.

Por último, se tiene el bloque de capa física, este se encarga de generar la sincronización requerida para conectar el controlador de memoria con el chip DDR SDRAM. Esta capa es utilizada además para inicializar el chip de memoria cuando se enciende el sistema.

4.1.2.2 Protocolo AXI para comunicación entre IP cores en ambiente Microblaze

Los diferentes IP cores que integran el diseño de un SoC deben comunicarse para poder funcionar como un sistema. El protocolo que se utiliza para las comunicaciones dentro de chip en sistemas basados en procesador Microblaze se denomina AXI4. Este, se deriva de la especificación AMBA y tiene 3 variaciones llamadas AXI4 Full, AXI4 Lite y AXI4 Stream. En la tabla 4.1 se presenta la utilización típica de cada una de las variaciones del protocolo según [12].

Tabla 4.1. Utilización típica de protocolos AXI

Protocolo	Utilización típica
AXI4	Comunicaciones mapeadas a memoria que requieran de alto desempeño.
AXI4 Lite	Enlaces mapeados a memoria que no requieran de alto rendimiento, ya que no permite ráfagas de datos.
AXI4 Stream	Comunicaciones de alta velocidad y que no requieran mapeo a memoria

Los protocolos AXI4 y AXI4 Lite son mapeados a memoria, bidireccionales y comparten la misma estructura de canales de comunicación. El protocolo AXI Stream no es mapeado a memoria y es unidireccional por lo que presenta una interfaz que difiere de los primeros, ya que las señales para acceso a memoria no son necesarias en su implementación. La descripción de los canales de estos protocolos se presenta en la tabla [4.2].

Tabla 4.2. Descripción de los canales de comunicación de protocolos AXI [12]

Protocolo	canales	Descripción
AXI4 y AXI4 lite	Read address	Dirección que el maestro va a leer del esclavo.
	Read data	Datos leídos del esclavo.
	Write address	Dirección del esclavo en la que el maestro va a escribir.
	Write data	Datos a escribir en el esclavo
	Write response	Respuesta de reconocimiento de escritura.
AXI4 Stream	Write data	Escribe datos desde una interfaz maestra en una interfaz esclava. Las señales que están presentes en este canal se denominan TDATA, TVALID y TREADY.

Estos protocolos están estructurados en transacciones, las cuales se componen de transferencias. Una transferencia en el contexto AXI4 significa el muestreo de un dato del mismo tamaño del bus de datos en un ciclo de reloj [12].

La forma en que se dan la lectura y escritura de datos mediante el protocolo AXI4 se detalla en las figura 4.2 y 4.3 respectivamente. En la figura 4.2 se observa cómo utilizando un solo ciclo de lectura (una transacción), es posible leer varias palabras de datos (varias transferencias) del esclavo. Esta capacidad del protocolo AXI4 se conoce como ráfagas de datos y es precisamente la principal diferencia que presenta con respecto a su versión Lite (no cuenta con ráfagas de datos). Esta capacidad también se muestra en la figura 4.3 en el cual se escriben varias palabras por transacción en el esclavo con una sola dirección de memoria.

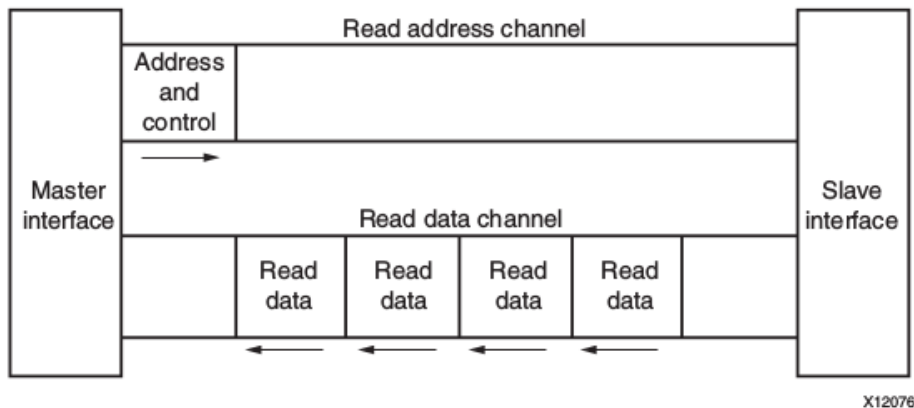


Figura 4.2. Estructura de una transacción de lectura AXI4 [12]

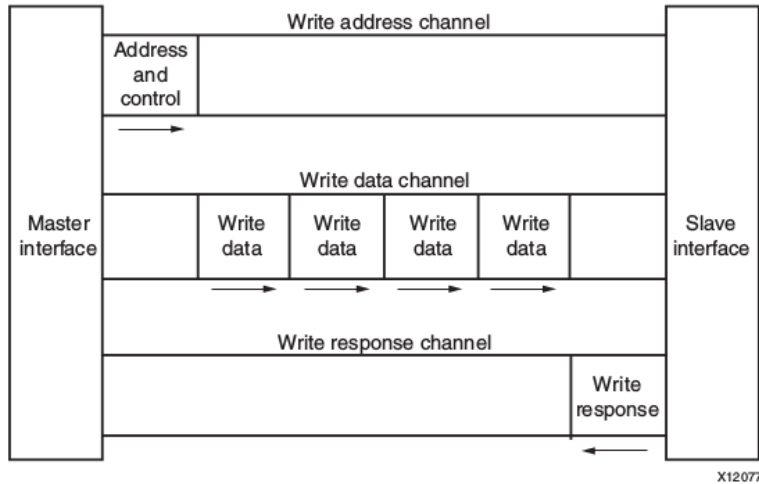


Figura 4.3. Estructura de una transacción de escritura AXI4

Ambas figuras (4.2 y 4.3) ilustran también el comportamiento del protocolo AXI4 Lite con la salvedad de que este transfiere solo un dato por transacción. En estas se muestra que los enlaces AXI4 y AXI4 Lite son full duplex dado que la información puede viajar en ambos sentidos simultáneamente.

Cuando estos protocolos funcionan en un bus con varios maestros y esclavos, la separación de canales de los mismos brinda amplia flexibilidad en cuanto a alternativas de interconexión [4]. En la tabla 4.3, se muestran las opciones que están disponibles para la implementación de un bus de tipo AXI4.

Tabla 4.3. Tipos de implementación de bus AXI [4]

Tipo de implementación de bus	Descripción
Bus de dirección y datos compartidos (SASD)	Un bus de dirección compartido se acopla con un bus de datos bidireccional que maneja escritura y lectura.
Bus de dirección compartido y múltiples buses de datos (SAMMD)	Un bus de dirección compartido se acopla con buses de lectura y escritura separados.
Múltiples buses de dirección y múltiples buses de datos (MAMMD)	Un bus de dirección separado para lecturas y escrituras es acoplado con un bus de datos separado para lectura y escritura.

En el protocolo AXI4-Stream se tiene un comportamiento como el que se observa en la figura 4.4. En esta se muestran las 3 señales básicas del protocolo. La señal que aparece en la figura como "INFORMATION" es también denominada TDATA. Las señales de la figura funcionan por hand shake lo cual significa que cuando el maestro coloca un dato en el bus y acierta la señal TVALID (indicando la validez de los datos) este tiene que

mantener el valor en el bus constante hasta que el esclavo acierte la señal TREADY y llegue un flanco positivo de reloj.

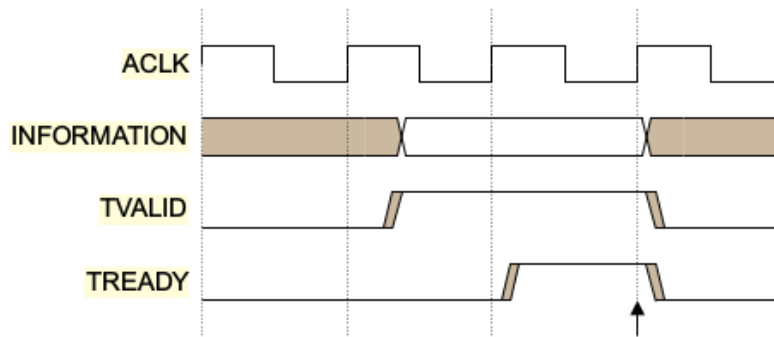


Figura 4.4. Proceso de transferencia de datos en interfaz AXI Stream [13]

4.1.2.3 Infraestructura de interconexión AXI

La forma de interconexión de IP cores con mapeo a memoria que se utiliza dentro de un sistema de tipo AXI4 es el IP core AXI Interconnect. En este dispositivo se genera la lógica de decodificación para implementar el mapa de memoria de un sistema basado en procesador Microblaze. Este mapa se crea utilizando el editor de direcciones de Vivado IDE y tiene las direcciones de las interfaces AXI esclavas de los periféricos conectados al dispositivo.

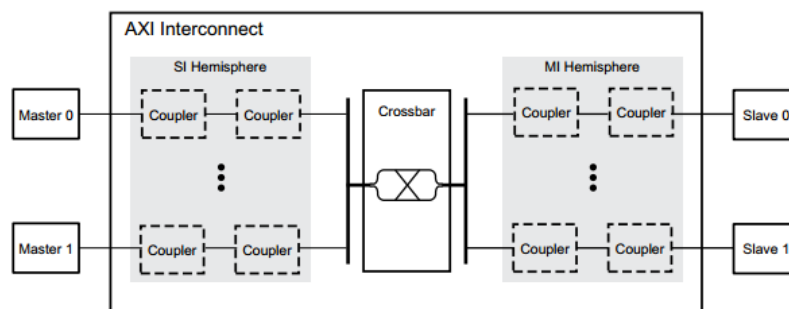


Figura 4.5. Diagrama de bloques internos de AXI Interconnect [24]

En la figura 4.5 se observa que en el núcleo de este IP core se encuentra una unidad denominada AXI Crossbar que es la encargada de realizar la función de enrutamiento de datos y de arbitraje. La presencia de esta unidad en el AXI Interconnect hace posible que un maestro pueda leer o escribir datos en cualquier esclavo conectado al bus. Esta se puede implementar en los modos SASD o SAMD (descritos en la tabla 4.3) que permiten optimizar en el diseño de una arquitectura el desempeño en transferencia de datos y área respectivamente.

En la figura 4.5 se puede observar que entre la unidad AXI Crossbar y las interfaces maestras y esclavas del IP core existen módulos de acoplamiento que están presentes para realizar las distintas funciones. Entre las funciones que se realizan en estos módulos de acoplamiento se encuentra la conversión de anchos de palabra para los buses de datos de maestros o esclavos que tengan anchos de palabra distintos. Este también permite la conversión de relojes para dispositivos AXI funcionando en dominios de reloj distintos y cumple funciones de almacenamiento temporal (función de buffer) para los distintos maestro o buses en el bus.

4.1.3 Consumo de área de IP cores

La estimación de utilización de área de FPGA es importante en las primeras etapas de desarrollo del diseño de un SoC, ya que este cuenta con la restricción del tamaño del chip FPGA en el cual se va a implementar.

La caracterización de un soft ip core es difícil de realizar debido a que el área que este ocupa en el chip, así como otros parámetros, dependen del resultado del proceso de síntesis, el dispositivo físico de implementación, entre otros factores. Por esta razón, en el caso de los IP cores del catálogo de Xilinx existe una guía de producto en donde se exponen estimaciones de consumo de área y ancho de banda para las configuraciones por defecto del dispositivo.

Las estimaciones que se encuentran en la guía de producto están dadas en términos de los recursos de las FPGA de la serie 7 de Xilinx. La descripción de los recursos disponibles en las FPGA de la serie 7 de Xilinx se lista en la tabla 4.4, en la que se describe la organización interna de los mismos.

Tabla 4.4. Recursos de las FPGA de la serie 7 de Xilinx [10]

Recurso	Organización
CLB, Flip Flops, Look up table (LUT)	Los CLB son las celdas que conforman la celda de la FPGA. Un CLB está compuesto por 2 compartimentos. Un compartimento consta de 4 LUTs, 8 Flip-Flops, lógica aritmética para acarreo y multiplexores. Algunos de estos Flip-Flops se pueden utilizar en modo LATCH. Entre un 25 y un 50% de los compartimentos pueden usar las LUTs como memoria o registros de desplazamiento.
Block Ram (BRAM)	Esta memoria está distribuida en la tela de FPGA y cada unidad tiene 2 puertos independientes, lógica FIFO (First In First Out) integrada configurable, lógica de corrección de errores y 36 Kb de memoria. Estos bloques se pueden separar y utilizar como bloques de 18 Kb de un solo puerto.
DSP	Los bloques DSP tienen un procesador de señal con acumulador de 48 bits y un multiplicador de 18x25 bits.

4.2 Herramientas de software de Xilinx para desarrollo de SoC

El flujo de diseño provisto por Xilinx para el desarrollo de sistemas en chip es tal y como se muestra en la figura 4.6. En esta se observa la función que tiene cada una de las herramientas de la suite de diseño de Xilinx hasta llegar a la implementación en FPGA.

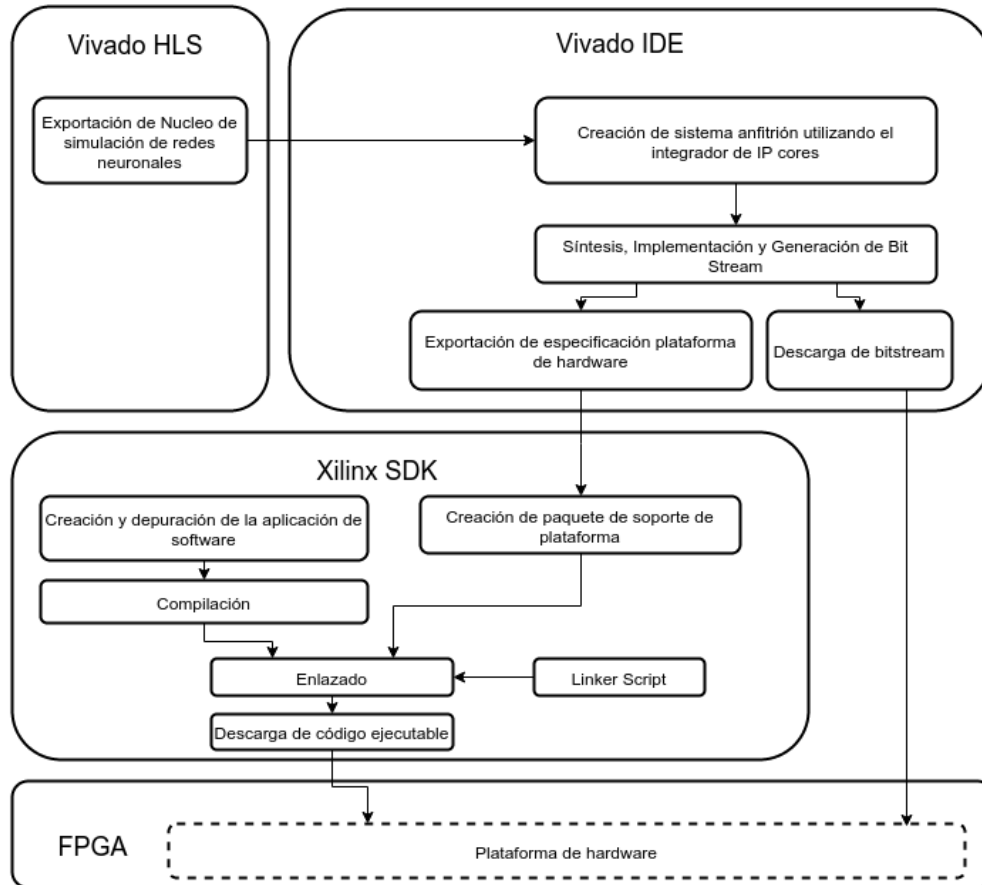


Figura 4.6. Flujo de diseño para desarrollo de SoC

4.2.1.1 Vivado HLS

Vivado HLS (High Level Synthesis), es una herramienta de desarrollo de hardware por medio de lenguajes de alto nivel como C, C++ o SystemC. Este programa toma una función descrita en este tipo de lenguajes y la traduce a un lenguaje HDL (los cuales son sintetizables como hardware). El usuario puede escoger la forma en que la herramienta de síntesis implementa ciertas partes del código mediante la utilización de pragmas o directivas TCL.

Los pragmas y las directivas TCL cumplen la misma función en el flujo de diseño, con la diferencia de que los primeros se agregan directamente en el código fuente y las directivas se incluyen por medio de un archivo [14].

Los diseños realizados en esta herramienta son exportables con la utilización de la función export RTL. El resultado de la exportación de un diseño depende del código fuente (en lenguaje de alto nivel) y de las directivas TCL o pragmas que el usuario especifique. Esta dependencia se ilustra en la figura 4.7.

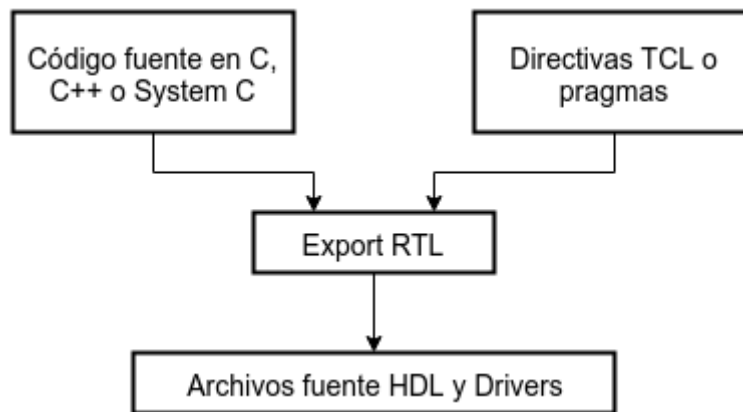


Figura 4.7. Entradas y productos de la función Export RTL

El producto de esta función es un directorio con el código HDL del producto de la síntesis de alto nivel. En caso de que al diseño se hayan agregado interfaces AXI4 esclavas por medio de directivas TCL, la herramienta también genera los archivos fuente en C del driver del IP core para su utilización en un ambiente Microblaze.

4.2.1.2 Vivado IDE

El software Vivado IDE (Integrated development environment) perteneciente a Vivado Design Suite ofrece la herramienta conocida como IP integrator (IPI). El IPI es un ambiente en el cual el usuario instancia y conecta soft IP cores de forma gráfica para armar SoCs que aprovechen las funcionalidades de los mismos. Los IP cores que se ofrecen en el IPI son muchas veces reconfigurables mediante interfaces gráficas para cumplir con las necesidades específicas del usuario, lo cual permite una gran flexibilidad de diseño y un tiempo de desarrollo menor que describir un módulo en un lenguaje HDL desde cero.

En la suite de diseño de Vivado es posible tomar los archivos HDL generados por medio de Vivado HLS e incorporar esos elementos de diseño en las plataformas que se diseñan en el IPI.

En Vivado IDE se realizan las tareas de síntesis, implementación y generación del bitstream para descargar en el chip FPGA. La etapa de implementación puede requerir

de la utilización de un archivo de restricciones XDC. En este archivo (provisto por el fabricante) el usuario puede asignar señales del diseño HDL a los pines físicos del chip.

La generación del bitstream se da seguida de la implementación, el producto de este proceso es una plataforma de hardware que es descargable en un chip FPGA. La plataforma se exporta luego al kit de desarrollo de software, para lo cual la herramienta genera un archivo XML con la descripción de la misma.

4.2.1.3 Kit de desarrollo de software de Xilinx

El kit de desarrollo de software de Xilinx es la herramienta para el desarrollo de aplicaciones de software para las plataformas que se exportan desde Vivado IDE. En la figura 4.6, se observa que luego de la exportación de la plataforma se crea un paquete de soporte. Este es creado por la utilidad libgen a partir de la especificación de la plataforma y de las librerías estándar de C.

Este paquete es una colección de bibliotecas estáticas estándar de C y drivers que forman la capa más baja de la aplicación de software [16]. Entre las bibliotecas estándar que se incluyen están libc.a y libm.a. La primera contiene funciones estándar como stdio y stdlib y la segunda contiene las rutinas para funciones matemáticas [15]. Las aplicaciones que se desarrollan para la plataforma de hardware se enlazan y se ejecutan sobre la interfaz de programación (API) que esta capa provee. La estructura lógica de la aplicación de software se muestra en la figura 4.8.

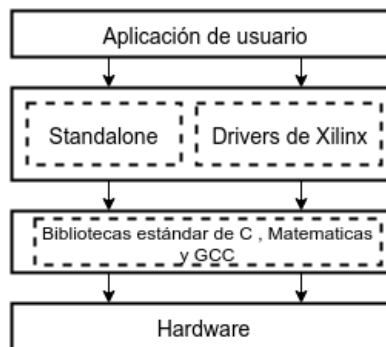


Figura 4.8. Estructura lógica de la aplicación de software

El kit de desarrollo de Xilinx ofrece la posibilidad de generar el paquete de soporte en modo Standalone o Xilkernel. Standalone es un ambiente de un solo hilo de ejecución que provee únicamente las funciones estándar de entrada/salida y de acceso a las características de hardware del procesador [16]. Xilkernel es un modo en el que se crea un sistema operativo ligero.

La aplicación de usuario se programa en lenguaje C o C++ y se puede depurar utilizando la consola XMD o la perspectiva de depuración del SDK. La compilación de esta aplicación se hace específicamente para la arquitectura del procesador Microblaze.

Luego, de que se compila la aplicación se tienen archivos objeto que deben ser enlazados contra las librerías del paquete de soporte de plataforma para generar un archivo ejecutable .elf (executable linkable file)[16]. El diagrama de la construcción del archivo ejecutable se puede observar en el recuadro rotulado como Xilinx SDK en la figura 4.6.

El SDK elimina la tarea de escritura del linker script al proveer al usuario de una interfaz gráfica por medio de la cual el usuario escoge a los espacios de memoria donde residirán las distintas secciones del programa.

4.3 Protocolos de comunicación PCIe, USB y Ethernet

4.3.1 Capas de encapsulamiento de datos

Los protocolos de comunicación se estructuran en capas de acuerdo a los requerimientos del espacio de aplicación para el cual fueron diseñados. Las diferentes capas presentes en un protocolo agregan diferentes tipos de información a la carga útil de un mensaje que se utiliza para enrutamiento, control y manejo de errores, entre otros.

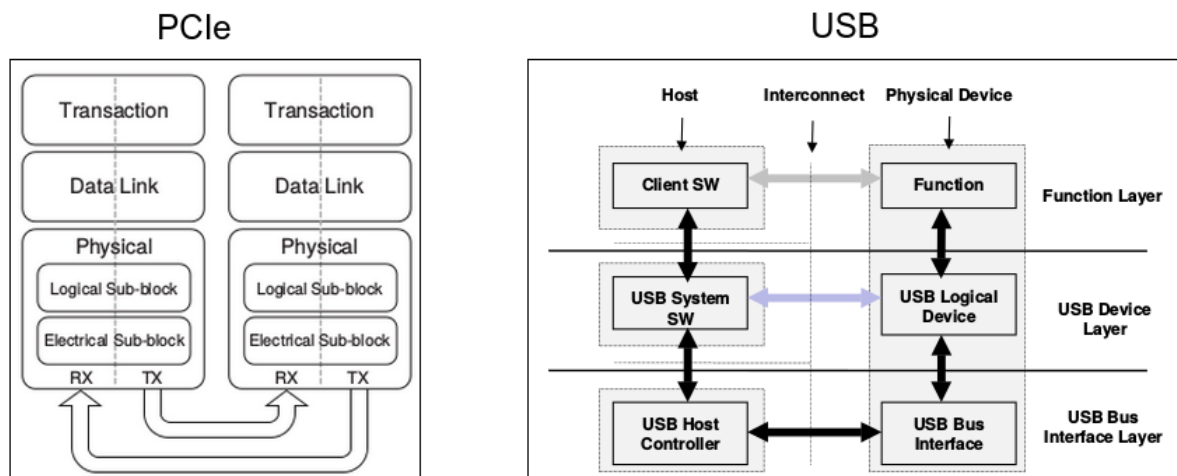


Figura 4.9. Estructura de capas PCIe y USB [18, 19]

En la figura 4.9 se puede ver que las capas presentes en los protocolos PCIe y USB, tienen un gran parecido y ejecutan funciones similares. Como se observa las capas de transacción (“Transaction” en la figura) de PCIe y de función (“Function” en la figura) en USB son análogas y es en estas donde se generan los datos de la carga útil del paquete de datos. La misma condición comparten las capas de enlace de datos (“Data Link” en la figura) de PCIe y la capa del dispositivo USB (“Device Layer” en la figura) las cuales se encargan de generar el CRC (cyclic redundancy check) por medio del cual se garantiza la integridad de los datos [18]. Por último la capa Physical y USB Bus interface Layer

están encargadas de codificar y decodificar los paquetes en señales eléctricas transferibles por el medio físico de conexión.

El protocolo Ethernet cuenta con la particularidad de que es normalmente utilizado en las comunicaciones como una sub capa del modelo OSI (Open system Interconnection). En la figura 4.10 se muestra específicamente el lugar que este ocupa dentro de dicho modelo. Este protocolo segmenta las funciones de la capa de enlace de datos del modelo OSI en las capas denominadas MAC (Media Access Control) y LLC (Logical Link Control). La capa MAC se encarga de agregar a las tramas de datos el campo de dirección de destino y de origen, además de la longitud de los datos. La capa LLC normalmente se utiliza como una interfaz lógica entre la capa MAC y la capa de red del modelo OSI.

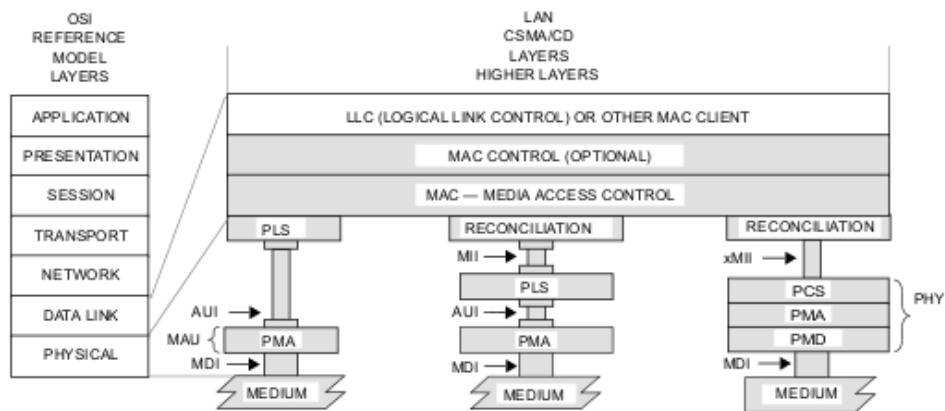


Figura 4.10. Estructura de capas de protocolo 802.3

IEEE 802.3 especifica diversos tipos de interfaz de acceso al medio en su capa física. Esta es la que finalmente se encarga del transporte de los datos mediante señales Eléctricas o en señales de luz en el caso de que el medio sea fibra óptica.

4.3.2 Paquetes de datos

Conforme los datos de un mensaje de una aplicación suben o bajan en la pila de capas de los protocolos, estos son encapsulados por datos que estas agregan para control de flujo del enlace, manejo de errores, enrutamiento, entre otros. Los paquetes de las capas establecidas encima de la capa física de PCIe, USB y Ethernet (IEEE 802.3) de los distintos protocolos se muestran en la figura 4.11.

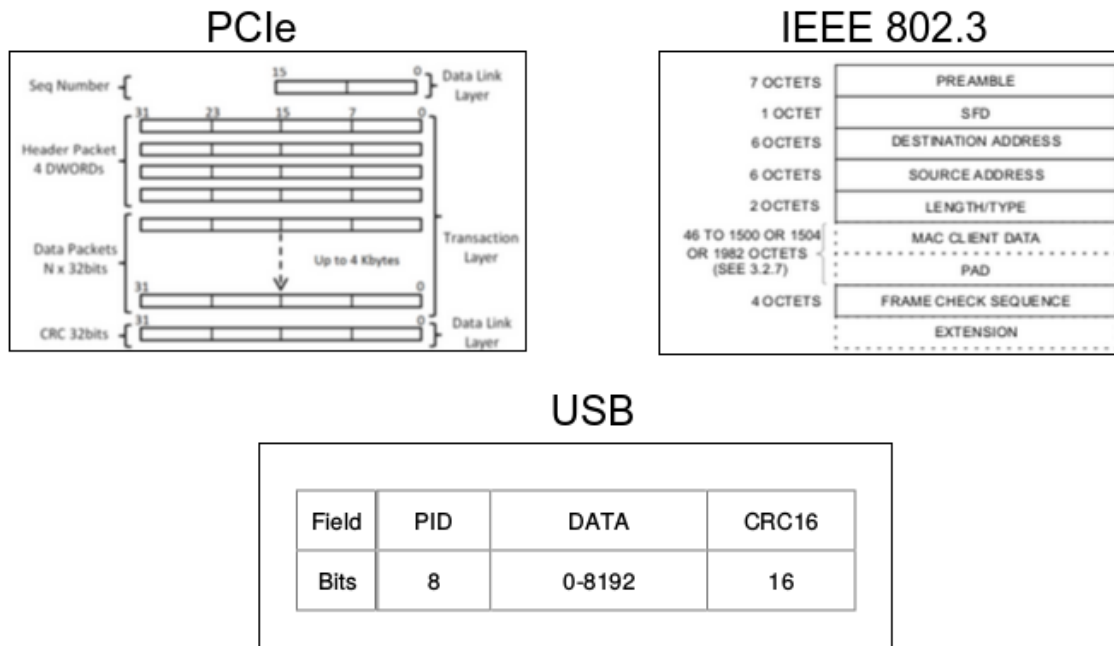


Figura 4.11. Estructura de paquetes de datos de PCIe, USB y IEEE 802.3 [17, 19, 20]

La información que se transporta por un enlace de datos perteneciente a la aplicación de un usuario que funciona sobre un protocolo se denomina carga útil. La fragmentación de tal información depende de la aplicación y de la capa superior del protocolo.

En los paquetes de datos que se muestran en 4.11 se pueden ver campos dedicados a datos. Estos son campos en los cuales no existe ningún tipo de limitación de formato para los datos. Estos son los encargados de contener la carga útil de un mensaje.

El tamaño de carga útil en un protocolo está limitado por la longitud máxima del campo de datos del paquete. Los límites que se encuentran para los paquetes de PCIe, USB y Ethernet (IEEE 802.3) son de 4Kb, 1Kb y 1,5 Kb respectivamente. A la carga útil de un mensaje en estos 3 protocolos se agrega normalmente un campo de chequeo de redundancia cíclica (CRC), el cual se utiliza para detección de errores. Este campo es calculado en el transmisor del mensaje en base a la información del paquete y se adiciona al paquete. El receptor luego calcula un CRC propio basado en el paquete recibido y lo compara contra el CRC que acompañaba al mismo.

Los encabezados de los paquetes de PCIe y Ethernet contienen información de direcciones de origen y destino para enrutamiento de datos. Además, presentan información relacionada con la longitud y formato de los datos en el paquete. Para USB el encabezado del paquete es un campo de 8 bits que corresponde al identificador del paquete y se utiliza para control de flujo.

El paquete que se muestra en la figura 4.11 corresponde con lo especificado en la revisión 2.0 como un paquete de datos sin embargo, es necesario tener en cuenta que junto al paquete se envía un campo de sincronización de 4 bytes, un campo de dirección de 7 bits, un campo de identificador de dispositivo de 4 bits y un campo de final de paquete de 11 bits.

4.4 Núcleo de procesamiento neuronal

4.4.1 Arquitectura del núcleo de procesamiento neuronal

Los rectángulos de la figura 4.12 rotulados como multiplexores de tiempo, corresponden a los núcleos en donde se hacen los cálculos para determinar el estado siguiente de la red neuronal. La red neuronal que está implementada en la versión original del NSN consta de 96 neuronas.

Las neuronas de la red están divididas en 8 grupos de 12 neuronas cada uno. La estrategia que siguieron los diseñadores del núcleo fue crear los 8 multiplexores de tiempo que se observan en la figura 4.12. Estos multiplexores operan concurrentemente y cada uno calcula el estado futuro de 12 neuronas de forma secuencial. Es importante mencionar que las operaciones matemáticas que se realizan en cada uno de los multiplexores de tiempo están realizadas con precisión simple de punto flotante.

En la misma figura se observa que existe una conexión lógica entre los multiplexores de tiempo. Esta conexión lógica existe por el hecho de que para calcular el estado futuro de una sola neurona, se necesita el valor de voltaje dendrítico de todas las neuronas de la red. El transporte de estos valores de tensión desde y hacia todas las neuronas es llevado a cabo por el control general del NSN.

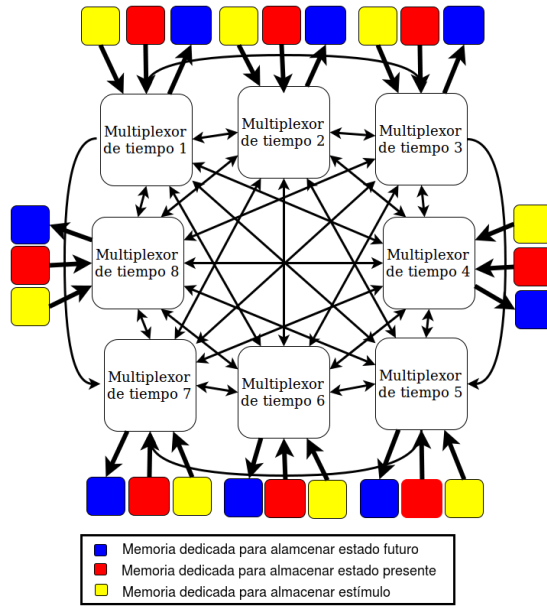


Figura 4.12. Organización lógica del NSN

El estado actual, el estímulo y el estado futuro del grupo de neuronas que cada uno de los multiplexores está encargado de computar, se alojan en las unidades de memoria que se muestran en la figura 4.12. En esta se puede ver que estas unidades existen de forma independiente, por lo cual cada uno de los multiplexores cuenta con acceso exclusivo a las mismas. Lo cual permite reducir el tiempo de escritura y lectura del multiplexor ya que este no tiene que esperar acceso [1].

4.4.2 Representación a nivel de bloque del NSN

La representación de nivel de bloque de la versión que se eligió implementar de este core tiene 5 puertos AXI4 Stream, un puerto AXI4 Lite y 4 señales de control (start, ready, done y idle) encapsuladas en el puerto **ap_ctrl**.

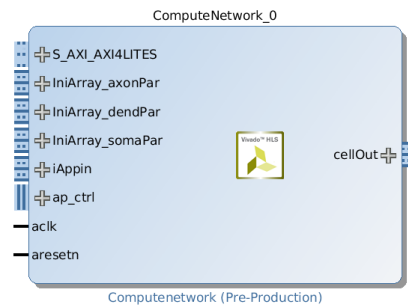


Figura 4.13. Representación de nivel de bloque de NSN

Los puertos que el NSN utiliza para leer la información necesaria para inicializarse son de tipo AXI4 Stream y se muestran en la figura 4.13 rotulados como **IniArray_axonPar**, **IniArray_dendPar** e **IniArray_somaPar**. Los estímulos para las neuronas se ingresan por medio del puerto **iAppin** y los voltajes de axón producidos en la simulación se exportan por medio del puerto **cellOut**.

El puerto AXI4 Lite que se muestra en la figura se utiliza para pasar parámetros como el tamaño de la red neuronal y el factor de multiplexado para la simulación. Este parámetro corresponde con el número de neuronas que se computan en cada multiplexor de tiempo de la figura 4.12. Además, a través de este puerto es posible dar una señal de inicialización, para establecer el estado inicial de las neuronas de la red.

La descripción de las funciones de las señales del puerto **ap_ctrl** se puede ver en la tabla 4.4. Este puerto es el medio con el que se puede controlar y obtener información del NSN desde otros módulos. Este puerto es generado por la síntesis de Vivado HLS.

Tabla 4.5. Descripción de señales de puerto ap_ctrl

Señales	Descripción
ap_start	Se utiliza para dar inicio a la operación del NSN
ap_done	Es controlada por el NSN e indica la finalización de un operación
ap_ready	Es controlada por el NSN e indica que está listo para recibir datos
ap_idle	Es controlada por el NSN e indica que el NSN está ocioso

4.4.3 Funcionamiento del núcleo de procesamiento neuronal

En la figura 4.14, se muestra un diagrama de flujo sobre la operación del NSN para un paso de simulación. En la figura se observa que la secuencia de eventos esta rotulada con números para fácil identificación.

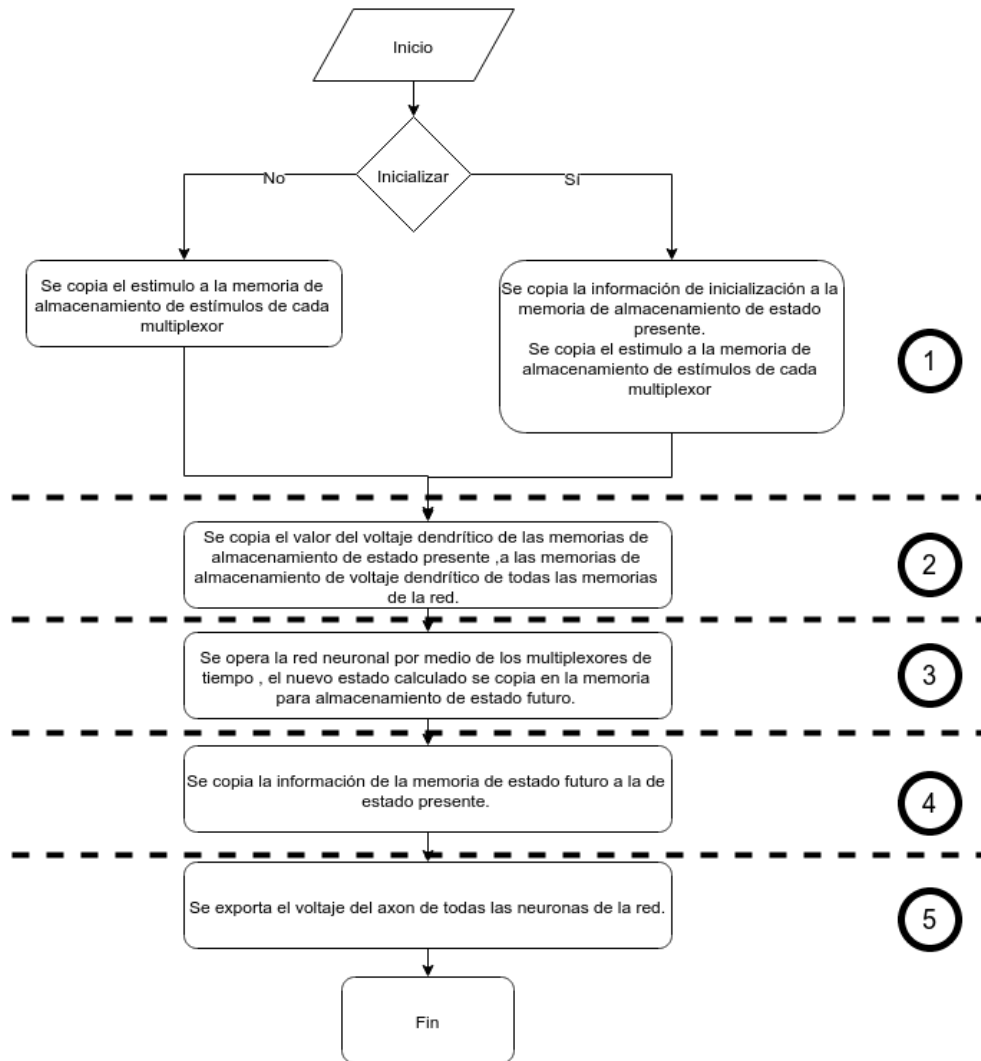


Figura 4.14. Diagrama de eventos en el NSN

Los primeros eventos que suceden en el diagrama de flujo y que están rotulados con el numero uno corresponden a la lectura y distribución de los valores de inicialización y de estímulo hacia las unidades de memoria dedicadas para estos datos.

Este método de inicialización se lleva a cabo al principio de las simulaciones y su función es cargar un estado inicial predeterminado en todas las neuronas que forman parte de la red que se va a simular. Esta función es realizada por el control general del NSN, el cual lee un arreglo de datos de inicialización en el que se describe el estado inicial de cada una de las neuronas de la red.

La estructura del arreglo de inicialización se muestra en la figura 4.15. Como se observa, el arreglo de inicialización cuenta con una cantidad n de estructuras de inicialización, donde n corresponde al tamaño de la red neuronal.

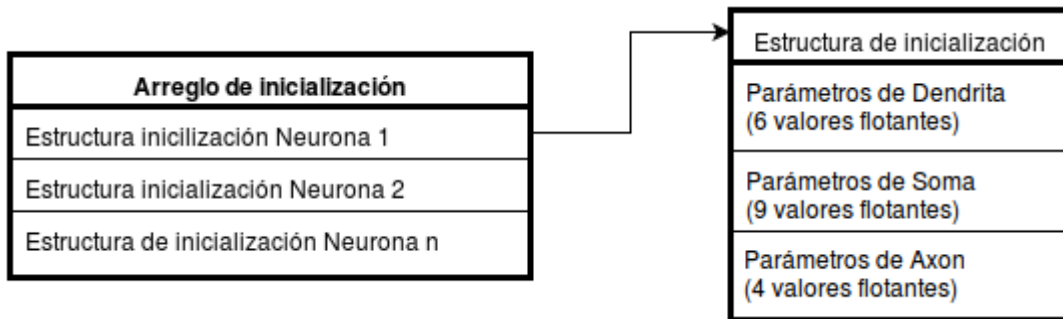


Figura 4.15. Estructura del arreglo inicialización

La información que contiene este arreglo de inicialización es distribuida entre las memorias de estado presente de cada uno de los multiplexores de tiempo del NSN. En estas memorias el estado de las neuronas es preservado mediante una estructura de datos como se muestra en la figura 4.16. De esta forma los parámetros de cada una de las estructuras de inicialización que ingresan al NSN se copian en un orden específico hacia los distintos campos que se muestran en la figura.

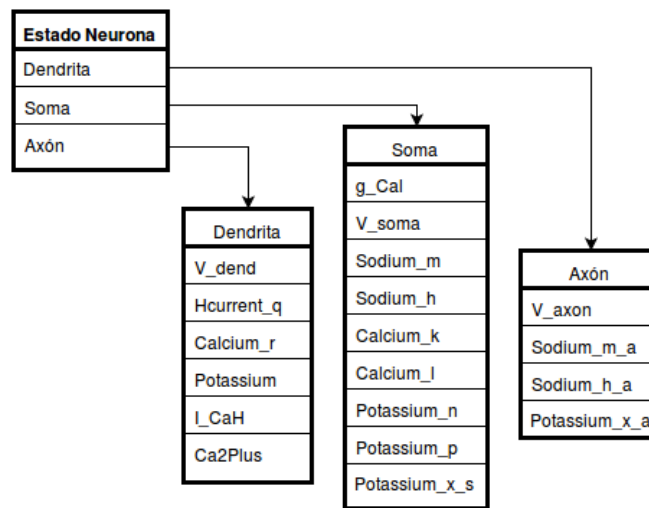


Figura 4.16. Estructura de la información de estado de las neuronas

En paralelo con el método de inicialización existe un mecanismo que se encarga de distribuir el estímulo que ingresa al NSN. El estímulo que ingresa hacia las neuronas tiene la forma de un arreglo simple de datos escalares de punto flotante como se muestra en la figura 4.17.

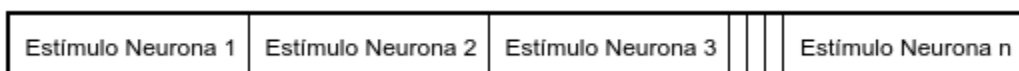


Figura 4.17. Arreglo de valores de estímulo

Luego, de haberse concluido con las tareas de inicialización el control general realiza la tarea rotulada como dos en la figura 4.14. Este procedimiento consiste en cargar los valores de voltaje dendrítico de cada una de las neuronas recién inicializadas hacia memorias dedicadas a almacenar estos valores.

Esto resulta en 8 arreglos (uno para cada multiplexor) que contienen los valores de voltaje dendrítico de toda la red neuronal como se observa en la figura 4.18. Esta distribución de voltajes dendríticos es la comunicación lógica que se representa en la figura 4.12.



Figura 4.18. Arreglos de voltaje dendrítico

El paso número tres del diagrama de flujo de la figura 4.14, es el momento en el que se computa el estado siguiente de toda la red neuronal. Los cálculos que se realizan en este estado de control son almacenados en cada una de las memorias de estado futuro que se muestran en la figura 4.12.

Luego, se copia el estado de la red calculado en el paso anterior hacia las memorias de estado presente. Esta transferencia de información es realizada por el control general y se realiza de forma paralela desde las 8 unidades de memoria de almacenamiento de estado futuro hacia las 8 unidades de almacenamiento de estado presente.

Por último, el control del NSN exporta el valor de voltaje del Axón de las neuronas de la red que está almacenado en las memorias de estado presente. Esto lo hace leyendo las memorias de estado presente y escribiendo estos valores en el puerto **cellOut** que se muestra en la figura 4.13.

5 Adaptación y evaluación del núcleo de simulación neuronal

5.1 Reducción

El NSN, fue entregado por parte del centro médico Erasmo en forma de código en lenguaje C. Este código en lenguaje C venía en dos versiones. Una de las versiones consistía del algoritmo para ejecución en una computadora convencional y una segunda versión para síntesis en Vivado HLS. Esta última, tenía la particularidad de que todo el manejo de memoria en la misma se realizaba de forma estática por limitaciones de la herramienta de síntesis Vivado HLS.

La primer tarea que se hizo con este código fue sintetizarlo, utilizando Vivado HLS. La síntesis se realizó seleccionando como objetivo una FPGA xc7a100tcs324-1 y un periodo de reloj de 10 ns. Los resultados del proceso de síntesis del código mencionado, tomados directamente del reporte de síntesis de Vivado HLS se pueden observar en la tabla 5.1.

Tabla 5.1. Resultados del proceso de síntesis de la versión original del NSN

Nombre	BRAM_18K	DSP48E	Flip Flops	LUT
Expresiones	-	-	-	3830
Estructuras FIFO	-	-	-	-
Instancias	-	1408	238360	270360
Memoria	1542	-	0	0
Multiplexores	-	-	-	10832
Registros	-	-	2062	-
Total	1542	1408	240422	285022
Disponible en FPGA	270	240	126800	63400
Porcentaje de utilización	571	586	189	449

En la tabla 5.1 se desglosa la estimación de la herramienta de síntesis sobre cómo el diseño sintetizado utiliza los recursos disponibles de la FPGA seleccionada. En la última fila de la tabla se muestra claramente que los porcentajes de utilización de este diseño original rebasan el 100% en todas las categorías de recursos. Esta situación hizo evidente la necesidad de reducir el tamaño del NSN.

Es importante destacar que el excesivo uso de recursos que hace este IP core, no se debe a que el diseño del mismo sea ineficiente. Como se mencionó, el mismo fue diseñado inicialmente para alojarse en una FPGA Virtex 7 (xc7vx485t-2ffg1761c) la cual tiene características superiores a la FPGA disponible para la implementación.

Uno de los objetivos de este proyecto consistía en poder descargar una versión funcional del IP core en un chip FPGA. La justificación para esto era la intención de

generar un sistema mínimo como prueba de concepto de la posibilidad de integración del mismo con un sistema de comunicación controlado por un procesador Microblaze.

Para reducir el tamaño del IP core, se analizó su arquitectura y la forma en que sus estructuras internas intercambian información. Este análisis implicó estudiar cuidadosamente el código en C que describe el hardware del NSN y la información encontrada en [1].

En la figura 4.12 se observa una representación de la arquitectura del NSN. Es claro, que debido a que este realiza cálculos de forma paralela, existen varias estructuras de hardware repetidas en su arquitectura. La estrategia seleccionada para reducir el consumo de área se basó en eliminar hardware que de alguna forma estaba replicado, siempre y cuando no se comprometiera la capacidad de realizar simulaciones acordes al modelo neuronal que el NSN implementa. Por defecto, al realizarse la eliminación de hardware se redujo también la capacidad de computación paralela del mismo. Esto, sin embargo no se consideró como un problema. Debido a que lo que se buscaba era contar con una versión reducida, que tuviera un comportamiento similar a la versión original. Esto para que las pruebas de concepto que se realizaran junto con un procesador Microblaze fueran lo más cercanas a una prueba con el NSN de tamaño completo.

La reducción se realizó eliminando las partes del código en C (para Vivado HLS) del NSN, para luego sintetizar el mismo y verificar el consumo de recursos por medio del reporte de síntesis de la herramienta.

Para identificar donde se localizaban los mayores gastos de recursos en la versión original del NSN. Se examinó el reporte de síntesis de Vivado HLS, el cual brinda un desglose detallado del consumo de hardware. Los resultados de este reporte para un solo multiplexor de tiempo se muestran en la tabla 5.2.

Tabla 5.2. Consumo de una instancia de un multiplexor de tiempo

Nombre del modulo	DSP48E	Flip Flops	LUT
Multiplexor de tiempo	176	29795	33795

Se eligió por lo tanto centrar la estrategia de reducción en suprimir estas estructuras debido a que eran las que provocaban el mayor consumo de recursos dentro del IP core. Es necesario mencionar que para escoger la cantidad de multiplexores de tiempo con que contaría la versión reducida del ip core se evaluó el consumo de recursos eliminando progresivamente la cantidad de multiplexores de tiempo. Se determinó entonces que el IP core únicamente sería implementable, si se dejaba solo un multiplexor de tiempo. La arquitectura resultante de este análisis se muestra en la figura 5.1.

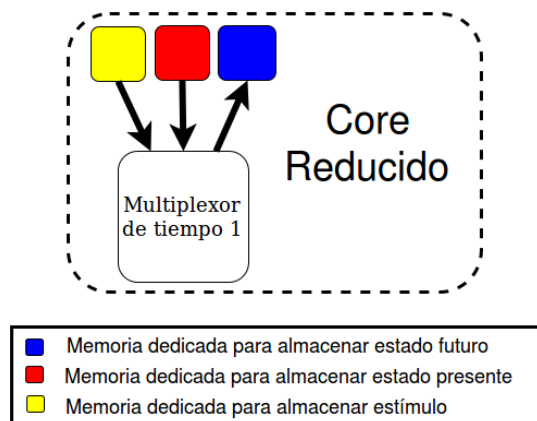


Figura 5.1. Arquitectura de versión reducida del NSN

De este modo se conservó la capacidad de realizar simulaciones concordantes con el modelo neuronal implementado en el NSN, con la salvedad de que las capacidades de computación serían claramente degeneradas. La versión reducida del IP core se configuró entonces para hacer simulaciones de 12 neuronas.

Luego de realizarse las respectivas modificaciones al código C (para síntesis en Vivado HLS) del NSN, era necesario comprobar que este era capaz de producir la misma salida que la versión C (para Vivado HLS) de tamaño original. Por lo tanto, se ejecutaron 120000 pasos de simulación en ambos programas (el de tamaño original y la versión reducida) en una computadora convencional para poder hacer dicha verificación.

El testbench utilizado para ejecutar dichas funciones fue el mismo que utilizaron los investigadores de Erasmus Brain Project en [1]. Como se mencionó, el mismo consta de 120000 pasos de simulación e inyecta un estímulo a todas las neuronas de la red de 6mV en el paso 20000.

El resultado de la ejecución de los mismos produjo los vectores de salida de los voltajes de axón de las neuronas que estos simulaban. Para la versión original del código C (para síntesis en Vivado HLS) se evaluaron las primeras 12 neuronas y para la versión reducida se tomaron las 12 neuronas que esta lograba simular. El porcentaje de error para esta comparación fue 0% para todos los pasos de simulación y todas las neuronas. Lo cual se debe a que el número de neuronas no afecta el resultado de las simulaciones en el modelo del NSN.

La gráfica del vector de salida de la versión reducida se muestra en la figura 5.2. En esta figura se puede ver que el vector de datos presenta 2 secciones oscilatorias en los extremos, así como 2 espigas producidas por el estímulo que introduce el testbench en el paso 20000. Esta es la misma forma que presenta la salida de la versión original pues como se vio el error entre ambas salidas fue 0.

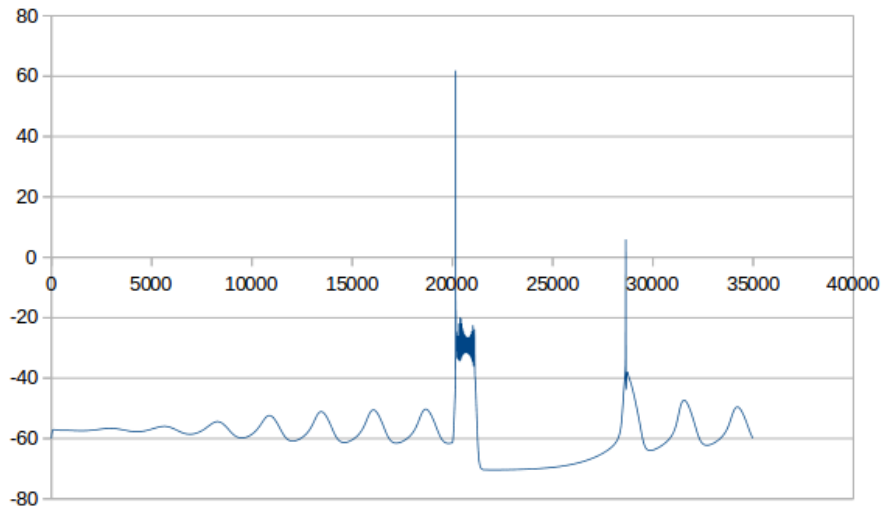


Figura 5.2. Voltaje de Axón de salida del NSN (reducido)

En la figura 5.3 se calculó el porcentaje de error relativo entre los valores obtenidos ejecutando la versión reducida y la versión original del algoritmo en C.

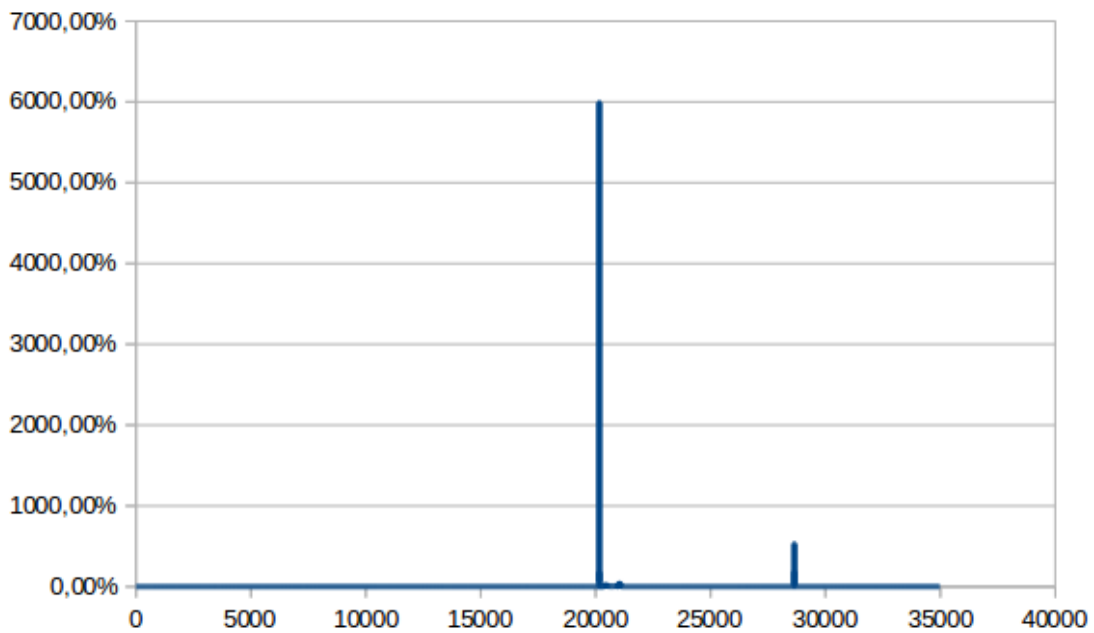


Figura 5.3. Comparación de los voltajes de axón de la primera neurona de la versión original y reducida del NSN

En la figura 5.3 se puede observar que en el momento en que llegan las espigas en la figura 5.2, el error crece notablemente. Este error se revisó en ambos vectores de datos y se determinó que el error dura 1 paso de simulación en el momento de llegada de las espigas. De esta forma el error es muy grande sin embargo es muy corto por lo cual se considera aceptable según las consideraciones que se hacen en [1] con respecto a la fidelidad de los datos de simulación de un modelo del olivo inferior.

Este error es además inherente a la implementación HLS realizada por los investigadores de Erasmus Brain Project por lo cual se puede afirmar que la reducción

hecha en este proyecto se ajusta perfectamente a las expectativas de comportamiento y fidelidad de datos de la implementación de hardware original del NSN.

Una vez que se comprobó el funcionamiento de la versión reducida del NSN, se procedió a sintetizarlo utilizando Vivado HLS. Obteniendo los resultados de síntesis que se muestran en la tabla 5.3.

Tabla 5.3. Resultados del proceso de síntesis de la versión reducida del NSN

Nombre	BRAM_18K	DSP48E	Flip Flops	LUT
Expresiones	-	-	0	419
FIFO	-	-	-	-
Instancia	-	176	29795	33775
Memoria	156	-	1280	120
Multiplexores	-	-	-	1382
Registros	-	-	488	-
Total	156	176	31563	35696
Disponible en FPGA	270	240	126800	63400
Porcentaje de utilización	57	73	24	56

Como se puede apreciar en la tabla 5.3 el consumo de recursos de la versión reducida si es menor a los existentes en la FPGA Artix 7 y por esto sí es implementable en la misma. Es importante mencionar que el modulo exportable de esta versión reducida al integrador de bloques de Vivado IDE, tiene la misma apariencia en cuanto a cantidad y tipo de puertos que la versión original mostrada en la figura 4.13.

5.2 Evaluación

La evaluación que se realizó del NSN comprendió los distintos tipos de interfaces presentes en el mismo. Además, también se analizó la tasa de consumo y producción de datos con respecto al tiempo, ya que a partir de estos se conocieron los requerimientos del sistema anfitrión en cuanto a almacenamiento de datos y rendimiento.

El consumo de datos del NSN se da en el momento de la inicialización y la recepción del estímulo de las neuronas que se simulan en el mismo. Por lo tanto, se tienen 2 tipos de consumo de datos, uno que se presenta solo al principio y otro que se da repetidamente a lo largo de la simulación. Se determinó en primera instancia que el consumo no sería elevado. Esto se debe a que el arreglo de inicialización y el arreglo de datos que describe el estímulo solo se consumen una vez, por lo que no representa una carga de información grande.

La producción de datos del NSN representaba un requerimiento mucho más exigente que el consumo. Esto debido a que una simulación produce gran cantidad de datos que se deben almacenar, ya que de otra forma se perderían debido a que el NSN no puede preservarlos.

La tasa de producción de dichos datos es posible encontrarla al analizar la latencia del dispositivo. El proceso de síntesis de la herramienta permite desglosar el valor estimado de la latencia para cada una de las etapas de la operación del NSN. Es importante mencionar que estos resultados dependen de la velocidad de la FPGA para la que se sintetizó el módulo, así como también del ciclo de reloj especificado. Además, la latencia obtenida también depende de las directivas TCL que se hayan utilizado durante la síntesis.

En la figura 5.4 se muestra la latencia en ciclos de reloj que presenta el NSN en cada una de sus etapas durante la operación para calcular un paso de simulación.

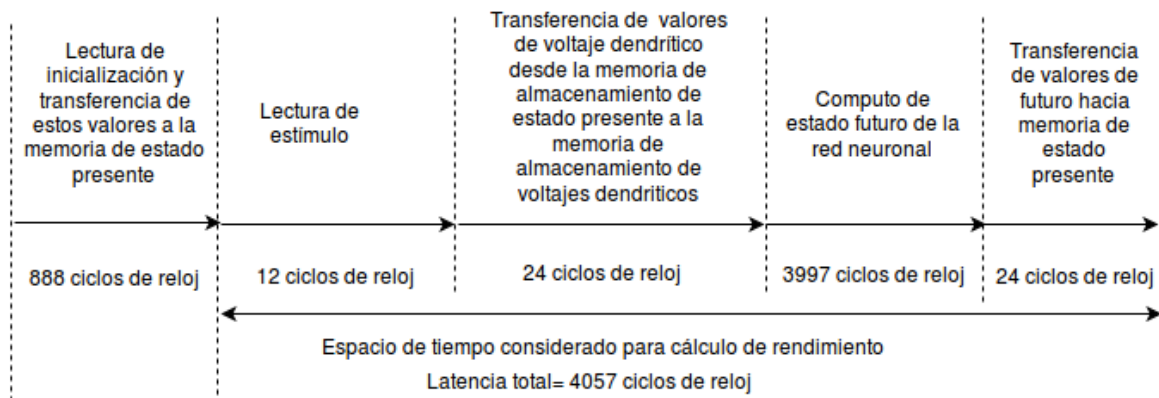


Figura 5.4. Diagrama de tiempo de la operación del NSN

En la figura se muestra el periodo de tiempo que se consideró para el cálculo de rendimiento del NSN. Este periodo de tiempo es el mínimo tiempo que le toma al dispositivo producir un nuevo estado para la red neuronal completa y la cantidad de datos que se consideró para dicho cálculo corresponde a la suma de bytes de todos los datos que describen el estado de la red neuronal. Por lo tanto, si la red neuronal consta de 12 neuronas y cada neurona cuenta con 19 datos de tipo flotante con precisión simple. La cantidad de bytes corresponde a 912 bytes. Entonces el rendimiento esperado del NSN que se obtiene es de 22,8 Mb/s.

Es necesario mencionar que este flujo de datos no es el que existe a la salida de la versión del NSN con la que se trabajó, ya que esta solo exporta los voltajes de axón de cada neurona de la red. Sin embargo, es importante diseñar la interfaz de descarga de datos desde del NSN para ese rendimiento ya que este considera todos los datos que podrían ser de utilidad en la aplicación experimental del mismo.

La interfaz de descarga de datos del NSN debería idealmente tener un ancho de banda que permita manejar el flujo de datos calculado de manera holgada. Ya que si se toma en cuenta que ese volumen de datos está calculado para un NSN de un octavo del

tamaño de la versión original. El volumen de datos de esta última sería 8 veces mayor (como máximo), lo que resultaría en un ancho de banda deseable de 182,4 Mb/s.

Es deseable que el NSN pueda producir varios segundos de simulación, ya que de esta forma las simulaciones brindarían mucha más información a los científicos que estudian las neuronas. Sin embargo, la intensa producción de datos durante una simulación no es manejable solo con la memoria con la que cuenta el NSN, ya que esta es muy limitada.

Las limitaciones de memoria del NSN se deben a que los bancos de memoria del mismo se implementan con recursos BRAM (Block Random Access Memory) de la FPGA y esta estrategia de implementación permite un rendimiento muy alto, pero hace que la cantidad de datos que puede retener el NSN sea muy reducida. Si bien se podría utilizar recursos de la FPGA para dotarlo de más memoria, esta sería una estrategia que no es escalable.

Esta característica del dispositivo condujo a la decisión de dotarlo de acceso a un volumen de memoria más grande en la cual se pudieran registrar los resultados de las simulaciones.

En un caso ideal el sistema anfitrión tendría que ser capaz de recolectar los datos que el NSN almacena en tal memoria y exportarlos por medio de una interfaz que brinde un ancho de banda similar a la tasa de producción de datos. Esto evitaría que la memoria principal del sistema se sature de información y por ende no habría que detener la simulación para exportar los datos hacia una computadora.

En el diseño del acceso a memoria del NSN que se observa en la figura 3.1 es necesario considerar la compatibilidad de las interfaces de datos y además tener presente el hecho de que el NSN no conoce internamente el mapa de memoria del sistema anfitrión. Este último, es un aspecto fundamental ya que esto requiere de un mecanismo mediante el cual los datos que viajan entre el NSN y la memoria se ubiquen en la misma a nivel lógico.

El consumo y producción de datos en el NSN se da por medio de arreglos con la información de cada neurona. La herramienta de síntesis (Vivado HLS) con la que se implementó el módulo sintetiza estas estructuras en puertos de nivel de bloque de tipo **ap_memory** o **ap_fifo** (first in first out). Los puertos utilizados en el NSN fueron de tipo **ap_fifo**, ya que son los únicos que admiten conexiones con interfaces de tipo AXI4 Stream.

En la herramienta de síntesis también era posible implementar otros tipos de puertos de nivel de bloque con interfaces AXI de tipo Lite y Full, sin embargo la implementación de estas hubiera requerido de modificaciones en la estructura de control interno del NSN. Esto no era deseable, porque de esta forma se incurría en un diseño en el que el funcionamiento del interno del NSN tenía que conocer del funcionamiento de su entorno.

Las interfaces AXI4 Stream tienen además varias ventajas con respecto a las interfaces Lite y Full. El área que requiere su implementación es menor a la que se requiere para implementar interfaces AXI4 Full y su rendimiento de transferencia de datos es similar.

Esto es posible gracias a que el protocolo AXI4 Stream no utiliza las señales de acceso a memoria y las comunicaciones se implementan en un solo sentido.

Esto hizo que se preservaran las interfaces AXI Stream en el diseño, lo que introdujo los requerimientos de conversión de este protocolo a AXI4-Full (que si es mapeado a memoria) y de controlar el mapeo de los datos en la memoria. Esto es manejable comparado con la opción de diseñar una interfaz desde cero que permita interactuar con los puertos de nivel de bloque tipo FIFO y que además permita que los datos sean mapeados adecuadamente a la memoria.

5.3 Modificaciones recomendadas para el NSN

El núcleo de simulación de redes neuronales cuenta con 4 puertos de entrada y un puerto de salida. Estos puertos son de tipo AXI Stream lo cual provoca que se tenga que recurrir a un core de DMA para que el núcleo pueda acceder a la memoria.

Lo que se logró durante este proyecto hace posible que el núcleo de simulación pueda funcionar físicamente en una FPGA. Esto, significa que el valor de este proyecto fue brindar la posibilidad de verificar el sistema experimentalmente. Anteriormente esto no había sido posible, por lo tanto la verificación del mismo se hizo mediante simulación.

Como se ha descrito previamente, el núcleo de simulación posee varias limitaciones. De acuerdo con el tipo de operación que se le dé al núcleo, ya sea en línea o fuera de línea se encuentran distintos problemas. Cuando el sistema se opera en línea se busca tener mucha capacidad de procesamiento para así poder simular la mayor cantidad de neuronas posible en tiempo real. En contraste cuando se opera el sistema fuera de línea es deseable contar con un volumen de memoria del mayor tamaño posible para así maximizar el número de neuronas que se puede simular en el sistema.

El diseño del sistema anfitrión para el núcleo de simulación, le dio al mismo la posibilidad de acceder al chip de memoria ram con que cuenta la tarjeta de desarrollo. Esto permite almacenar los resultados de la simulación, una capacidad con la que antes no se contaba. Potencialmente, esto debería servir para que el núcleo pueda aumentar el número de neuronas que puede simular. Sin embargo, en el diseño actual esto no es posible debido a que para esto hay que efectuar algunas modificaciones de forma en el NSN. Las modificaciones propuestas para maximizar el número de neuronas que el sistema puede simular se muestran la figura 5.5.

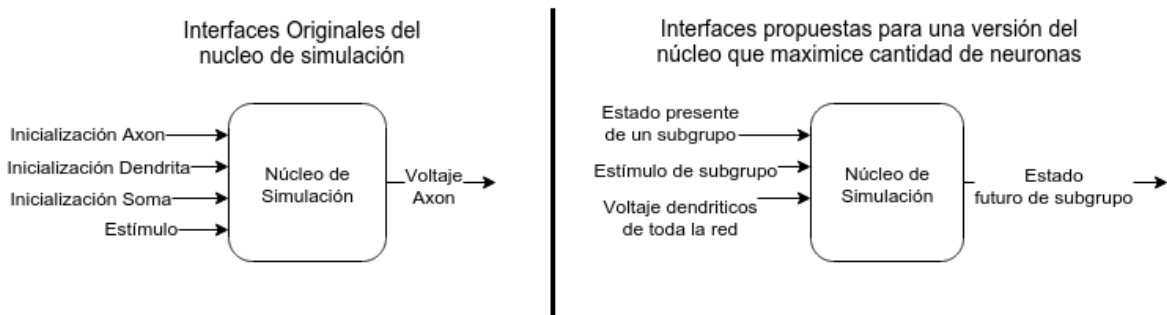


Figura 5.5. Modificaciones a nivel de bloque

En esta figura se muestra que en la arquitectura propuesta se reemplazan los puertos de inicialización por un puerto de estado presente y se adiciona un puerto para introducir los voltajes dendríticos de toda la red neuronal. El puerto de salida de la arquitectura original se reemplaza por un puerto que exporte el estado futuro de las neuronas.

El cambio en los puertos obedece a un cambio en el funcionamiento del núcleo de simulación, el cual ahora en vez de operar sobre toda la red neuronal, lo hará sobre subgrupos de la red neuronal. Lo cual significa que un paso de simulación de la red neuronal se calculara por fases, en donde el número de fases corresponde con el número de subgrupos en que se divide la red neuronal. Esto es posible gracias a que perfectamente se puede calcular el estado futuro de una neurona, solo teniendo los valores de voltaje dendrítico de todas las neuronas de la red para el paso de simulación presente.

Es por esta razón, que se adicionó un puerto para introducir los voltajes dendríticos de todas las neuronas desde la memoria. De modo que en la primera fase de cálculo se lee el arreglo de voltajes dendríticos presentes en el estado anterior de la red neuronal y ese arreglo se usa para calcular el valor futuro de todas las neuronas de los subgrupos de la red. La comparación del comportamiento descrito con respecto al comportamiento original se puede apreciar en la figura 5.6

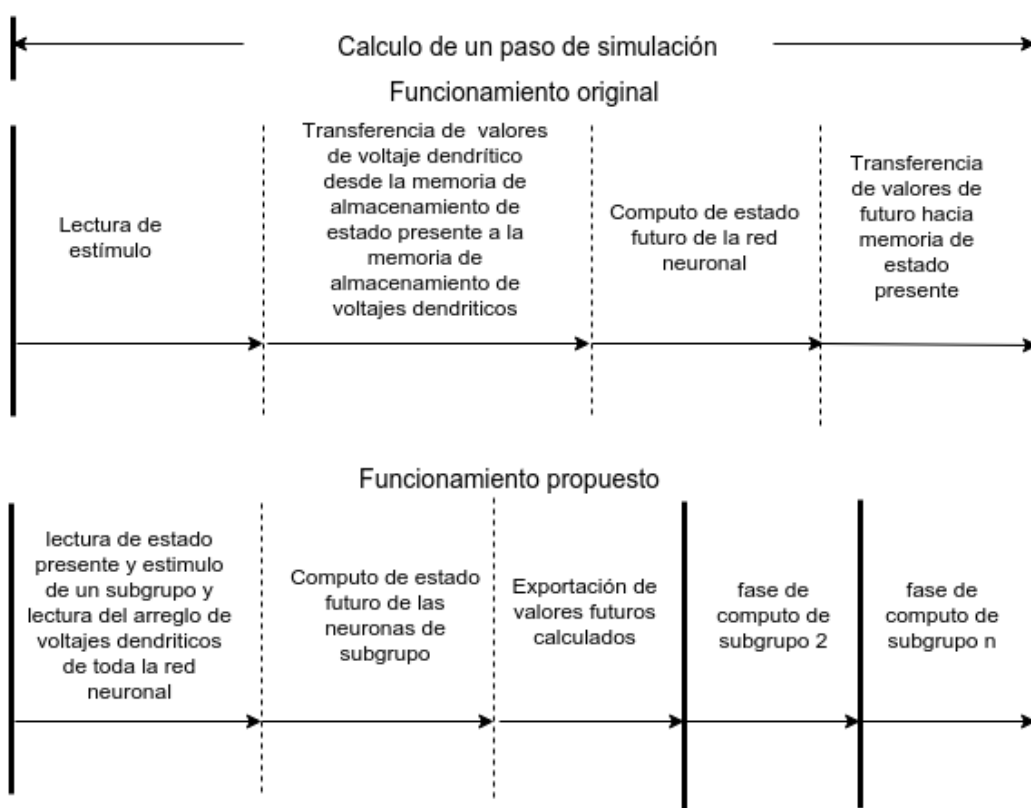


Figura 5.6. Modificación propuesta para fases de simulación

Este mecanismo permite que sea el chip de memoria DDR SDRAM el que almacene el estado de la red neuronal y no la memoria interna del núcleo de simulación. Esto permite simular redes de mucho mayor tamaño ya que ahora se va a utilizar un volumen de memoria mucho más grande.

Este aumento en el tamaño de las simulaciones se hace a expensas de sacrificar velocidad de simulación, pero esta situación es aceptable debido a que el sistema se propone para simulaciones fuera de línea.

6 Desarrollo de un sistema embebido para soportar el núcleo de simulación

6.1 Estrategia de diseño y propuesta

El sistema para albergar el NSN consiste en un SoC sobre FPGA. Las plataformas embebidas de este tipo tienen 2 partes fundamentales, el hardware y el software. Para establecer los componentes principales con los que debía contar el diseño de la solución se elaboró una tabla de requerimientos y restricciones.

Los requerimientos y restricciones se generaron partiendo del estudio y análisis de las necesidades de área, tipos de interfaces de datos y velocidad de transmisión de datos de la versión del NSN denominada IO_FullInter_8HWCells_RuntimeMuxFactor. Este análisis, además tomó en cuenta que en este proyecto no se buscaba diseñar un sistema capaz de hacer simulaciones en línea (tiempo real), pero sí maximizar las capacidades de computación del sistema anfitrión con el NSN integrado al procesador a un Microblaze. Para satisfacer este requerimiento se planteó una estrategia de carga y descarga de datos para la cual se diseñó el hardware y el software de SoC.

Es importante mencionar que el hardware fue sujeto de restricciones que no provenían del NSN. Éste obtuvo limitaciones provenientes de parámetros como la cantidad de recursos disponibles en el chip FPGA y las interfaces de los dispositivos físicos presentes en la tarjeta de desarrollo en la que se probaron las distintas partes del sistema.

Las opciones de hardware consistían principalmente en la colección de IP cores disponibles en el IPI de Vivado IDE. La selección de los IP cores que se incluyeron en la solución, se basó en un estudio de la bibliografía en el que se recabó la información de consumo de área, desempeño en la transferencia de datos y tipo de interfaces presentes en cada IP core. Para cumplir un requerimiento en donde se presentaban varias opciones de diseño, se realizaron tablas en donde se ponderaron las ventajas y desventajas de cada una de las alternativas que se podían utilizar para satisfacerlo. El diseño del hardware se hizo también considerando la estrategia de carga y descarga de datos seleccionada en la evaluación del IP core del olivo inferior.

La arquitectura utilizada para la interconexión de los distintos componentes utilizados en el sistema anfitrión del NSN, se hizo considerando el tipo de protocolo involucrado en la comunicación y las exigencias de ancho de banda de las comunicaciones.

El diseño de la solución de software consta de un paquete de soporte de plataforma (bsp) y de la aplicación de software. La generación del paquete de soporte de plataforma estuvo fuera de las decisiones de diseño. Esto se debe a que los drivers que se incluyen en este, son generados automáticamente por la herramienta de desarrollo

de software de acuerdo a los ip cores que sean incluidos por el usuario en el diseño del hardware. Sin embargo, la inclusión de bibliotecas y demás decisiones de construcción del paquete se realizaron en concordancia con los requerimientos identificados y analizando las ventajas y desventajas de las opciones disponibles.

El propósito de la aplicación de software es la de coordinar la comunicación entre el SoC y una computadora anfitriona, así como también controlar la carga y la descarga de datos al NSN.

Las decisiones del diseño de la aplicación final de software se fundamentaron en un análisis general del flujo de datos necesario para implementar la estrategia de carga y descarga de datos escogida. A partir de este insumo, se identificó cuáles periféricos iban a ser atendidos por sondeo y cuales por interrupción. Así, se estableció la prioridad con que se iban a atender las interrupciones de cada periférico.

La realización del segundo objetivo de este proyecto, fue de igual forma condicionada por el análisis de las necesidades del core de modelado del olivo inferior. Para la escogencia de una interfaz ideal se estudiaron las referencias bibliográficas encontradas acerca de las interfaces de PCIe y USBv2. Estas últimas interfaces se contrastaron contra una interfaz de tipo ethernet en términos de consumo de área y velocidad de transferencia de datos.

6.2 Análisis de requerimientos y restricciones

Los requerimientos más importantes identificados para el NSN fueron la necesidad de una interfaz de comunicación de alta velocidad y de acceso a un volumen de memoria fuera de chip. La comunicación de alta velocidad era deseable ya que como cualquier acelerador de hardware el propósito del NSN es que un científico del área de la neurociencia pueda desarrollar simulaciones neuronales en el dispositivo en una fracción del tiempo que lo haría en el procesador principal de una computadora convencional. El diseño del NSN ya era eficaz para acelerar el proceso de simulación de manera sustancial, sin embargo lo ideal es poder transferir los datos que este produce lo más rápido posible.

El manejo de las comunicaciones es un aspecto fundamental del diseño del SoC anfitrión ya que los protocolos de comunicación son muchas veces estructurados en capas lo cual hace necesario que los datos que van a ser transmitidos sean previamente procesados para adicionarles un campo de encabezado que varía según el protocolo. Esta necesidad de procesamiento hizo evidente que el sistema requería de software. Además, los diferentes módulos que se podrían integrar como parte de la solución para lograr la funcionalidad deseada requieren de rutinas de inicialización y coordinación general a nivel de sistema. Era por lo tanto imperativo que se integrara un procesador de propósito general al sistema.

El acceso a memoria era necesario principalmente para posibilitar el registro de los resultados de las simulaciones neuronales del NSN. Potencialmente, esto podría servir para incrementar el número de neuronas que el NSN puede simular. Es importante que el tiempo de acceso a la memoria fuera minimizado, ya que de esta forma se mejora el desempeño del NSN. Como se explicó en el capítulo anterior, las interfaces de datos presentes en el dispositivo están implementadas por medio del protocolo AXI Stream el cual no cuenta con señales para acceder a memoria. Esto introduce la necesidad de agregar un dispositivo capaz de traducir este protocolo a AXI4. Además, es necesario que el mismo pueda manejar el proceso de escribir y leer de la memoria principal.

La solución que se planteó desde un inicio para albergar el NSN fue un SoC sobre FPGA el cual contaría con todos los módulos necesarios para satisfacer dichas necesidades. Este enfoque de solución cuenta con la limitación principal del espacio en la FPGA. Como se expuso en el capítulo anterior el NSN es un módulo que requiere de una gran cantidad de recursos de chip para ser implementado. Lo restringió el diseño del SoC anfitrión a utilizar módulos con bajo consumo de área.

Los módulos que interactúan con el exterior del sistema anfitrión fueron restringidos por los periféricos disponibles en la tarjeta de desarrollo nexys 4 ddr. En esta tarjeta la memoria fuera de chip disponible era un chip de 128 Mb de tipo DDR SDRAM. La interfaz física que se encontró para implementar la comunicación de alta velocidad fue un Ethernet PHY para velocidades de transmisión de 10/100 Mb/s. La tarjeta cuenta con un puerto JTAG en el cual está mapeado un puerto UART el cual está disponible para diseños del usuario.

6.3 Detalle de la arquitectura de solución

La arquitectura detallada de la arquitectura diseñada para ser implementada se muestra en la figura 6.1. En esta se puede observar que los distintos IP cores que componen el diseño están conectados al procesador por medio de dos buses. Como se observa en la misma, el bus que conecta el procesador con los IP cores periféricos de comunicaciones es de tipo AXI Lite. El segundo bus que se muestra en el diagrama general del sistema es un bus AXI4 al cual se conecta el procesador y el NSN para acceder a memoria por medio de módulos DMA.

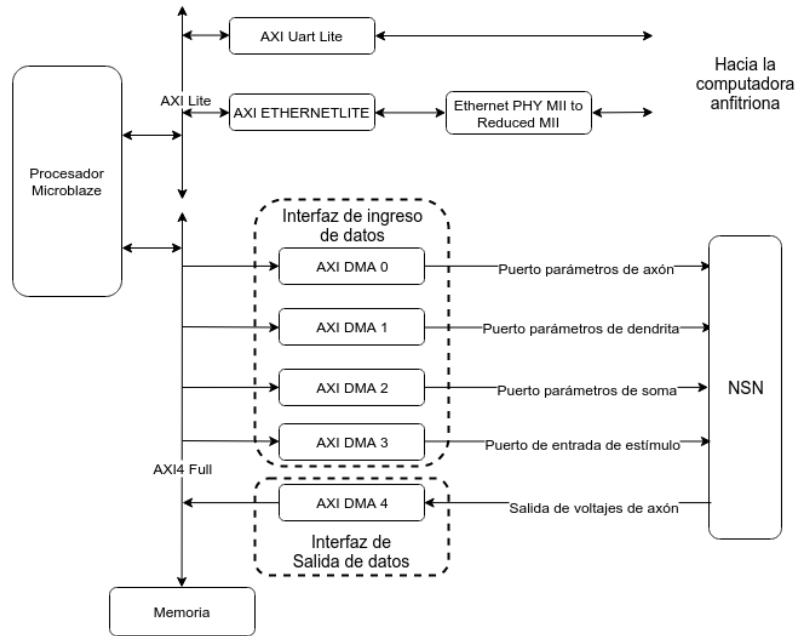


Figura.6.1. Diagrama general de arquitectura implementada

El diseño de la arquitectura se dividió de esta forma para optimizar el uso de ancho de banda del bus de la memoria y evitar el desperdicio de recursos de área en el chip. Para no desperdiciar ancho de banda en este bus era necesario evitar conectar dispositivos que no necesitaran acceso a la memoria, pues ocuparían ancho de banda innecesariamente.

En la figura 6.1 se observa también que el módulo de comunicación Ethernet no está conectado directamente a la memoria como se planteó en la figura 3.1, esto se debe a que la arquitectura que se implementó tuvo que ajustarse a las características de la tarjeta Nexys4 DDR. Las razones para este detalle de la implementación se verán en la sección de implementación dispositivos de entrada/salida.

Las razones existen para la presencia de cinco módulos de DMA contenidos dentro de las interfaces de entrada y salida de datos del NSN se detallarán posteriormente en la sección de acceso directo a memoria del NSN.

6.3.1 Implementación de buses de datos

Los buses que se muestran en la figura 6.1 se implementaron con el IP core AXI Interconnect. Este es el dispositivo que contiene la lógica de decodificación del mapa de memoria del sistema. Los buses AXI4 Full y AXI Lite se implementaron como se muestra en la figura 6.2 y 6.3. Como se observa en 6.2, en el bus AXI Lite se ubican solo

interfaces de configuración o interfaces de datos de baja velocidad de dispositivos de como AXI UART Lite y el AXI Ethernet.

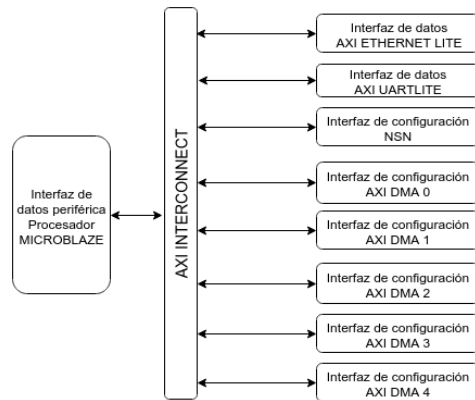


Figura 6.2. Detalle de interfaces conectadas a bus AXI4 Lite

En la figura 6.3 se muestra la implementación del bus AXI4 Full en el cual solo se encuentran las interfaces de cache de datos e instrucciones del procesador MICROBLAZE y las interfaces de datos de lectura y escritura de los módulos DMA que están contenidos en el bloque de entrada y salida de datos del NSN de la figura 6.1.

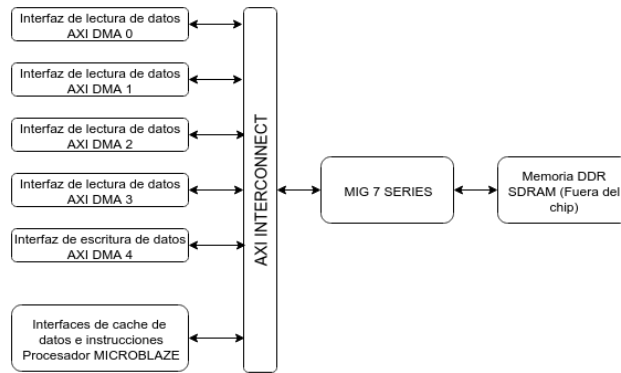


Figura 6.3. Detalle de interfaces conectadas a bus AXI4 Full

En ambas figuras se observa la presencia de varios dispositivos AXI DMA. Estos corresponden a los módulos DMA que están contenidos en los bloques de entrada y salida de datos del NSN en la figura 6.1. Es importante mencionar que en ambas figuras se hace referencia a diferentes interfaces del mismo modulo.

6.3.2 El procesador

A partir de las necesidades expuestas se decidió integrar un procesador de propósito general al sistema. El propósito del mismo era funcionar como control general para ejecutar las rutinas de inicialización de los diferentes módulos presentes en el sistema y para desarrollar tareas de pre procesamiento para paquetes de la interfaz de transmisión de alta velocidad. Inclusive, este tuvo utilidad en las tareas de depuración del SoC. El NSN se desarrolló en Vivado HLS, lo cual hace que los productos exportables del mismo fueran mucho más manejables en el ambiente Vivado y por eso fue que se eligió como ambiente de desarrollo.

En el ambiente Vivado los procesadores embebidos disponibles son el Microblaze y ZYNQ-7000. El ZYNQ no es un IP core blando, lo que quiere decir que para poder utilizarlo en un diseño se debe tener un chip de tipo ZYNQ. El espectro de posibilidades quedó reducido a la opción del procesador Microblaze, el cual sí es un soft IP core.

La implementación de este núcleo se hizo por medio del integrador de IP cores de Vivado en donde se instanció, para luego configurarlo según las necesidades de la aplicación. En la implementación del hardware del SoC de este proyecto el procesador se instanció con la mayoría de sus configuraciones por defecto, sin embargo se activaron las memorias cache de datos e instrucciones según las recomendaciones encontradas en [11,12].

Para el buen funcionamiento del procesador fue necesario agregar un generador de reloj y un generador de reset. El sistema resultante con las configuraciones por defecto se muestra en la figura 6.4. En esta se observa un módulo de depuración (Microblaze Debug Module), el cual facilita la depuración del software que se ejecuta en el procesador. Este dispositivo establece una comunicación por una interfaz JTAG con el depurador del kit de desarrollo de Xilinx.

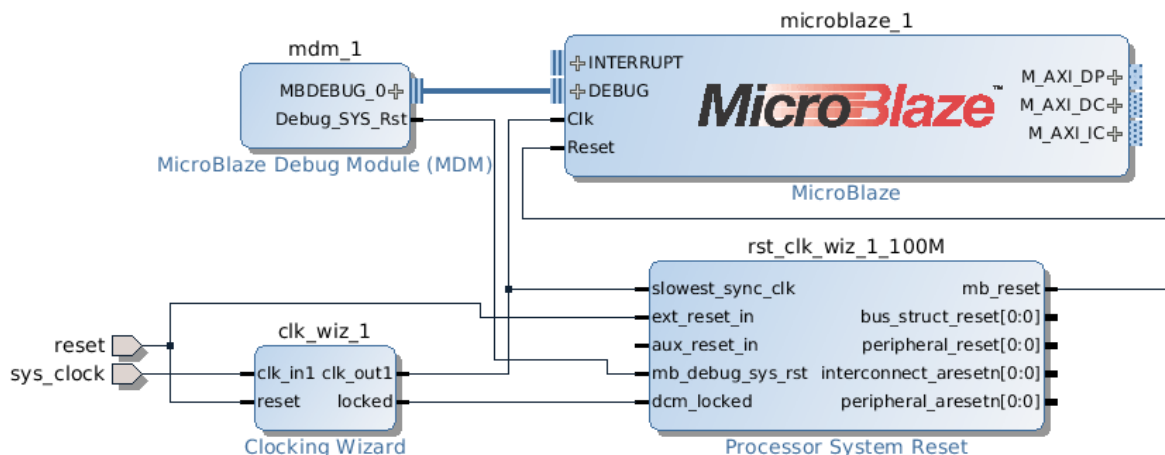


Figura 6.4. Sistema básico con procesador Microblaze

La interfaz periférica de datos **M_AXI_DP** se conectó a las interfaces de los dispositivos periféricos como se detallan en la figura 6.2, mientras que las interfaces de cache de datos e instrucciones **M_AXI_DC** y **M_AXI_IC** se conectaron a la memoria de la forma que se observa en la figura 6.3. El mapa de memoria que se obtuvo para el procesador Microblaze, de acuerdo con dichos esquemas de conexión se muestra en la tabla 6.1.

Tabla 6.1. Mapa de memoria del procesador MICROBLAZE del SoC anfitrión

Periférico	Dirección base	Espacio de direcciones
AXI UARLITE	0x40600000	64 K
AXI ETHERNET LITE	0x40E00000	64 K
AXI DMA 0	0x41E00000	64 K
AXI DMA 1	0x41E10000	64 K
AXI DMA 2	0x41E20000	64 K
AXI DMA 3	0x41E30000	64 K
AXI DMA 4	0x41E40000	64 K
NSN	0x44A00000	64 K
MIG 7SERIES (controlador de memoria)	0x80000000	128 M

6.3.3 Interfaz con la memoria

En la propuesta de diseño se explicó la necesidad del SoC de acceder a un chip de memoria externo a la FPGA. El chip de memoria disponible en la tarjeta Nexys 4 DDR del proyecto era un Micron MT47H64M16HR-25:H. Según la documentación del fabricante de la tarjeta [5], se recomienda que para un funcionamiento adecuado del chip de memoria, se tiene que incluir un controlador de memoria y una interfaz física en el diseño de la FPGA. El fabricante recomienda las soluciones que se muestran en la tabla 6.2.

Tabla 6.2. Soluciones recomendadas por Digilent para interfaz con memoria

Nombre de la solución	Ventajas	Desventajas
Modulo adaptador DDR-SRAM de Digilent	Instancia un controlador de memoria y utiliza un bus SRAM asíncrono para conectarse con lógica del usuario.	Se compromete el ancho de banda, para ganar simplicidad en la implementación del diseño.
Xilinx 7 series memory interface solutions core	Es una solución optimizada para controlar chips de memoria DDR SDRAM y genera una interfaz AXI4 para	La implementación es más compleja debido a que se requiere configurar parámetros de funcionamiento interno del chip de memoria

	conectarse con la lógica del usuario.	
--	---------------------------------------	--

De acuerdo con la información mostrada en la tabla 6.1, se seleccionó la opción del Xilinx 7 series memory interface solutions core. La implementación de este controlador es más compleja que la opción provista por Digilent, pero esta ofrece un bus de datos AXI4 para conectarse con el resto del sistema, característica que resultó conveniente dado que este es el protocolo que estaba presente en las interfaces de los IP cores que necesitaban acceso a la memoria. Adicionalmente, la solución está optimizada específicamente para chips DDR2 SDRAM y por último esta solución es fácilmente portable a implementaciones en otras tarjetas de desarrollo con FPGAs Xilinx de la serie 7.

La configuración de este IP core se realizó por medio de una interfaz gráfica que se despliega cuando se instancia el mismo. Por simplicidad y porque no era de interés para los objetivos de este proyecto ahondar en los conceptos físicos que intervienen en el diseño de un controlador de memoria se siguieron los parámetros de configuración provistos en la hoja de datos de la tarjeta que se muestran en el anexo 11.1.

Las configuraciones de interés en el diseño del SoC, eran las correspondientes con la interfaz AXI4 esclava que genera el IP core, ya que es finalmente mediante esta interfaz que el mismo es integrado al resto del sistema. Estas se muestran en la tabla 6.2 y se escogieron de acuerdo con las necesidades del sistema en chip y del tamaño de la memoria disponible.

Tabla 6.3. Parámetros de configuración de la interfaz AXI esclava del controlador de memoria

Nombre del parámetro	Valor
Ancho de dirección AXI	27 bits
Ancho de identificador AXI	32 bits
Esquema de prioridad	Round robin

El ancho de dirección AXI corresponde con el tamaño de la memoria disponible en la tarjeta la cual es de 128 Mb, luego para este mismo bus el ancho de datos AXI se dimensionó en 32 bits ya que este es el ancho que se seleccionó para todos los maestros que acceden al bus de memoria.

El esquema de prioridad que se utiliza entre la interfaz AXI esclava y la interfaz de usuario del controlador de memoria es round-robin. Este se necesita debido a que el protocolo AXI4 exige un bus de dirección de lectura y escritura independientes mientras que la memoria solo tiene un bus de dirección. Este problema exige que se divida la prioridad de acceso para escritura y lectura [6]. El perfil de solicitudes de lectura y escritura a la memoria era difícil de estimar por lo tanto se escogió el esquema round-robin, ya que en caso de que existan solicitudes de ambos tipos (lectura y escritura) este divide el ancho de banda. Adicionalmente, si en un determinado

momento no hay solicitudes de algún tipo este esquema otorga el ancho de banda al tipo de solicitud que sí se esté utilizando.

6.3.4 Interfaz de acceso directo a memoria del núcleo de simulación neuronal

El bus de memoria que se implementó en el SoC es de tipo AXI4, por lo tanto todos los IP cores que fueran a hacer uso de la memoria debían cumplir con dicho protocolo. Como se expuso en la sección 5.2 se decidió utilizar las interfaces de tipo AXI Stream en el NSN.

Las interfaces AXI4 Stream no pueden ser utilizadas para conectarse con una memoria ya que no tienen señales de dirección. Aún si estas tuvieran tales señales el NSN no podría acceder a la memoria directamente debido a que el NSN no tiene conocimiento del mapa de memoria del SoC anfitrión. Por lo tanto, se añadió un dispositivo de intermediación entre cada interfaz de NSN y la memoria.

El dispositivo escogido para cumplir con dicha tarea es el IP core AXI DMA (Direct Memory Access). La representación a nivel de bloque de este dispositivo se puede observar en las figuras 6.5 y 6.6.

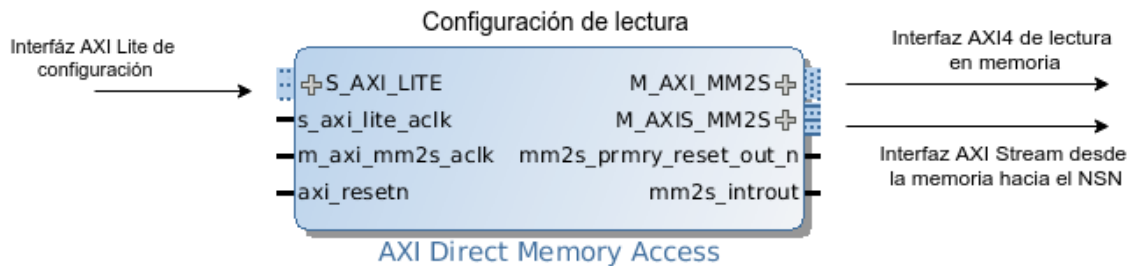


Figura 6.5 Representación de nivel de bloque de módulo DMA de lectura

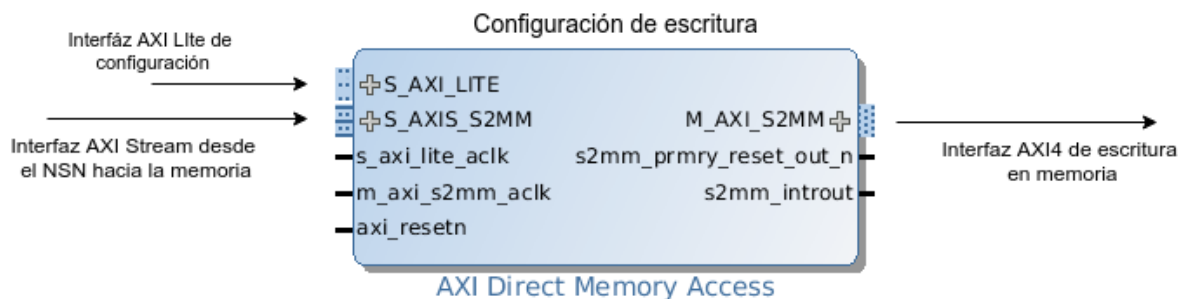


Figura 6.6. Representación de nivel de bloque de módulo DMA de escritura

Como se mencionó, el protocolo AXI4 Stream implementa la comunicación en un solo sentido. Esto quiere decir que en un bus AXI4 Stream solo existe un maestro y este solo puede escribir. Por esta razón, para que el NSN pueda consumir datos tenía que existir un dispositivo a su entrada que sea un maestro AXI Stream y que el mismo escriba en

las interfaces esclavas del NSN. En una forma similar para que el NSN pudiera exportar datos debía tener un esclavo en el cual pueda escribir pues la interfaz **cellOut** del mismo es maestra.

El IP core AXI DMA tiene varias formas de utilizarse. En la implementación realizada este se utilizó en su configuración simple. Esta permite que el mismo se comporte como maestro y como esclavo según el requerimiento que se tenga.

El modo de funcionamiento requerido para poder escribir en las interfaces del NSN se denomina modo de lectura y se muestra en la figura 6.5. Esta tiene una interfaz Stream maestra rotulada como **M_AXIS_MM2S** que se conectó con el NSN y una interfaz AXI4 llamada **M_AXI_MM2S** la cual provee de acceso a memoria al DMA. Para el caso de los datos de salida se tiene una situación similar pero inversa. En vez de la configuración de lectura se utiliza la de escritura la cual se presenta en la figura 6.6, ya que esta tiene una interfaz AXI Stream esclava para que el NSN pueda escribir datos en ella y también una interfaz AXI4 para mover los datos a la memoria.

El AXI DMA en la configuración utilizada solo puede manejar un canal de DMA, por lo cual el NSN requirió de 4 módulos DMA en configuración de lectura y uno en configuración de escritura. Esto se puede ver claramente en la figura 6.5.

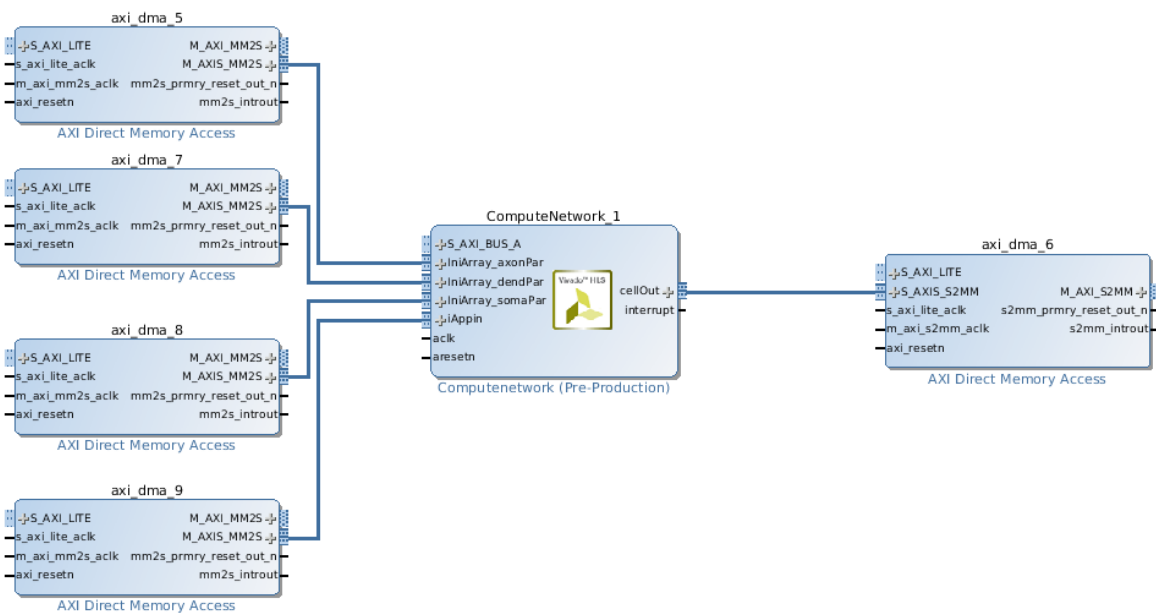


Figura 6.7. Interconexión de los módulos DMA con el NSN

Las interfaces AXI4 que aparecen en los módulos DMA de entrada y salida de datos del NSN se conectaron luego al interconector AXI (bus de memoria) que se muestra en la figura 6.3 para que estos pudieran acceder a la memoria.

Los dispositivos de DMA tienen que ser inicializados para operar bajo el modo deseado. Además, las transferencias desde o hacia la memoria requieren la intervención del

procesador cada vez que son realizadas. Esto es requerido para configurar la dirección de memoria de escritura/lectura y el tamaño de la transferencia.

Estas tareas de control de los módulos de DMA son llevadas a cabo por el procesador a través de las interfaces AXI Lite que se muestran en las figuras 6.3 y 6.4. Dichas interfaces fueron conectadas al bus AXI Lite como se muestra en la figura 6.2.

Este procedimiento requiere ser programado en el software que es ejecutado por el procesador Microblaze. La rutinas de inicialización que el procesador ejecuta para los módulos de DMA tiene la secuencia expuesta en la figura 6.6.

```
Estado_dispositivo //Estructura de datos con estado del hardware
Configuracion_dispositivo //Estructura de datos con configuración del hardware

Configuracion_dispositivo = funcion consulta tabla de modulos en hardware(Identificador del modulo)
    Si no hay configuracion valida entonces se produce error

funcion Inicialización de estado del dispositivo (Estado_dispositivo, Configuracion_dispositivo)
    Si inicialización del estado no es exitosa se produce error

funcion desactivar_interrupciones (Estado_dispositivo, Interrupciones a desactivar)
```

Figura 6.8. Rutina de inicialización general

La inicialización consiste en que el software revise la lista de configuraciones de los dispositivos DMA presentes en el sistema. Esta lista es creada por la herramienta cuando se exporta la plataforma de hardware al SDK de Xilinx. El procedimiento de búsqueda en esta lista se hace con el identificador del dispositivo que se está configurando y el mismo devuelve la configuración presente en el hardware. Luego se inicializa la estructura de datos que retiene el estado del dispositivo físico con la función de inicialización de estado. Por último, se desactivan las interrupciones ya que el programa funciona por sondeo.

La rutina de inicialización que se muestra en la figura 6.8, es necesario realizarla para todos los dispositivos DMA que se utilizaron en el diseño, de forma que en el programa en lenguaje C esta se realiza 5 veces.

Los dispositivos DMA requieren que se reserve espacio en la memoria principal del sistema para que estos lean o escriban en ella según sea su modo de operación. Por este motivo fue necesario establecer un mapa de memoria el cual respetarían todas las transferencias de datos para evitar solapamiento de datos en la memoria. El mapa de memoria utilizado se muestra en la figura 6.9.

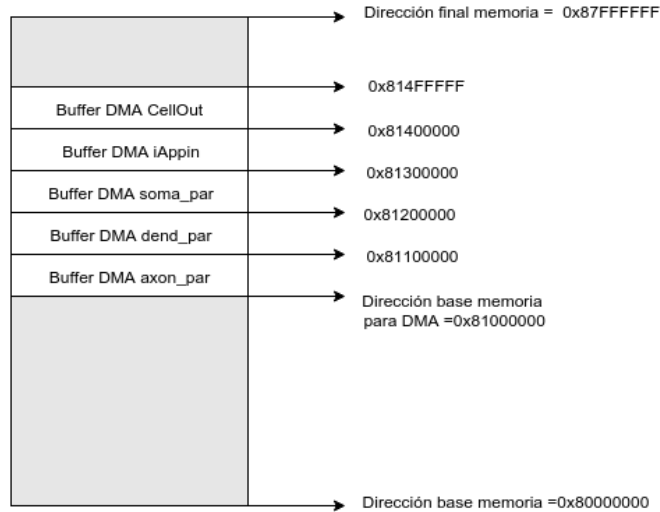


Figura 6.9. Mapa de memoria para los buffer de DMA

La memoria reservada para las operaciones de los dispositivos DMA se reservó con un desplazamiento de 16 Mb con respecto al inicio de la memoria. Esto se debe a que el programa a ejecutar por el Microblaze para las tareas de control también será almacenado en esta memoria. El mismo sin embargo, estará ubicado al inicio de la memoria y tiene un tamaño aproximado de 26 Kb por lo que un desplazamiento de 16Mb para almacenar los datos del NSN brinda la posibilidad de que el tamaño del programa pueda crecer si es requerido.

6.3.5 Dispositivos de E/S

Los dispositivos de Entrada/Salida que se implementaron son de tipo UART y Ethernet. Estos son IP cores del catálogo del IPI de Vivado y son respectivamente el AXI UART Lite y el AXI Ethernet Lite.

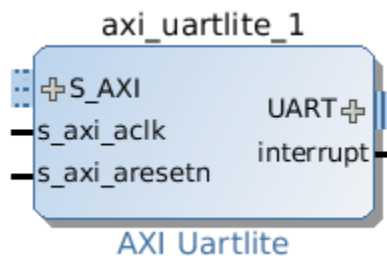


Figura 6.10. Representación de nivel de bloque del AXI UART Lite en el IPI

El AXI UART Lite se conectó al bus AXI Lite de la forma en que aparece en la figura 6.1. El diagrama de nivel de bloque se presenta en la figura 6.10. Este se utilizó para establecer una comunicación serie desde una computadora hacia el sistema anfitrión. El mismo se configuró a 9600 b/s, 8 bits de datos y sin paridad. La herramienta de

software configura al mismo como dispositivo estándar de entrada y salida en las configuraciones del paquete de soporte de plataforma. Gracias a esto se puede utilizar la función `xil_printf` para transmitir datos por medio de este puerto.

El IP core de Ethernet se conectó de la misma forma que el de tipo UART. Como se mencionó el diseño del sistema anfitrión estuvo restringido por las interfaces físicas disponibles en la tarjeta Nexys4 DDR. En esta tarjeta el Ethernet PHY (chip de Ethernet de acceso al medio), está conectado como se muestra en la figura 6.11.

En la figura 6.11, se observa que el chip integrado en la tarjeta es el LAN8720A, el cual está conectado a la FPGA por medio de una interfaz RMI (Reduced Media Independent Interface). Normalmente este tipo de interfaz se utiliza en aplicaciones que requieren una baja utilización de pines.

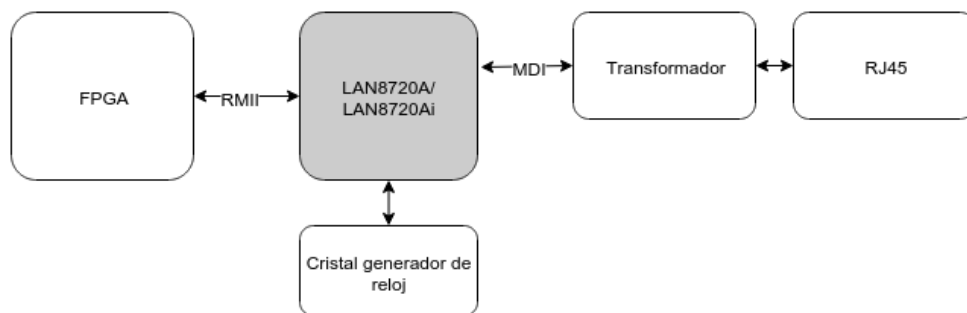


Figura 6.11. Interconexión de Ethernet PHY con FPGA

En la figura 6.10 se muestra el detalle de los pines presentes en el chip LAN 8720. Las señales **TXD**, **TXEN**, **RXD**, **RXEN** y **CRS_DV** son la que componen la interfaz RMI. Las señales **MDIO** y **MDC** son interfaces para configurar los registros internos del chip.

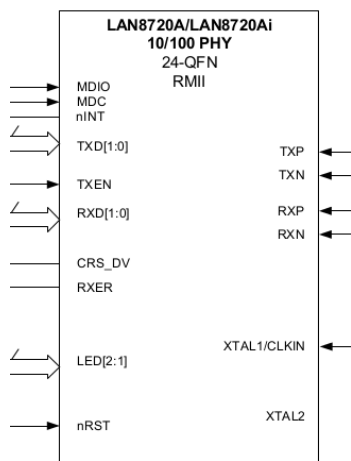


Figura 6.12. Diagrama de bloque de chip LAN 8720

Las opciones analizadas para implementar la conexión Ethernet del SoC anfitrión fueron el AXI Ethernet Subsystem y el AXI Ethernet Lite. El primero es de hecho una versión más completa del segundo, permitiendo velocidades de hasta 1Gbps. Por otro lado, el AXI Ethernet Subsystem es un sistema que funciona por DMA, el cual hubiera permitido una implementación similar a la que se muestra en la figura 3.1, en donde se planteó que el IP core de Ethernet pudiera contar con acceso directo a la memoria.

Desafortunadamente la licencia de Vivado Design Suite con la que se trabajó durante este proyecto no tiene este IP core licenciado, razón por la cual fue imposible implementarlo. De esta forma el IP core escogido para la implementación fue el AXI Ethernet Lite que se muestra en la figura 6.13.

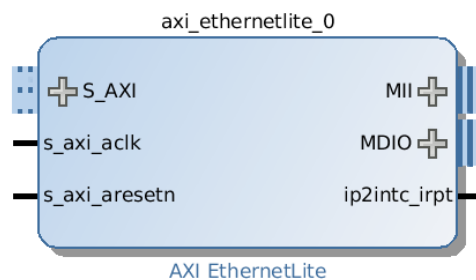


Figura 6.13. Representación a nivel de bloque del AXI Ethernet Lite en el IPI de Vivado

La interfaz MAC (acceso al medio) presente en el AXI Ethernet Lite es de tipo MII (Media Independent Interface) la cual es incompatible con la interfaz del Ethernet PHY presente en la tarjeta Nexys4 DDR. Siguiendo las recomendaciones que se encontraron en [5,7], se utilizó el IP core acoplador Ethernet PHY MII to reduced MII para conectar el AXI Ethernet core de la forma que aparece en la figura 6.14.

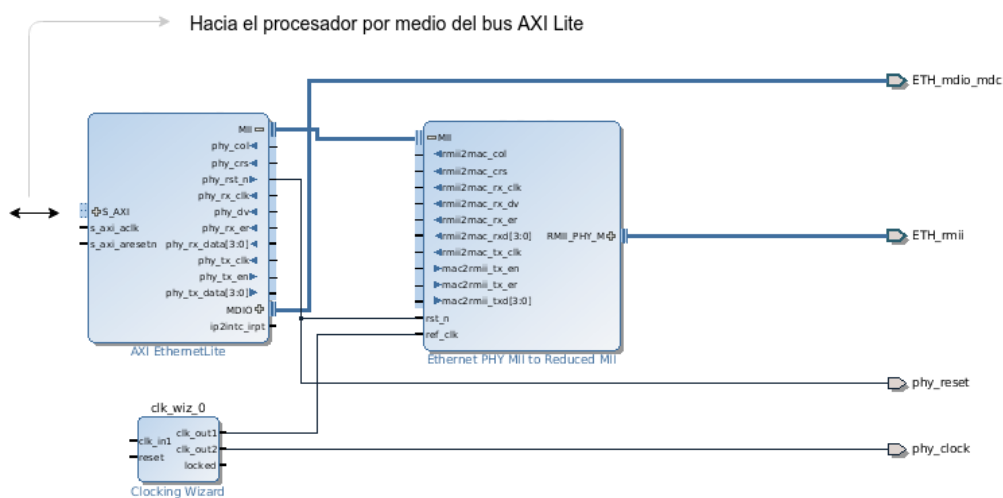


Figura 6.14. Conexión de acoplador con AXI Ethernet Lite

En la figura 6.14, se puede observar que la interfaz MII del AXI Ethernet Lite se conectó a la interfaz del mismo nombre del dispositivo de acople. Adicionalmente, la señal **phy_rst** se exportó para conectarla a la señal **nRST** del chip LAN 8720 que se muestra en la figura 6.12. En la figura 6.14, también se observa que se generaron relojes adicionales, ya que el acoplador y el Ethernet PHY requerían de relojes de referencia. Estos relojes se generaron mediante la utilización de un IP core denominado Clocking Wizard. En el mismo los relojes se configuraron a una frecuencia de 50 MHz, con un desfase de 45 grados.

En [5] el fabricante de la tarjeta recomienda este desfase relativo pero no se menciona cuál de los dos relojes debe estar atrasado. Ante esta situación, se consideró que este desfase se recomienda debido a la latencia que implica el IP core de acople, lo cual significa que a la salida de este acoplador todos los eventos suceden con un atraso. Por lo tanto, se decidió que el reloj atrasado 45 grados fuera el que se exportó para conectarse con la señal **CLKIN** que se observa en la figura 6.12.

El software para controlar esta interfaz no llegó a ser implementado debido a razones que serán expuestas en la sección siguiente.

6.4 Evaluación del diseño y propuesta

La evaluación del diseño se hizo por secciones para poder aislar los errores. Las partes que debían ser probadas en el sistema eran la comunicación UART, la comunicación Ethernet, el acceso a memoria y por último la realización de simulaciones exitosas en el NSN funcionando dentro del sistema anfitrión en la FPGA.

Las pruebas realizadas para verificar todas las partes del sistema excepto las simulaciones en el NSN siguieron la estrategia de prueba que se resume en la figura 6.15.

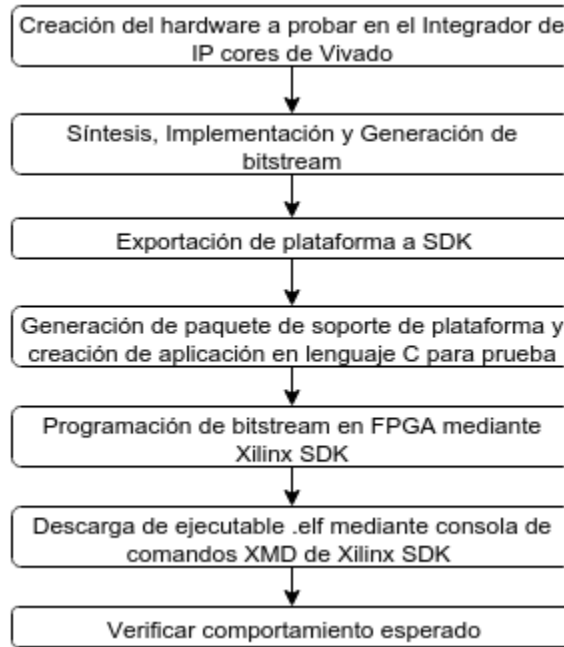


Figura 6.15. Estrategia de prueba utilizada durante evaluación

El uso de la consola XMD del modo que se detalla en la figura 6.15, necesita la utilización de comandos especiales. La función de esta consola es establecer comunicación con el módulo de depuración que se integró al diseño del procesador Microblaze. Una vez establecida la comunicación se utilizan los comandos que se listan en la tabla 6.4 para controlar la descarga de ejecutables al procesador Microblaze del sistema.

Tabla 6.4. Comandos de control para consola XMD

Comando	Descripción
connect mb mdm	Establece la conexión entre la computadora y el módulo de depuración del procesador
rst	Reinicia el procesador
stop	Detiene el procesador
dow <archivo.elf>	Descarga un archivo ejecutable de tipo ELF (Executable Linkable File) al espacio que se haya configurado en el linker script del programa.
run	Inicia la ejecución del programa descargado

6.4.1 Evaluación del acceso a memoria

La primera parte que se probó del sistema fue el acceso al chip de memoria externo. El sistema que se utilizó para las pruebas de memoria conto con la adición de una BRAM local desde donde se ejecutó un programa para probar el funcionamiento del chip de memoria. Para probar la memoria se utilizó un programa de prueba que viene incluido

entre las plantillas del Xilinx SDK. Este programa lo que hace es escribir y leer en formato de 32, 16 y 8 bits en la memoria. El resultado de esta prueba se muestra en la figura 6.16, donde se muestra puede leer que la memoria pasó las pruebas de escritura y lectura satisfactoriamente.

```
Testing memory region: mig_7series_0
  Memory Controller: mig_7series
    Base Address: 0x80000000
      Size: 0x08000000 bytes
    32-bit test: PASSED!
    16-bit test: PASSED!
    8-bit test: PASSED!
--Memory Test Application Complete--
```

Figura 6.16. Resultado de la prueba de memoria

6.4.2 Evaluación dispositivos de Entrada/Salida

6.4.2.1 Evaluación AXI UARTLITE

El IP core de comunicación UART es un dispositivo muy utilizado debido a su robustez y simplicidad de implementación. Por esta razón probar el dispositivo fue relativamente sencillo, ya que se utilizó una plantilla que ofrece el Xilinx SDK para probar el funcionamiento de la misma. La plantilla consiste en un programa que imprime periódicamente “Hello World” en el puerto UART y se realizó utilizando el AXI UartLite a 9600 bps, 8 bits de datos, 1 bit de parada y sin utilizar bits de paridad. En esta prueba se utilizó una computadora con sistema operativo Ubuntu 14.04 en la cual se utilizó el software GTKterm para examinar el puerto serie. Los resultados de esta prueba fueron completamente satisfactorios, ya que por el puerto se recibió el mensaje esperado.

6.4.2.2 Evaluación AXI ETHERNET LITE

La evaluación de la interfaz Ethernet se hizo utilizando un programa de prueba provisto en el SDK de Xilinx la cual se denomina LWIP echo server. Este programa utiliza la biblioteca LwIP (Light Weight IP) para implementar un servidor que haga eco a los mensajes que se envíen por telnet a la interfaz Ethernet implementada como se describió en la figura 6.14. Además, para esta prueba se adicionó al sistema anfitrión un IP core temporizador, ya que la biblioteca LwIP requiere de un cronómetro para monitorear el estado de las sesiones TCP.

La topología que se utilizó durante las pruebas de esta interfaz utilizando el programa de prueba LwIP echo server se muestra en la figura 6.17. En el programa de prueba fue

necesario configurar la dirección ip local, la máscara de subred, la puerta de enlace predeterminada, así como el puerto TCP que se utilizaría para establecer el servidor telnet. Las configuraciones utilizadas durante las pruebas se muestran en la tabla 6.5.

Tabla 6.5. Parámetros de configuración para prueba con interfaz Ethernet

Parámetro de configuración	Valor
Dirección IP	192.168.0.30
Máscara de subred	255.255.255.0
Dirección IP puerta de enlace predeterminada	192.168.0.1
Puerto TCP	7

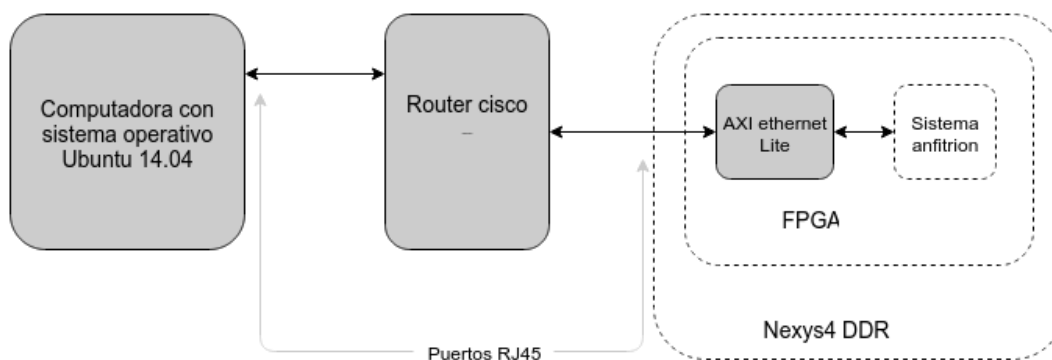


Figura 6.17. Topología de prueba utilizada en la evaluación de la interfaz

En esta prueba se empleó una computadora como se observa en la figura 6.17. En esta se instaló el software Wireshark Network Analyzer para monitorear el puerto Ethernet y verificar la llegada de paquetes provenientes del SoC anfitrión.

Una vez que se conectaron apropiadamente todos los equipos de la figura 6.17, se programó la FPGA con el bitstream de la plataforma de hardware que integraba la interfaz de Ethernet. Luego de haberse programado la FPGA, se procedió a seguir las instrucciones disponibles en la documentación del programa de prueba. De esta forma, se descargó el archivo ejecutable del programa por medio de la consola XMD del SDK en el sistema anfitrión.

Luego, en concordancia con la documentación se utilizó el comando 'telnet 192.168.0.30 7' en una terminal bash en la computadora de la figura 6.17 para abrir la sesión telnet entre esta y el sistema anfitrión funcionando en la FPGA. Esta prueba se repitió varias veces obteniéndose resultados muy diversos.

Los primeras veces que se realizó este experimento se descargaba el archivo ejecutable

en el sistema anfitrión y luego se emitía el comando que se mencionó desde la computadora para tratar de establecer comunicación. Estos esfuerzos sin embargo, fueron infructuosos debido a que el enlace nunca se daba. La terminal en la computadora desde donde se intentaba iniciar la sesión siempre devolvía un mensaje de host inalcanzable.

Ante esta situación se consultó la documentación de la biblioteca LwIP, por lo que se siguieron las recomendaciones señaladas en [8], en donde se recomienda encender el modo de depuración de la biblioteca en las configuraciones del paquete de soporte de plataforma en el SDK.

El modo de depuración de la biblioteca contribuyó a determinar que existía un problema con la cantidad de memoria reservada para el HEAP del programa de prueba. En la configuración por defecto del programa de prueba el tamaño del Heap era de 1Kb. Se pensó que el problema de conexión era debido a que la memoria disponible se acababa y el procesador se bloqueaba cuando trataba de acceder un espacio de memoria no permitido. Por esta razón, el Heap se aumentó considerablemente a 16 Mb.

Luego de que se realizó esta modificación, se tuvieron las primeras conexiones exitosas. De esta forma cuando se emitía el comando de telnet desde la computadora, se establecía una sesión en la que el sistema anfitrión respondía cualquier cadena de caracteres que se le enviara. Este éxito sin embargo no significó una operación completamente funcional de la interfaz, dado que la operación era poco robusta, lenta y en algunos casos la conexión no se establecía adecuadamente.

El modo de depuración de la biblioteca permitía ver como se formaban y con qué frecuencia se enviaban los paquetes desde el sistema anfitrión cuando se hacían las pruebas. Como se mencionó, en la computadora de prueba se utilizaba la herramienta de software Wireshark para monitorizar los paquetes que entraban por la interfaz Ethernet. Utilizando esta herramienta se notó que había muchos paquetes que se perdían. La razón para esta pérdida de paquetes nunca se pudo determinar.

Las pruebas siguieron efectuándose con el modo de depuración de la biblioteca LwIP activado. Mediante estas se determinó que durante la inicialización que se realizaba en la ejecución del programa de prueba, la biblioteca nunca reconocía adecuadamente la interfaz de Ethernet que se implementó en el sistema anfitrión.

La biblioteca LwIP mantiene una lista de las interfaces de red presentes en el sistema. Esta lista se configura al principio de la ejecución del programa de prueba mediante la utilización de una función llamada **XemacAdd**. Esta función toma una estructura de datos que describe la interfaz de Ethernet y la agrega en la lista de interfaces activas de la biblioteca. Esta función nunca logró ser exitosa durante la ejecución de las pruebas, ya que siempre devolvía un error generado desde adentro del cuerpo de la función. Este error se generaba en caso de que se tratara de agregar una interfaz de tipo desconocido.

Al analizarse el cuerpo de la función se vio que la misma tenía una estructura de casos,

en donde cada uno de estos casos contenía los procedimientos para agregar interfaces de tipo SGMII, RGMII y MII a la lista de interfaces de la biblioteca. La función sin embargo, no contemplaba la utilización de una interfaz de tipo RMII (interfaz del Ethernet PHY de la Nexys4 DDR), por lo cual esta función entraba en su caso por defecto.

Este hecho condujo a la conclusión de que el problema con la biblioteca LwIP y la interfaz RMII del Ethernet PHY consistía en la adaptación que se hizo con el IP core Ethernet PHY MII to REDUCED MII. La utilización de este core debía hacer transparente la comunicación entre el AXI ETHERNET LITE y el Ethernet PHY de la tarjeta Nexys4 DDR, sin embargo esto no sucedió así. El acople colocado entre estos dos en la figura 6.14 nunca logró presentar al procesador la interfaz real del Ethernet PHY (RMII) como una interfaz MII.

Debido a la complejidad implícita en la depuración de un sistema Ethernet y considerando las limitaciones de tiempo presentes en este proyecto se decidió dejar la evaluación de esta interfaz como un trabajo para el futuro.

6.4.3 Recomendaciones para la interfaz Ethernet

La combinación de la biblioteca LwIP y una interfaz de tipo RMII no resultó ser una buena combinación de diseño. La implementación realizada en este proyecto escogió esta combinación debido a que era la opción que ofrecía la posibilidad de implementar una interfaz Ethernet en un plazo aceptable. Es preciso considerar que solo se contaba con una Nexys4 DDR y que se necesitaba la pila de protocolos TCP/IP para mantener un control del flujo de los datos que se producían en el NSN.

En el futuro el problema de esta interfaz funcionando con LwIP se podría solucionar portando el diseño a una plataforma que cuente con un Ethernet PHY que sea de tipo MII o superior. Esto evitaría el uso del acople utilizado en este proyecto y así la biblioteca si reconociera la interfaz del Ethernet PHY. Además, ya que el sistema está hecho en Vivado esta tarea se hace sencilla y no consume mucho tiempo.

Una segunda opción que conllevaría mayor inversión de tiempo es conservar la plataforma de hardware actual y utilizar el driver que ofrece el AXI ETHERNET LITE directamente. Con esto se buscaría configurar directamente los registros internos del Ethernet PHY por medio de la interfaz de programación provista por el driver. Esto tiene varias implicaciones pues si se quiere utilizar una topología de prueba como la de la figura 6.17, se tendría que programar en la aplicación del usuario el procesamiento que se supone debía desempeñar la biblioteca LwIP.

Esta opción puede ser compleja si se considera que el driver del AXI Ethernet Lite no tiene en su interfaz de programación una forma de adicionar información de protocolos que estén por encima de la capa de enlace de datos. Por ejemplo, este no ofrece funciones para adicionar en un paquete las direcciones IP de origen y destino.

6.4.4 Evaluación de una simulación en el NSN

La estrategia adoptada para probar el funcionamiento adecuado del NSN difiere un poco con respecto a lo que se hizo para probar otras secciones del sistema. Esto se debe a la complejidad inherente que tiene la estructura de consumo de datos del NSN.

En la depuración del software controlador de las simulaciones del NSN se utilizó la perspectiva de depuración del kit de desarrollo de software de Xilinx. Esta perspectiva permite conectarse a un sistema Microblaze por medio de la interfaz de depuración del mismo. Esta conexión hacia el procesador permite ejecutar el programa línea por línea, monitorizar el contenido de las diversas variables del programa que ejecuta el procesador y ver el contenido de espacios específicos de la memoria del sistema en tiempo de ejecución.

La verificación del funcionamiento de los módulos DMA conectados a las interfaces de entrada y salida del NSN, se realizó mediante la utilización del IP core denominado Integrated Logic Analyzer (ILA). Este IP core es un monitor de interfaces AXI que se conectó de la forma que se muestra en la figura 6.16. Este permite al usuario de Vivado ver el contenido del bus en tiempo de ejecución.

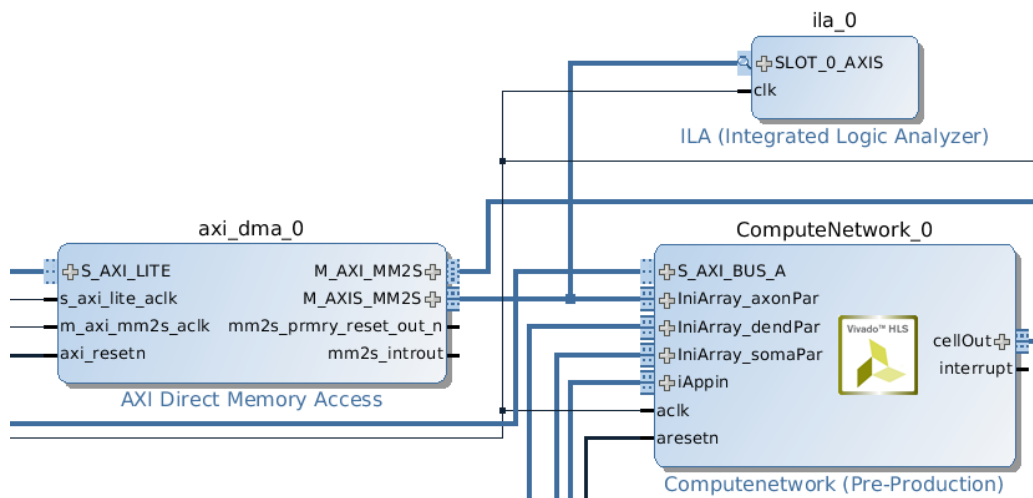


Figura 6.18. Conexión de Integrated Logic Analyzer

Como se observa en la figura 6.18, este monitor se conectó en paralelo a la interfaz del NSN denominada **IniArray_axonPar**. Así, se pudo analizar el contenido de los datos que salían del módulo de DMA y los datos que entraban al NSN.

La forma de determinar que los datos de las simulaciones ejecutadas por el NSN funcionando dentro del sistema anfitrión en la FPGA eran correctos, fue reproducir la metodología de prueba que los investigadores de Erasmus Brain Project utilizaron en [1].

Esta metodología consiste en realizar una simulación de 120000 pasos, en la que se inicializan todas las neuronas de la red del NSN en el mismo valor y de la misma forma se les introduce un estímulo de 500 pasos de duración en el paso número 20000 de la simulación. Ante este estímulo, se espera que todas las neuronas de la red respondan con una espiga.

En la figura 6.19, se puede ver un diagrama de flujo en el que se detalla el programa elaborado para ejecutarse en el procesador Microblaze y reproducir el experimento realizado por los investigadores de Erasmus Brain Project.

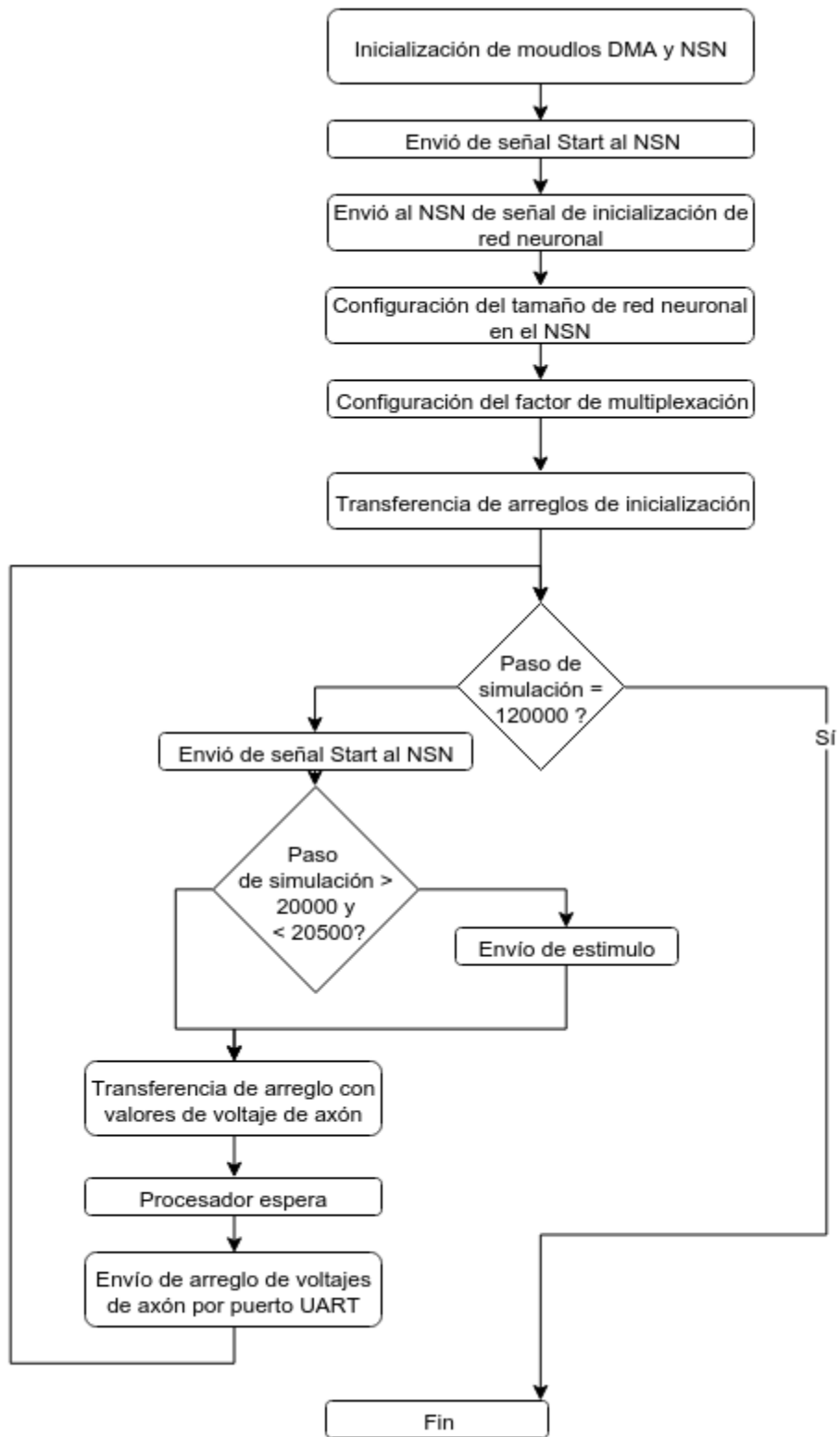


Figura 6.19. Diagrama de flujo de programa para experimento

Los datos producidos por el NSN en este experimento, se exportaban por medio del puerto UART del sistema anfitrión el cual finalmente se conectó a una computadora en la cual se recibieron los datos de la simulación que se muestran en la figura 6.20, en la cual se muestra el voltaje de axón de una neurona.

Es importante mencionar que la obtención del total de los datos duro aproximadamente 16 minutos. Lo cual se debe a que el puerto UART del sistema anfitrión se utilizó en una velocidad relativamente baja (9600 b/s) en comparación a lo que sería lograble con una interfaz Ethernet.

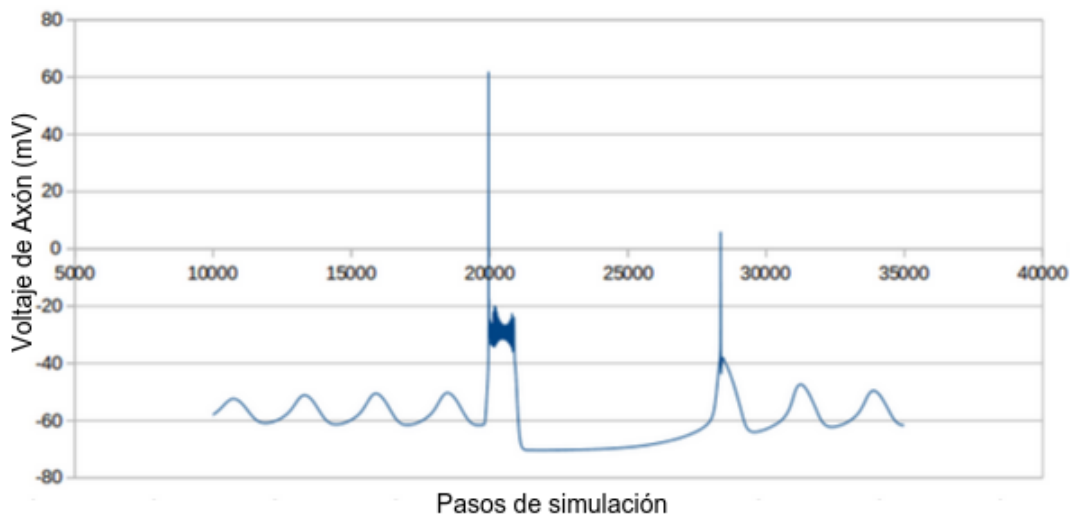


Figura 6.20. Resultado de simulación en FPGA

Los resultados mostrados en esta figura se pueden comparar con los expuestos en [1], para la metodología mencionada. En la figura 6.19, se pueden ver el grafico de dichos resultados, los cuales se obtuvieron mediante simulación utilizando Questasim 10.1 y para el NSN con una precisión simple de punto flotante (misma precisión de la versión del NSN con la que se trabajó).

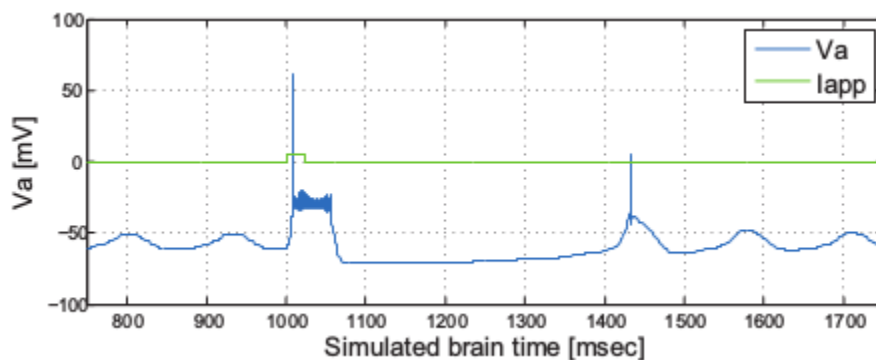


Figura 6.21. Resultados de simulación neuronal con precisión simple

El gráfico que se muestra en la figura 6.21, tiene el eje horizontal con unidades de mili segundos lo cual sucede porque un paso de simulación corresponde a 50 micro segundos de actividad biológica. Esto quiere decir que mediante esta relación se pueden comparar los gráficos de esta figura y la 6.20. Por ejemplo, el tiempo $t=1000\text{ms}$ en la figura 6.21 corresponde con el paso 20000 de la figura 6.20.

Las señales que se muestran en azul y verde en la figura 6.21 corresponden con el voltaje de axón de una neurona y con el estímulo introducido respectivamente.

Al compararse las señales de las figura 6.20 y 6.21 se puede ver que el comportamiento de los valores producidos por el NSN funcionando en el SoC anfitrión es en principio concordante con la respuesta del experimento realizado en [1]. Esto debido a que ambas señales experimentan oscilaciones 10mVpp aproximadamente y la espiga que sucede en respuesta al estímulo introducido corresponde con un valor mayor que 50 mV en ambos casos. Por esta razón se puede afirmar que las simulaciones en el NSN funcionando dentro del sistema anfitrión tienen una concordante con la forma de la señal esperada. Teórico.

Una vez que se comprobó que el comportamiento del NSN fuera concordante, era necesario determinar más exhaustivamente la integridad de los datos producidos. Por esta razón, se tomó el vector de datos exportado por el NSN y se comparó contra la salida producida por el algoritmo en lenguaje C ejecutado en una computadora. Esta comparación se hizo calculando el porcentaje de error relativo existente entre ambas señales. El porcentaje de error relativo se calculó con la expresión en (1).

$$\%error = \frac{(Valor\ teórico - Valor\ experimental)}{Valor\ teórico} \times 100 \quad (1)$$

El resultado de este cálculo para 35000 pasos de simulación se muestra en la figura 6.22.

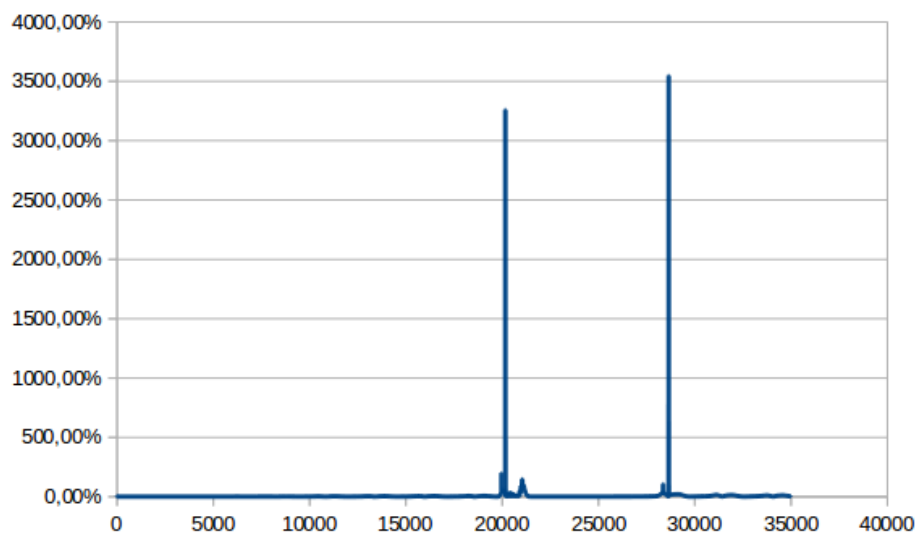


Figura 6.22. Porcentaje de error entre implementación de hardware y versión C original del NSN

En la figura 6.22, se puede ver que existe un error relativo muy grande que coincide con la presencia de espigas en la señal de la figura 6.20. Este error también está presente en la figura 5.3, en la cual se grafica el porcentaje de error entre el algoritmo C original y la versión en C para síntesis en Vivado HLS utilizando el mismo testbench que se utilizó para extraer los datos graficados en 6.20. Por lo tanto, las espigas de error que se observan en la figura 6.22 eran esperables ya que la respuesta del hardware debía ser en teoría muy similar al comportamiento obtenido de la ejecución del algoritmo HLS C.

En la misma figura, las subespigas (de menor tamaño) que se observan cerca de las espigas principales son atribuibles a que el hardware está implementado con una biblioteca matemática (HLS MATH) que difiere ligeramente de la biblioteca estándar de C. Según lo expuesto en [14] esto es esperable para los diseños que utilizan funciones de punto flotante.

Es preciso mencionar que pueden existir fuentes de error secundarias como la ocurrencia de inversiones aleatorias de bits en la FPGA y errores ocurridos durante la extracción del vector de datos por medio de puerto UART.

La fidelidad de datos alcanzada es por lo tanto satisfactoria con respecto a lo que el equipo de Erasmus Brain Project esperaba de la implementación en hardware del NSN.

6.4.5 Propuesta para interfaz mejorada de acceso directo a memoria del NSN

La prueba realizada utilizando el algoritmo de la figura 6.19, tiene el problema de que el procesador no conoce exactamente el momento en el que el AXI DMA a la salida del NSN termina de escribir los datos en la memoria. Esto implica que el procesador tenga que esperar un tiempo prudencial antes de intentar acceder a los datos recién escritos por el AXI DMA, porque en caso contrario podría leer datos que no corresponden a valores correctos de simulación.

Esto hace que la implementación actual no aproveche al máximo el poder computacional que ofrece el NSN. Esto es aceptable para la implementación que se hizo en este proyecto, sin embargo es necesario poder evitar esta situación en el futuro.

La implementación de la interfaz que escribe en memoria (interfaz que presenta mayor flujo de datos), se realizó con el IP core AXI DMA en configuración simple por razones de simplicidad y consumo de área. Sin embargo, entre las opciones que presenta este IP core está el modo Scatter-Gather. En este modo de operación el procesador del sistema solo tiene que configurar el comportamiento del AXI DMA una vez y este continuará haciendo transferencias DMA de la forma en que el usuario haya especificado.

Este modo de operación permite un manejo de los buffers mucho más acoplado a las necesidades de una interfaz con intensa producción de datos. Esto se debe a que el driver, maneja la memoria en 4 grupos. Estos grupos son la memoria que está bajo control del software, la que está bajo control del hardware, memoria en pre

procesamiento y memoria post procesada. De esta forma el driver del AXI DMA se encarga de devolver la memoria al hardware del AXI DMA cuando ya fue operada por el procesador y viceversa. Así, la aplicación del usuario recibe los datos cuando el driver le otorga control sobre el espacio de memoria donde residen los mismos. Entonces la aplicación ejecutándose en el procesador puede operar sobre la memoria que está bajo su control sin preocuparse por integridad de datos.

Además, el driver permite al procesador funcionar por sondeo o interrupción. Así, cuando se quiere una implementación por sondeo el driver cuenta con una función que la aplicación puede llamar constantemente hasta que existan datos que necesitan ser procesados. En el caso del modo de interrupción, la aplicación puede implementar una rutina que atienda la interrupción y procese los datos recibidos. Es importante mencionar que cualquiera de estas dos implementaciones no requiere que el procesador espere para operarlos, o sea que el flujo de datos sería tan rápido como el NSN los produzca.

7 Desarrollo de las interfaces de comunicación

El SoC anfitrión debía contar con una interfaz de comunicación de alta velocidad. El análisis para escoger el protocolo de dicha interfaz se limitó a protocolos que podían

funcionar con los puertos encontrados en la tarjeta de desarrollo VC707, ya que en el futuro el sistema será implementado y escalado en esta tarjeta. Por lo tanto, el estudio para seleccionar una interfaz ideal para el NSN incluyó los protocolos Ethernet, USB y PCI express.

Las versiones de los protocolos que se tomaron en cuenta para el estudio, se limitaron a la versión 2 para el USB, la segunda generación para PCI Express e IEEE 802.3 para Ethernet. En la tabla 7.1 se muestra los puertos disponibles en la tarjeta VC707 y los IP cores disponibles para agregar a un sistema Microblaze para utilizar dichos puertos.

Tabla 7.1. Tipos de interfaces y opciones de conectividad para tarjeta VC707

Protocolo	Puerto disponible en tarjeta de desarrollo VC707	IP core de Vivado para implementar interfaz
PCI Express	Conector PCI express de 8 líneas, este hace transferencia de datos de 2,5 Gt/s para aplicaciones de generación 1 y de 5 GT/s para aplicaciones de generación 2.	-Virtex-7 FPGA Gen 3 Integrated Block for PCI Express -AXI memory mapped to PCI Express core
USB	Transceptor USB3320 USB 2.0 ULPI	AXI Universal Serial Bus (USB) 2.0 Device v5.0
Ethernet	Ethernet PHY Marvel Alaska 88E1111 para comunicaciones de 10, 100 y 1000 MB/s	AXI Ethernet Subsystem

Así, primero se planteó una posible forma de escalar las capacidades de computación de la aplicación, para luego buscar los parámetros ideales de una interfaz de comunicación que podría satisfacer las necesidades de un solo NSN, así como las de la propuesta que se hace en la siguiente sección. Luego, se evalúan los IP cores disponibles en el catálogo de Xilinx para implementar cada uno de los protocolos y se termina con una recomendación para maximizar las capacidades de comunicación del NSN funcionando individualmente y en red.

7.1 Propuesta de red que integre varios NSN

La versión del NSN con la que se desarrolló el sistema anfitrión no está diseñada, para funcionar junto a otros dispositivos de su mismo tipo. Sin embargo, es posible hacer las modificaciones necesarias para implementar un sistema como el de la figura 7.1.

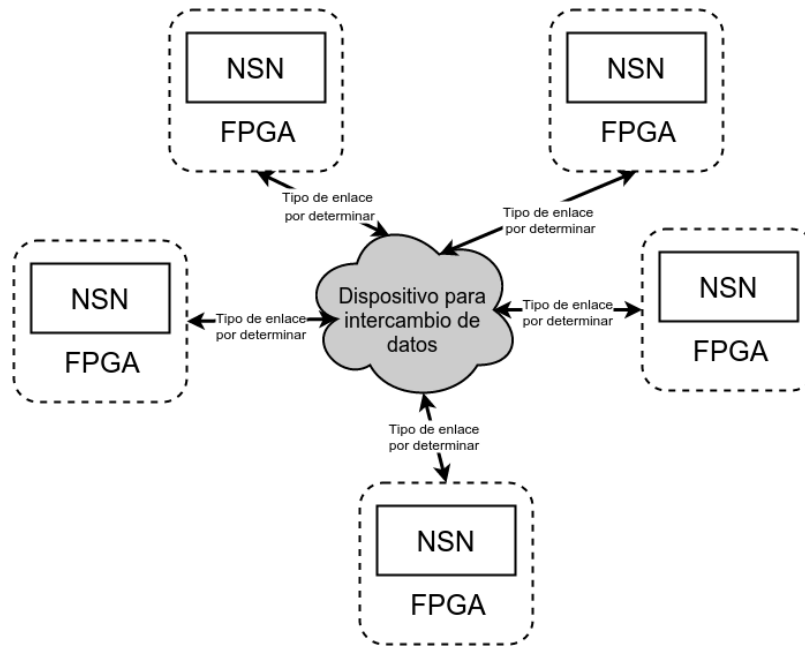


Figura 7.1. Red de múltiples NSNs

La finalidad de que los NSN estén interconectados es que de esta forma se puedan simular redes neuronales más grandes. Esto quiere decir que esta interconexión significa que todos los NSNs de la red trabajarían juntos para computar una sola red neuronal. Para poder funcionar de tal forma, los NSNs deberán poder intercambiar la información de los voltajes dendríticos, de las neuronas que están encargados de calcular. Esto quiere decir que cada uno de los NSN del sistema deberá conocer el voltaje dendrítico de todas las neuronas de la red.

7.2 Análisis de las especificaciones de los protocolos Ethernet, USB v2 y PCIe

Los protocolos de comunicación están descritos mediante extensas especificaciones en las que es posible encontrar una gran variedad de aspectos involucrados en el desempeño de los mismos. La extensión de estos documentos es tal, que es inútil e impráctico tratar elaborar un análisis que comprenda todos y cada uno de los aspectos que estos detallan.

El análisis de las especificaciones de los protocolos se hizo bajo criterios seleccionados de acuerdo a las necesidades actuales y futuras del NSN. La necesidad más inmediata del NSN con respecto a comunicación es una interfaz que provea del mayor ancho de banda posible. Sin embargo, es probable que Erasmus Brain Project busque implementar sistemas multi-FPGA como el planteado en la sección anterior, en los cuales interactúen varios NSN. Esto hace que el escalamiento sea también una característica deseable de dicha interfaz.

La selección de criterios realizada obedeció también a la metodología expuesta en [17], en la cual se analizan 3 enlaces seriales de alta velocidad (PCIe, USB y LLI) y se describe las interrelaciones de los distintos parámetros de los mismos. Esto permitió determinar cómo influyen estos criterios en la idoneidad de los protocolos para el caso específico de la interfaz del NSN.

Por lo tanto, los criterios que se consideraron determinantes en el desempeño de una futura implementación de una interfaz de alta velocidad para el NSN son la arquitectura a nivel de sistema, la estructura de control del enlace, la estructura de los paquetes de datos y la eficiencia de los datos.

7.2.1 Arquitectura a nivel de sistema

La arquitectura a nivel de sistema es un aspecto que está relacionado con la escalabilidad para la implementación de una red de NSN con múltiples FPGA como la que se muestra en la figura 7.1.

En la figura 7.2, se puede observar las topologías físicas de los protocolos PCIe y USB. En estas topologías se puede ver que ambos sistemas están diseñados alrededor de una unidad central de procesamiento y que la diferencia entre estos está en el tipo de dispositivo físico que sirve como intermediario entre los periféricos y dicha unidad. Es importante destacar que estas topologías físicas son configuraciones típicas y pueden existir variaciones. Además, están diseñadas jerárquicamente para servir a un CPU porque son tipos de enlaces destinados a funcionar a nivel de una computadora individual.

A diferencia de estos, Ethernet está diseñado para funcionar a nivel de red de computadoras. Por esta razón, para Ethernet no existe una topología física típica dado que las redes de computadoras pueden tener topologías variables y no existe la jerarquía que está presente en USB o PCIe.

El acceso de los periféricos a la unidad central de procesamiento es el principal diferenciador entre PCIe y USB. Este acceso es en ambos casos provisto por un dispositivo de interconexión intermediario. En el caso de PCIe este dispositivo intermediario es de tipo SWITCH, mientras que en USB es de tipo HUB.

Estos dispositivos se diferencian por el manejo de las colisiones en el medio físico de interconexión. Un HUB es básicamente una repetidora que propaga los mensajes entre interfaces. En este dispositivo todas las interfaces comparten el mismo medio físico, por lo tanto los periféricos tienen que esperar para acceder al host que se observa en la parte superior en el recuadro derecho de la figura 7.2.

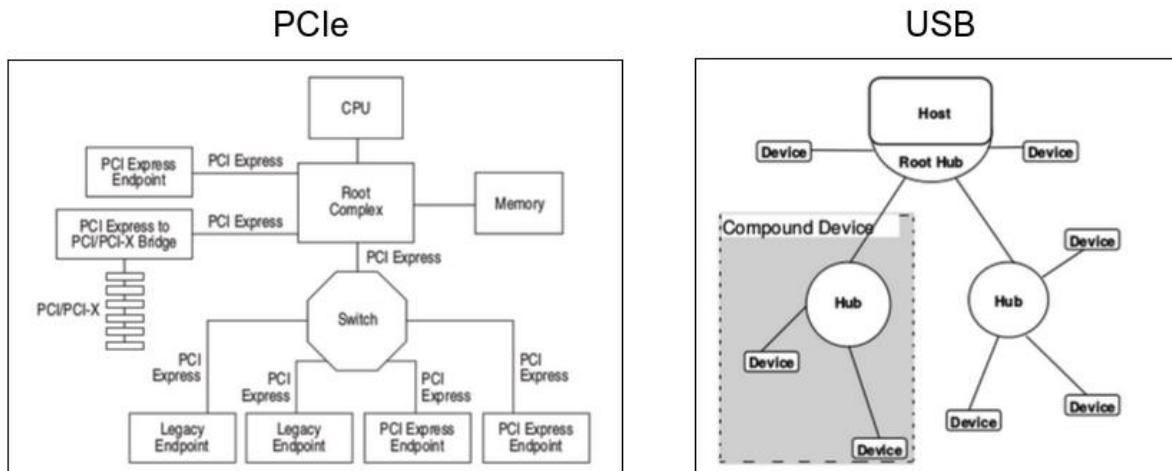


Figura 7.2. Arquitecturas de sistemas PCIe y USB [18, 19]

Un SWITCH separa los dominios de colisión de los dispositivos, lo cual significa que las diferentes interfaces de un SWITCH no comparten el mismo medio. Esto permite una utilización más eficiente del ancho de banda, lo cual implica que conforme se agregan dispositivos al SWITCH de un sistema PCIe, el ancho de banda disponible para los periféricos permanece aproximadamente constante.

La situación contraria se encuentra en la implementación de tipo HUB de USB, en la cual el ancho de banda se va a degradar sensiblemente conforme se aumentan los dispositivos interconectados. La eficiencia en la utilización en el ancho de banda no es una bondad del protocolo USB, sin embargo este permite conectar una mayor cantidad de periféricos que PCIe.

La razón por la cual USB presenta tal ventaja sobre USB, está relacionada con la interfaz física que se encuentra en ambos casos. La cantidad de puertos USB de una computadora es fácilmente expandible mediante la utilización de dispositivos HUB externos. Se puede utilizar dispositivos de tipo SWITCH externos, para expandir la cantidad de puertos PCIe, pero son más costosos que un HUB USB. Si se evalúa este factor para el caso de un enlace de tipo Ethernet se encontrará que en el mercado hay disponibles dispositivos SWITCH para Ethernet a los cuales se puede conectar varios dispositivos, con lo cual se podría construir una red escalable como la que se propone en la figura 7.1.

En la tecnología Ethernet, la cantidad de dispositivos que permite un SWITCH es muy variable, estos pueden llegar a interconectar hasta 24 dispositivos. Sin embargo, conforme aumenta el número de puertos de un SWITCH, la electrónica utilizada a lo interno del dispositivo tiene que ser más compleja y por ende el costo del SWITCH es mayor.

7.2.2 Estructura de control

Las arquitecturas que se observan en la figura 7.2, también afecta directamente la estructura de control de cada protocolo. Por ejemplo, en el caso del protocolo USB las comunicaciones se dan por medio de transacciones, las cuales solo pueden ser iniciadas por el dispositivo que aparece como host en el recuadro etiquetado como USB de la figura. Esto representaría una restricción, si se desea utilizar este protocolo para una implementación como la de la figura 7.1 o para una implementación simple con un único NSN, ya que se tendría que diseñar un protocolo de intercambio de datos en el cual el dispositivo periférico tenga que esperar para poder exportar los datos que produce.

En el protocolo PCIe las comunicaciones se pueden dar mapeadas a memoria lo cual puede servir para establecer una comunicación por medio de memoria compartida con otros dispositivos o con el procesador principal del sistema. Por último el protocolo Ethernet tiene una estructura descentralizada en la cual se tendría un gran espacio de decisión sobre la forma en que el intercambio de información se implementaría.

7.2.3 Tasa de transferencia de datos

La tasa de transferencia de datos que ofrece cada uno de estos enlaces es un aspecto que afecta la capacidad de computación de una red como la que se propone en la figura 7.1. En la tabla 7.2 se puede ver la velocidad de los 3 enlaces en cuestión.

Tabla 7.2. Velocidad de transferencia de datos teóricos de PCIe, USB y Ethernet

Protocolo	Velocidad del enlace
PCI Express 2 Gen	5,0 Gb/s
USB v2	480 Mb/s
Ethernet 802.3	1000 Mb/s

Las velocidades de transferencia que se exponen en la tabla 7.2, corresponden con los valores teóricos disponibles en las especificaciones de cada protocolo. En la práctica estos valores son difícilmente alcanzables. La cantidad de información con la que se calculan estas tasas de transferencia corresponde con toda la información que es capaz de cruzar por el enlace en un espacio de tiempo (información de control sumada a la carga útil), por lo cual es importante mencionar que el valor esperable para la tasa de transferencia de información útil en la implementación física de todos los enlaces es menor a la expuesta en la tabla.

La velocidad de transferencia de datos del protocolo PCIe que se muestra en la tabla 7.2, es la velocidad de una sola línea de datos. Esto quiere decir que 5,0 Gb/s corresponde con la tasa de transferencia para para un enlace de tipo de x1. El tipo de puerto PCIe que se encuentra en la tarjeta de desarrollo VC707 es de tipo x8 que según [18] provee un ancho de banda de 40 Gb/s.

La tasa de transferencia expuesta para el enlace USB es la correspondiente al modo de operación HIGH-SPEED, el cual está diseñado para manejar dispositivos de almacenamiento o de video.

Por último, el ancho de banda mostrado para el enlace de tipo Ethernet, es la máxima velocidad de Ethernet (IEEE 802.3) que permite el chip Ethernet PHY de tipo SGMII de la tarjeta VC707.

En conclusión, la interfaz que permite el mayor ancho de banda es PCIe, ya que ofrece 40 veces más velocidad si se compara contra Ethernet 802.3 y 80 veces más velocidad si la comparación se hace contra USB v2.

7.2.4 Paquetes y eficiencia de los datos

La estructura de los paquetes de datos afecta directamente la eficiencia de los datos que es lograble con cada uno de los protocolos. La eficiencia de los datos es la razón que existe entre la carga útil y la información de control de un paquete de datos. Este es un parámetro importante debido a que un enlace con alta tasa de transferencia de datos pero con una estructura de control compleja (que ocupa gran ancho de banda), podría ser superado por un enlace de menor desempeño y con una estructura de control más simple (ocupa ancho de banda menor).

Los encabezados de los paquetes de datos a diferencia de los datos son de longitud constante. Esto implica que indiferentemente del tamaño de la carga útil, es necesario reservar un ancho de banda fijo para el control del enlace. Lo cual reduce el ancho de banda para los datos, sin embargo es un gasto que se justifica, si se obtiene una estructura de comunicación robusta.

La longitud del encabezado va a afectar la eficiencia de los datos lograble conforme se varía la carga útil en un paquete. La eficiencia de datos se estimó para cada uno de los protocolos de acuerdo a los parámetros que se muestran en la tabla 7.3. La cual resume los diferentes campos de encabezado que se encuentran en los paquetes.

Es importante destacar que los campos detallados en esta tabla corresponden a modos de operación específicos de los protocolos PCIe y USB. En el caso de PCIe la información detallada es válida cuando este opera mapeado a memoria. Del mismo modo para USB la información de la tabla 7.3 corresponde al tipo de transferencia de datos Isócrono.

Estos modos de operación se escogieron teniendo en cuenta una posible implementación de la arquitectura de red que se muestra en la figura 7.1. El protocolo PCIe se escogió mapeado a memoria porque así los múltiples NSNs presentes en la figura 7.1 podrían compartir la memoria para que esta actúe como un medio de comunicación virtual.

En el caso de USB se escogió el modo de transferencia Isócrono, ya que este es el modo en la especificación USB para el cual se garantiza un determinado ancho de banda para un dispositivo periférico.

Tabla 7.3. Campos tomados en cuenta para cálculo de eficiencia [17, 19, 20]

Protocolo	Descripción de encabezado	Tamaño de encabezado
PCIe	2 bytes para número de secuencia, 4 bytes para de formato, longitud y direccionamiento del paquete	6 bytes
USB	8 bytes de sincronización, 2 bytes de identificador de paquete, 2 bytes identificador del dispositivo, 2 bytes de CRC y 24 bytes de retraso entre paquetes (Esta cantidad de bytes corresponde al encabezado de una transacción Isócrona, la cual se compone de un paquete de datos y un paquete TOKEN)	38 bytes
Ethernet 802.3	7 bytes de preámbulo, 1 byte de SFD, 12 bytes de direcciones de origen y destino, 2 bytes de longitud y 4 bytes de CRC	26 bytes

De acuerdo con los datos de la tabla 7.3 se calculó la eficiencia de datos contra la variación en el tamaño de la carga útil. Los resultados de este cálculo se graficaron en la figura 7.6.

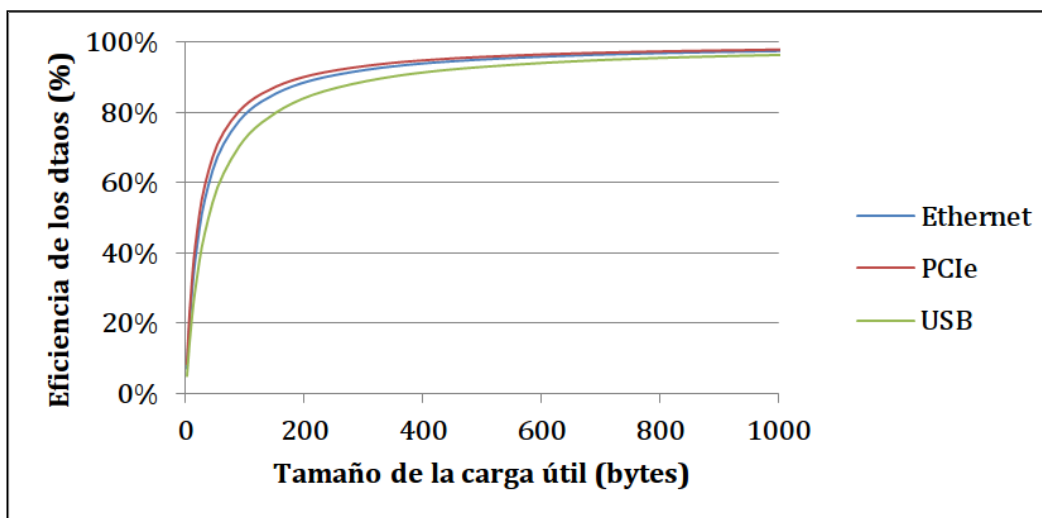


Figura 7.3. Eficiencia de datos PCIe, USB y Ethernet

En la figura 7.4 se puede ver que el comportamiento de los tres protocolos para tamaños de carga útil cercanos a 1 Kb es muy similar. Sin embargo, PCIe y Ethernet demuestran ser entre un 5 y 7% superiores para tamaños de carga útil menores que 256 bytes.

El grafico de la figura se extiende hasta 1Kb debido a que esta es la carga útil máxima del protocolo USB v2 para el modo de operación HIGH-SPEED. El nivel de eficiencia de PCIe y Ethernet adelante de este límite se mantiene aumentando de forma casi despreciable hasta alcanzar los valores de carga útil máxima de ambos protocolos (4 Kb para PCIe y 1,5 Kb para Ethernet).

En la evaluación de la eficiencia de los datos se tiene por lo tanto un virtual empate entre los tres protocolos, los cuales presentan una eficiencia de datos de más de 90 % para tamaños de carga mayores de 400 bytes, cifra que mejora hasta un 96% para tamaños de carga útil mayores a 600 bytes.

El protocolo USB presenta la particularidad de que el nivel de eficiencia mostrado fue calculado para el tipo de transferencia de mayor eficiencia de datos disponible en la especificación [19]. El problema que surge para este protocolo es el hecho de que para garantizar el ancho de banda en dicho tipo de transferencia el enlace se expone a potenciales pérdidas de datos [19]. Esto se debe a la falta de una estructura de manejo de errores bajo tal modo de operación. Eventualmente, esto podría comprometer la integridad de los datos de las simulaciones que se realicen en el NSN. Además, es difícil garantizar el ancho de banda para varios dispositivos periféricos funcionando bajo este tipo de transferencia dada la implementación de tipo HUB del protocolo USB.

Por último, la eficiencia de los datos presente en Ethernet y PCIe permitiría mayor flexibilidad en el intercambio de datos en una red de NSNs como la de la figura 7.1. Esto debido a que ambos presentan una buena eficiencia para un amplio espectro de valores de carga útil. Por lo tanto, cuando se implemente la red se podrían utilizar diversos tamaños de carga útil sin desperdiciar ancho de banda.

7.3 Análisis de IP cores disponibles para implementación

Los IP cores disponibles para implementar una interfaz de comunicación de alta velocidad se pueden ver en la tabla 7.1. En el diseño de un SoC sobre FPGA el área es un aspecto crítico debido que se busca minimizar la utilización de recursos en la interfaz y destinar la mayor cantidad de área para alojar el NSN o versiones incluso más grandes que las actualmente disponibles.

En la tabla 7.4, se reportan estimaciones de consumo de recursos de área de los IP cores de comunicación. El consumo de recursos reportado para el Integrated Block for PCI express y el AXI Ethernet Subsystem es una estimación provista por Xilinx en la guía de producto de ambos núcleos. Dicha estimación se deriva del reporte de post síntesis generado para chips FPGA Virtex-7. En el caso de USB el valor que se reporta corresponde a la estimación post síntesis del core generado para un chip FPGA Kintex 7.

Las estimaciones que se muestran en la tabla 7.4, dependen en gran medida de las configuraciones que se elijan para los módulos. El Integrated Block for PCI Express se escogió en 8 líneas de datos ya que es lo máximo que ofrece el puerto PCIe de la tarjeta

VC707. Bajo este modo de operación se permite además utilizar una carga útil de entre 128 y 1024 bytes lo cual da flexibilidad al diseño de la aplicación.

El consumo reportado para el AXI Universal serial Bus 2.0 Device, corresponde al consumo con módulo de DMA integrado dado que esta es la configuración que permitiría mayor tasa de transferencia de datos.

El Ethernet PHY de tipo SGMII presente en la tarjeta VC707, condicionó los parámetros de configuración para los cuales se reportó el consumo de área del AXI Ethernet Subsystem de la tabla 7.4. Esto debido a que se decidió incorporar la opción de checksum de recepción y transmisión al mismo, ya que así esta tarea no se tiene que realizar en el procesador del sistema. Además, no se tomaron en cuenta las opciones de manejo de información relacionada con VLAN debido a que esta es una tecnología que está destinada a la administración efectiva de redes locales de computadoras cuando se tienen varias redes en un sistema. En la aplicación que se desea implementar el número de redes no será mayor a uno.

Tabla 7.4. Utilización de recursos de área para IP cores de comunicación [21, 22, 23]

IP core	Modo de Operación	Utilización de recursos				
		LUT	FF	BRAM 18K	BRAM 36K	DSP
Virtex-7 FPGA Gen 3 Integrated Block for PCI Express v4.1	8 líneas, Capacidad de carga útil 128- 1024 bytes	3399	4818	12	3	0
AXI Universal Serial Bus (USB) 2.0 Device v5.0	Capacidad de DMA	2237	1952	0	3	0
AXI Ethernet Subsystem	SGMII, Checksum en transmisión y recepción y opciones para	5201	6812	0	4	0

	VLAN desactivadas					
--	----------------------	--	--	--	--	--

Mediante la información que se desprende de la tabla 7.4, se puede comparar el consumo de recursos de los IP cores. Es importante mencionar que si bien la estimación para el AXI USB Device no está generada para un chip Virtex 7, se puede considerar buena teniendo en cuenta que esta fue generada para un chip Kintex 7, el cual forma parte de la familia de la serie 7 de Xilinx. En esta familia la gama de chips de mayor desempeño es Virtex 7 seguida por Kintex 7. Por lo tanto, la huella de área del AXI USB Device se podría esperar menor o igual para un chip Virtex 7.

El AXI USB Device es por lo tanto el IP core que menos recursos de área utiliza, seguido por el AXI Ethernet Subsystem y por Integrated Block for PCI Express. Este IP core no es el que mejor rendimiento tiene en cuanto a transferencia de datos. Sin embargo, si la futura implementación tiene un presupuesto de área reducido para comunicaciones se podría recurrir a la utilización de este módulo.

7.4 Recomendaciones para interfaz de comunicación

La comunicación del NSN es un aspecto muy importante de la aplicación ya que va a determinar la escalabilidad de la misma y su practicidad. Es importante tomar en cuenta, cómo es que la aplicación podría ser usada por científicos que estudian el Olivo Inferior del cerebro. Ya que precisamente el mayor aporte de esta aplicación es la aceleración de los procesos de simulación que se realizan en diferentes investigaciones del área.

Los diferentes aspectos de los protocolos que han sido revisados, han dado a conocer las ventajas que tiene cada uno de los protocolos que se tomaron en cuenta. En la tabla 7.5 se puede encontrar un resumen de tales ventajas contra los aspectos que fueron analizados. Los protocolos se consideraron convenientes de acuerdo a como estos ayudaran en la implementación de una red como la de la figura 7.1, así como en el buen desempeño de una interfaz operando individualmente.

Tabla 7.5. Ventajas de los protocolos seleccionados en los parámetros revisados

Parámetro	Mejor (es) Protocolo (s)	Ventajas
Eficiencia de uso de Ancho de Banda	PCIe Ethernet	Un manejo eficiente del ancho de banda permite que sea posible que los dispositivos tengan el mismo ancho de banda para varios tamaños de red.

Escalabilidad de tamaño de red	Ethernet	Esto permite que se pueda ampliar fácilmente la red que se plantea inicialmente sin hacer cambios significativos en la implementación.
Velocidad de enlace	PCIe	Una velocidad de enlace alta, posibilita que las simulaciones se hagan lo más cercano posible a tiempo real.
Estructura de control	Ethernet	Una estructura de control descentralizada da una gran flexibilidad de diseño para una red multi-FPGA.
Eficiencia de los datos	PCIe y Ethernet	Una buena eficiencia de datos para diferentes tamaños de carga útil, permite una utilización eficiente del ancho de banda para varios tamaños de redes.
Consumo de área	USB v2	Minimizar el consumo de área permite reservar más espacio en el chip para el hardware de procesamiento de datos lo que eventualmente podría elevar la capacidad de computación.

Al considerarse la eficiencia de uso de ancho de banda, la velocidad de enlace y eficiencia de los datos en los paquetes se puede establecer que el protocolo PCIe es el ideal para formar una red de NSNs con la mayor capacidad de computación posible. La tecnología PCIe permitiría conectar entre 4 y 5 NSNs (cantidad de ranuras PCIe en una tarjeta madre).

El protocolo de Ethernet por su parte se podría utilizar para una implementación que brinde la posibilidad de probar distintas topologías para la red de la figura 7.1. Es posible que mediante una interfaz Ethernet se pueda construir una red más grande que con PCIe, sin embargo sería necesario evaluar si es viable en términos de costo tomando en cuenta que la comunicación entre NSNs se degradaría de gran forma con respecto a una comunicación de tipo PCIe.

Una interfaz Ethernet podría funcionar muy bien para implementarse con un único NSN, ya que este tipo de interfaz es de fácil manejo para el usuario. Además, el NSN se podría conectar y desconectar de la computadora en caliente lo cual representa comodidad para los científicos usuarios de la aplicación.

El nivel de difusión de la tecnología USB tendría un efecto similar al de Ethernet en la aceptación por parte del usuario. Por lo cual sería valioso explorar una implementación que integre esta interfaz, dado que incluso este es el IP core que tiene menor huella de área.

8 Conclusiones

Se diseñó e implementó un sistema anfitrión para FPGA basado en procesador Microblaze con capacidad de manejar la transferencia de datos entre el núcleo de simulación neuronal y un chip de memoria DDR SDRAM externo mediante IP cores de acceso directo a memoria. Además, dicho sistema ha sido dotado de un puerto UART para exportar los datos producidos durante simulaciones en el núcleo.

Se ha portado exitosamente el núcleo de simulaciones neuronales a dicho sistema anfitrión. Este ha sido adaptado para disminuir el uso de recursos de hardware. Su funcionalidad ha sido comprobada mediante la ejecución de simulaciones neuronales del olivo inferior sobre el sistema implementado. Dichas simulaciones probaron ser fieles a la respuesta esperada del algoritmo del NSN funcionando en software.

Se ha logrado conectividad Ethernet parcial del sistema anfitrión con una terminal Bash de Ubuntu por protocolo TCP/IP. No obstante, no ha sido posible garantizar conectividad 100% confiable, debido a problemas de compatibilidad y licenciamiento

de los IP cores necesarios para hacer funcionar esta interfaz en la plataforma FPGA destino.

Se ha propuesto una interfaz óptima para el sistema luego de un análisis que incluyó arquitectura, escalabilidad, estructura de control y eficiencia de los datos. Este análisis determino que la interfaz más adecuada para las necesidades actuales y futuras del NSN es una interfaz PCIe implementada con el Virtex-7 FPGA Gen 3 Integrated Block for PCI Express v4.1.

9 Recomendaciones

9.1 Optimización del sistema anfitrión y el NSN

9.1.1 Interfaces de comunicación de alta velocidad del sistema anfitrión

El sistema anfitrión necesita una interfaz con alta velocidad de transferencia de datos para exportar con mayor rapidez los resultados de las simulaciones que el NSN produce. Los problemas de compatibilidad y pérdida de paquetes en el enlace se pueden evitar utilizando una tarjeta que presente un Ethernet PHY de tipo MII (ver sección 6.4.3).

La interfaz más adecuada para el sistema anfitrión es la de tipo PCIe. Esta interfaz provee de un ancho de banda ideal para que el NSN pueda funcionar como acelerador de una computadora (ver sección 7.4). Además, esta interfaz permite un escalamiento aceptable con lo cual sería posible implementar redes de múltiples NSN.

9.1.2 Arquitectura de control e interfaces del NSN

Es posible mejorar el desempeño actual del NSN funcionando adentro del sistema anfitrión. La velocidad de transferencia de datos desde el NSN hacia la memoria DDR SDRAM se puede elevar mediante la utilización de un módulo DMA en configuración Scatter-Gather (ver sección 6.4.5). Esto evitaría que el procesador espere innecesariamente a que el NSN termine de operar y de esta forma se aprovecharía al máximo la velocidad de computación del mismo.

La forma en que el NSN consume y opera sobre los datos se puede modificar para que las simulaciones se puedan realizar por partes. Así, el NSN no tendría que retener el estado de toda la red neuronal sino que retendría solo los datos sobre los cuales está operando (ver sección 5.3). Esto permitiría hacer simulaciones de mayor tamaño dado que el estado de la red neuronal se retendría en la memoria DDR SDRAM de la tarjeta de desarrollo que es de mayor tamaño que la memoria del chip FPGA.

9.2 Metodología de aprendizaje en las herramientas

9.2.1 Documentación

El navegador de documentos de Vivado es una herramienta que enlista toda la documentación relacionada con el ambiente. Los recursos que se pueden acceder con el Navegador de documentos incluye las guías de producto de todos los IP cores, diseños de ejemplo y manuales de uso de Vivado IDE, Vivado HLS y el Kit de desarrollo de Xilinx. Por lo tanto, la utilización de esta herramienta es recomendada para todos los usuarios novatos en el ambiente.

9.2.2 Tarjeta de desarrollo

Xilinx libera diseños de ejemplo para los usuarios de Vivado. Estos diseños normalmente se acompañan de un informe técnico (paper) denominado Xilinx Application (XAPP). Es posible descargar estos ejemplos de la web como Bitstreams listos para programar en FPGA o como proyectos modificables de Vivado IDE. Los

diseños son normalmente dirigidos hacia tarjetas de desarrollo específicas. La tarjeta KC705 es la tarjeta de desarrollo para la cual existen mayor número de ejemplos, por lo cual se recomienda la obtención de la misma para acelerar el proceso de aprendizaje.

10 Bibliografía

[1] Smaragdos, G., Isaza, S., Van eijk, M., Sourdis, I. & Strydis, C. (2014, 26 de Febrero). *FPGA-based Biophysically-Meaningful Modeling of Olivocerebellar Neurons* Rotterdam: Departamento neurociencia, Centro Médico Erasmo.

[2] Izhikevich, E. (2004). *Which Model to Use for Cortical Spiking Neurons* San Diego, CA: The Neurosciences Institute.

[3] Flynn, M. & Luk, W. (2011). *Computer System Design* Hoboken, New Jersey: John Wiley & Sons.

[4] Dutt, N. & Pasricha, S. (2008). *On-Chip Communication Architectures: System on Chip Interconnect* Burlington, Massachusetts: Morgan Kaufmann Publishers.

[5] Digilent (2014, 11 de Septiembre). *Nexys4 DDR FPGA Board Reference Manual* Recuperado el 25 de Diciembre del 2015, de https://reference.digilentinc.com/_media/nexys4-ddr:nexys4ddr_rm.pdf

[6] Xilinx (2014, 02 de Abril). *Zynq-7000 SoC and 7 Series Devices Memory Interface Solutions v2.0* Recuperado el 28 de Diciembre del 2015, de http://www.xilinx.com/support/documentation/ip_documentation/mig_7series/v2_

0/ug586_7Series_MIS.pdf

[7] Xilinx (2014, 02 de Abril). *MII to RMI core v2.0 LogiCORE IP Product Guide* Recuperado el 28 de Diciembre del 2015, de http://www.xilinx.com/support/documentation/ip_documentation/mii_to_rmii/v2_0/pg146-mii-to-rmii.pdf

[8] Atmel (s. f.). *AT04055: Using the lwIP Network Stack* Recuperado el 28 de Diciembre del 2015, de http://www.atmel.com/Images/Atmel-42233-Using-the-lwIP-Network-Stack_AP-Note_AT04055.pdf

[9] Wolf, W. (2009). *Modern VLSI Design: IP-Based Design* (4 ed.) Boston: Pearson Education.

[10] Xilinx (2014, 17 de Diciembre). *7 Series FPGAs Overview* Recuperado el 03 de Enero del 2016, de http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[11] Xilinx (2014, 2 de Abril). *Microblaze Processor Reference Guide* Recuperado el 24 de diciembre del 2015, de http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_2/ug984-vivado-microblaze-ref.pdf

[12] Xilinx (2015, 24 de Junio). *AXI Reference guide* Recuperado el 03 de Enero del 2016, de http://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf

[13] ARM (2010, 03 de Marzo). *AMBA 4 AXI4-Stream Protocol* (1 ed.)

[14] Xilinx (2013, 19 de Junio). *UG902 High-Level Synthesis* Recuperado el 26 de Septiembre del 2015, de http://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_2/ug902-vivado-high-level-synthesis.pdf

[15] Xilinx (2009, 02 de Diciembre). *OS and Libraries Document Collection* Recuperado el 04 de Enero del 2016, de http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/oslib_rm.pdf

[16] Xilinx (2014). *Xilinx Software Development Kit Help* Recuperado el 04 de Enero del 2016, de http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_3/SDK_Doc/index.html

[17] Saadé, J., Pétrot, F., Picco, A., Huloux, J. & Goulahsen, A. (2013). *A System-Level*

Overview and Comparison of Three High-Speed Serial Links: USB 3.0, PCI Express 2.0 and LLI 1.0 Grenoble: STMicroelectronics.

[18] PCI-SIG (2006, 20 de Diciembre). PCI Express Base Specification Revision 2.0.

[19] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips (2000, 27 de Abril). Universal Serial Bus Specification (2 ed.)

[20] IEEE Standards Association (2012, 28 de Diciembre). IEEE Standard for Ethernet New York: IEEE.

[21] Xilinx (2012, 24 de Abril). Virtex-7 FPGA Gen3 Integrated Block for PCI Express Product Guide Recuperado el 26 de Diciembre del 2015, de http://www.xilinx.com/support/documentation/ip_documentation/pcie3_7x/v1_1/pg023_v7_pcie_gen3.pdf

[22] Xilinx (2015, 18 de Noviembre). AXI Universal Serial Bus (USB) 2.0 Device v5.0 LogiCORE IP Product Guide Recuperado el 11 de Enero del 2016, de http://www.xilinx.com/support/documentation/ip_documentation/axi_usb2_device/v5_0/pg137-axi-usb2-device.pdf

[23] Xilinx (2014, 01 de Octubre). AXI Ethernet Subsystem v6.2 Product Guide Recuperado el 11 de Enero del 2016, de http://www.xilinx.com/support/documentation/ip_documentation/axi_ethernet/v6_2/pg138-axi-ethernet.pdf

[24] Xilinx (2015, 18 de Noviembre). AXI Interconnect v2.1 LogiCORE IP Product Guide Recuperado el 14 de Enero del 2016, de http://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v2_1/pg059-axi-interconnect.pdf

11 Anexos

11.1 Tabla de configuraciones físicas del ip core controlador de memoria

Parámetro	Valor
Tipo de memoria	DDR2 SDRAM
Periodo de reloj máximo	3000 ps (ancho de banda 667 Mbps)
Periodo de reloj recomendado	3077 ps (ancho de banda 650 Mbps)
Numero de parte	MT47H64M16HR-25E
Ancho de palabra	16 bits
Máscara de datos	Habilitado
Pin de selección de chip	Habilitado
Rtt (nominal) On-die termination	50 ohms
Voltaje de referencia interno	Habilitado
Impedancia interna	50 ohms