

PONTIFICIA UNIVERSIDAD JAVERIANA CALI

Computación de Borde con FPGA para IoT

por

Manuel A. Valderrama

Dirigido por

Eugenio Tamura

Monografía presentado en cumplimiento parcial para
el título de Magister en Ingeniería

en la
Facultad de Ingeniería
Departamento de Electrónica y Ciencias de la Computación
Maestría en Ingeniería

29 de julio de 2018

Índice General

Índice General	III
Índice de Figuras	V
Índice de Tablas	VII
Agradecimientos	IX
Abreviaciones	X
1 Introducción	5
2 Definición del Problema	7
2.1 Procesamiento de la Información	7
2.2 Costo del Procesamiento y Almacenamiento	7
3 Objetivos	9
3.1 Objetivo General	9
3.2 Objetivos Específicos	9
3.3 Resultados Esperados	10
4 Justificación	11
4.1 Aceleración del Procesamiento	11
4.2 Reducción de Costos	11
4.3 Enriquecimiento de la Información	11
4.3.1 Fusión de Datos	12
5 Alcances	13
6 Marco Teórico	15
6.1 Estado del Arte	15
6.1.1 Computación de Borde	15
6.1.2 Computación Consciente del Contexto	17
6.1.3 Monitoreo y predicción de inundaciones	19
6.2 Síntesis de Alto Nivel <i>HLS</i>	20
6.2.1 Herramientas de Síntesis	21
6.2.1.1 Esquema General de Síntesis	22
6.2.1.2 Optimizaciones de Hardware	22
6.2.3 Software de Síntesis de Hardware Vivado Design Suite	33
6.4 Red Neuronal Artificial	35
6.4.1 Neurona Básica	35
6.4.1.1 Función de Propagación	36
6.4.1.2 Función de Activación	36
6.4.2 Perceptrón Multi-Capa	37
7 Recursos	39
7.1 Hardware	39
7.2 Software	39

8 Metodología	41
9 Elección de la Plataforma de Desarrollo	43
9.1 Elección de la plataforma	43
10 Diseño de Software y Hardware	45
10.1 Diseño de Hardware	45
10.1.1 Diseño de la Red Neuronal Artificial	46
10.1.2 Función de salida de la red neuronal	47
10.1.3 Código fuente Red Neuronal Artificial	47
10.1.4 Puertos de entrada y salida AXI	48
10.1.5 Optimizaciones de Hardware	49
10.1.6 Implementación del hardware	51
10.2 Diseño de Software	54
10.2.1 Petalinux SDK	54
10.2.1.1 Configuración de Petalinux SDK	56
10.2.2 Diseño del driver	58
10.2.3 Diseño del controlador de reconfiguración	58
11 Diseño Experimental	63
12 Comparación Modelo de la ANN	65
13 Análisis de Resultados	69
13.1 Benchmark	69
13.2 Análisis de tiempo y recursos	71
13.2.1 Tiempo de ejecución	71
13.2.2 Recursos utilizados	72
14 Conclusiones	75
 Bibliografía	77

Índice de Figuras

6.1	Evolución y aplicaciones de IoT[1].	15
6.2	Distribución del Internet de las Cosas (Operación)[2].	16
6.3	Escalas de tiempos de servicios de la computación de borde vs la nube[3].	17
6.4	Red de sensores estructurada por capas [4].	18
6.5	Directiva ARRAY_MAP en modo horizontal [5]	24
6.6	Distribución en RAM del Mapeo Horizontal [5]	24
6.7	Directiva ARRAY_MAP en modo vertical [5]	25
6.8	Distribución en RAM del Mapeo Vertical [5]	25
6.9	Modos de operación de la directiva ARRAY_PARTITION [5]	25
6.10	Modos de operación de la Directiva ARRAY_RESHAPE [5]	26
6.11	Funcionamiento de la Directiva DATA PACK [5]	27
6.12	Sistema secuencial [5]	28
6.13	Sistema concurrente a nivel de tareas [5]	28
6.14	Funcionamiento de la directiva DATAFLOW [5]	28
6.15	Funcionamiento de la Directiva LOOP MERGE [5]	30
6.16	Directiva PIPELINE aplicada a funciones y bucles [5]	31
6.17	Comportamiento del modo PIPELINEREWIND [5]	31
6.18	Comportamiento del modo PIPELINEFLUSH [5]	32
6.19	Funcionamiento de la Directiva UNROLL [5]	32
6.20	Vivado Design Suite GUI	34
6.21	Neurona básica	35
6.22	MLP completamente conectada	37
10.1	Diseño de la red neuronal	46
10.2	Red neuronal en árbol	47
10.3	Memory burst transaction [6]	49
10.4	Latencia e intervalo de las optimizaciones en float	50
10.5	Ganancia en rendimiento usando float	51
10.6	Latencia e intervalo de las optimizaciones fijo	51
10.7	Ganancia en rendimiento en float	52
10.8	Diferencia entre punto fijo y flotante	52
10.9	Implementación final de hardware	53
10.10	Esquema de driver en espacio de usuario[7].	55
10.11	Estructura de arranque de sistemas ZYNQ [8]	56
10.12	Diagrama de clases de la aplicación	59
10.13	Esquema de funcionamiento ANN	60
10.14	Esquema de funcionamiento ANN reconfigurada	60

10.15	Esquema de funcionamiento múltiples aplicaciones	61
11.1	Esquema de conexión	64
12.1	Curva ROC de detección	65
12.2	Correlación cruzada entre modelos	67
13.1	Benchmark resultados de la implementación	70
13.2	Rendimiento del acelerador en Hardware	71
13.3	Recursos utilizados en la FPGA	72
13.4	Factor de ganancia recursos contra tiempo	73

Índice de Tablas

6.1	Atributos que causan o previenen inundaciones	19
6.2	Sintetizadores de alto nivel más comunes	21
6.3	Directivas disponibles en Vivado HLS [5]	23
6.4	Mapa de Direcciones del Zynq 7000 Processing System [8]	34
6.5	Funciones de Activación	37
9.1	Familias de FPGA	43
9.2	Comparación tarjetas de desarrollo	44
10.1	Tabla uso de recursos post síntesis	53

Agradecimientos

El autor agradece la cooperación de todos los miembros de la alianza *Centro de Excelencia y Apropiación en Internet de las Cosas (CEA-IoT)*. El autor agradece también a todas las instituciones que soportaron este trabajo: el *Ministerio de Tecnologías de la Información y las Comunicaciones - MinTIC* y el *Departamento Administrativo de Ciencia, Tecnología e Innovación - Colciencias* a través del *Fondo Nacional de Finciamiento para la Ciencia, la Tecnología y la Innovación Francisco José de Caldas* (Proyecto FP44842-502-2015).

Abreviaciones

AMBA	Advanced Microcontroller Bus Architecture
ANN	Artificial Neural Network
AXI	Advanced eXtensible Interface
BRAM	Block Random Access Memory
DSP	Digital Signal Processor
FF	Flip Flop
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
HLS	High Level Synthesis
IoT	Internet of Things
IP	Intellectual Property
LUT	Look Up Table
MLP	Multi Layer Perceptron
OS	Operating System
PL	Programmable Logic
PS	Processing System
QoS	Quality of Service
RTL	Register Transfer Language
SoC	System on a Chip
TDP	Thermal Design Power

Resumen

Muchas de las soluciones de software que se pueden encontrar en el mercado actualmente tienen algún grado de procesamiento en la nube, lo que proporciona hardware y software bajo demanda, brindando así al desarrollador herramientas que hace 10 años eran impensables y hoy están al alcance de un click.

Las virtudes de la computación en la nube, aunque amplias, implican altos costos operativos y en algunos casos limitan el acceso no por su presupuesto sino por la propia naturaleza de la tecnología involucrada en las comunicaciones (vía Internet), que adiciona dificultades de tipo operativas en aquellas aplicaciones en las que el tiempo de respuesta es un factor clave. En estos casos, la computación en la nube no se puede tomar ni siquiera como una opción.

En este trabajo se pretende hacer uso de la computación de borde y la aceleración por hardware usando FPGAs como una herramienta que optimiza el ancho de banda y la cuota de datos enviados a la nube para su posterior procesamiento, con el objetivo de incrementar el número de proyectos de investigación en el ramo del Internet de las Cosas en el país.

Abstract

Currently, many of the software solutions that can be found in the market have some degree of processing in the cloud, thus providing hardware and software on demand, bringing to the developer tools that 10 years ago were unthinkable but today can be reached with a single click.

The virtues of cloud computing, although broad, imply high operating costs and in some cases limit the access not necessarily because of budget, but by the very nature of the technology involved in communications (via the Internet), which adds operational difficulties in those applications where the response time is a key factor. In these cases, cloud computing cannot be even taken as an option.

In this work, we will use edge computing and hardware acceleration using FPGAs as a tool that allows optimizing the bandwidth and data rate sent to the cloud for further processing, with the aim of increasing the number of research projects in the field of Internet of Things in the country.

Capítulo 1

Introducción

La computación en la nube como medio de procesamiento y análisis se convirtió en una herramienta con recursos casi ilimitados para los desarrolladores desde su masificación hace unos pocos años. La computación en la nube cuenta con múltiples bondades; entre ellas encontramos el Internet de las Cosas (*IoT*) como una de las aplicaciones más importantes dada la gran cantidad de datos y recursos necesarios para procesar los altos volúmenes de información generados por las redes de sensores a los que usualmente se encuentra acoplada. Hoy, encontramos el término computación de borde, que aunque es un esquema relativamente nuevo de computación, ha llamado la atención de múltiples investigadores dado que implica llevar parte de los servicios de la nube al lugar donde realmente se producen los datos [3]. Realizar parte del procesamiento en el borde le permite al desarrollador acceder a aplicaciones sensibles a la latencia incapaces de operar directamente sobre la nube, disminución de los costos de analítica, reducciones en el almacenamiento y volumen de datos procesados al interior de la nube, entre otros.

Aparte de las diferentes virtudes que ofrece la computación de borde, también encontramos marcadas limitaciones como la reducida potencia de cálculo de algunos de los dispositivos. En este sentido, la aceleración por hardware y la reconfiguración parcial dinámica, juegan un rol importante, aportando potencia de cálculo en un espacio reducido en función de las necesidades de procesamiento requeridas por la aplicación.

El desarrollo de este documento, resultado del trabajo alrededor de la temática anteriormente enunciada, tiene la siguiente estructura:

- El capítulo 1 contiene la introducción al documento.
- En el capítulo 2 se explica la definición del problema al que este documento quiere abordar y dar solución.
- El capítulo 3 enumera los diferentes objetivos que se pretenden cumplir al final del proyecto.
- El capítulo número 4 explica la justificación de la necesidad de realizar y llevar a cabo el proyecto.

- En el capítulo número 5 se exponen los alcances del proyecto a nivel de tareas a realizar basado en los diferentes objetivos.
- Todo el capítulo 6 expone el marco teórico del tema general que aborda todo el proyecto.
- En el capítulo número 7 se discriminan los recursos necesarios para llevar a cabo el proyecto a buen término.
- El capítulo 8 expone la metodología usada en el proyecto.
- En el capítulo número 9 se exponen los criterios de elección para la plataforma de desarrollo.
- El diseño de software y hardware se trata en el capítulo 10; aquí se muestra sistemáticamente el proceso de diseño para ambas partes del proyecto.
- El capítulo número 11 trata el diseño experimental; en este se tratan las diferentes actividades para realizar las pruebas con las que se obtendrán los diferentes resultados.
- En el capítulo número 12 se expone comparativamente la red neuronal software usando Matlab y la red neuronal implementada en hardware con el objetivo de validar el modelo final implementado.
- El análisis de los resultados de la implementación se trata en el capítulo número 13.
- Finalmente, las conclusiones se exponen en el capítulo número 14.

Capítulo 2

Definición del Problema

Determinar de forma rápida y eficaz la solución a un algoritmo complejo visto desde una óptica de IoT implica tener en consideración múltiples factores. Uno de ellos es la potencia de cálculo usualmente medida en Tera Flops y la cantidad de datos a analizar sin que se saturen los recursos disponibles en el diseño de la arquitectura de IoT. Un esquema de operación simple basado en Internet of Things podría no cumplir con los requisitos necesarios para la operación dependiendo del volumen de los datos y el tipo de procesamiento que se realice sobre los mismos; relegar todo este procesamiento a la nube podría ser simplemente muy costoso y/o imposible de implementar debido a limitaciones netamente técnicas.

2.1. Procesamiento de la Información

Resolver el problema de la potencia de procesamiento desde un esquema de IoT ha traído consigo múltiples retos que no pueden ser resueltos por un esquema de operación basado en *Cloud Computing*, es decir, que problemas como la alta latencia, la pérdida de conexión entre la fuente y el repositorio de datos (pérdida de información), el uso del ancho de banda y la administración de los recursos son inevitables, a menos que la aplicación sea insensible a ellos. Uno de los problemas más comunes es claramente la latencia; esta implica enviar grandes cantidades de información a través de Internet usando un ancho de banda determinado puede retrasar el procesamiento hasta que toda la información esté disponible en el servidor de la nube; este problema se suma al tiempo que tarda en responder el servidor, imponiendo unas restricciones en el tiempo de respuesta mínimo que define claramente la IEEE [9, 10].

2.2. Costo del Procesamiento y Almacenamiento

El procesamiento centralizado ofrecido por los proveedores de computación en la nube es robusto en general, pero se ve limitado debido a los altos costos operativos para cierto

tipo de aplicaciones que no requieren mucha potencia de cálculo, despliegues de gran cantidad de servidores y altos niveles de tráfico de red hacia los servidores.

Por otra parte, el almacenamiento en la nube también implica altos costos, originados tanto por la persistencia de los datos como por la cantidad de transacciones que se realizan al medio de almacenamiento [11–14].

Con todo esto, se hace necesario encontrar soluciones capaces de analizar, transformar, tomar decisiones y enviar gran cantidad de información conservando el TDP característico de los procesadores orientados a IoT.

Capítulo 3

Objetivos

3.1. Objetivo General

Proponer una plataforma de computación de borde para IoT que proporcione aceleración por hardware que, a través de la reconfiguración parcial, adicione funcionalidad de manera dinámica, con el objetivo de permitir que la plataforma se ajuste de acuerdo al contexto de ejecución.

3.2. Objetivos Específicos

- Definir los requerimientos y los lineamientos requeridos por la plataforma.
- Desarrollar un módulo de fusión de datos sintetizable en hardware, instrumentado y empaquetado como módulo IP.
- Integrar e implementar los módulos IP de fusión de datos como plataforma consciente del contexto usando reconfiguración parcial.
- Comparar el rendimiento de la plataforma desarrollada con una implementación diseñada completamente en software.

3.3. Resultados Esperados

Se espera tener una plataforma mixta hardware-software que sea capaz de procesar datos provenientes de múltiples sensores de variables físicas relacionadas entre sí para combinarlos en un estimado que posteriormente será enviado a la nube para su procesamiento y análisis.

Para efectos de desarrollo se dispondrá de una base de datos con la que la plataforma interactuará simulando la captura de datos a lo largo del tiempo.

- Los datos procesados contendrán información más rica y útil que la proporcionada por los valores individuales.
- El tiempo de procesamiento de los datos usando aceleración por HW deberá ser menor que su contraparte SW.

Capítulo 4

Justificación

4.1. Aceleración del Procesamiento

Una plataforma de procesamiento concurrente como una FPGA provee aceleración del procesamiento según la configuración adoptada por el desarrollador. Dependiendo de la cantidad de hardware asociado y el diseño del algoritmo, una FPGA es capaz de acelerar varias veces los resultados por unidad de tiempo a diferencia de su contraparte software, como ocurre por ejemplo, en casos como la ejecución de consultas a bases de datos, criptografía, simulaciones, entre otros [15–18].

4.2. Reducción de Costos

Llevar parte de los servicios de la nube al campo implica la reducción de costos operativos. Esta migración, conocida como computación de borde, implica para el desarrollador invertir más recursos en servidores locales de procesamiento y almacenamiento de datos que tienen un costo fijo de adquisición y un costo bajo de operación en contraste con los servicios de la nube [10, 11].

4.3. Enriquecimiento de la Información

Los datos en crudo de una o múltiples fuentes no aporta la suficiente información para derivar conclusiones en un modelo cuyo objetivo es realizar el análisis de los datos. El procesamiento de datos en el borde permite al desarrollador analizar muchos más detalles, tomar decisiones o incluso observar y predecir el comportamiento de las variables implicadas en la medición, con lo cual el desarrollador obtiene una visión más amplia de lo que ocurre en el medio observado [4, 19, 20].

4.3.1. Fusión de Datos

La fusión de datos como práctica de enriquecimiento de la información es bien conocida. Esta, permite al desarrollador ampliar el espectro de la información disponible a partir de diferentes fuentes de información; estas pueden ser diferentes tipos de datos, sensores o incluso imágenes. En el caso de un modelo de detección de inundación es posible usar la información proveniente de diferentes nodos de medición a lo largo de un río y combinarlos usando Redes Neuronales Artificiales para determinar el riesgo de inundación dentro de una ventana de tiempo específica [21–23].

Con esto, una plataforma de procesamiento de borde equipada con software y hardware especializado se encuentra en la capacidad de procesar grandes bloques de datos provenientes de campo, siendo capaz de ajustar, según la necesidad de procesamiento los recursos de hardware con el objetivo de reducir la cantidad de datos enviados a la nube, reducir los volúmenes de almacenamiento, reducir la latencia al actuar directamente sobre el origen de los datos y a su vez facilitar al desarrollador el procesamiento y analítica aplicable a los datos.

Capítulo 5

Alcances

- 1. Definir los requerimientos y los lineamientos requeridos por la plataforma**
 - Definir las características de la plataforma según la tecnología de cómputo y fuentes de datos disponibles.
- 2. Desarrollar un módulo de fusión de datos sintetizable en hardware, instrumentado y empaquetado como módulo IP**
 - Desarrollo de una Red Neuronal Artificial en C++ que permita el procesamiento y fusión de los datos provenientes de múltiples sensores para luego ser sintetizado usando HLS con optimizaciones a nivel de hardware que permitan un alto nivel de concurrencia.
- 3. Integrar e implementar los módulos IP de fusión de datos como plataforma consciente del contexto usando reconfiguración parcial**
 - Se espera desarrollar una plataforma que procese múltiples bloques de datos dependiendo del estado del sistema, cantidad de nodos sincronizados o frecuencia de muestreo requerida. Esto se deberá garantizar reconfigurando dinámicamente la FPGA con lo cual se añadirían más recursos de hardware al procesamiento de la información que llega al nodo de computación de borde.
 - La cantidad de señales en paralelo que se puedan analizar dependerá del alcance número 1, los recursos de hardware (cantidad de bloques lógicos de la FPGA) con los que cuente la plataforma de desarrollo y el resultado final de las optimizaciones de hardware y síntesis.
- 4. Comparar el rendimiento de la plataforma desarrollada con una implementación diseñada completamente en software**
 - Análisis de rendimiento de los módulos según el tiempo de procesamiento y los recursos de HW utilizados contra una implementación netamente SW.

Capítulo 6

Marco Teórico

6.1. Estado del Arte

6.1.1. Computación de Borde

El IoT, definido como una red distribuida de dispositivos heterogéneos (sensores, actuadores y procesadores) que intercambian mensajes entre ellos sin intervención humana, ha cobrado particular interés por parte de los investigadores, dado que miles de millones de dispositivos se estarán comunicando en un futuro cercano. La Figura 6.1, muestra la tendencia de la capacidad de procesamiento y la cantidad de servicios que el IoT puede ofrecer, a la par que también aumentan los retos de seguridad y administración de los datos [1, 24].

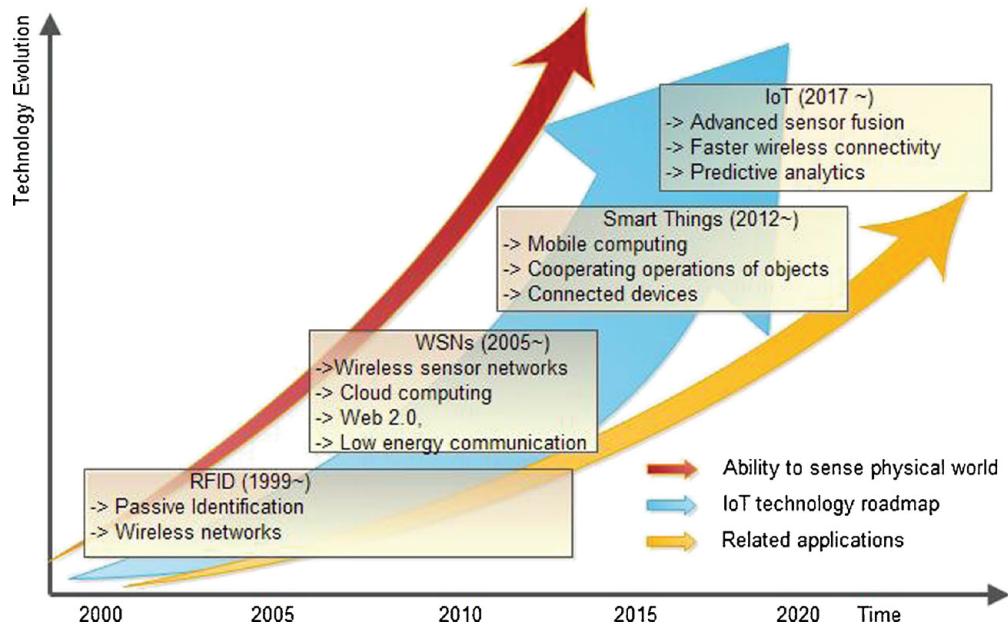


FIGURA 6.1: Evolución y aplicaciones de IoT[1].

Con el incremento en la cantidad de servicios, también surgen otras necesidades. Algunas de ellas y quizás las más críticas en la arquitectura pensada para IoT, son la latencia de los datos y el rendimiento del sistema [2]. Por ello, se propone llevar algunos de los servicios de la nube más cerca del nivel al que se encuentran los datos así como emplear múltiples niveles, según las necesidades de las diferentes aplicaciones, como lo muestra la Figura 6.2. En el caso del nivel *Extremo* ó *MIST*, se tienen niveles de procesamiento bajos; en este nivel, son los mismos nodos sensores y actuadores, los que toman decisiones operativas a un nivel muy bajo dadas las limitaciones en los procesadores que requieren para funcionar. A nivel de *Borde* ó *FOG*, se tiene mayor potencia de cálculo permitiendo realizar labores de almacenamiento, analítica, toma de decisiones, etc. Esto es posible, dado que en algunas aplicaciones, el dispositivo de borde, también conocido como Gateway, hace la labor de establecer un puente entre el extremo y la nube, contando con una radio potente y una conexión a Internet y alimentación eléctrica estables; es por estas necesidades que el gateway dentro de su concepción debe contar con hardware más potente que los demás dispositivos. Finalmente, sobre la última capa se encuentra la *Nube* ó *CLOUD*, el cual es el pilar más importante del Internet de las Cosas, ofreciendo múltiples servicios, con recursos prácticamente ilimitados en procesamiento, capaz de escalar dinámicamente conforme las necesidades lo requieran [3].

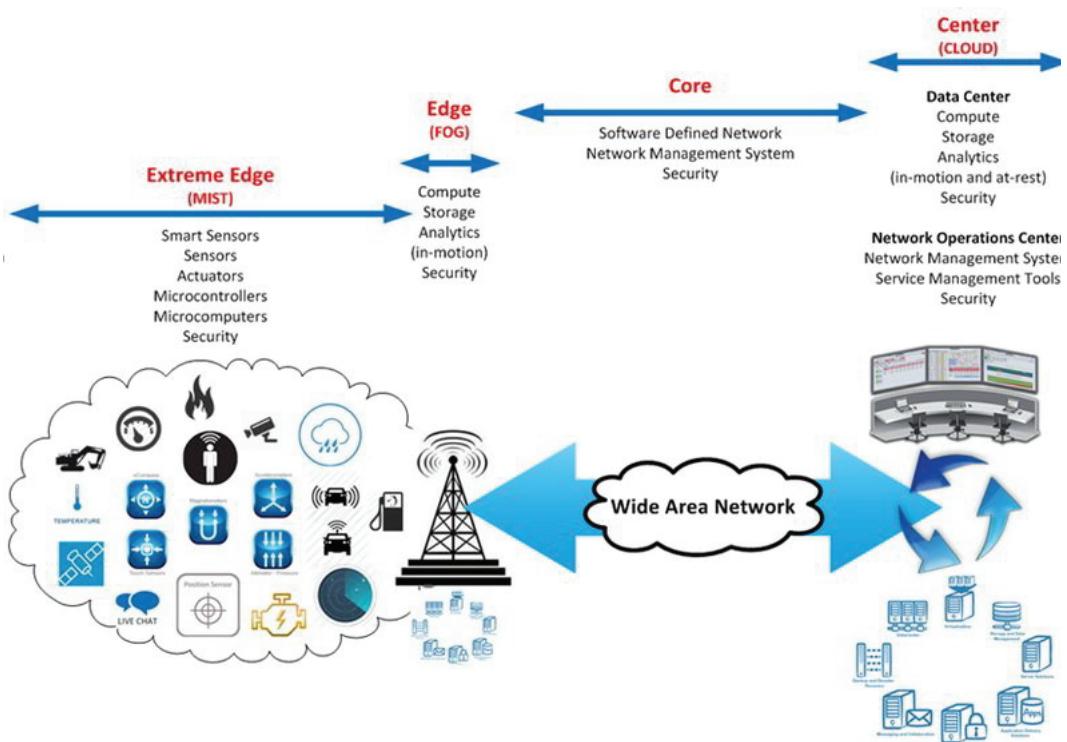


FIGURA 6.2: Distribución del Internet de las Cosas (Operación)[2].

Entonces, como se sugiere, la computación de borde está orientada a comunicación entre dispositivos, con una orientación a un almacenamiento y toma de decisiones eficientes. Esto implica directamente reducción en tiempo de procesamiento, almacenamiento, visualización y manejo de los datos como se muestra en la Figura 6.3.

Al final, la nube como el pilar fundamental, ofrece potencia de cálculo y por ende más servicios [25], pero algunas de las aplicaciones únicamente requieren parte de estos servicios, que pueden ser resueltos en el borde por un gateway. Las aplicaciones que hacen uso de la computación de borde pueden ser de cualquier tipo; por ejemplo, encontramos modelos de programación para aplicaciones que requieran procesamiento de borde a gran escala [26], Gateways inteligentes capaces de enrutar de mejor manera los paquetes de red que tienen como destino la nube [27], modelos de *centros de datos* para el procesamiento de datos efectivos en el borde [28], QoS distribuido dentro de la red de IoT [29]. Como se ve, el concepto es ampliamente usado, pero el terreno sobre la aceleración por hardware sigue sin ser explorado, dadas las complicaciones que requiere su programación. Con sintetizadores de alto nivel apenas madurando, se espera avanzar hacia la creación de APIs ó middlewares que permitan a los desarrolladores incrementar la potencia de cálculo en el borde como lo sugieren las patentes [30, 31].

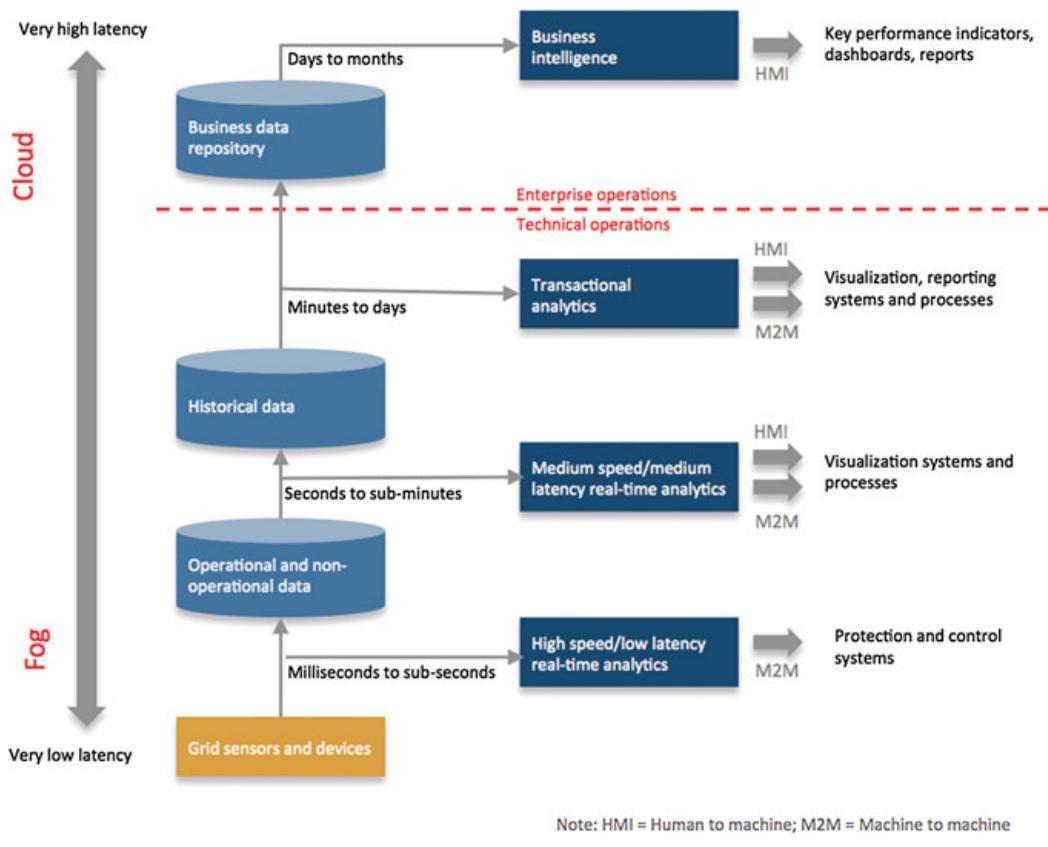


FIGURA 6.3: Escalas de tiempos de servicios de la computación de borde vs la nube[3].

6.1.2. Computación Consciente del Contexto

Cuando se habla de grandes cantidades de datos recolectada por miles de sensores dentro de un esquema de IoT, se pone sobre la mesa una pregunta: ¿Cómo interpretar los datos?. Es importante notar que por más información que se tenga disponible, si no se analiza, interpreta y entiende dentro de un contexto dado, esta información no tiene valor alguno.

La computación consciente del contexto, vista desde una perspectiva de IoT, propone una forma de mediar con la información desde diferentes flancos. Actualmente hablamos de la implementación de soluciones middleware capaces de añadir recursos que se encarguen de abstraer, fusionar y comprimir los datos, además de añadir servicios como QoS, paradigmas de programación, adaptabilidad, escalabilidad, recursos de hardware y lo más importante, seguridad de los datos [32].

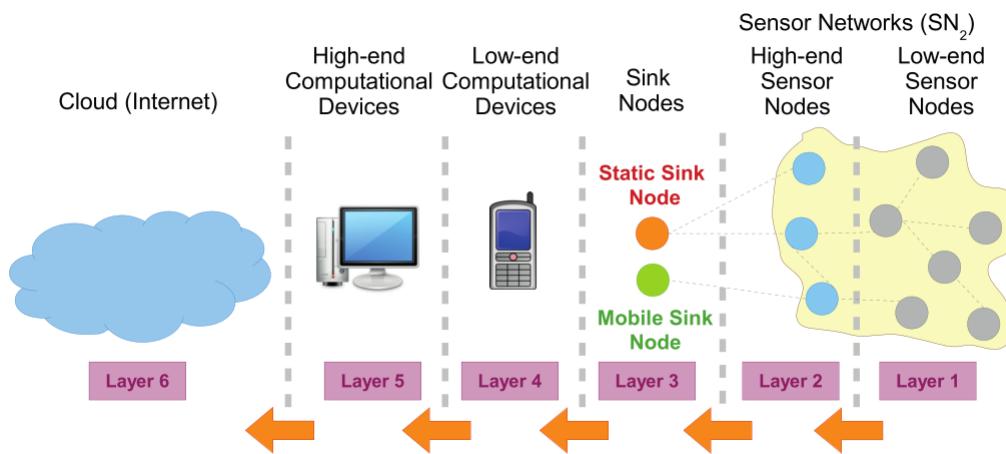


FIGURA 6.4: Red de sensores estructurada por capas [4].

La Figura 6.4 ofrece una vista general de la forma como se distribuye una red de sensores por capas, desde los elementos más básicos hasta los más complejos. En las primeras tres capas se observa el flujo de datos que va desde los dispositivos menos potentes hasta los más potentes en la capa 3, que muchas veces hacen las veces de enrutadores de borde de red. En las capas 4 y 5 se tienen dispositivos computacionalmente mucho más potentes; estos son conocidos como Gateways, capaces de llevar la información del campo hacia la nube. Finalmente, se tiene la Nube en Internet, donde la información capturada puede ser transformada y procesada para su análisis. Cada una de las capas cumple un rol específico en el que la eficiencia es la clave[33]. Nótese que el procesamiento se podría realizar entre las dos primeras capas reduciendo considerablemente el costo de las comunicaciones; sin embargo, las limitaciones en capacidad de cómputo y energéticas impedirían realizar un análisis detallado de la información recolectada por la red.

En función de esta necesidad, los investigadores definieron siete grandes características que permiten enmarcar una solución para el IoT dentro del contexto de la computación consciente del contexto que, necesariamente, dadas las condiciones, deben ser realizadas entre las capas cuatro a seis [34]. Estas son: inteligencia, arquitectura, sistemas complejos, magnitud, tiempo, espacio y “Todo como un Servicio” o “All as a Service”. Algunas de estas cobran particular interés, como inteligencia, pues determina que para usar la información recolectada por diferentes fuentes en crudo se debe transformar para lograr lo que se denomina *información de alto nivel*, que lo que quiere decir es que esta información nueva enriquecida y puesta en un contexto, pueda ser usada como insumo para otro tipo de operaciones. Otra de las características que cobra particular interés es la

de tiempo, pues dados los altos volúmenes de transacciones de datos se hacen necesarios mecanismos que permitan operar en tiempo real, siendo una restricción importante cuando se modela una solución de este tipo.

A partir de este punto, se introduce el concepto “*Principios Administrativos de Diseño Consciente del Contexto*”, que básicamente se define como, los requerimientos para la aplicación de un middleware consciente del contexto de operación, que se encuentre en capacidad de analizar, transformar e interactuar con el medio para darle sentido a la información [35–37]. Uno de los requerimientos más tangibles es el llamado *optimización de recursos*, el cual implica que, cualquier mejora en los recursos de procesamiento, por más pequeña que sea, tendrá un gran impacto en el procesamiento dados los altos volúmenes de información que el IoT puede llegar a procesar.

Finalmente, tenemos la definición estricta de Schilit [38] en la que, una aplicación consciente del contexto, no es más que un dispositivo que cumple unas reglas básicas según la operación que realiza, desde la toma simple de decisiones basado en eventos hasta la resolución de los mismos usando ventanas de oportunidad en las que las condiciones del evento establecen las fronteras.

6.1.3. Monitoreo y predicción de inundaciones

Las inundaciones son uno de los fenómenos naturales más comunes y destructivos que puede enfrentar la humanidad; esto debido a los altos niveles de precipitación, el cambio climático, y represamientos ya sean naturales o artificiales. Existen diferentes atributos que pueden favorecer o prevenir una inundación; estos son mostrados en la Tabla 6.1. En ella, se dispone información, que según estudios [39], son los factores determinantes de mayor ocurrencia en lugares donde ocurren inundaciones.

Categoría	Atributos	Aumento del orden de ocurrencia ->					
Causantes de Inundación	Precipitación Media	Muy poca	Poca	Moderada	Alta	Extrema	
	Velocidad de Captación	Constante	Poco flujo	Flujo moderado	Flujo Alto	Flujo Extremo	
	Orden del flujo	1	2	3	4	5	
Prevención de Inundación	Densidad del bosque por área	Bosque Denso >1000	Bosque Espeso 800-1000	Bosque Pequeño 500-800	Tierra de Cultivo 100-500	Tierra Estéril <100	
	Estación actual	Invierno	Primavera	Otoño	Verano	Monzón	
	Sistemas de drenaje	El mejor	Bueno	Promedio	Malo	La peor	
	Tipo de suelo	Arenoso	Granular	Orgánico	Arcilla	Compacto	

TABLA 6.1: Atributos que causan o previenen inundaciones

Las investigaciones alrededor del tema van desde monitorear las precipitaciones remotamente desde satélite[40, 41] con resultados observables que muestran altos niveles de incertidumbre, pasando por modelos fuzzy que se ven altamente beneficiados por su habilidad de crear relaciones unificadas alrededor de la información hidrológica, hasta el uso de redes neuronales para determinar los niveles del agua a lo largo de múltiples estaciones meteorológicas donde la correlación entre los datos es importante y determinante para realizar las predicciones[42–46].

Según [39], un sistema de monitoreo eficiente debería tener múltiples factores ejecutados en secuencia.

- Recopilación correcta de los datos.

- Análisis efectivo de la información.
- Notificaciones tempranas de los resultados.

Para ello se proponen diferentes soluciones; una de ellas, es una solución multicapa en la que se realicen estas operaciones, siendo la capa más baja la capa IoT donde se adquieran los datos, una capa de computación de borde, una capa de análisis de datos y finalmente, la capa más alta conocida como la capa de presentación. Cada una de estas capas realiza una actividad específica, siendo la capa de computación de borde la usada para reducir la latencia de toda la solución.

La predicción oportuna y efectiva de inundaciones depende de varios factores; algunos de ellos están clasificados en la Tabla 6.1 y hay otros, como la ubicación estratégica de los nodos de medición, siendo este el factor que más puede influenciar en la predicción, pues es necesario que la información no se vea solapada y que el modelo se encuentre correctamente establecido para la zona. Por ejemplo, en [39], se propone una distribución hexagonal de los diferentes nodos; en [45] se propone distribuir los sensores a lo largo de la cuenca del río a analizar. Por otro lado, la precisión de la predicción se ve afectada por la configuración y tipo de modelo usado, como por ejemplo, modelos hidrológicos e hidrodinámicos que permiten tener simulaciones bastante precisas [23, 47] que son complejos y requieren análisis, medidas de los ríos y conocimientos específicos para que sean factibles. Finalmente, están los basados en redes neuronales, que son adaptables a la gran cantidad de información y variables que se requieren para realizar una predicción sin conocer específicamente la relación entre las entradas y las salidas del sistema; lo que se busca con estos modelos es tener una predicción en línea, con bajas latencias y sin necesidad de realizar un post-procesamiento de los datos en servidores lejos de la zona de interés. Estas aproximaciones han sido usadas para estimar el nivel de los ríos usando hardware embebido [21–23, 48–51].

6.2. Síntesis de Alto Nivel *HLS*

La síntesis de alto nivel (*HLS*, por las siglas en inglés de *High-Level Synthesis*), le permite al desarrollador programar hardware usando lenguajes con altos niveles de abstracción como lo son C/C++/System C/C#, sin necesidad de hacer uso de lenguajes de descripción de hardware como VHDL o Verilog, cuyo desarrollo y simulación es lento y complejo comparado con el tiempo que toma desarrollar en un lenguaje de alto nivel. Actualmente, la tercera generación de compiladores de HLS ofrece a los desarrolladores ventajas significativas frente a las revisiones anteriores como lo son la optimización del proceso de síntesis de hardware, la estandarización del uso de C, C++, System C o C# como lenguajes de entrada al software de síntesis y la posibilidad de adicionar directivas de optimización sin realizar modificaciones al código fuente, co-simulación entre los diferentes lenguajes de entrada y el RTL objetivo, entre otras [52].

Las bondades de HLS frente a los paradigmas de programación de hardware tradicionales son muchas como se mencionó anteriormente. Una de ellas y quizás de las más importantes, es la aplicación de optimizaciones de hardware que, según el contexto,

puede incrementar el rendimiento de un circuito, modificando la cantidad de hardware dedicado al procesamiento de cada tarea [53, 54].

Por otra parte, HLS provee al desarrollador la opción de crear hardware reconFigurable de forma dinámica; esto es conocido como reconFiguración parcial o *PR* por sus siglas en inglés. La aplicación de la reconFiguración parcial le permite al desarrollador integrar múltiples módulos de hardware con diferentes algoritmos que pueden ser instanciados sobre la superficie de la FPGA de forma dinámica [55–57], con el objetivo de tener en un único dispositivo, múltiples funciones sin la necesidad de reprogramar la FPGA completamente.

Entre los proyectos orientados a usar netamente HLS como medio de prototipado e implementación de sistemas hardware, se pueden encontrar soluciones desde, optimizar procesos de kernel de un S.O., pasando por acceder y modificar bases de datos de forma concurrente usando paralelismo y finalizando con la fusión de datos [58–65].

6.2.1. Herramientas de Síntesis

Dentro de las herramientas de síntesis de alto nivel es posible encontrar una amplia gama de desarrollos; entre ellos, los más modernos son:

Compilador	Desarrollador	Lenguaje	Salida
A++	Altera	C/C++ Verilog	VHDL Verilog
Vivado HLS	Xilinx	C/C++ System C	VHDL Verilog
Stratus	Cadence	C/C++ System C	RTL
Hastlayer	Lombiq Technology	.NET C/C++/F#	VHDL
LegUp HLS	LegUp Computing	C	Verilog
Kiwi	Universidad de Cambridge	C#	Verilog
HercuLeS	Ajax Compilers	C/NAC	VHDL
Matlab HDL Coder	Matlab/Xilinx	Matlab Script	VHDL

TABLA 6.2: Sintetizadores de alto nivel más comunes

Entre los desarrolladores de la Tabla 6.2, encontramos a Altera y Xilinx, desarrolladores de chips FPGA, así como importantes desarrolladores de software como Mathworks con su software de síntesis HDL Coder.

6.2.1.1. Esquema General de Síntesis

En su núcleo, los sintetizadores de alto nivel siguen un mismo esquema de compilación, el cual sigue los siguientes pasos:

- Procesamiento léxico
- Optimización del algoritmo
- Análisis de control y flujo de datos
- Procesamiento de bibliotecas
- Asignación de recursos
- Agendamiento
- Unión de funciones
- Unión de registros
- Procesamiento de salidas
- Reducción de entradas

La diferencia entre cada uno de los sintetizadores radica principalmente en la cantidad de lenguajes de entrada, la cantidad de optimizaciones y la calidad de las mismas aplicadas al hardware final.

6.2.1.2. Optimizaciones de Hardware

Cada sintetizador puede ofrecer una gama diferente de optimizaciones; estas pueden ser aplicadas al código fuente usando una directiva del lenguaje, como por ejemplo, **pragma** en el caso de C y sus derivados. A través de ella, se le indican directamente al compilador, parámetros de entrada como configuraciones dependiendo del tipo de optimización a aplicar.

Para el caso particular de Vivado HLS, la Tabla 6.3 muestra las directivas de optimización y una breve descripción del propósito.

Directiva	Descripción
ARRAY_MAP	Permite combinar diferentes arreglos de tamaño reducido en uno solo de gran tamaño con el objetivo de reducir la cantidad de BRAM requerida en la solución.
ARRAY_PARTITION	Permite transformar un arreglo de gran tamaño en sub arreglos de menor tamaño buscando reducir las contenciones en las memorias BRAM Dual Port.
ARRAY_RESHAPE	Permite transformar un arreglo con muchos elementos de tamaño reducido a uno de menor cantidad de elementos individuales pero con mayor ancho de datos. Esta directiva es útil para reducir la cantidad de accesos a memoria RAM.
DATA_PACK	Empaquea los campos de una estructura en un solo arreglo con mayor ancho de datos.
DATAFLOW	Habilita un pipeline a nivel de tareas, permitiendo que funciones y bucles operen de forma concurrente.
LATENCY	Permite especificar la latencia del circuito a implementar; esta puede ser tanto mínima como máxima.
LOOP_FLATTEN	Permite combinar bucles anidados en un solo bucle; esta directiva mejora considerablemente la latencia del circuito únicamente si los bucles son considerados perfectos.
LOOP_MERGE	Permite combinar bucles consecutivos en uno solo mejorando la latencia del circuito.
PIPELINE	Permite reducir el intervalo de iniciación, permitiendo que las tareas a nivel de función o bucle se ejecuten concurrentemente.
UNROLL	Usada para desglosar un bucle <code>for</code> en múltiples operaciones independientes; esta directiva busca incrementar la productividad haciendo uso del paralelismo.
CLOCK	Permite cambiar el dominio de tiempo de una función.
INLINE	Usada para crear optimizaciones lógicas entre funciones, reduciendo así la latencia y la sobrecarga de llamados a la función.

TABLA 6.3: Directivas disponibles en Vivado HLS [5]

La directiva ARRAY_MAP permite realizar el mapeo de los arreglos de dos formas diferentes que pueden beneficiar de diferentes formas el circuito, los cuales son el mapeo horizontal y el mapeo vertical.

Mapeo Horizontal: El resultante de aplicar la directiva ARRAY_MAP con la opción de mapeo horizontal implica la creación de un nuevo arreglo con los diferentes arreglos concatenados uno tras otro.

En la Figura 6.5, se aprecia la aplicación de la directiva ARRAY_MAP con Mapeo Horizontal. Este tipo de implementaciones permite reducir la cantidad de BRAM presente en el circuito.

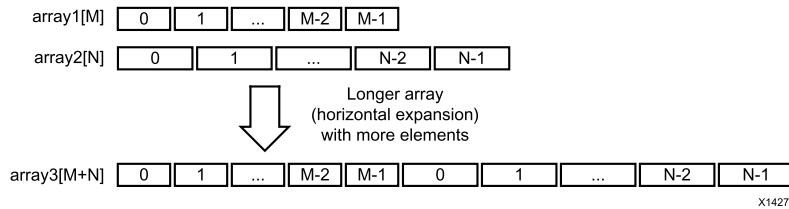


FIGURA 6.5: Directiva ARRAY_MAP en modo horizontal [5]

En la Figura 6.6 se ve la aplicación en RAM de la directiva. Se puede notar que se toma una BRAM del tamaño de datos más ancho. Los bits sobrantes son descartados.

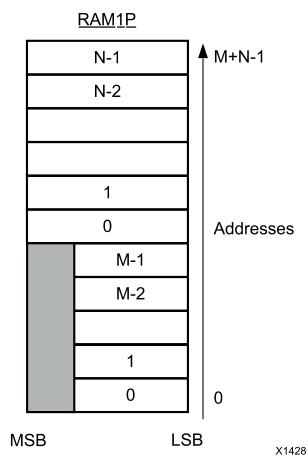


FIGURA 6.6: Distribución en RAM del Mapeo Horizontal [5]

Implementaciones de este tipo traen complicaciones que pueden afectar de manera directa el rendimiento del circuito, pues dadas las grandes dimensiones y la limitada cantidad de puertos, se pueden presentar contenciones que darán lugar al efecto cuello de botella, donde la unidad de control deberá agendar los accesos a la RAM de tal forma que se extraigan los datos según la cantidad de peticiones.

Mapeo Vertical: Esta implementación consiste en la creación de un nuevo arreglo donde se concatenan los datos uno junto a otro; esto implica la creación de un arreglo de tamaño igual a la cantidad de datos individuales pero con un ancho de bits igual a la suma de los anchos de los arreglos a unir.

En la Figura 6.7 se puede observar una representación del Mapeo Vertical; en este caso se han tomado dos arreglos de ancho e índices diferentes y se han concatenado elemento a elemento para formar un arreglo de mapeo vertical.

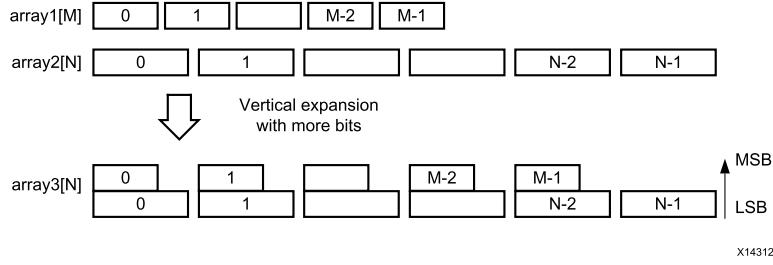


FIGURA 6.7: Directiva ARRAY_MAP en modo vertical [5]

El orden de concatenado se da a elección del diseñador, el cual le indica al sintetizador cuál subarreglo deberá ocupar la parte baja del elemento del arreglo y cuál la parte alta.

En la Figura 6.8, se aprecia cómo sería la distribución en la memoria RAM del mapeo vertical. En este caso, uno de los arreglos tiene mayor número de elementos que el otro y los bits restantes son puestos a tierra por el sintetizador.

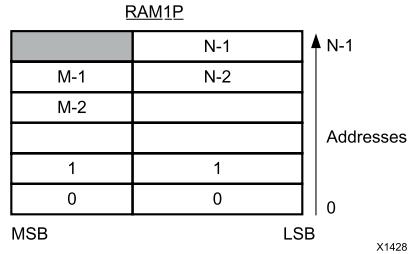


FIGURA 6.8: Distribución en RAM del Mapeo Vertical [5]

Otra de las directivas de Vivado HLS es la llamada ARRAY_PARTITION. Esta directiva permite al programador incrementar el rendimiento del circuito cuando este se ve envuelto en un cuello de botella debido a las contenciones; esto se debe al número limitado de puertos con los que cuentan las memorias Block RAM de la FPGA y que el sistema intenta acceder simultáneamente a diferentes entradas que se encuentran en la misma memoria BRAM, como ocurre al aplicar la directiva PIPELINE.

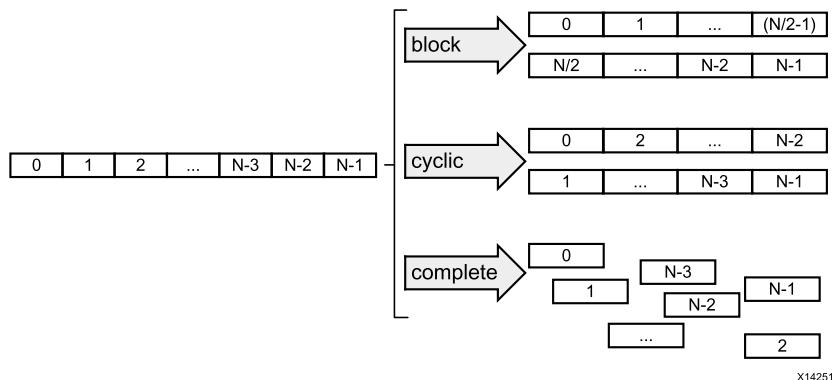


FIGURA 6.9: Modos de operación de la directiva ARRAY_PARTITION [5]

En la Figura 6.9 se pueden observar los modos de operación de la directiva ARRAY_PARTITION, los cuales se detallarán a continuación.

- **Bloque:** En este caso, el arreglo que se desea particionar será dividido en un número de acuerdo a un factor especificado por el diseñador y las necesidades del hardware.
- **Cíclico:** Este modo permite particionar el arreglo de modo que se intercalen los índices del arreglo. En este modo también es posible adicionar un factor de partición.
- **Completo:** En modo completo el sintetizador transformará el arreglo en registros individuales permitiendo acceder a cada elemento de forma individual; esto implica un aumento significativo de los recursos de hardware usados para cumplir el funcionamiento del circuito.

Continuando con el detalle de las directivas encontradas en Vivado HLS encontramos la directiva ARRAY_RESHAPE, la cual es una combinación de dos directivas anteriormente mencionadas: ARRAY_MAP en modo vertical y ARRAY_PARTITION. Esta directiva permite reducir el número de BRAM a usarse en la solución final y a su vez mantiene el beneficio de particionar los arreglos, cual es el acceso paralelo a los datos contenidos en la memoria.

La Figura 6.10 nos muestra un ejemplo claro del funcionamiento de la directiva.

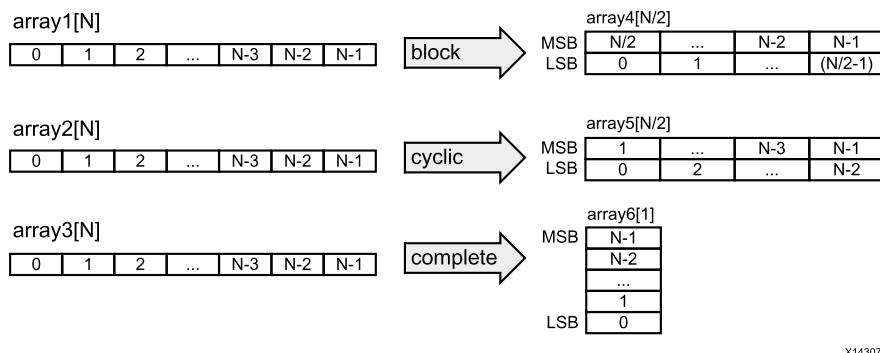


FIGURA 6.10: Modos de operación de la Directiva ARRAY_RESHAPE [5]

En este caso particular se han elegido tres arreglos diferentes de tamaño N; en ellos se ha elegido una partición en factor de dos para los modos BLOQUE y CICLICO. Como se mencionó anteriormente, cada uno de los elementos del índice se concatena usando el modo vertical del ARRAY_MAP. En el caso del modo COMPLETO se concatena cada uno de los elementos para formar un solo elemento de ancho N, siendo el bit menos significativo el primer elemento del índice original y el bit más significativo el último bit del último elemento del índice original.

Dentro de las posibilidades de desarrollo usando síntesis de alto nivel con lenguajes como C/C++, surge la necesidad de adicionar el tipo de dato STRUCT o estructura;

esto permite crear un tipo de dato el cual puede combinar diferentes tipos de datos en su interior. Estos datos no son más que referencias a posiciones de memoria, las cuales usan desplazamientos relativos (offsets) según el tipo de dato a acceder.

Sin embargo, a nivel de hardware, el acceso a los elementos individuales puede ser complejo dada la cantidad de elementos que la estructura puede contener. Por omisión, el sintetizador asume que cada arreglo dentro una estructura es una BRAM independiente, es por esto que se debe crear una lógica de control independiente para cada una de estas memorias.

En la Figura 6.11 se observa el funcionamiento de la directiva DATA_PACK, la cual permite optimizar el acceso a la estructura.

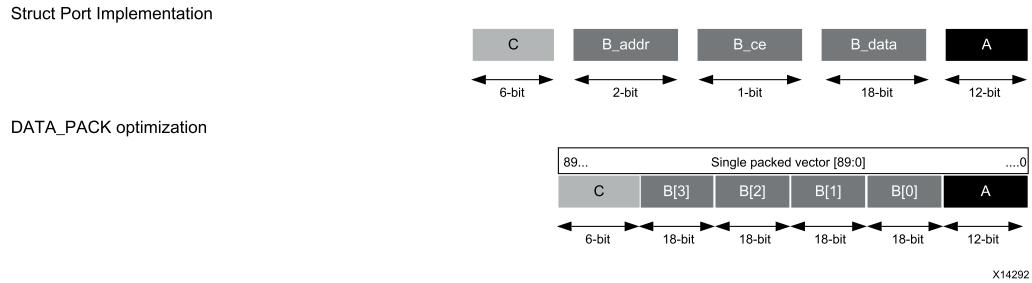


FIGURA 6.11: Funcionamiento de la Directiva DATA PACK [5]

En este caso particular, el elemento A es un elemento individual de tipo int12, el elemento B es un arreglo de 4 elementos de tamaño int18 y finalmente el elemento C, es un elemento individual de tamaño int6. Como se infiere de la parte superior de la imagen, la implementación de la estructura implica que el sintetizador haga uso de dos LUT RAM para ocupar los elementos A y C y una BRAM para ocupar el elemento B con su respectiva lógica de control. Posterior a la optimización, es posible tener una sola BRAM conteniendo todos los elementos de la estructura; de esta forma, se puede reducir el hardware necesario para el funcionamiento del circuito reduciendo así la latencia del mismo.

Dentro de las posibilidades de realizar operaciones concurrentes con Vivado HLS podemos encontrar la directiva DATAFLOW. Esta directiva transforma el código de tal forma que es posible tener un paralelismo a nivel de tareas o procesos dentro del mismo algoritmo haciendo uso de pipeline.

En la Figura 6.12 podemos apreciar el esquema de un circuito resuelto de manera secuencial usando funciones.

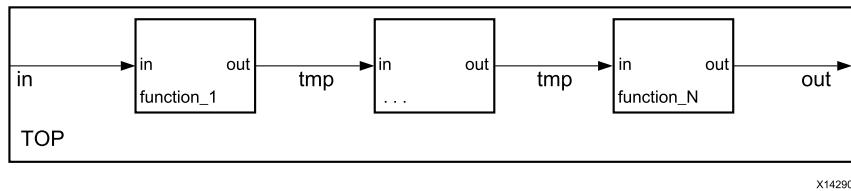


FIGURA 6.12: Sistema secuencial [5]

De esta forma es usual atacar la mayor cantidad de problemas si tienen una baja complejidad y cuentan con dependencias a nivel de datos como la variable `tmp` que se pasan las funciones en el módulo `TOP`.

Como se mencionó antes, la directiva DATAFLOW es capaz de transformar el código de tal forma que se crean canales de comunicación entre las diferentes funciones dividiendo las tareas entre procesos. Esto se puede observar en la Figura 6.13, donde la creación de los canales asegura que no se deba esperar a que termine la operación para continuar con las demás tareas.

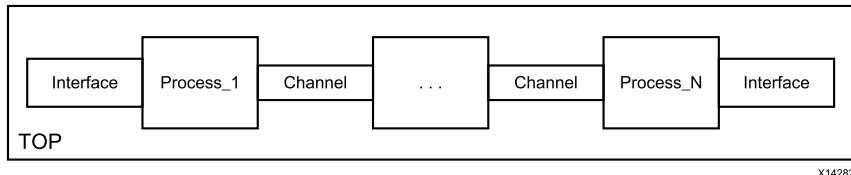


FIGURA 6.13: Sistema concurrente a nivel de tareas [5]

En la Figura 6.14 podemos ver un ejemplo claro del funcionamiento de la directiva DATAFLOW.

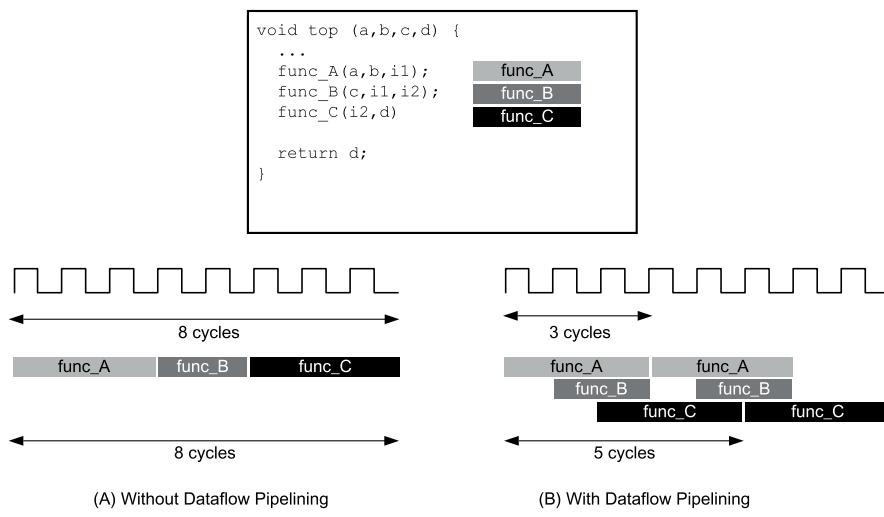


FIGURA 6.14: Funcionamiento de la directiva DATAFLOW [5]

En el lado (A) es posible ver cómo la operación tarda hasta 8 ciclos desde que se inicia la operación con `func_A` y la salida del sistema `d` es escrita en memoria por `func_C`. Usando la directiva DATAFLOW es posible aumentar la tasa a la que se obtienen resultados dada la operación en paralelo de las funciones usadas para calcular los resultados; en este caso, en el lado (B) vemos cómo el sistema es capaz de aceptar una nueva entrada al sistema cada 3 ciclos de reloj a diferencia de cada 8 ciclos sin la optimización y además, es posible obtener resultados cada 5 ciclos de reloj desde que ingresa un dato nuevo al circuito.

La directiva DATAFLOW requiere de condiciones específicas para garantizar la síntesis del circuito dentro de las cuales se deben cumplir los siguientes requerimientos:

- El sistema no debe tener ningún tipo de dependencia de datos.
- Las funciones no deben tener retroalimentación.
- Todas las funciones se deben ejecutar sin condiciones (`if/else`).
- Los límites de los bucles deben ser fijos.
- El sistema no puede tener paradas condicionales.

Vivado HLS es capaz de controlar la latencia de un bucle o una porción de código como tal; para esto es necesario usar la directiva LATENCY. Esta directiva restringe al sintetizador a alcanzar los valores de latencia indicados por el diseñador; en caso de no poder satisfacerlos, dadas las restricciones de hardware y propagación de los datos, el sintetizador tratará de alcanzar el nivel más bajo posible al indicado en el parámetro de la directiva.

Continuando con las mejoras en latencia de los circuitos usando Vivado HLS, podemos encontrar directivas como LOOP_FLATTEN; esta directiva permite aplanar bucles anidados permitiendo reducir la latencia del bucle completo pues se elimina la lógica de control entre bucles.

La directiva LOOP_FLATTEN requiere que el bucle a aplanar sea Perfecto o Semi-Perfecto; en el caso de ser un bucle imperfecto la directiva no será aplicada.

- **Bucle Anidado Perfecto:** Un bucle anidado perfecto es aquel en el cual no hay lógica entre bucles y únicamente el bucle más interior contiene lógica; además, los límites de los bucles son constantes.
- **Bucle Anidado Semi-Perfecto:** Igual que el Bucle Anidado Perfecto, excepto porque los límites del bucle más externo pueden ser variables.

Además de la directiva de LOOP_FLATTEN podemos encontrar la directiva LOOP_MERGE. Esta directiva permite combinar bucles consecutivos no anidados e implica una mejora en la latencia general del sistema al reducir algunos ciclos de reloj entre la transición de bucles; este tiempo muerto se debe a que el circuito debe realizar una

inicialización de variables, que por lo general no toma más de unos cuantos ciclos de reloj.

La Figura 6.15 muestra un ejemplo de cómo una típica operación con dos bucles puede tener un impacto negativo en la ejecución del circuito.

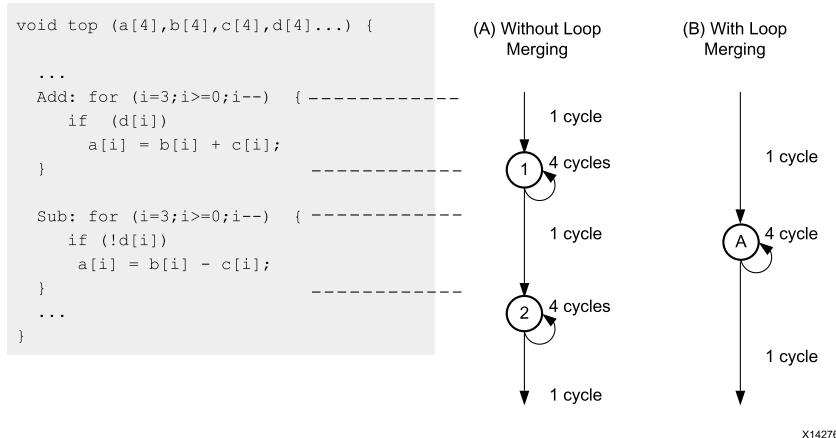


FIGURA 6.15: Funcionamiento de la Directiva LOOP MERGE [5]

En este caso, sin la aplicación de la directiva, vemos que al sistema le toma 5 ciclos extra con respecto a aplicar la directiva al código fuente, pues se combinan ambos bucles en un solo estado de ejecución.

Es importante tener en cuenta que la directiva LOOP_MERGE es únicamente aplicable si los bucles no cuentan con ningún tipo de dependencia entre sí.

La directiva PIPELINE provista por el software le da la posibilidad al diseñador de agregar etapas de pipeline a la ejecución de funciones y bucles dentro del código fuente del circuito a sintetizar; esta directiva se encarga de adicionar los estados de la unidad de control y el hardware necesario para el funcionamiento.

Como se describió anteriormente, la directiva PIPELINE puede ser aplicada tanto en funciones como en bucles. En la Figura 6.16 se puede apreciar el comportamiento de la directiva aplicada a funciones en la parte izquierda y el comportamiento de la directiva aplicada a bucles en la parte derecha. En la parte inferior tenemos el comportamiento de las peticiones de lectura y escritura producido por ambas implementaciones. En el caso de aplicar la directiva a bucles, podemos notar que la unidad de control debe adicionar burbujas y liberar los registros cuando se requiere iniciar una nueva iteración del bucle. Esta, aunque es una operación necesaria, puede incrementar la latencia del circuito.

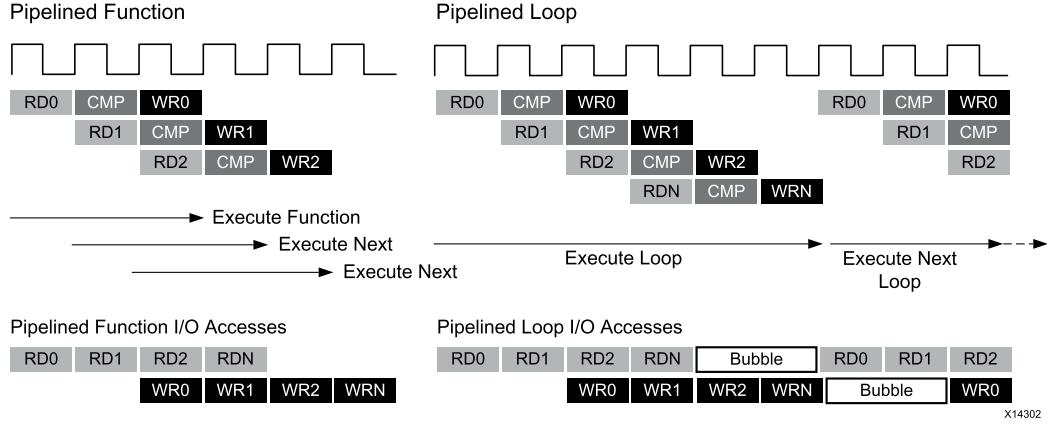


FIGURA 6.16: Directiva PIPELINE aplicada a funciones y bucles [5]

La aplicación de la directiva PIPELINE provee al diseñador la capacidad de explotar en mayor medida el hardware que se está diseñando. Sin embargo, esta directiva trae consigo problemas como la adición de burbujas que claramente incrementan la latencia y el hecho de que el sistema deba desocupar el pipeline cuando los datos no estén disponibles para ser computados por el hardware; estos problemas pueden ser solucionados fácilmente con los modos PIPELINE_REWIND y PIPELINE_FLUSH respectivamente.

La Figura 6.17 nos muestra el comportamiento de la directiva PIPELINE en su modo REWIND.

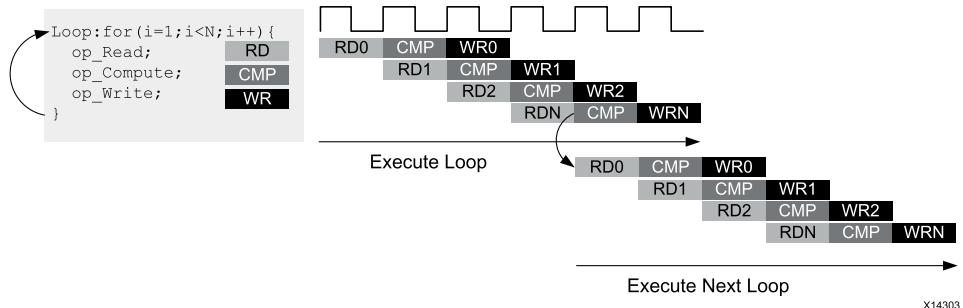


FIGURA 6.17: Comportamiento del modo PIPELINE_REWIND [5]

Este modo permite reducir la latencia del circuito al ejecutar bucles, pues no requiere agregar burbujas ni evacuar las etapas del pipeline para continuar con la siguiente iteración. Este tipo de modos requiere una gran cantidad de hardware para su implementación y más aún cuando se realizan operaciones complicadas con bucles anidados. Es posible entonces que el sintetizador no pueda aplicar la directiva con este modo dada la complejidad y las dependencias de datos.

Otro de los problemas del pipeline es la dependencia de datos. La dependencia de datos cuando se corre un circuito con pipeline implica que este no puede continuar con las operaciones hasta que los datos se encuentren disponibles provocando que las etapas

del pipe sean evacuadas. Esto trae consigo un incremento de la latencia, pues idealmente estas etapas deberían permanecer llenas. Con la aplicación del modo FLUSH de la directiva PIPELINE este tipo de eventos no provocará la evacuación de las etapas del pipeline, pues la unidad de control detecta el momento en que los recursos no se encuentran disponibles y congela la ejecución del hardware hasta que los datos se encuentran disponibles para ser computados nuevamente por el hardware; ver Figura 6.18.

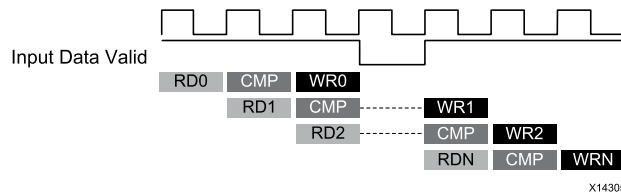


FIGURA 6.18: Comportamiento del modo PIPELINE_FLUSH [5]

Continuando con las directivas que favorecen la concurrencia encontramos la directiva UNROLL. Esta directiva permite dividir el hardware de un bucle especificando un factor de división. El factor de división de la directiva por omisión es un unroll de tamaño completo; ver Figura 6.19. Cada una de las divisiones que se realizan usando la directiva UNROLL crea instancias nuevas con los mismos recursos de hardware que el circuito original; la diferencia radica en la unidad de control y la forma en que se administra la ejecución del algoritmo usando las diferentes particiones; ver Figura 6.19.

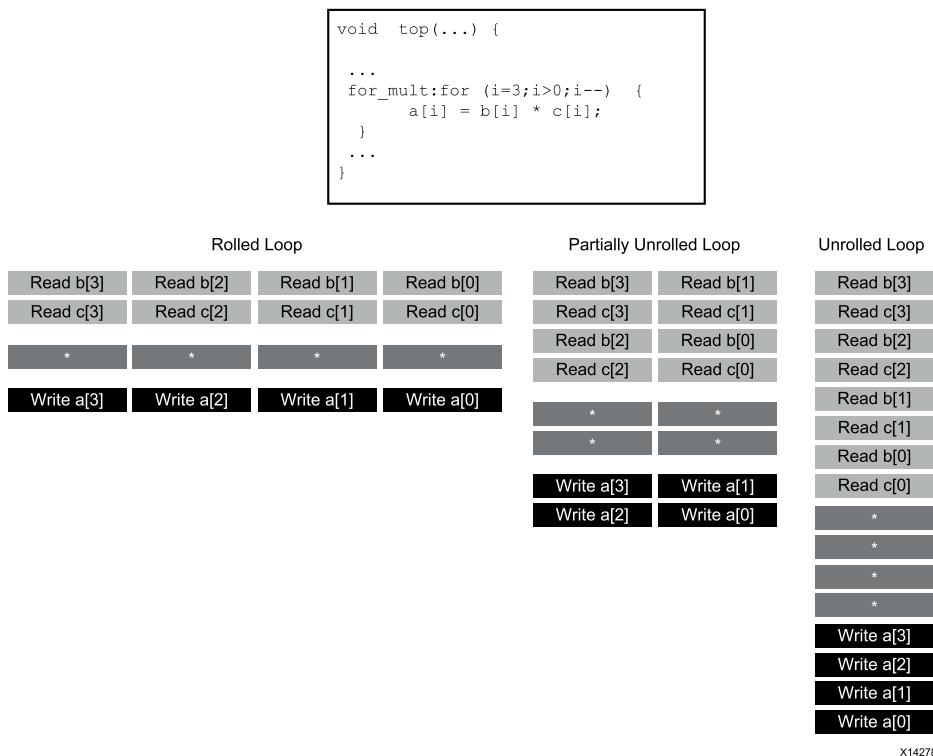


FIGURA 6.19: Funcionamiento de la Directiva UNROLL [5]

Un bucle puede tener tres estados. El más común es el estado natural en el cual no se ha aplicado la directiva; este caso se conoce como un ROLLED LOOP. El segundo estado es aplicando la directiva con factor de dos o más; este estado se conoce como PARTIALLY UNROLLED LOOP. Finalmente el estado ideal, que implica consumir grandes cantidades de hardware es el FULLY UNROLLED LOOP; en este caso, cada una de las operaciones del circuito se realiza en un hardware dedicado.

La directiva UNROLL es capaz de reducir considerablemente la latencia del circuito tras su aplicación, pero su aplicación puede traer consecuencias como cuellos de botella si las memorias BRAM no han sido particionadas correctamente.

La directiva CLOCK es muy importante cuando se diseña usando HLS pues esta, permite al diseñador usar diferentes dominios de tiempo para cada uno de sus circuitos y funciones dentro del mismo módulo. Esto le da la libertad de validar el funcionamiento de su circuito con diferentes frecuencias de reloj, lo cual, si se aprovecha, puede incrementar el rendimiento general del circuito.

Finalmente, Vivado HLS cuenta con la directiva INLINE. Esta directiva le permite al programador expandir en línea funciones pequeñas dentro de aquellas que las llaman con mayor frecuencia, con lo cual, la aplicación de esta directiva elimina ciclos de establecimiento de señales entre funciones además de eliminar el hardware de control de las interfaces de las funciones.

La mayoría de herramientas de síntesis de alto nivel permite crear directivas múltiples por dominio, dando la posibilidad de aplicar una o más directivas a una porción de código específica. Por ejemplo, a un bucle podría aplicarse directivas de PIPELINE y UNROLL al mismo tiempo, lo cual incrementaría el performance de la aplicación. La única restricción de este tipo de modificaciones usando directivas son las dependencias de datos; este tipo de problemas podría evitar que la directiva sea aplicada correctamente por el sintetizador.

6.3. Software de Síntesis de Hardware Vivado Design Suite

Aplicaciones como Vivado HLS, además de la síntesis de alto nivel y simulación del hardware, no permiten una implementación final en la FPGA. Para este fin existen herramientas de síntesis de hardware como [Vivado Design Suite \[66\]](#); este software transforma el código en VHDL arrojado por la síntesis de alto nivel en un bitstream, el cual contiene la configuración de la lógica de la FPGA. Vivado Design Suite es un software complejo que conoce la arquitectura de la FPGA y se encarga de los aspectos de la optimización final de tiempo, área y potencia de la implementación deseada.

En la Figura 6.20 se puede ver un ejemplo claro de cómo el software interconecta los bloques IP Core. En hardware es común ver que un bloque IP tenga múltiples interfaces de entrada y salida. Por ejemplo, un módulo que desee interactuar con una memoria BRAM deberá contar con señales de control compatibles con dicha RAM; conectar individualmente estas señales implica tiempo y si se habla de interfaces más complejas como las interfaces AXI, que requieren de 45 señales en su formato FULL AXI4, la

conexión requerirá aún más tiempo. Vivado Design Suite reconoce la interface de cada IP Core y las agrupa, permitiendo así realizar una sola conexión por interface.

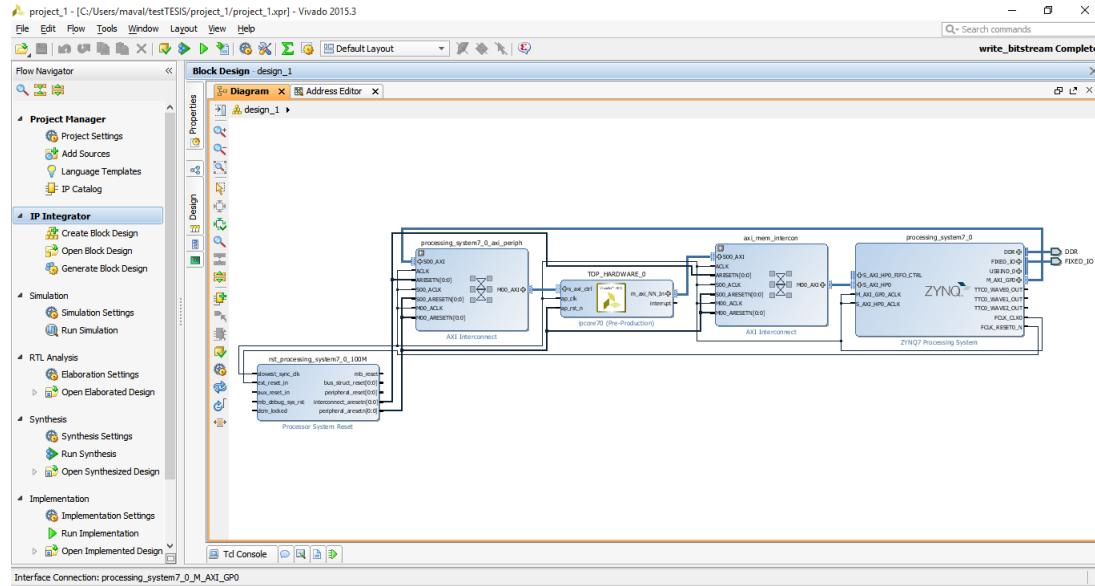


FIGURA 6.20: Vivado Design Suite GUI

En el caso de las interfaces mapeadas a memoria como las interfaces AXI, Vivado Design Suite se encarga de mostrarle al disenador el lugar de la memoria donde el sistema será mapeado; esto se hace según la Tabla 6.4. Dependiendo del puerto al que se ha conectado el módulo al procesador ARM, quien es el que tiene el controlador de la memoria RAM DDR3, la herramienta entregará un rango de memoria disponible para el usuario.

Address Range	CPUs y ACP	AXI_HP	Others Bus Master	Notas
0000_0000 to 0003_FFFF	OCM	OCM	OCM	Address not filtered by SCU and OCM is mapped low
	DDR	OCM	OCM	Address filtered by SCU and OCM is mapped low
	DDR			Address filtered by SCU and OCM is not mapped low
				Address not filtered by SCU and OCM is not mapped low
0004_0000 to 0007_FFFF	DDR			Address filtered by SCU
0008_0000 to 000F_FFFF	DDR	DDR	DDR	Address filtered by SCU
0010_0000 to 3FFF_FFFF	DDR	DDR	DDR	Address not filtered by SCU
4000_0000 to 7FFF_FFFF	PL		PL	Accessible to all interconnect masters
8000_0000 to BFFF_FFFF	PL		PL	General Purpose Port #0 to the PL, M_AXI_GP0
E000_0000 to E02F_FFFF	IOP		IOP	General Purpose Port #1 to the PL, M_AXI_GP1
E100_0000 to E5FF_FFFF	SMC		SMC	I/O Peripheral registers
F800_0000 to F800_0BFF	SLCR		SLCR	SMC Memories
F800_1000 to F880_FFFF	PS		PS	SLCR registers
F890_0000 to F8F0_2FFF	CPU		PS	PS System registers
FC00_0000 to FDFF_FFFF	Quad-SPI		Quad-SPI	CPU Private registers
FFFFC_0000 to FFFF_FFFF	OCM	OCM	OCM	Quad-SPI linear address for linear mode
			OCM	OCM is mapped high
			OCM	OCM is not mapped high

TABLA 6.4: Mapa de Direcciones del Zynq 7000 Processing System [8]

6.4. Red Neuronal Artificial

Las redes neuronales (*ANN*, por las siglas en inglés de *Artificial Neural Network*), han surgido como un mecanismo para solucionar problemas en diferentes ámbitos. Esto se debe a que estas tienen la capacidad de diagnosticar, detectar, predecir, monitorear, entre otras tareas [67–70]. La principal ventaja de estos sistemas es la facilidad que tienen para adaptarse a las necesidades de un problema. Solo se necesita un buen entrenamiento y una buena topología para que pueda cumplir con las exigencias del proyecto.

Para el desarrollo de ANN, existen dos técnicas muy conocidas, tanto para el entrenamiento como para la topología. La primera es *BPA* (*Back-Propagation Algorithm*). Permite propagar el error de la salida por cada una de las capas de la red, de forma que el error de la salida se va reduciendo debido a la modificación de los pesos sinápticos. La segunda es *MLP* (*Multi-Layer Perceptron*); esta es históricamente la más usada para el desarrollo de redes. Esto es debido a los buenos resultados que se han obtenido al implementarla [71].

6.4.1. Neurona Básica

Una neurona artificial o perceptrón es un modelo matemático basado en su contraparte biológica. Una neurona es una célula compuesta por una cantidad indefinida de conductos de entrada llamados dendritas y un conducto de salida llamado axón. Los conductos le permiten a la neurona comunicarse con otras neuronas, de forma que las señales de los sensores de entrada se van propagando a través de cada una. El perceptrón es la unidad de procesamiento básico de una ANN; está compuesto por cuatro partes esenciales: nodos de entrada, pesos sinápticos, función de propagación y función de salida, como se muestra en la Figura 6.21.

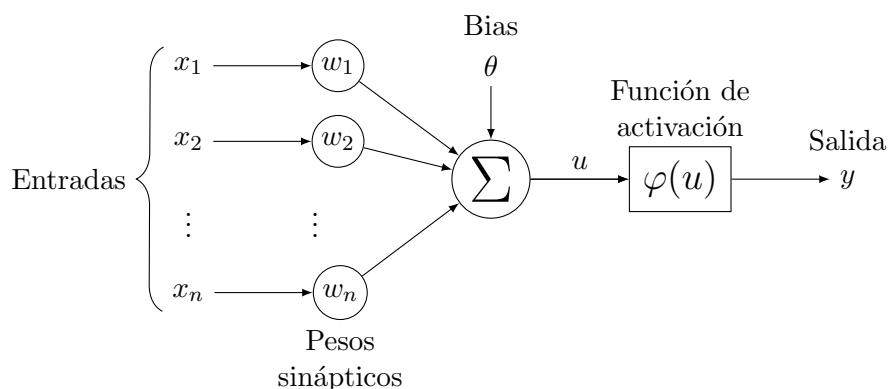


FIGURA 6.21: Neurona básica

6.4.1.1. Función de Propagación

Para el modelo matemático, las señales de entrada son expresadas por el conjunto $X = (x_1, x_2, x_3, \dots, x_n)^T$. Para simular el proceso de reorganización de las conexiones sinápticas de las neuronas biológicas, se asocia a cada elemento del conjunto de entrada un peso sináptico que le permite darle a la conexión un valor de excitación o de inhibición dependiendo del signo. Los pesos sinápticos se expresan por el conjunto $W = (w_1, w_2, w_3, \dots, w_n)^T$.

La función de propagación se encarga de multiplicar el conjunto de entrada y el conjunto de pesos sinápticos, término a término, para luego hacer la sumatoria de los elementos del vector resultante y así generar un valor de salida, u . La expresión completa para la función de propagación está dada por la Ecuación 6.1

$$u = \sum_{i=1}^n w_i x_i + \theta \quad (6.1)$$

donde $x_i \in X$, $w_i \in W$ y θ es el umbral de inhibición o excitación de la neurona.

6.4.1.2. Función de Activación

La función de activación surge como una necesidad de tener una salida de la neurona acotada a un rango determinado; esto se debe a que la salida en la neurona biológica se encuentra acotada en amplitud. En la Ecuación 6.2 se muestra la expresión matemática respecto a u .

$$y = \varphi(u) \quad (6.2)$$

Existen diferentes tipos de funciones de activación. La elección de estas depende del objetivo del diseño:

Función de Activación	Expresión Matemática
Paso Unitario	$\varphi(u) = \begin{cases} 1.0 & \text{si } u > 0 \\ 0.0 & \text{en otros casos.} \end{cases}$
Rampa	$\varphi(u) = \max\{0.0, \min\{1.0, u + 0.5\}\}$
Sigmoidal	$\varphi(u) = \frac{1}{1 + e^{-u}}$
Bipolar Sigmoid	$\varphi(u) = \frac{1 - e^{-u}}{1 + e^{-u}}$
Tangente Sigmoidal	$\varphi(u) = \frac{2}{1 + e^{-2u}} - 1$

TABLA 6.5: Funciones de Activación

6.4.2. Perceptrón Multi-Capa

Un MLP *Multilayer Perceptron* es una red de neuronas que está compuesta por una capa de nodos de entrada, una o dos capas ocultas y una capa de salida. Todas las capas están compuestas por neuronas artificiales a excepción de la capa de entrada donde los nodos son las señales de ingreso a la red. Las conexiones entre capas dependen de la configuración de la red; puede ser completamente conectada si cada uno de los nodos de una capa se conecta con todos los nodos de la siguiente, véase Figura 6.22. Por otro lado, puede tener una conexión personalizada y no estar completamente conectada.

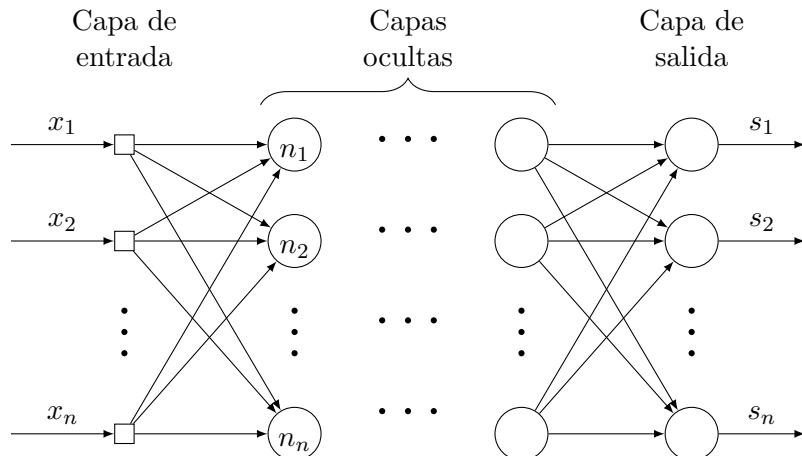


FIGURA 6.22: MLP completamente conectada

Capítulo 7

Recursos

7.1. Hardware

- Tarjeta de Desarrollo ZYBO Zynq-7000 ARM/FPGA SoC:
 - Xilinx Zynq-7000 AP SoC XC7Z010-1CLG400C
 - BRAM 18K: 120 Unidades.
 - DSP48E: 80 Unidades.
 - FF: 35200 Unidades.
 - LUT: 17600 Unidades.
 - Dual-core ARM CortexTM-A9
 - 512 MB DDR3
 - 128 MB Quad-SPI Flash
 - 4 GB SD card
 - Onboard USB-JTAG Programming
 - 10/100/1000 Ethernet
 - USB OTG 2.0 and USB-UART
 - Analog Devices ADAU1761 SigmaDSP® Stereo, Low Power, 96 kHz, 24-Bit Audio Codec
 - Analog Devices ADV7511 High Performance 225 MHz HDMI Transmitter (1080p HDMI, 8-bit VGA, 128x32 OLED)
 - PS & PL I/O expansion (FMC, Pmod, XADC)

7.2. Software

- Vivado Design Suite 2017.4 [66]: Programa para síntesis e implementación de hardware a partir de una descripción usando VHDL o Verilog.

- [Vivado HLS 2017.4 \[72\]](#): Programa para síntesis de alto nivel a un lenguaje de descripción de hardware (VHDL o Verilog).
- [Petalinux SDK 17.4 \[73\]](#).
- [PuTTY \[74\]](#).

Capítulo 8

Metodología

Para el desarrollo de los objetivos se seguirán los siguientes lineamientos.

1. **Definir los requerimientos y los lineamientos requeridos por la plataforma**
 - Se definen los recursos de hardware necesarios para establecer que tarjeta de desarrollo es suficiente para implementar una plataforma de computación de borde.
 - *Escritura:* Documentar los hallazgos encontrados.
2. **Desarrollar un módulo de fusión de datos sintetizable en hardware, instrumentado y empaquetado como módulo IP**
 - Se desarrollará un módulo de fusión de datos de alto rendimiento usando ANN que permita procesar grandes bloques de información proveniente de múltiples fuentes de datos.
 - *Verificación:* Se realiza un análisis de compatibilidad entre lenguajes de programación y herramientas disponibles.
 - *Escritura:* Se detalla el proceso de diseño e implementación sobre el hardware.
3. **Integrar e implementar los módulos IP de fusión de datos como plataforma consciente del contexto usando reconfiguración parcial**
 - *Programación HLS:* Se programan los algoritmos de fusión usando síntesis de alto nivel HLS, añadiendo múltiples optimizaciones de hardware según sea necesario con el objetivo de mejorar la concurrencia y disminuir la latencia total del circuito.
 - *Programación RP:* Se realiza la configuración de los módulos que harán uso de la reconfiguración parcial.
 - *Escritura:* Se reportan los recursos y métodos de hardware usados para la implementación.

4. Comparar el rendimiento de la plataforma desarrollada con una implementación diseñada completamente en software

- *Pruebas iniciales:* Se realiza la verificación de funcionamiento del sistema completo.
- *Correcciones:* Se corrigen posibles errores de implementación y diseño.
- *Pruebas finales:* Se realizan pruebas de rendimiento y funcionalidad del sistema en su última etapa.
- *Evaluación final:* Se evalúa el desempeño del sistema en su totalidad, con lo cual se obtienen los resultados a reportar en el documento.
- *Escritura:* Se reportan los resultados obtenidos.

Capítulo 9

Elección de la Plataforma de Desarrollo

9.1. Elección de la plataforma

Se realiza un análisis primario con las familias de FPGA disponibles comercialmente por Xilinx que permiten reconfiguración dinámica y la instrumentación necesaria para servir como Gateway de IoT como la posibilidad de correr un SO y conexión a red.

La Tabla 9.1 muestra las familias de SoC dentro de la categoría de precio reducido y rendimiento medio. Se nota que por cantidad de recursos la opción mas viable es la familia ZYNQ 7000, ofreciendo hasta el doble de celdas lógicas y casi tres veces la cantidad de DSPs que otras plataformas dentro de su categoría, volviéndola rápidamente la familia a examinar.

SoC	ZYNQ 7K	ARTIX 7	SPARTAN 7
MAX LOGIC CELLS [k]	444	215	102
MAX MEM [Mb]	26.5	13	4.2
DSPs	2020	740	160
TRANSCEIVER SPEEDS [Gb/s]	12.5	6.6	NA
MAX I/O PINS	250	500	400

TABLA 9.1: Familias de FPGA

La Tabla 9.2 muestra las diferentes plataformas de desarrollo disponibles y de fabricantes como AVNET y DIGILENT. En ella vemos que los diferentes SoC de cada una de las plataformas cuentan con diferentes recursos de hardware y características extra adicionadas por el fabricante.

Para elegir la plataforma de desarrollo ideal se debe tener en cuenta el objetivo de lo que se está desarrollando y más importante aún, el para qué se está desarrollando. En este caso, se desea implementar un algoritmo altamente concurrente con el objetivo de operar bajo un esquema de IoT. Por lo tanto, es importante tener en cuenta que se requiere que

Nombre	X Z7K EKIT	MICROZED	PICOZED	ZEDBOARD	MINIZED	PYNQ-Z1	ZYBO N	ZYBO O
SoC	ZC702	ZC7010	ZCZ010	XC7Z020	XC7Z007S	XC7Z020	XC7Z020	XC7Z010
Price	895	266	178	475	89	199	299	199
Max LogicCells	85000	28000	28000	85000	23000	53200	53200	28000
Max BRAM Mb	4.9	1.92	1.92	4.9	1.8	4.9	4.9	1.92
DSPs	220	80	80	220	66	220	220	80
DDR RAM MB	1024	1024	1024	512	512	512	1024	512
Notas			No headers		No headers			

TABLA 9.2: Comparación tarjetas de desarrollo

los recursos de hardware sean altos, que tenga un stack de red y los jacks de conexión, todo esto bajo un kernel de Linux que permita administración remota usando SSH. Desde esta óptica, la mayoría de tarjetas cumplen con los requisitos. Algunas, como las Picozed y Minized, del fabricante AVNet no cuentan con el jack RJ45 de red, entre otros headers; para estas tarjetas, es necesario adicionar accesorios que incrementarían el precio base. Finalmente, se debe tener en cuenta el precio y el soporte del fabricante. En este caso, basado en la cantidad de recursos y el precio, la mejor plataforma sería la **PYNQ-Z1**. Aunque cuenta con 512MB de RAM DDR3, por su precio es imbatible frente a las otras plataformas; de requerirse más memoria RAM, la mejor opción sería la versión nueva de ZYBO, ambas del fabricante DIGILENT.

Dentro de este esquema de desarrollo es necesario aclarar que independientemente de la plataforma de desarrollo elegida, la implementación debería ser transversal a todas las familias de Xilinx que usen VIVADO como sintetizador de hardware, siendo las diferencias entre los SoC la velocidad de transferencia y eficiencia de la FPGA en el diseño de sus bloques lógicos, pudiendo obtener latencias más bajas en FPGAs más costosas.

Capítulo 10

Diseño de Software y Hardware

La investigación basada en el estado del arte Capítulo 6, determinó que la estructura que mejor se acopla al tipo de problema es una ANN de tres capas, tres neuronas en la capa de entrada para analizar las variables de Flujo, Precipitación y Humedad Relativa, una capa oculta con mínimo tres neuronas y finalmente una neurona en la capa de salida. Por otra parte, también determinó que es necesario escalar el sistema de detección proporcionalmente según la necesidad de resolución en la medición, esto es necesario cuando se desea determinar con mayor precisión el riesgo de inundación del área examinada.

10.1. Diseño de Hardware

Diseñar el hardware implica tener en cuenta las diferentes etapas del desarrollo del mismo; en este caso, es necesario conocer cómo funcionan el sintetizador de alto nivel Vivado HLS y el sintetizador de hardware Vivado Design Suite. Se estableció una metodología que permite establecer el mejor diseño de hardware: en primera instancia se debe realizar un análisis de cuáles son las posibles optimizaciones que puedan reducir la latencia total del circuito una vez sintetizado y cuáles podrían incrementarla.

1. Desarrollar la ANN usando lenguaje C++.
2. Sintetizar el hardware usando Vivado HLS.
3. Analizar latencia y uso de recursos obtenidos en el reporte de síntesis.
4. Aplicar optimizaciones de hardware.
5. Repetir desde 2 hasta obtener el mínimo entre latencia y uso de recursos.

Para este circuito se determinó que la latencia mínima se podría obtener usando las siguientes optimizaciones luego de analizar el código de la ANN, los bloques de memoria requeridos para realizar los bursts de memoria que provee la interfaz AXI, la cantidad de bucles y la cantidad de tareas que se realizan al interior del hardware.

1. Dataflow: Aplicable a tareas
2. Pipeline: Aplicable a bucles
3. Unroll: Aplicable a bucles

10.1.1. Diseño de la Red Neuronal Artificial

Como se mencionó en el Capítulo 7, las redes neuronales son sistemas concurrentes capaces de aprender y realizar tareas para las que han sido entrenadas; por ejemplo, detección de patrones, clasificación, implementación de modelos matemáticos y fusión de datos. Por esta razón, es que se decide aplicarla a este problema, pues se desconoce la interacción entre las variables meteorológicas y físicas que afectan el nivel de los ríos provocando una inundación.

La Red Neuronal Artificial a implementar debe componerse de una capa de entrada con tres neuronas, un mínimo de tres neuronas en la capa oculta de la red y finalmente una neurona de salida, todo esto por nodo de medición en campo. Así pues, si se requieren analizar los datos provenientes de múltiples fuentes, se deberán instanciar tantas Redes Neuronales como nodos disponibles se tengan. La figura 10.1 muestra el diseño de la red implementada, en ella se puede observar el diseño básico, cada nodo de sensores se compone de una tripleta de neuronas a la entrada y en la capa oculta para ser por una neurona en la capa de salida la cual es la encargada de entregar el resultado de la estimación.

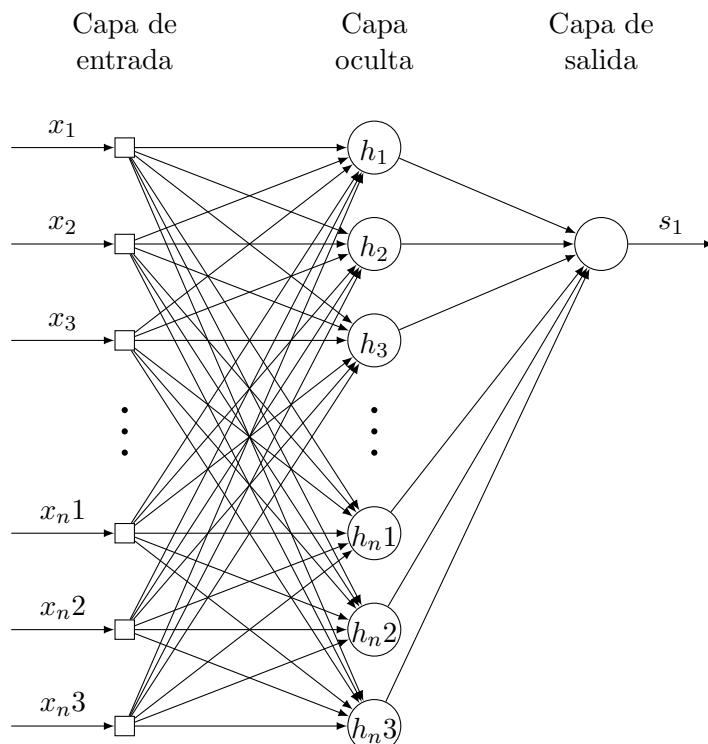


FIGURA 10.1: Diseño de la red neuronal

10.1.2. Función de salida de la red neuronal

Uno de los parámetros que más influye en la veracidad de la estimación de una red neuronal es la función de activación de las neuronas, ver Tabla 6.5. Según el estado del arte, la función de activación más común para este tipo de problemas es la función tangente sigmoidal [43], representada en la Ecuación 10.1 y la Figura 10.2.

$$\varphi(u) = \frac{2}{1 + e^{-2u}} - 1 \quad (10.1)$$

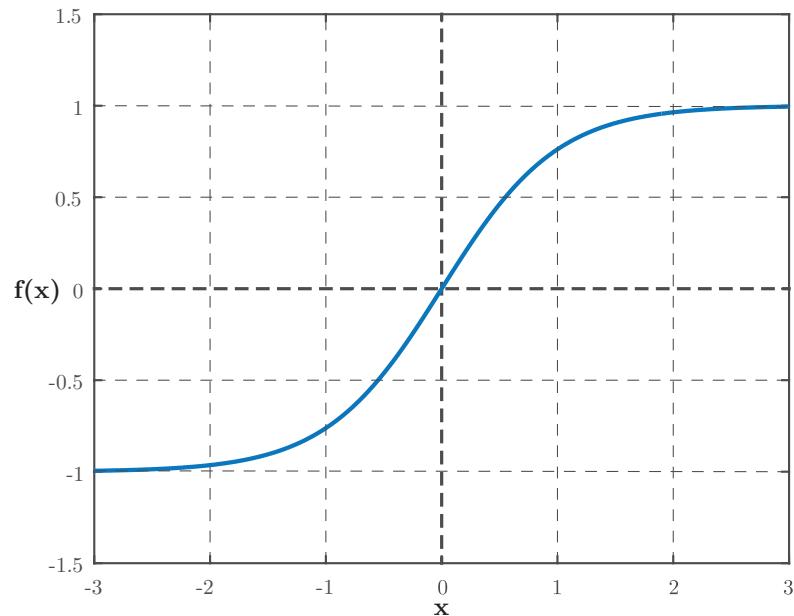


FIGURA 10.2: Tangente sigmoidal

La Ecuación 10.1 muestra que, para calcular la función tangente sigmoidal, es necesario realizar cálculos de complejos como los exponentiales que se pueden ver altamente afectados por el error que pueda introducir la unidad que los pueda calcular. Por otra parte, implementar este tipo de cálculos directamente sobre el hardware de la FPGA sería altamente costoso en términos de recursos y latencia por el simple hecho de utilizar punto flotante. Para resolver el problema se optó por realizar una implementación donde se discretiza la función continua, de tal forma que el bus de direcciones alcance máximo 12bits, es decir 4096 posiciones de memoria (se eligen 12b debido a que el sintetizador presenta problemas si se elige un bus de direcciones más grande al mapear una memoria ROM).

10.1.3. Código fuente Red Neuronal Artificial

A continuación se muestra de forma simplificada el diseño de la ANN en código fuente usando el lenguaje de programación C++

```

TOPANN ( fix *Inputs, fix *Outputs, fix *layerWeight, fix *layerBias,
          fix *outputLayerWeight, fix *outputLayerBias )
{
    // Creación de buffers de memoria
    fix BurstInputs[DATASET_INPUTS];// Buffer local que garantiza el Burst
    fix layerResult[NEURONS];      // Permite reciclar el HW de las capas

    memcpy ( BurstInputs, (const fix *) Inputs,
             DATASET_INPUTS * sizeof(fix) ); // Mecanismo que activa el Burst

    // Ejecución del algoritmo de la ANN
    ANN ( BurstInputs, BurstOutputs, layerResult, Weights, ... )

    // Burst de salida
    memcpy ( (fix *) Outputs, layerResult, sizeof (fix) * OUTPUTS );
}

```

Como se observa en el código anterior, la única diferencia entre una red de 3 entradas y una de N entradas son las definiciones de DATASET_INPUTS y NEURONS, que definen respectivamente, cuántas entradas va a tener la red y la cantidad de neuronas en la capa oculta. Es por esto que se hace necesario sintetizar la cantidad de circuitos como variaciones de estas configuraciones.

10.1.4. Puertos de entrada y salida AXI

Dado que se requieren manipular grandes cantidades de datos provenientes de la red de sensores, se hace necesario que la ANN cuente con la instrumentación necesaria. En este caso, se optó por usar el máximo ancho de banda posible disponible en la arquitectura ZYNQ 7000; este es otorgado por el bus AMBA. Para lograr el máximo ancho de banda se requiere que el hardware realice un llamado en bloque a la memoria partiendo de una dirección base otorgada como dependencia en la entidad más alta como un apuntador (`* Inputs` y `* Outputs`) y realizar un llamado a la función `memcpy`, la cual activa el pipelining y las optimizaciones del bus en tiempo de síntesis; este proceso se conoce como MEMORY BURST TRANSACTION, véase la Figura 10.3.

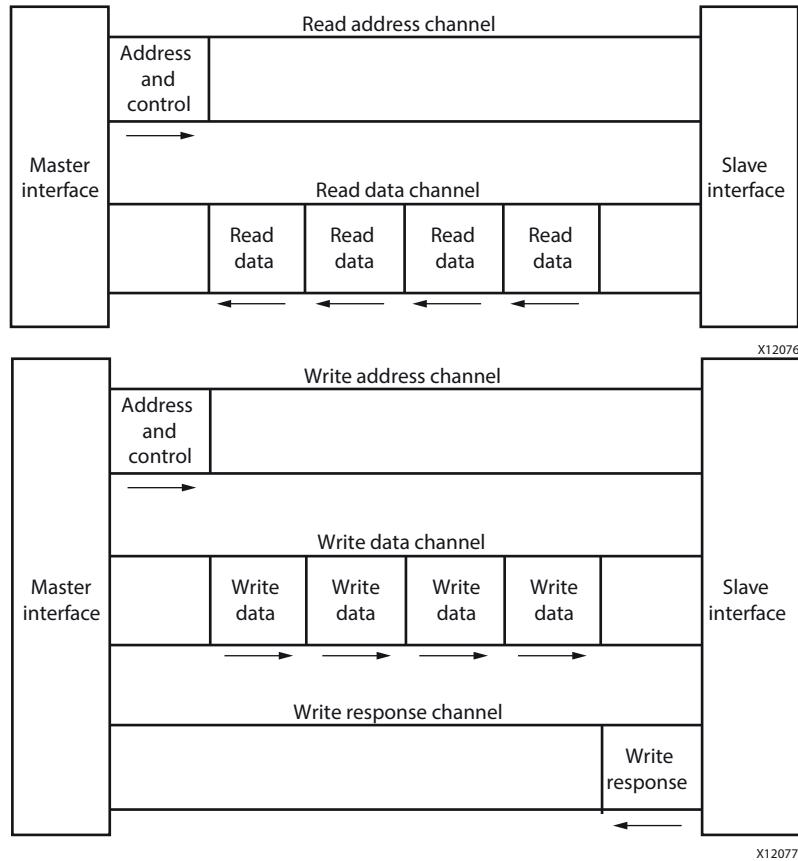


FIGURA 10.3: Memory burst transaction [6]

Finalmente, es necesario determinar qué tipo de datos se usarán para realizar las operaciones a nivel de los datos. En este caso, para garantizar una baja latencia y un bajo uso de recursos, se optó por un ancho del bus de 32b definido en código como **fix**, el cual es una definición de 32b con 8b para la parte entera y 24b para la parte decimal, la cual puede ser ajustada de acuerdo a la necesidad.

10.1.5. Optimizaciones de Hardware

En la sub sección anterior se definió el diseño general de la ANN sin optimizaciones de hardware más allá de las dadas por la interfaz AXI. En esta sub sección se adicionarán las optimizaciones y los criterios de diseño para dejar una u otra optimización.

Previo a la síntesis, es necesario determinar cuáles son las optimizaciones adecuadas para el tipo de problema: del espectro de opciones, es necesario conocer bien el manual del Vivado HLS y las condiciones que deben cumplirse para aplicar las directivas, pues su disposición puede ser tanto positiva como negativa dependiendo de cómo se implementen. Tras analizar el código fuente, se estableció que, debido a la dependencia de los datos entre capas, el circuito se ve altamente beneficiado por mecanismos como el pipeline y el paralelismo de las diferentes tareas que se realizan. Una vez aplicadas las optimizaciones,

es necesario analizar el reporte de síntesis y establecer si existen problemas con el ancho de banda de la memoria, el cual es un problema típico de este tipo de circuitos, en los que, debido a la alta concurrencia, el acceso a la RAM se ve afectado, teniendo que arbitrarse el acceso, aumentando en consecuencia de manera drástica la latencia total del circuito.

La Figura 10.4 muestra la latencia e intervalo de datos para las diferentes optimizaciones. Estas están definidas en clocks y cada una brinda información importante para la implementación final; el intervalo indica cada cuántos clocks es posible introducir un nuevo dato al circuito para que pueda ser procesado y la latencia indica cuánto tarda un dato para ser procesado. En este caso, la combinación de las directivas DATAFLOW y PIPELINE permite obtener el menor intervalo en punto flotante, pero se ve superado por PIPELINE puro en cuanto a reducción de la latencia.

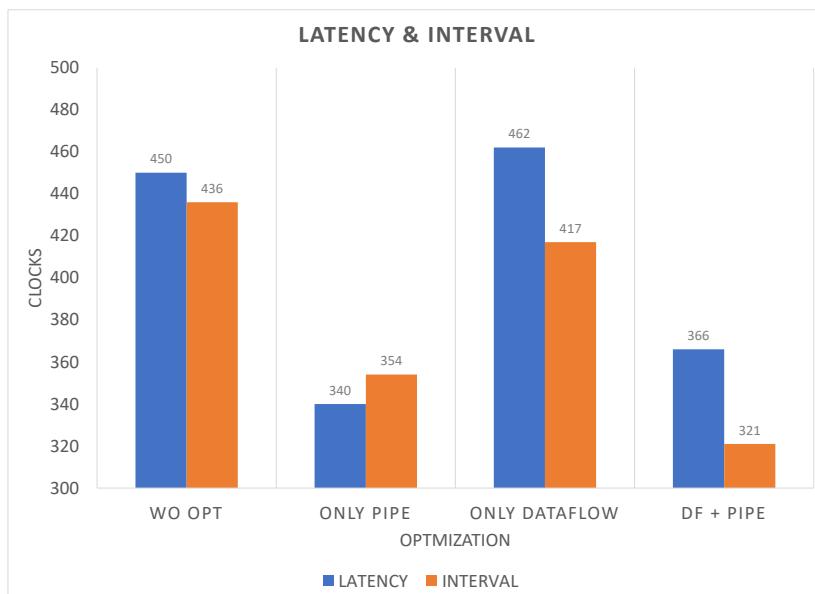


FIGURA 10.4: Latencia e intervalo de las optimizaciones en float

La Figura 10.5 muestra la ganancia de cada una de las optimizaciones contra la referencia (Sin optimizaciones), lo cual indica que para punto flotante, es mejor aplicar PIPELINE, obteniendo así una reducción casi del 30 % de latencia total del circuito y 20 % más intervalo.

La Figura 10.6 muestra los resultados de síntesis para las mismas optimizaciones usando punto fijo.

En este caso se observa un claro ganador; este puede ser corroborado por la Figura 10.7 con 3.5 veces en mejora del intervalo y 2.5 veces de la latencia frente a la referencia.

La Figura 10.8 muestra las diferencias de ganancia entre punto fijo y flotante. Es claro que las operaciones en punto fijo son menos costosas en términos de hardware; por tanto, el punto fijo y la combinación de DATAFLOW y PIPELINE son la combinación de optimizaciones a sintetizar e implementar en hardware.

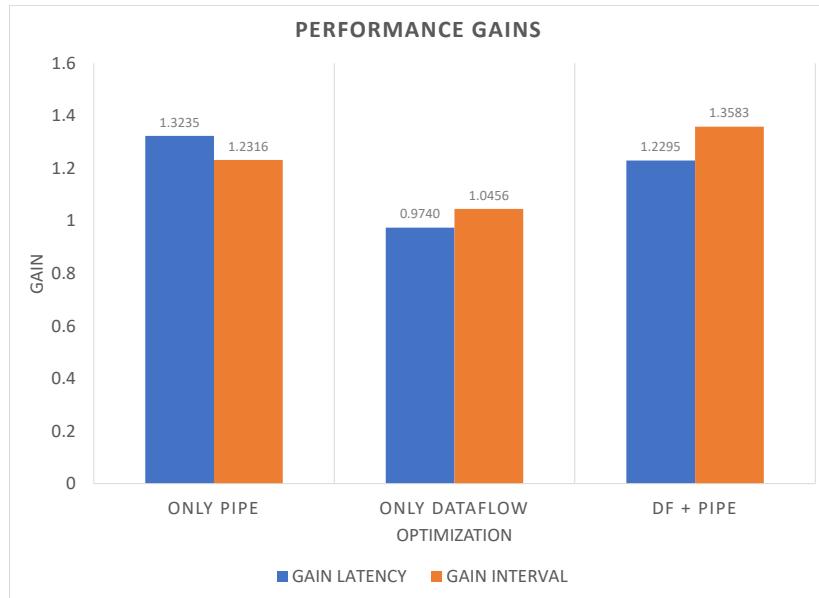


FIGURA 10.5: Ganancia en rendimiento usando float

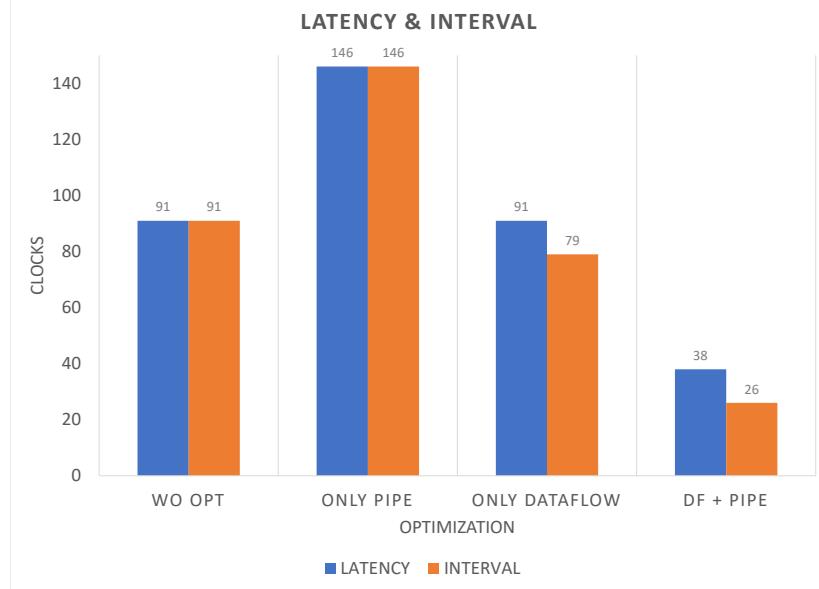


FIGURA 10.6: Latencia e intervalo de las optimizaciones fijo

10.1.6. Implementación del hardware

En esta sub sección se hablará de la implementación de hardware usando el sintetizador de hardware Vivado. La Figura 10.9 muestra la implementación final. En ella se observan 3 conectores AXI en el módulo **TOPANN_0**; los de la derecha representan los maestros AXI HP, encargados de iniciar las transacciones de datos desde y hacia la memoria RAM con sus respectivos Bursts; el conector AXI Lite a la izquierda del módulo corresponde al esclavo encargado de recibir los datos de los pesos desde la memoria RAM. Se determinó

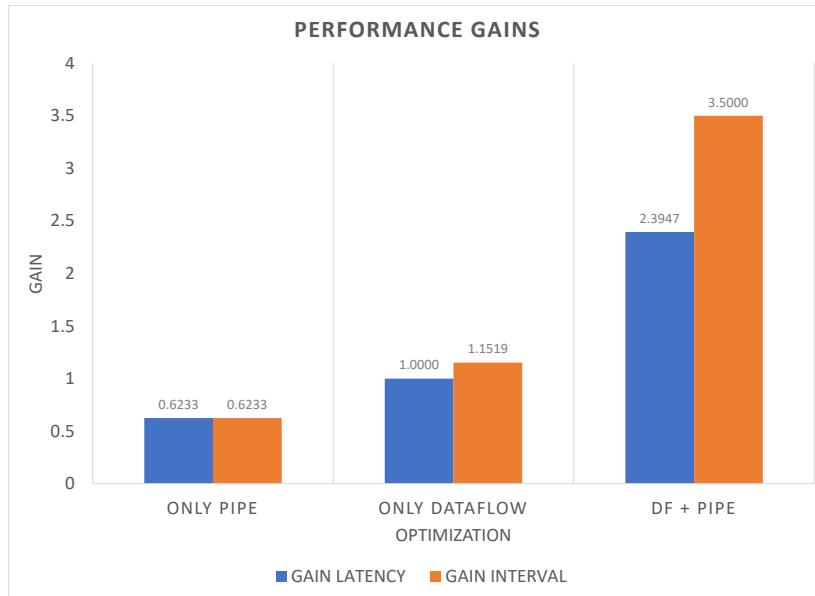


FIGURA 10.7: Ganancia en rendimiento en float

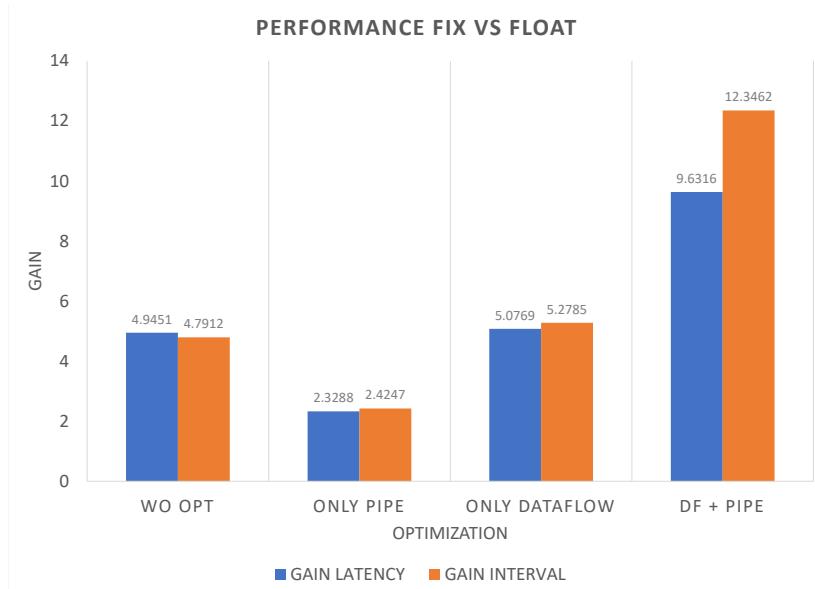


FIGURA 10.8: Diferencia entre punto fijo y flotante

que era la mejor implementación, dado que los AXI HP otorgan un ancho de banda más amplio en tanto que los pesos de la red que no cambian en el tiempo, se pueden trabajar con un puerto AXI Lite.

La Tabla 10.1 enseña la cantidad de recursos de la FPGA usados por la implementación propuesta una vez ha sido sintetizado el hardware necesario. En ella, se muestra en porcentaje usado del hardware disponible de la tarjeta de desarrollo el consumo de cada una de las unidades lógicas y de procesamiento de la FPGA por cada una de las optimizaciones propuestas. Se observa que la implementación de **DATAFLOW** más **PIPELINE**

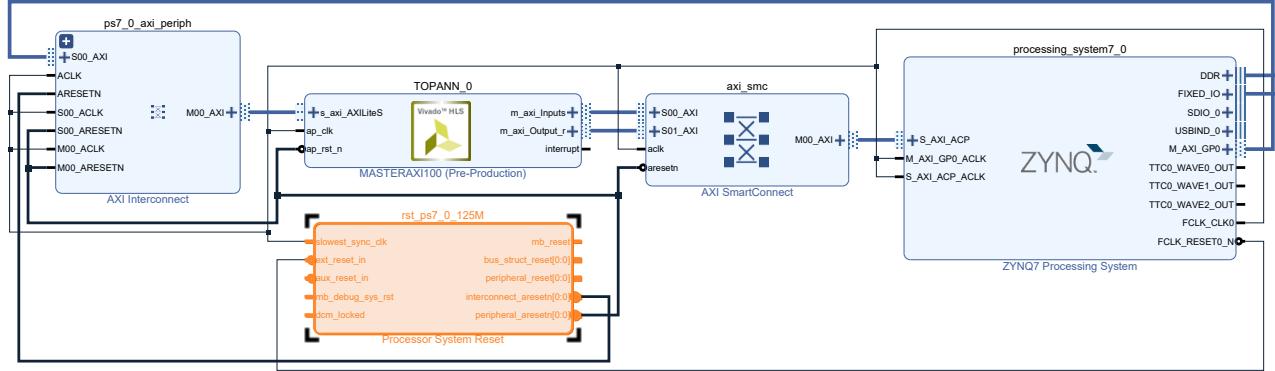


FIGURA 10.9: Implementación final de hardware

escogida para ser implementada es la que presenta mayor uso de recursos. Esto se debe al alto nivel de concurrencia que ocurre a nivel de las tareas y los bucles, otorgando un paralelismo máximo. Las implementaciones de **DATAFLOW**, **PIPE UNROLL**, y **NO OPT** son similares en cuanto a uso de hardware y su diferencia debería ser notable en tiempo de ejecución. Finalmente, en búsqueda de un equilibrio entre los recursos y el rendimiento del bloque, se configuró el sintetizador en modo de optimización de área; esto lo que garantiza es que se reutilicen los recursos de hardware disponibles de tal modo que se use la menor cantidad para implementar el circuito. En este escenario, se observa que la síntesis con optimización de área otorga una reducción de aproximadamente el 50% en promedio de los recursos totales utilizados frente a la solución numéricamente más deseable.

	LUT %	LUT RAM %	FF %	BRAM %	DSP %	AVERAGE %
DF PIPE AREA OPT	27	12	15	13	15	16.4
DF PIPE	53	22	25	35	35	34
DATA FLOW	38	24	19	13	15	21.8
PIPE UNROLL	38	23	21	15	20	23.4
NO OPT	37	23	19	13	15	21.4

TABLA 10.1: Tabla uso de recursos post síntesis

Con esto se tiene un conjunto de pruebas de cuatro circuitos diferentes para ser analizados en la plataforma de pruebas.

10.2. Diseño de Software

El diseño consta de múltiples partes. Una de ellas es el diseño de la ANN como tal; el diseño de hardware y optimizaciones a implementar; y, el esquema de reconfiguración dinámico basado en el contexto de ejecución, en este caso el estado o riesgo de inundación que pueda tener el río en un momento determinado. En esta sección se analizarán los criterios para el diseño del software; en este caso, es la capa que interconecta el hardware diseñado en la sección anterior con el procesador principal ARM A9, usando como medio el kernel de Linux.

En este caso, el diseño del software requiere conocer sobre el user space driver del kernel de Linux, que a su vez implica mapear las direcciones físicas de memoria entregadas por el módulo IP implementado en la FPGA a un espacio de memoria virtual manejado por el kernel.

La Figura 10.10 muestra la mecánica de un driver funcionando bajo un esquema de espacio de usuario: el dispositivo hardware, en este caso “Device”, recibe y escribe la información que necesita bajo una dirección física de memoria. El usuario, para acceder a esas direcciones de memoria, debe conocer la dirección base física a la cual el dispositivo está accediendo, el cual es provisto por el diseñador del hardware o en este caso el sintetizador HLS y el tamaño de paginación de memoria que entrega el SO. Con esto, el driver puede mapear la dirección de memoria a una virtual a la cual la aplicación pueda acceder como una referencia de memoria; en el caso de C/C++, como un apuntador que puede ser manipulado según sea necesario.

10.2.1. Petalinux SDK

Petalinux es una distribución especial del proyecto Yocto [75], cuya razón de existir es ayudar a los desarrolladores a crear sus propias distribuciones del kernel de Linux para dispositivos embebidos. Xilinx desarrolló su propia versión del kernel basado en Yocto llamándolo Petalinux y su conjunto de herramientas conocidas como Petalinux SDK. Petalinux SDK cuenta con un gran conjunto de herramientas de compilación y ayudas al desarrollador para simplificar el proceso de personalizar y compilar el kernel de Linux para ser distribuido dentro de una de las tarjetas de desarrollo de Xilinx.

Para que un bloque IP funcione dentro de un chip ZYNQ de Xilinx debe cumplir varios requisitos.

- La FPGA debe ser configurada por el JTAG o por el PS7.
- Los registros de memoria y del procesador deben ser fijados usando el procedimiento estándar de arranque descrito en la Figura 10.11.
- Si se requiere una implementación con el kernel Petalinux, se debe garantizar que el kernel sea compilado usando Petalinux SDK con el DeviceTree y el Board Support Package (bitstream incluido) creado usando la herramienta de síntesis Vivado.

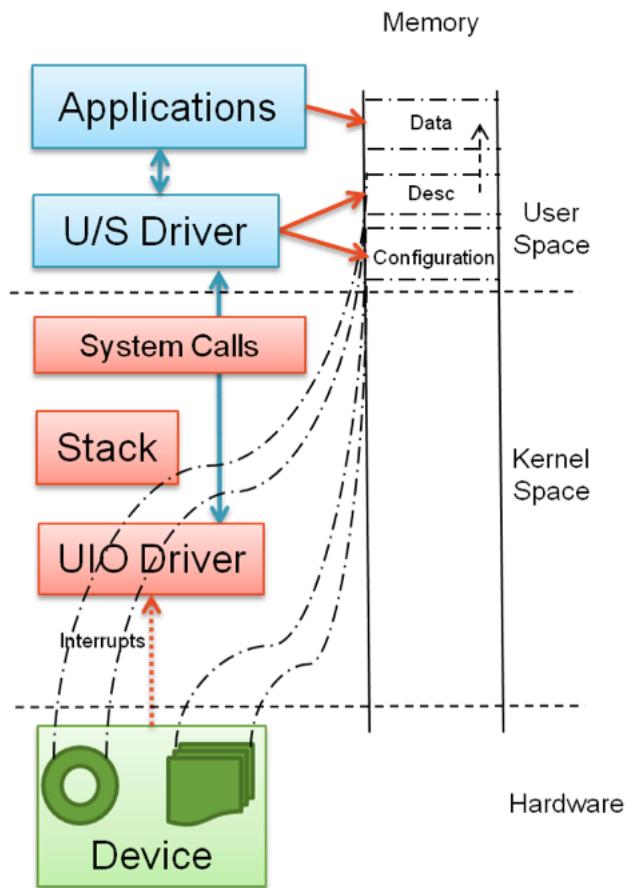


FIGURA 10.10: Esquema de driver en espacio de usuario[7].

- Se debe inicializar el bloque IP usando los registros otorgados por la herramienta de síntesis de alto nivel Vivado HLS.

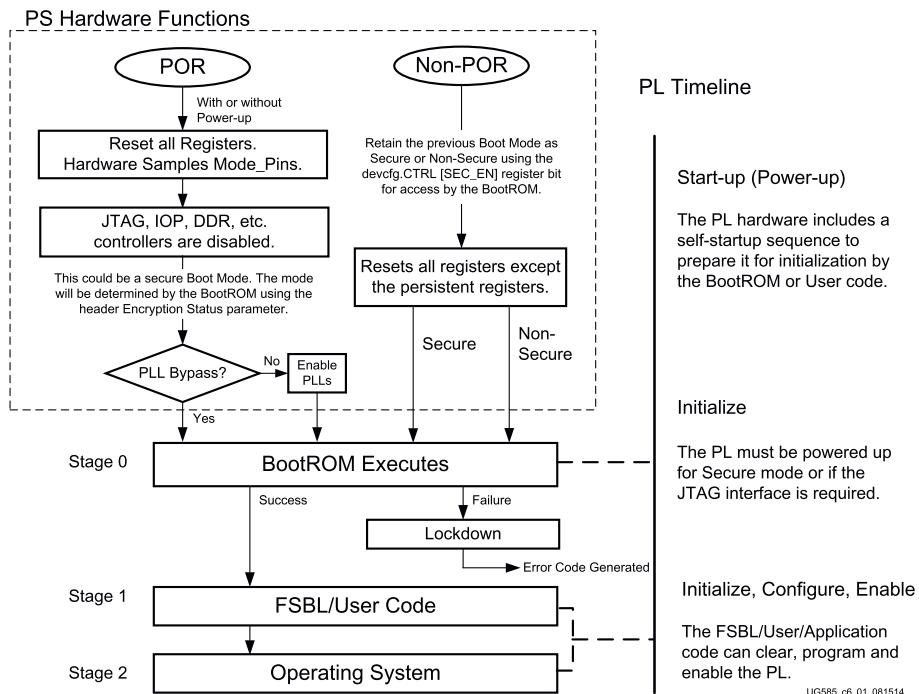


FIGURA 10.11: Estructura de arranque de sistemas ZYNQ [8]

10.2.1.1. Configuración de Petalinux SDK

Dado el volumen de datos y la posibilidad de cambiar el software en línea sin recompilar el kernel de Linux se decidió que la solución debería contar con las siguientes características:

- Persistencia de los datos.
- Conexión de red Ethernet.
- Conexión serial a 115200 baudios.
- Ejecutar software usando el User Space de Linux.

Para cumplir con lo anterior, se configura el kernel de Linux de la siguiente forma:

```
// usando bash de Linux
petalinux-config
// Una vez en el menú configurar de la siguiente manera
Image Package Configuration -> Root File System Type -> SD Card
// Salvar los cambios y salir del menú

// Usando bash de Linux
petalinux-config -c kernel
```

```
//Configurar de la siguiente manera
Device Drivers ->
<*>User I/O platform driver with generic IRQ handling
<*>User Platform Driver with generic irq and dynamic memory
// Salvar y salir del menú
```

Se debe garantizar la configuración del Device Tree Blob de la siguiente manera:

```
// Válido para Petalinux SDK 2017.4 y compatibles
// Modificar el archivo system-user.dtsi
// Ubicación: ./carpeta-proyecto/proj-spec/meta-user/recipes-bsp/device-tree/files

/include/ "system-conf.dtsi"
{

chosen {
bootargs = "console=ttyPS0,115200 earlyprintk uio_pdrv_genirq.of_id=generic-uio
            root=/dev/mmcblk0p2 rw rootwait";
};

};

&TOPANN_0 {
compatible = "generic-uio";
reg = <0x43c00000 0x10000>;
xlnx,s-axi-axilites-addr-width = <0x8>;
xlnx,s-axi-axilites-data-width = <0x20>;
};

&gem0 {
phy-handle = <&ethernet_phy>;
/* phy-reset-gpio = <&axi_gpio_eth 0 0>;
phy-reset-active-low;
phy-reset-duration = <10>; */
ethernet_phy: ethernet-phy@1 { /* rtl8211e-vl */
reg = <1>;
device_type = "ethernet-phy";
};
};
```

De esta forma, con las configuraciones dadas anteriormente, es posible compilar el kernel de Linux y con el driver adecuado debería ser posible que el hardware funcione dentro de la FPGA y sea controlado desde Petalinux.

10.2.2. Diseño del driver

Para el driver se tomó la decisión de usar el lenguaje C++ dado que permite el uso del paradigma de programación orientada a objetos, el cual permite crear diferentes módulos con funcionalidad determinada que luego pueden ser heredados a otros objetos para mejorar o ampliar el funcionamiento de los mismos. En este caso, se usaron diferentes clases u objetos que permitirían al desarrollador mapear la memoria, inicializar el módulo de aceleración de hardware con las secuencias de bits requeridos por la interfaz AXI y la operación del módulo de como tal, es decir la carga de los pesos de la ANN, los datos a procesar y los resultados de la operación.

La implementación del driver se ve en la Figura 10.12. En ella se resumieron los métodos usados para mapear la memoria y manipular los datos que llegan y van hacia el módulo acelerador en la FPGA. La clase `ANNService` recibe como dependencia las direcciones de memoria mapeadas en la clase `Driver` mediante la clase `Controller`. La clase `ANNService` a su vez, usa la clase `ANN.DAO`, que recibe como dependencia, una estructura de almacenamiento desacoplando la aplicación del método de persistencia. En este caso, dado que se busca obtener el performance de la aplicación como tal, se usó persistencia en memoria.

10.2.3. Diseño del controlador de reconfiguración

Teniendo en cuenta que se requiere que el controlador monitoree el estado del río mediante el estado de la ANN, se optó por un diseño reactivo del controlador, donde el riesgo de inundación disparado por la ANN activa los mecanismos de reconfiguración, la velocidad de muestreo y la recepción de datos proveniente de los nodos aledaños.

La Figura 10.13 muestra cómo estaría configurada la FPGA en su modo lazy o flojo, pudiendo atender los datos provenientes de los nodos aledaños cuando el Controlador detecta un riesgo de inundación; es decir que, la salida de la red “ANN 1” sobrepasa el umbral de entre 70 % y 80 %. Cuando el PS detecta una salida de la “ANN 1” sobre ese umbral, el procesador reconfigura automáticamente la FPGA para recibir un mayor número de entradas provenientes de campo. Dado que es el mismo circuito, todos estos datos pasarían por la misma interfaz AXI High Performance.

Al reconfigurar el circuito se espera que la cantidad de área asignada al circuito cambie. En este caso, la variación está dada en función de las entradas a analizar (Neuronas de Entrada), pues se incrementa la cantidad de Look Up Tables y Block RAMs necesarias para operar la ANN.

Esto aplica también para configuraciones o aplicaciones en las que dinámicamente se adiciona hardware; la diferencia con la Figura 10.14, es que se crean y se usan nuevas interfaces AXI HP que se unen al bus AMBA.

La Figura 10.15 muestra un esquema de configuración para múltiples aplicaciones ejecutándose en la FPGA de forma concurrente. Cada una de las instancias de las aplicaciones posee su propia interfaz AXI otorgando acceso directo a la RAM, por lo que

User Space Driver - Class Diagram

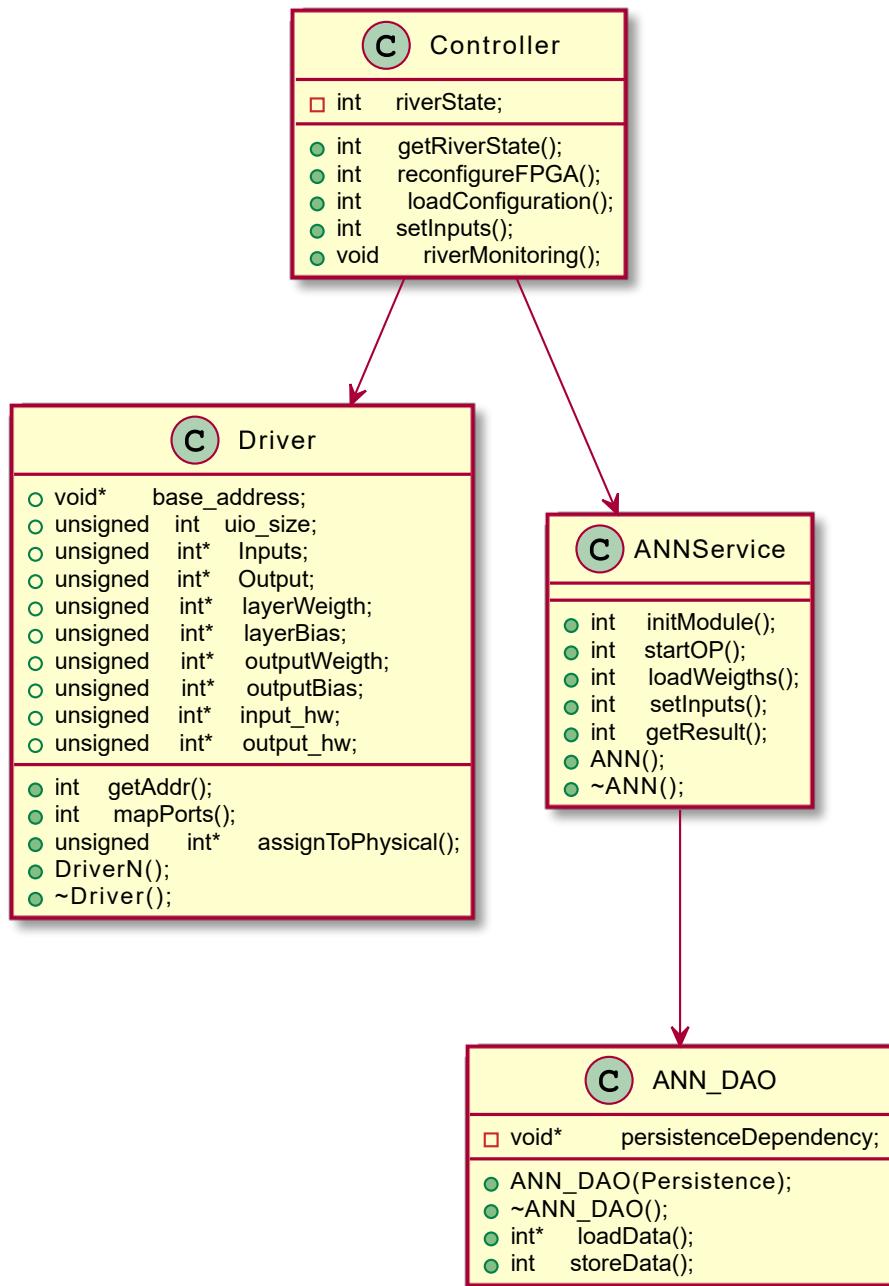


FIGURA 10.12: Diagrama de clases de la aplicación

cada una podría realizar actividades diferentes y la única actividad que debería realizar el controlador es alimentar y monitorear las direcciones de memoria correspondientes a cada uno de los bloques.

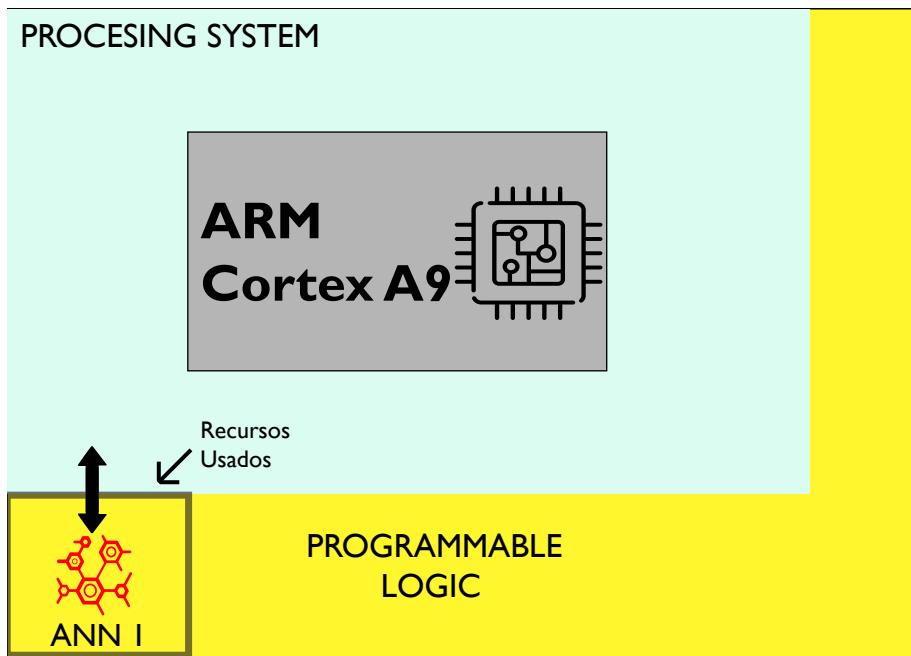


FIGURA 10.13: Esquema de funcionamiento ANN

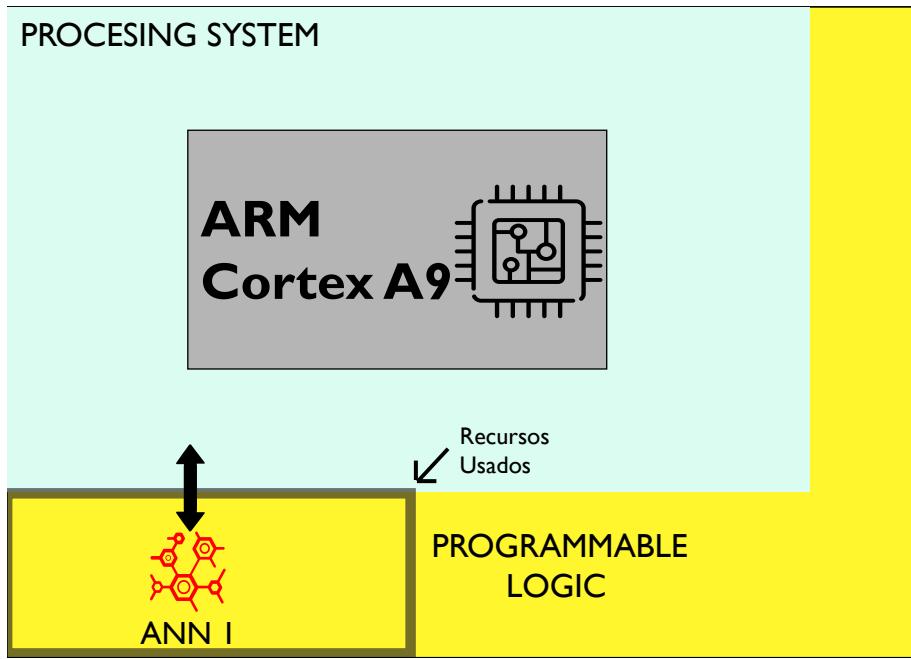


FIGURA 10.14: Esquema de funcionamiento ANN reconfigurada

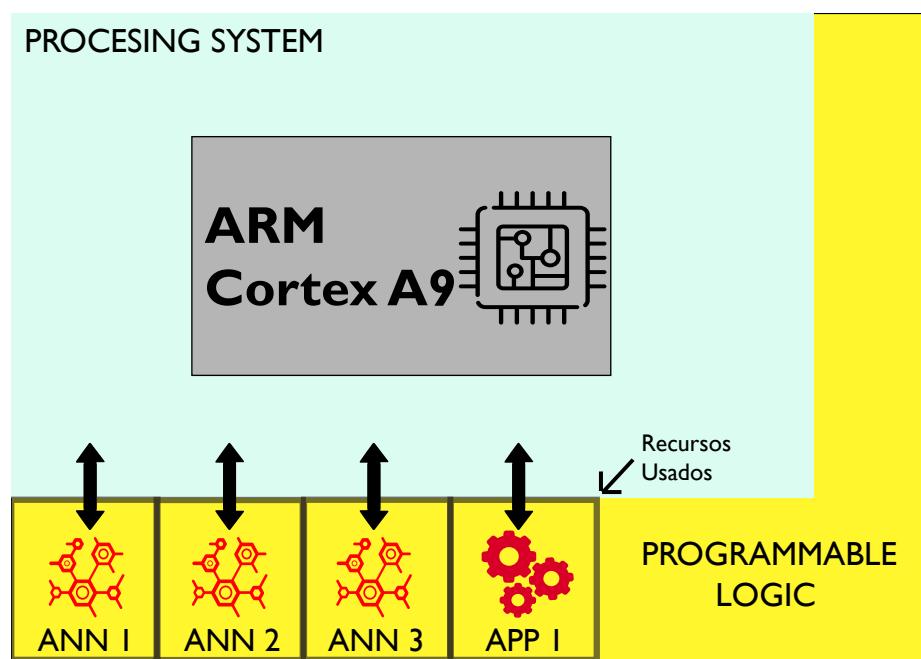


FIGURA 10.15: Esquema de funcionamiento múltiples aplicaciones

Capítulo 11

Diseño Experimental

Basado en el diseño del Capítulo 10, se desarrollan las siguientes actividades con el objetivo de encontrar el mejor resultado para la implementación buscada; en este caso, un sistema embebido capaz de operar concurrentemente con un sistema operativo que se encargue de gestionar una conexión de red hacia la nube.

Se usa como banco de pruebas la tarjeta de desarrollo ZYBO con la distribución de Petalinux 2017.4 con Kernel de Linux versión 4.9.0 y File System por defecto de Petalinux 2017.4.

- Se analizan los tiempos de ejecución del código a implementar en Hardware junto con la versión Software mediante un Benchmark. Este medirá el tiempo de ejecución a lo largo de tres iteraciones; en el caso del HW, se medirá el rendimiento del mismo a las frecuencias de 100MHz y 150MHz, pudiendo así determinar la eficiencia del circuito.
- Se analizarán los tiempos de ejecución del Hardware a lo largo de 4 estrategias de optimización obtenidas mediante HLS así como la cantidad de recursos usados en cada solución.
- Se analizará el circuito con mejor rendimiento contra él mismo, modificando la estrategia de síntesis a una de optimización de área buscando reducir la cantidad de recursos usados por la solución.
- Finalmente se determina la relación rendimiento contra uso de recursos buscando la eficiencia más alta.
- Se determinará la ganancia SW vs HW; posteriormente, se analizará la viabilidad de la implementación en campo.

La conexión entre el computador y la tarjeta de desarrollo se realiza dentro de una red local usando un enrutador de red como intermediario 11.1. La manipulación de la consola de Linux se realiza usando PuTTY y la manipulación de los archivos de configuración se realiza usando un cliente SFTP con FileZilla.

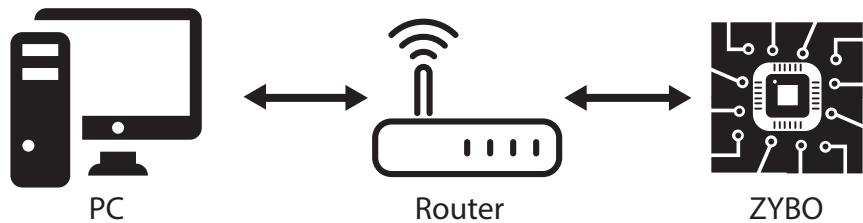


FIGURA 11.1: Esquema de conexión

Capítulo 12

Comparación Modelo de la ANN

Es importante mencionar que el rendimiento de una ANN depende directamente de la calidad del entrenamiento y del conjunto de datos elegido. Por lo tanto, en este capítulo se analizarán comparativamente los niveles de detección entre un modelo de ANN usando las herramientas de Matlab (nntool) frente al modelo en hardware propuesto en este documento en el Capítulo 10, Figura 10.1. El objetivo es demostrar que el modelo propuesto es válido y puede ser implementado a nivel de hardware directamente. Para esto se eligió mostrar la detección del modelo de la red en punto fijo usando el entrenamiento de Matlab.

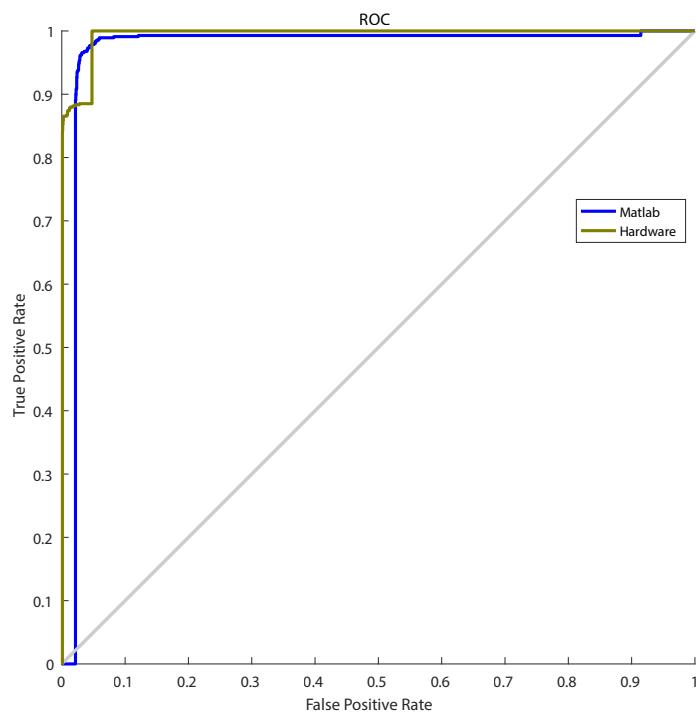


FIGURA 12.1: Curva ROC de detección

La Figura 12.1 muestra la curva ROC o *Receiver Operating Characteristic* por sus siglas en inglés. Esta gráfica lo que muestra es la calidad del clasificador al discriminar entre un valor de detección positivo y otro negativo basado en el conjunto de datos de entrenamiento. Particularmente, una curva ROC muestra lo siguiente según su forma [76]:

- Se muestra el balance entre la sensibilidad (eje vertical) y la especificidad (eje horizontal), donde un incremento en la sensibilidad va acompañado de un detrimento de la especificidad.
- Entre más cerca la curva esté al eje vertical izquierdo y al eje horizontal superior, más efectivo es el clasificador.
- Entre más cerca esté la curva a la diagonal, menos efectivo es el clasificador.

A partir de la curva ROC es entonces posible determinar que el modelo se encuentra en capacidad de detectar eficazmente el riesgo de inundación y que el modelo en Matlab presenta un desempeño aceptable para el propósito planteado, cual es el de presentar una plataforma de computación de borde con capacidad de acelerar el procesamiento y análisis de múltiples fuentes de datos. Por ello, no se invirtió más tiempo en sintonizar los parámetros de entrenamiento del modelo, lo que ocasiona que este presente algunas detecciones falsas; esto mayoritariamente se debe a las fluctuaciones generadas por el transitorio entre los estados de inundación y sequía propios del conjunto de datos, generando falsos positivos y/o negativos. Esto es posible corregirlo ajustando los parámetros del entrenamiento a tal punto que las fluctuaciones en la detección sean menos notorias, pero particularmente para este problema las transiciones podrían seguir presentes. Ajustando la salida de la ANN con un circuito de retraso que pueda amortiguar los cambios de estado permitiría filtrar la salida de la ANN de tal forma que no se vea afectada la medición final.

Finalmente, se hace necesario comparar punto a punto la salida de la ANN con el fin de determinar que, efectivamente tanto el modelo en Matlab como su correspondiente implementación en hardware, están entregando una equivalencia para el conjunto de datos entregado. Para ello se eligió calcular el índice de correlación y la correlación cruzada. En el primer caso, el índice de correlación es de 0.9329, siendo un valor de 1 una perfecta relación positiva entre dos señales [77, 78]. La Figura 12.2 muestra la correlación cruzada entre las señales provenientes del modelo de Matlab y el calculado con el entrenamiento sobre el hardware. Se ve claramente que, según la teoría, la forma triangular sugiere que se trata de una correlación entre dos señales muy similares entre sí, confirmando que el entrenamiento y la detección de ambas redes es equivalente.

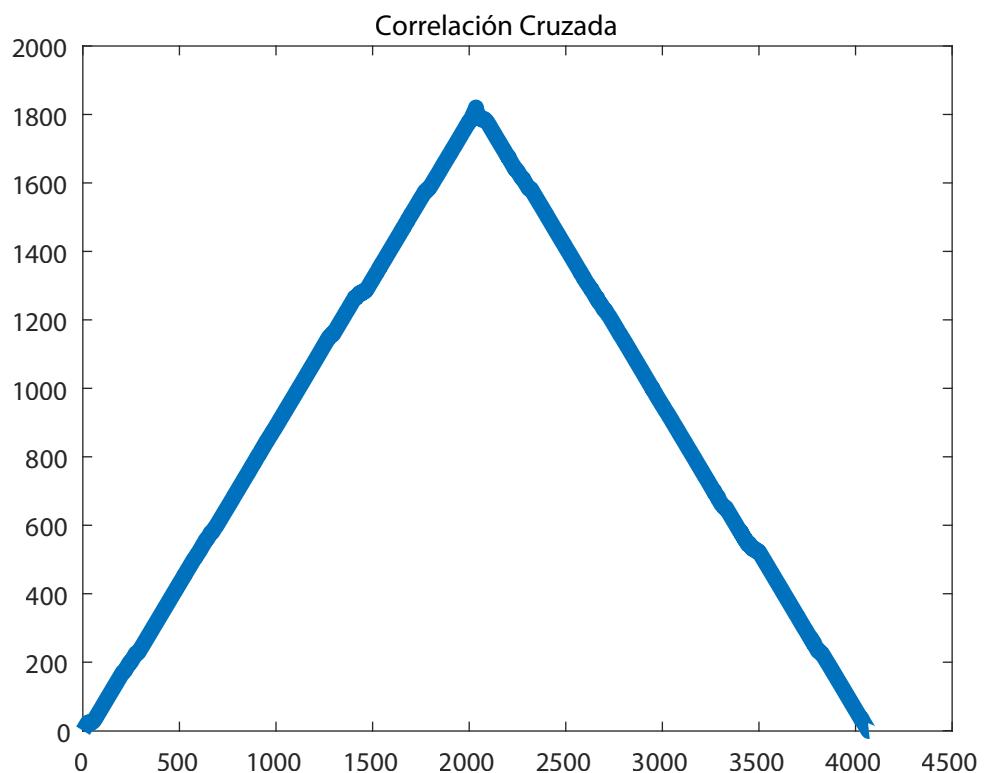


FIGURA 12.2: Correlación cruzada entre modelos

Capítulo 13

Análisis de Resultados

En este capítulo se muestran los resultados obtenidos en la implementación del hardware. Para esto se siguió la metodología descrita en el capítulo [11](#):

13.1. Benchmark

Dada la necesidad de medir el rendimiento del circuito implementado en HW contra la versión SW, se determinó que la mejor métrica se obtendría realizando un Benchmark de tres etapas con ejecuciones de 100 (**SW_ 100M**) y 300 millones de operaciones por ejecución (**SW_ 300M**), con la implementación en HW funcionando a 100MHz (**HW_ 100M_ 100MHz**, **HW_ 300M_ 100MHz**) y 150MHz (**HW_ 100M_ 150MHz**, **HW_ 300M_ 150MHz**), que son las frecuencias de reloj recomendadas por el sintetizador de alto nivel. La prueba consiste en cargar las neuronas de entrada y calcular el tiempo que le tomó a ambas aproximaciones realizar el cálculo del resultado.

Como se observa en la Figura [13.1](#), las implementaciones **HW_ 100M_ 100MHz**, **HW_ 100M_ 150MHz**, **HW_ 300M_ 100MHz** y **HW_ 300M_ 150MHz** comparado con las implementaciones **SW_ 100M** y **SW_ 300M** la aceleración es aproximadamente cuatro veces. Esta prueba lo que busca es saturar el bus AMBA cargando y exigiendo al máximo el pipeline del bus usando las optimizaciones burst de la interfaz AXI. A diferencia de la implementación HW, la implementación software realiza una carga de las neuronas de entrada de forma secuencial adicionando un tiempo de carga alto debido a los múltiples llamados al bus que se encarga de la RAM.

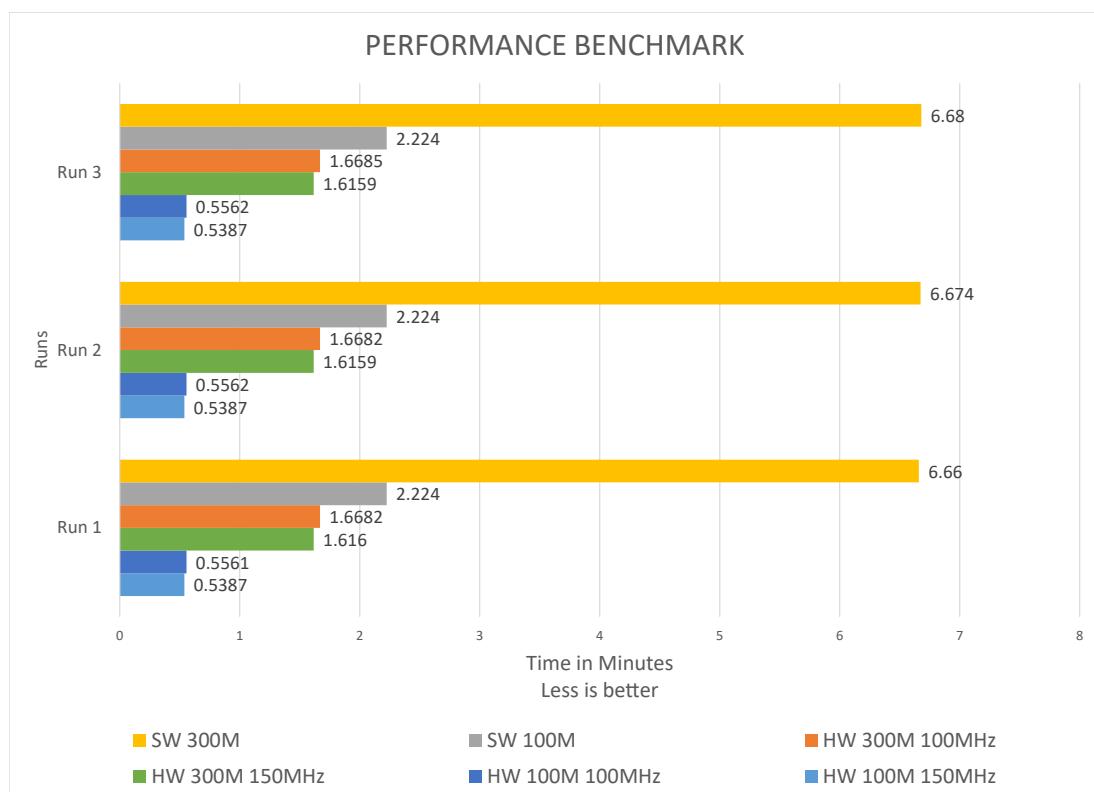


FIGURA 13.1: Benchmark resultados de la implementación

13.2. Análisis de tiempo y recursos

13.2.1. Tiempo de ejecución

En la Figura 13.2 se observan los resultados de la implementación de diferentes estrategias de optimización del hardware; en este caso tenemos:

- WO-OPT: Sin optimizaciones.
- P-UNROLL: Pipelining a nivel de bucles combinado con Unroll a los bucles.
- DATAFLOW: Pipelining a nivel de tareas.
- AO: Optimización de área.
- PO: Pipelining a nivel de tareas, bucles y unroll

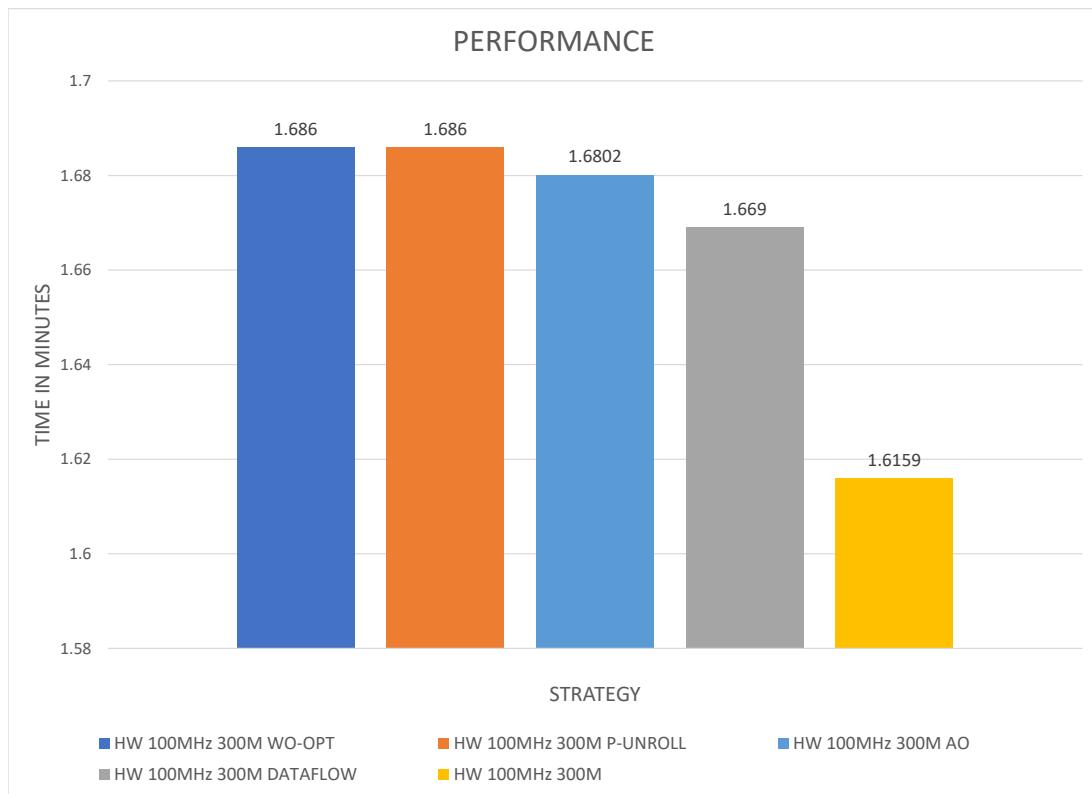


FIGURA 13.2: Rendimiento del acelerador en Hardware

La Figura muestra claramente que la estrategia “PO” ofrece el mejor rendimiento en tiempo de ejecución de las tareas, siendo la estrategia a implementar en producción.

13.2.2. Recursos utilizados

Una vez determinados los tiempos de ejecución, se hace necesario tener en cuenta el área o la cantidad de recursos necesarios para cumplir las diferentes tareas de la solución. En este caso se analizó la cantidad de recursos para cada estrategia teniendo en cuenta el reporte del sintetizador de hardware Vivado.

La Figura 13.3 muestra el porcentaje de recursos usados por cada una de las diferentes estrategias. Es importante notar que la estrategia “AO” usa 51 % menos LUT, 42 % menos DSPs y 37 % menos BRAM que su contraparte “PO” para un rendimiento casi marginal en tiempo.

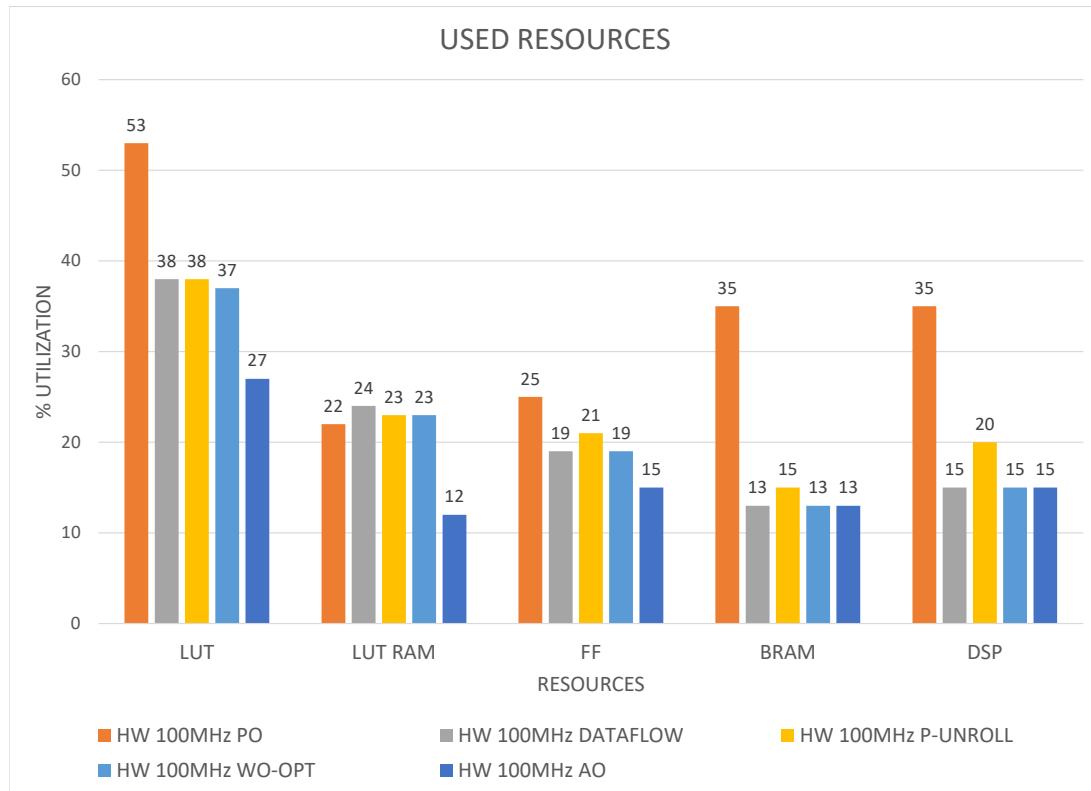


FIGURA 13.3: Recursos utilizados en la FPGA

La Figura 13.4 muestra claramente que la estrategia “AO” es aproximadamente 46 % más eficiente que su contraparte “PO”. De esta forma, es posible determinar que, al usar menos área vamos a obtener un mejor delta de eficiencia, pues al usar menos recursos es posible adicionar más módulos de hardware o, en otros casos, usar menos potencia en tiempo de ejecución.

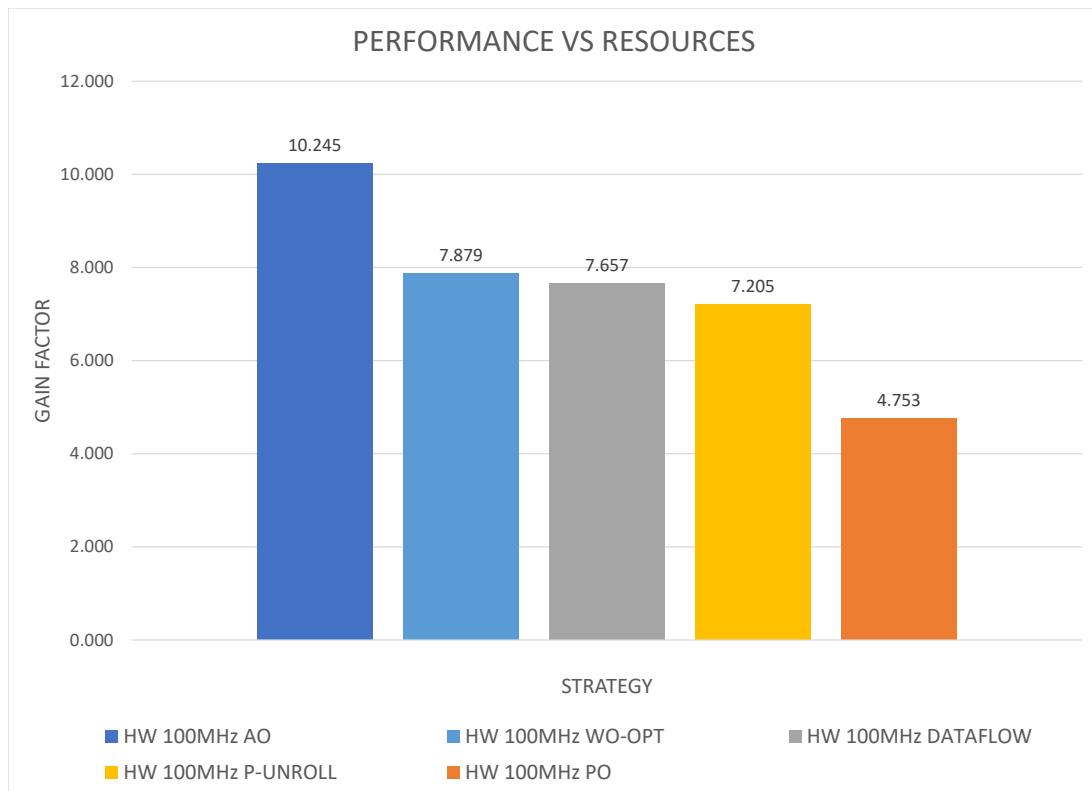


FIGURA 13.4: Factor de ganancia recursos contra tiempo

Capítulo 14

Conclusiones

De acuerdo con la revisión del estado del arte no se encontró una solución que articule las dimensiones de aceleración y reconfiguración dinámica del hardware en una plataforma de computación de borde usando un esquema orientado al Internet de las Cosas, por lo cual se considera que el enfoque que se plantea en este trabajo es la principal contribución del mismo.

En particular, se muestra que el procesamiento y fusión de altos volúmenes de datos en el borde de una red IoT es posible usando dispositivos orientados al Internet de las Cosas con hardware para incrementar la potencia de cálculo y software especializado para coordinar y adicionar funcionalidad basado en esquemas de reconfiguración dinámica del hardware. Con un diseño de hardware y las optimizaciones propuestas, es posible realizar múltiples operaciones sobre los datos concurrentemente con un bajo footprint sobre el procesador, relegando únicamente la carga operativa a la FPGA y dejando que el procesador atienda los diferentes servicios de red que implican una conexión con la Nube.

La elección del hardware y los recursos necesarios para la implementación de una plataforma de computación de borde están basados netamente en las necesidades que se desean cumplir. En este caso, hay indicios de que con un hardware (procesador, memoria, almacenamiento) relativamente limitado es posible prototipar y posteriormente desplegar en producción un dispositivo enteramente capaz de realizar el trabajo de otros dispositivos con TDP mucho más altos.

Se determinó que es posible implementar directamente el modelo Hardware entrenado, garantizando el funcionamiento y detección basado en un modelo software usando las herramientas de desarrollo y entrenamiento de Matlab.

Más aún, una implementación directamente sobre el hardware permite tener control fino sobre las diferentes etapas de optimización, manipulación de los datos y consumo de recursos de hardware a diferencia de implementaciones usando las herramientas de Matlab como System Generator que permiten sintetizar hardware a un nivel muy alto sin opciones de optimización y consumo de recursos para sistemas como el propuesto en el que el hardware es muy limitado.

Como parte de la funcionalidad de la plataforma, se integraron exitosamente los múltiples módulos de hardware como bloques IP en forma de repositorio; estos se encuentran almacenados al interior de la memoria micro SD que contiene el File System y la configuración de arranque del sistema operativo. Esto con el objetivo de que la plataforma de computación de borde tenga a disposición local los módulos que permitan la reconfiguración de la FPGA en tiempo de ejecución.

El diseño de hardware propuesto proporciona una ganancia de 4x la velocidad de cálculo sobre su contraparte software con el mismo diseño base, convirtiéndolo en una solución mucho más eficiente para el procesamiento de los datos en el borde de una red IoT. Además, con las optimizaciones de área correctas es posible integrar múltiples módulos IP que realicen diferentes operaciones y transformaciones sobre los datos, sacrificando rendimiento que puede o no ser notable, dependiendo de la complejidad y optimizaciones de hardware que requiera el circuito.

La implementación de una solución como la propuesta requiere de pasos muy precisos en la configuración del kernel, más aún si se desean características específicas como un sistema de archivos de Ubuntu o módulos de hardware como una DMA para video o para mover grandes bloques de datos de una región a otra de memoria. Con una arquitectura como la propuesta es posible configurar la FPGA en tiempo de ejecución del Kernel de Linux y adicionar nuevos binarios y configuraciones al sistema de archivos usando un servidor sftp y un cliente como FileZilla.

Se estableció que el ancho en bits del bus de datos del circuito hardware no influye sobre la latencia total del circuito, con lo cual se pudo implementar un ancho de datos de 32 bits, garantizando así una alta resolución para las operaciones sin perder rendimiento.

Durante la etapa de desarrollo inicial del hardware se determinó que el canal de datos de alta velocidad ofrecido por la interfaz AXI Full es el requerido para una aplicación donde el alto volumen de datos a procesar es un factor determinante para el rendimiento total de la solución. Esto se evidencia al implementar la solución con la interfaz AXI Lite, con la cual se obtuvo un rendimiento más bajo que la solución netamente software.

Bibliografía

- [1] S. Li, L. Da Xu, and S. Zhao, “The internet of things: a survey,” *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.
- [2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, ACM, 2012.
- [3] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, “Fog computing: A platform for internet of things and analytics,” in *Big Data and Internet of Things: A Roadmap for Smart Environments*, pp. 169–186, Springer, 2014.
- [4] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context aware computing for the internet of things: A survey,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.
- [5] *Vivado Design Suite User Guide HLS*, 9 2017.
- [6] *Vivado Design Suite AXI Reference Guide*, 12 2017.
- [7] R. M. Hemant Agrawal, “Device drivers in user space — embedded.” <https://www.embedded.com/design/operating-systems/4401769/Device-drivers-in-user-space>, 11 2012. (Accessed on 05/25/2018).
- [8] *Zynq-7000 All Programmable SoC Technical Reference Manual*, 12 2017.
- [9] J. Mattai and M. Joseph, *Real-Time Systems: specification, verification, and analysis*. Prentice Hall PTR, 1995.
- [10] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” pp. 13–16, 2012.
- [11] Microsoft, “Pricing calculator — Microsoft Azure.” <https://goo.gl/rvv4Ql>, 2016. (Accessed on 11/01/2016).
- [12] Amazon, “Amazon web services pricing calculator.” <https://goo.gl/R5PwWq>, 2016. (Accessed on 11/02/2016).
- [13] Google, “Google cloud platform pricing calculator — google cloud platform.” <https://goo.gl/E4zCnk>, 2016. (Accessed on 11/02/2016).
- [14] Ubidots, “Ubidots pricing.” <https://goo.gl/a5vyym>, 2016. (Accessed on 11/02/2016).

- [15] B. Salami, G. MALAZGIRT, O. Arcas-Abella, A. Yurdakul, and N. Sonmez, “Ax-ledb: A novel programmable query processing platform on FPGA,” *Microprocessors and Microsystems*, 2017.
- [16] Z. Liu, J. Großschädl, Z. Hu, K. Järvinen, H. Wang, and I. Verbauwhede, “Elliptic curve cryptography with efficiently computable endomorphisms and its hardware implementations for the internet of things,” *IEEE Transactions on Computers*, vol. 66, no. 5, pp. 773–785, 2017.
- [17] G. Mingas, L. Bottolo, and C.-S. Bouganis, “Particle mcmc algorithms and architectures for accelerating inference in state-space models,” *International Journal of Approximate Reasoning*, 2016.
- [18] F. Mehdipour, B. Javadi, and A. Mahanti, “Fog-engine: towards big data analytics in the fog,” pp. 640–646, 2016.
- [19] F. T. Mahmoudi, F. Samadzadegan, and P. Reinartz, “Object recognition based on the context aware decision-level fusion in multiviews imagery,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 1, pp. 12–22, 2015.
- [20] A. Abrardo, M. Martalò, and G. Ferrari, “Information fusion for efficient target detection in large-scale surveillance wireless sensor networks,” *Information Fusion*, vol. 38, pp. 55–64, 2017.
- [21] P. Mitra, R. Ray, R. Chatterjee, R. Basu, P. Saha, S. Raha, R. Barman, S. Patra, S. S. Biswas, and S. Saha, “Flood forecasting using internet of things and artificial neural networks,” in *Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2016 IEEE 7th Annual*, pp. 1–5, IEEE, 2016.
- [22] S. H. Elsafi, “Artificial neural networks (anns) for flood forecasting at dongola station in the river nile, sudan,” *Alexandria Engineering Journal*, vol. 53, no. 3, pp. 655 – 662, 2014.
- [23] M. A. M. Sahagun, J. C. D. Cruz, and R. G. Garcia, “Wireless sensor nodes for flood forecasting using artificial neural network,” in *2017 IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, pp. 1–6, Dec 2017.
- [24] I. Stojmenovic and S. Wen, “The fog computing paradigm: scenarios and security issues,” in *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*, pp. 1–8, IEEE, 2014.
- [25] H. Madsen, B. Burtschy, G. Albeanu, and F. Popentiu-Vladicescu, “Reliability in the utility computing era: towards reliable fog computing,” in *2013 20th International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 43–46, IEEE, 2013.
- [26] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, “Mobile fog: A programming model for large-scale applications on the internet of things,” in *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, pp. 15–20, ACM, 2013.

- [27] M. Aazam and E.-N. Huh, "Fog computing and smart gateway based communication for cloud of things," in *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*, pp. 464–470, IEEE, 2014.
- [28] M. Aazam and E.-N. Huh, "Fog computing micro datacenter based dynamic resource estimation and pricing model for iot," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pp. 687–694, IEEE, 2015.
- [29] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Distributed qos-aware scheduling in storm," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, pp. 344–347, ACM, 2015.
- [30] V. Subramanian, D. Chan, A. K. Moghe, R. Pan, and F. Bonomi, "Dynamic coding for network traffic by fog computing node," Jan. 5 2016. US Patent 9,232,433.
- [31] J. Zhu, D. Chan, M. M. S. PRABHU, P. NATARAJAN, and H. Hu, "Improving web sites performance using edge servers in fog computing architecture," May 29 2013. US Patent App. 13/904,327.
- [32] M. M. Molla and S. I. Ahamed, "A survey of middleware for sensor network and challenges," in *Parallel Processing Workshops, 2006. ICPP 2006 Workshops. 2006 International Conference on*, pp. 6–pp, IEEE, 2006.
- [33] S.-H. Sho, K.-S. Kim, W.-r. Jun, J.-S. Kim, S.-H. Kim, and J.-D. Lee, "Ttcg: three-tier context gathering technique for mobile devices," in *Proceedings of the 5th international conference on Pervasive services*, pp. 157–162, ACM, 2008.
- [34] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé, "Vision and challenges for realising the internet of things," *Cluster of European Research Projects on the Internet of Things, European Commision*, 2010.
- [35] D. Martin, C. Lamsfus, and A. Alzua, "Automatic context data life cycle management framework," in *Pervasive Computing and Applications (ICPCA), 2010 5th International Conference on*, pp. 330–335, IEEE, 2010.
- [36] F. Ramparany, R. Poortinga, M. Stikic, J. Schmalenstroer, and T. Prante, "An open context information management infrastructure-the ist-amigo project," 2007.
- [37] A. M. Bernardos, P. Tarrio, and J. R. Casar, "A data fusion framework for context-aware mobile services," in *Multisensor Fusion and Integration for Intelligent Systems, 2008. MFI 2008. IEEE International Conference on*, pp. 606–613, IEEE, 2008.
- [38] B. Schilit, N. Adams, and R. Want, "Context-aware computing applications," in *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pp. 85–90, IEEE, 1994.
- [39] S. K. Sood, R. Sandhu, K. Singla, and V. Chang, "Iot, big data and hpc based smart flood management framework," *Sustainable Computing: Informatics and Systems*, 2017.

- [40] F. Hossain, E. N. Anagnostou, and T. Dinku, "Sensitivity analyses of satellite rainfall retrieval and sampling error on flood prediction uncertainty," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 42, pp. 130–139, Jan 2004.
- [41] G. Merkuryeva, Y. Merkuryev, B. V. Sokolov, S. Potryasaev, V. A. Zelentsov, and A. Lektauers, "Advanced river flood monitoring, modelling and forecasting," *Journal of Computational Science*, vol. 10, pp. 77 – 85, 2015.
- [42] P. K.-T. Nguyen, L. H.-C. Chua, and L. H. Son, "Flood forecasting in large rivers with data-driven models," *Natural Hazards*, vol. 71, pp. 767–784, Mar 2014.
- [43] P. Mitra, R. Ray, R. Chatterjee, R. Basu, P. Saha, S. Raha, R. Barman, S. Patra, S. S. Biswas, and S. Saha, "Flood forecasting using internet of things and artificial neural networks," in *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 1–5, Oct 2016.
- [44] C. Wu and K.-W. Chau, "Evaluation of several algorithms in forecasting flood," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pp. 111–116, Springer, 2006.
- [45] C. Wu and K. Chau, "A flood forecasting neural network model with genetic algorithm," *International journal of environment and pollution*, vol. 28, no. 3-4, pp. 261–273, 2006.
- [46] P. Mitra, R. Ray, R. Chatterjee, R. Basu, P. Saha, S. Raha, R. Barman, S. Patra, S. S. Biswas, and S. Saha, "Flood forecasting using internet of things and artificial neural networks," in *Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2016 IEEE 7th Annual*, pp. 1–5, IEEE, 2016.
- [47] G. Merkuryeva, Y. Merkuryev, B. V. Sokolov, S. Potryasaev, V. A. Zelentsov, and A. Lektauers, "Advanced river flood monitoring, modelling and forecasting," *Journal of Computational Science*, vol. 10, pp. 77–85, 2015.
- [48] S. H. Elsafi, "Artificial neural networks (anns) for flood forecasting at dongola station in the river nile, sudan," *Alexandria Engineering Journal*, vol. 53, no. 3, pp. 655 – 662, 2014.
- [49] M. Campolo, A. Soldati, and P. Andreussi, "Artificial neural network approach to flood forecasting in the river arno," *Hydrological sciences journal*, vol. 48, no. 3, pp. 381–398, 2003.
- [50] R. Adnan, A. M. Samad, M. Tajjudin, and F. A. Ruslan, "Modeling of flood water level prediction using improved rbfnn structure," in *Control System, Computing and Engineering (ICCSCE), 2015 IEEE International Conference on*, pp. 552–556, IEEE, 2015.
- [51] T. Egawa, K. Suzuki, Y. Ichikawa, T. Iizaka, T. Matsui, and Y. Shikagawa, "A water flow forecasting for dam using neural networks and regression models," in *2011 IEEE Power and Energy Society General Meeting*, pp. 1–6, July 2011.
- [52] G. Martin and G. Smith, "High-level synthesis: Past, present, and future," *IEEE Design & Test of Computers*, no. 4, pp. 18–25, 2009.

- [53] Q. Huang, R. Lian, A. Canis, J. Choi, R. Xi, N. Calagar, S. Brown, and J. Anderson, “The effect of compiler optimizations on high-level synthesis-generated hardware,” *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 8, no. 3, p. 14, 2015.
- [54] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. D. Brown, and J. H. Anderson, “Legup: An open-source high-level synthesis tool for fpga-based processor/accelerator systems,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 2, p. 24, 2013.
- [55] C. Kao, “Benefits of partial reconfiguration,” *Xcell journal*, vol. 55, pp. 65–67, 2005.
- [56] P. Wehner, C. Piberger, and D. Görhringer, “Using json to manage communication between services in the internet of things,” in *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2014 9th International Symposium on*, pp. 1–4, IEEE, 2014.
- [57] S. G. Owens, *Design modifications and platform implementation procedures for supporting dynamic partial reconfiguration of FPGA applications*. PhD thesis, Mississippi State University, 2013.
- [58] C. Liu, C. L. Yu, and H. K.-H. So, “A soft coarse-grained reconfigurable array based high-level synthesis methodology: promoting design productivity and exploring extreme fpga frequency,” pp. 228–228, 2013.
- [59] J. S. Monson and B. L. Hutchings, “Using source-level transformations to improve high-level synthesis debug and validation on FPGAs,” pp. 5–8, 2015.
- [60] D. Navarro, Ó. Lucí, L. A. Barragán, I. Urriza, Ó. Jiménez, *et al.*, “High-level synthesis for accelerating the fpga implementation of computationally demanding control algorithms for power converters,” *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1371–1379, 2013.
- [61] G. A. Malazgirt, N. Sonmez, A. Yurdakul, A. Cristal, and O. Unsal, “High level synthesis based hardware accelerator design for processing sql queries,” pp. 27–32, 2015.
- [62] J. Choi, S. Brown, and J. Anderson, “From software threads to parallel hardware in high-level synthesis for fpgas,” pp. 270–277, 2013.
- [63] C. Alias, A. Darte, and A. Plesco, “Optimizing remote accesses for offloaded kernels: application to high-level synthesis for fpga,” pp. 575–580, 2013.
- [64] R. Zhao, M. Tan, S. Dai, and Z. Zhang, “Area-efficient pipelining for fpga-targeted high-level synthesis,” p. 157, 2015.
- [65] W. Liu, H. Chen, and L. Ma, “Moving object detection and tracking based on zynq fpga and arm soc,” in *IET International Radar Conference 2015*, pp. 1–4, IET, 2015.
- [66] “Vivado design suite.” <http://www.xilinx.com/products/design-tools/vivado.html>. (Visited on 03/30/2018).

- [67] F. Amato, A. Lopez, E. M. Peña-Méndez, P. Vaňhara, A. Hampl, and J. Havel, “Artificial neural networks in medical diagnosis,” *Journal of applied biomedicine*, vol. 11, no. 2, pp. 47–58, 2013.
- [68] K. P. Singh, A. Basant, A. Malik, and G. Jain, “Artificial neural network modeling of the river water quality a case study,” *Ecological Modelling*, vol. 220, no. 6, pp. 888–895, 2009.
- [69] C. Sivapathasekaran, S. Mukherjee, A. Ray, A. Gupta, and R. Sen, “Artificial neural network modeling and genetic algorithm based medium optimization for the improved production of marine biosurfactant,” *Bioresource technology*, vol. 101, no. 8, pp. 2884–2887, 2010.
- [70] M. K. D. Kiani, B. Ghobadian, T. Tavakoli, A. Nikbakht, and G. Najafi, “Application of artificial neural networks for the prediction of performance and exhaust emissions in si engine using ethanol-gasoline blends,” *Energy*, vol. 35, no. 1, pp. 65–69, 2010.
- [71] B. M. Wilamowski, “Neural network architectures and learning algorithms,” *Industrial Electronics Magazine, IEEE*, vol. 3, no. 4, pp. 56–63, 2009.
- [72] “Vivado high-level synthesis.” <http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html>. (Visited on 03/30/2018).
- [73] “Petalinux tools.” <http://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html>. (Visited on 03/30/2018).
- [74] “Download putty - a free ssh and telnet client for windows.” <http://www.putty.org/>. (Visited on 03/30/2018).
- [75] “Yocto project – it’s not an embedded linux distribution – it creates a custom one for you.” <https://www.yoctoproject.org/>. (Accessed on 06/04/2018).
- [76] M. Greiner, D. Pfeiffer, and R. Smith, “Principles and practical application of the receiver-operating characteristic analysis for diagnostic tests,” *Preventive veterinary medicine*, vol. 45, no. 1-2, pp. 23–41, 2000.
- [77] “Correlation coefficient – from wolfram mathworld.” <http://mathworld.wolfram.com/CorrelationCoefficient.html>. (Accessed on 07/25/2018).
- [78] S. Smith, *Digital signal processing: a practical guide for engineers and scientists*. Elsevier, 2013.