UNIVERSIDAD D CÓRDOBA

Programa de Doctorado de Computación Avanzada, Energía y Plasmas
Universidad de Córdoba

Tesis Doctoral

Diseño de Sistemas Embebidos Reconfigurables Empleando Elementos de Interconexión en Chip Definidos por Software

Design of Reconfigurable Embedded Systems Using Software-Defined On-Chip Interconnect Elements

---

Memoria presentada para optar al grado de Doctor por

Salvador Ibarra Delgado

Director: Dra. María Brox Jimenez
Co-Director: Manuel Agustín Ortiz López
Co-Director: Remberto Sandoval Aréchiga

Córdoba, 9 de mayo de 2021

TITULO: *Diseño de sistemas embebidos reconfigurables empleando elementos de interconexión en Chip definidos por Software*

AUTOR: *Salvador Ibarra Delgado*

Campus de Rabanales
Ctra. Nacional IV, Km. 396 A
14071 Córdoba

https://www.uco.es/ucopress/index.php/es/
ucopress@uco.es

TÍTULO DE LA TESIS: Diseño de Sistemas Embebidos Reconfigurables Empleando Elementos de Interconexión en Chip Definidos por Software

DOCTORANDO/A: Salvador Ibarra Delgado

INFORME RAZONADO DE LOS DIRECTORES DE LA TESIS

Esta Tesis Doctoral se centra en el desarrollo de la arquitectura de un sistema de interconexión tipo bus que opera bajo el paradigma de Redes definidas por Software en un Sistema en Chip. Los resultados incluidos en la Tesis demuestran que la infraestructura desarrollada es altamente flexible, reconfigurable, escalable, reutilizable y fácil de administrar, por lo que se incrementan las prestaciones otorgadas por los modelos tradicionales de interconexión tipo bus.

Durante el desarrollo de la Tesis Doctoral, el doctorando ha demostrado una gran capacidad de organización y trabajo autónomo adaptándose y buscando soluciones para resolver los diferentes problemas que han ido apareciendo en el transcurso de la misma. Por tanto, se pone de manifiesto que el doctorando ha adquirido las competencias necesarias para el desarrollo de una prometedora carrera científica.

Por todo lo reflejado en este informe, los directores consideramos que esta Tesis Doctoral posee un gran calidad y prueba de ello es que los resultados derivados de ella han sido publicados en una revista de segundo cuartil y otra de cuarto cuartil (JCR):

-Ibarra-Delgado S., Sandoval-Arechiga R.,Gómez-Rodríguez J.R., Ortíz-López M. and Brox M. A Bandwidth Control Arbitration for SoC Interconnections Performing Applications with Task Dpendencies. Micromachines 2020, 11(12), 1063.

-Ibarra-Delgado S., Sandoval-Arechiga R., Brox M. and Ortíz-López M. Throughput Unfairness in Fair Arbitration Interconnection-Buses for Aerospace Embedded Systems. IEEE Latin America Transactions 2020, 18(9), 1606.

Por todo ello, se autoriza la presentación de la Tesis Doctoral.

Córdoba, España, 22 de abril de 2021

Firma de los directores

BROX JIMENEZ MARIA - 30964938S

Firmado digitalmente por BROX JIMENEZ MARIA - 30964938S
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-30964938S, givenName=MARIA, sn=BROX JIMENEZ, cn=BROX JIMENEZ MARIA - 30964938S
Fecha: 2021.04.25 08:08:58 +02'00'

Fdo.: Dra. María Brox Jiménez

Firmado por ORTIZ LOPEZ MANUEL AGUSTIN - 24199639M el día 26/04/2021 con un certificado emitido por AC

Fdo.: Dr. Manuel Agustín Ortiz López

Fdo.: Dr. Remberto Sandoval Aréchiga

Abstract

The set of applications that today run in the processing elements inside a System-on-Chip (SoC) require that the SoC interconnection system allows them to meet the Quality of Service (QoS) requirements that were established. Current interconnection systems must be flexible, reconfigurable, scalable, reusable and easy to manage. This document presents a solution for a bus type interconnection system, based on the Software Defined Network (SDN) paradigm. The work shows the general architecture of the interconnection system and demonstrates that this architecture meets the characteristics mentioned above. The work puts special emphasis on the infrastructure layer of the system — hardware —. However, it also establishes the elements to be included in the network operating system layer and its interrelation with the adjacent layers. This work also shows a new arbitration policy based on budgets that allows the differentiated use of the bus.The policy presents the best behavior when the system works in scenarios with applications executing dependent tasks, which are very common nowadays. The system was modeled in SystemC with clock cycle accuracy. The contributions made in this work can be extrapolated to other SoC interconnection systems because the challenges they share are similar.

Resumen

El conjunto de aplicaciones que hoy en día se ejecutan en los elementos de procesamiento en el interior de un Sistema en Chip (System-on-Chip (SoC), en inglés) requieren que el sistema de interconexión que los conecta, les permita cumplir con los requerimientos de Calidad en el Servicio (Quality of Service (QoS), en inglés) que les fueron establecidos. Los actuales sistemas de interconexión deben ser flexibles, reconfigurables, escalables, reutilizables y fáciles de administrar. En este documento se presenta una solución para un sistema de interconexión tipo *bus* basado en el paradigma de Redes Definidas por Software (Software Defined Network (SDN), en inglés). El trabajo muestra la arquitectura general del sistema de interconexión y en él se demuestra que la arquitectura cumple con las características anteriormente mencionadas. El trabajo pone especial énfasis en la capa de infraestructura del sistema – hardware–. Además, se incluyen los elementos de la capa de sistema operativo de red y se establecen sus interrelaciones con las capas adyacentes. En este trabajo también se muestra una nueva política de arbitraje basada en presupuestos la cual permite el uso diferenciado del bus. La política muestra un mejor comportamiento cuando el sistema funciona en escenarios ejecutando aplicaciones con tareas dependientes, los cuales son muy comunes en la actualidad. El sistema fue modelado en SystemC con precisión de ciclo de reloj. Las contribuciones realizadas en este trabajo pueden ser extrapoladas a otros sistemas de interconexión en SoC debido a que los retos que comparten son similares.

Índice general

Índice de figuras

Índice de tablas

Siglas

AA Administrador de Aplicaciones. 81

AAPRR Asynchronous Adaptive Priority Round-Robin. 42

ABA Age-Based Arbitration. 42

ABL Age-Based Lottery. 41

AES Advanced Encryption Standard. 71

AG Administración General. 83

AP Arbitraje Probabilístico. 42

API Application Programming Interface. 4, 45, 48, 50, 52

AR Administrador de Red. 61, 62, 64

ASIC Application Specific Integrated Circuits. 14

AT Administración de Tráfico. 85, 88

AWRR Adaptively Weighted Round-Robin. 42

BDAM Base de Datos de Aplicaciones Mapeadas. 87

C-SDN Controlador-SDN. 45, 47, 48, 64, 73, 82, 83, 85, 148

CB Controlador del Bus. 72, 75, 83, 98, 111–113, 148

CBA Credit-Based Arbitration. 43

CCSPA Credit-Controller Static-Priority Arbitration. 44

CDB Controlador de Datos del Bus. 21

CI Circuito Integrado. 56

CInf Capa de Infraestructura. 60, 91

CR Configuración de Red. 85, 88

CRR Classified Round-Robin. 36

DesR Descubrimiento de Red. 85

DR Dispositivos de Reenvío. 47, 97

dTDMA dynamic-TDMA. 40

EP Elementos de Procesamiento. 17, 19–22, 24–28, 33, 40, 43, 55, 57, 60, 62, 63, 67, 68, 86, 87

ES Elastic Scheduling. 40

FP Función de Programación. 85

FPGA Field Programmable Gate Array. 21

FR Funciones de Red. 59, 82, 85

GALS Globally Asynchronuos, Locally Synchronous. 16

GCT Grafo de Comunicación de Tareas. 18

HDL Hardware Description Language. 16

IN Interfaz Norte. 49, 52, 60, 82, 83

IPCore Intellectual Property Core. 1–4, 6–8, 14–16, 18, 19, 21, 25, 27, 60, 64, 68, 71, 72, 75, 78, 85, 87, 92, 96–98, 102, 106–110, 120, 121, 130, 139, 141–144, 147, 151–153

IR Interfaz de Red. 21, 24, 41, 60–62, 72, 75, 76, 78, 82, 85, 90, 93–99, 101–105, 108–112, 116, 118–120, 144

IRE Interfaz de Red Esclava. 98, 106–108, 110, 123

IRM Interfaz de Red Maestra. 98, 99, 103, 106, 107, 122

IS Interfaz Sur. 48, 49, 52, 59, 60, 90, 93

LTY Lottery. 34, 37, 38, 41, 43

MEF Máquina de Estados Finitos. 93, 95, 98, 99, 101, 105, 107, 113, 155–160

MOB Modo de Operación del Bus. 83

NDPE Número de Datos Por Enviar. 100

NoC Network-on-Chip. 5, 17, 18, 22–25, 27, 41, 42, 54–56, 62, 63, 67, 86, 87

NofC Network-of-Chips. 56

PA Plano de Aplicación. 57, 81, 82

PA Plano de Administración. 49

PAR Plano de Administración de la Red. 59, 82, 83, 85, 87

PC Plano de Control. 45, 47, 48, 50, 59, 72, 82, 93

PCI Placa de Circuito Impreso. 56

PD Priority Division. 40

PD Plano de Datos. 45, 48, 50

PF Prioridad Fija. 38, 39, 41, 43

PPD Plano de Procesamiento de Datos. 60, 72, 75, 76, 91, 93

PRD Plano de Reenvío de Datos. 60, 72, 75, 93, 98

PV Prioridad Variable. 39

QoS Quality of Service. 2, 3, 6, 17, 28, 36, 39, 54, 57, 59, 64, 81, 124, 125, 131, 133, 149, 151, 153

RB_Lottery Real-Time Bandwidth Lottery. 38

RCT Registros de Control de Tráfico. 100

RDPE Registro de Datos Por Enviar. 100, 110

RDPO Registro de Datos Por Operar. 109, 110

RE Recolector de Estadística. 101, 103

REPR Registro de Espera Por Reconocimiento. 107, 110

RGC Registros Generales de Control. 83

RR Receptor de Red. 99, 103

RR Round-Robin. 30, 31, 36, 38, 39, 41–44

RRPE Registro de Reconocimiento Por Enviar. 110

RT_lottery Real-Time lottery. 37, 38

RTL Register Transfer Level. 14

RTRE Registro de Tiempo de Recolección de Estadística. 83, 88

SDBoC Software Defined Bus-on-Chip. 72, 73, 79–83, 85, 88, 91, 92, 96, 97, 99, 101–103, 105, 106, 109, 111, 118–120, 124, 125, 139, 142–144, 146–149, 152, 153

SDI Software Defined Interconnection. 57

SDN Software Defined Network. 4–6, 8–10, 12, 13, 44, 45, 47, 49, 52–57, 59, 60, 62–65, 67, 72, 77, 79–81, 85, 88, 91, 118, 120, 124, 125, 139, 146, 151–154

SDNoC Software Defined Network-on-Chip. 55, 61, 62, 153

SeCMN Sistema-en-Chip con Múltiples Núcleos. 56

SM Self-Motivated. 39

SO Sistema Operativo. 80, 81, 96

SO Sistema Operativo. 57

SoC System-on-Chip. 1–10, 12–23, 27, 28, 41, 44, 53, 55–59, 64, 66, 72, 77–79, 81, 86, 88, 91, 98, 99, 124–127, 131–133, 138, 149, 151, 153, 154

SOR Sistema Operativo de Red. 45, 47, 49, 52, 57–60, 72, 82, 83, 124

SR Servicios de Red. 82, 83, 97

SuDO Supervised Debt and Opportunistic. 114, 115

TDMA Time Division Multiple Access. 33, 36, 37, 40

TF Tablas de Flujo. 45

TIF Tabla de Información de Flujos. 85

TR Transmisor de Red. 99–101, 104

TS Tabla de Scripts. 82, 83, 87, 88, 93

UC Unidad de Control. 72

USF Unidad Selectora de Flujo. 100, 101, 103

VCP Verificador de Cumplimiento de Plazos. 37

VR Visor de Red. 83

WRR Weighted Round-Robin. 35, 42, 101

Glosario

Igualdad de Servicio se refiere a que todos los solicitantes de un servicio que otorga un nodo de la red obtengan la misma proporción de uso del mismo, no importando que tan lejos (lógicamente) se encuentren de él. 42, 57

Interfaz de Programación de Aplicaciones es un conjunto de definiciones y protocolos que son utilizados para desarrollar e integrar el software de las aplicaciones. 4

Lenguaje de Transferencia de Registros es un lenguaje de bajo nivel cuya operación se basa en transferir información de un lugar a otro a nivel de registros de hardware. 14

Lenguajes de Descripción de Hardware es un lenguaje especializado que se utiliza para definir la estructura, diseño y operación de circuitos electrónicos. 16

Machine Learning es una disciplina científica del ámbito de la Inteligencia Artificial que crea sistemas que aprenden automáticamente. 56

Mejor Esfuerzo tipo de transmisión donde los paquetes son conmutados sin ninguna prioridad, no es necesario hardware o mecanismos de control para su traslado. 36, 39

Núcleo de Propiedad Intelectual se refiere a la descripción lógica de un circuito electrónico, comúnmente distribuido como un bloque sintetizable en un lenguaje de descripción de hardware como VHDL o Verilog. 1

Procesadores-Software es un procesador que es totalmente implementado con la lógica programable de un dispositivo lógico programable. 56

Servicio Garantizado tipo de transmisión donde los paquetes son categorizados basados en tipos de flujo los cuales tienen asignados prioridades. Los flujos con mayor prioridad son utilizados comúnmente para aplicaciones de tiempo real. 36, 39

broadcast es una forma de transmisión de información donde un solo emisor envía información a todos los nodos receptores en su universo. 25, 85, 99

chiplet componente de hardware, completamente funcional que puede ser agrupado con otros componentes para construir un chip más grande. 56

end-to-end son servicios que proveen soluciones completas desde su solicitud hasta su entrega, haciendo transparente las capas o procesos por las cuales la solicitud y la respuesta tienen que transitar. 23

esclavo elemento de procesamiento que responde a las transacciones iniciadas por un maestro en el modelo maestro-esclavo. 21

flip-flop es un dispositivo electrónico capaz de permanecer en uno de dos estados posibles por un tiempo indefinido, utilizado en sistemas digitales para almacenar información. 31

flit es la unidad de transmisión lógica en un sistema de interconexión. 39, 62

four-phase dual-rail Es un protocolo *handshake* de cuatro fases con codificación de doble carril. 42

hot-spot Un nodo en la red que está siendo intensamente utilizado por los demás nodos de la red en un momento determinado. 42, 64

maestro elemento de procesamiento iniciador de las transacciones en un modelo maestro - esclavo. 21

mapeo de memoria se dice que un dispositivo trabaja bajo este modelo cuando los elementos que lo usan acceden a todas sus funcionalidades como si fueran direcciones de memoria. 7, 8, 93

multicast en este tipo de transmisión los destinatarios de la información es un grupo específico. 99, 103, 106

phit es la unidad de transmisión física del canal de comunicación por donde viajan los paquetes en un sistema de interconexión. 61, 62, 85, 126

pipeline es una técnica que intenta mantener ocupada cada parte de un procesador, dividiendo las instrucciones en un conjunto de pasos secuenciales ejecutadas por diferentes unidades del procesador. 26

ruteador también conocido como enrutador o direccionador, proviene del término inglés router. Es un elemento de hardware encargado de interconectar elementos de cómputo que operan en el marco de una red. 42

scripts Código de programación que contiene comandos que serán ejecutados secuencialmente, utilizados comúnmente para controlar un elemento específico. 15, 58

streaming Se dice que un dispositivo trabaja bajo este modelo cuando por un lado recibe a modo de ráfagas los datos con que va a operar y por otro lado genera una ráfagas con los resultados. 7

testbench Conjunto de elementos utilizados para probar un diseño, generalmente desarrollado en lenguajes especializados para validación y verificación. 16

tiempo de ejecución proviene del término en inglés runtime. Indica que un sistema se encuentra en operación. 6, 10, 39–41, 54, 55, 64, 65, 67, 73, 85, 124, 125, 148, 151–153

unicast en este tipo de transmisión los destinatarios de la información enviada por la red se identifican individualmente. 99

Capítulo 1

Introducción

1.1. Antecedentes

En la actualidad, la posibilidad de poder integrar millones de transistores en una oblea de silicio, ha propiciado el surgimiento de arquitecturas que permiten incorporar miles de elementos de procesamiento en un solo circuito. La mayoría de estos elementos son integrados utilizando el modelo de SoC como se menciona en Chen et al. (2017). Basta realizar una simple búsqueda de este tópico en las principales bases de datos científicas para percibir el interés que la comunidad científica y tecnológica ha tenido al respecto. La base de datos IEEEXplore (2020) muestra que en las diferentes revistas y congresos que forman parte de esta asociación, se han publicado cerca de dos mil artículos al respecto en los últimos cinco años. En promedio, un artículo por día. La revista *Microprocessors and Microsystems* de Elsevier (2020) muestra cerca de quinientos resultados en el mismo periodo, casi cien artículos por año. Este interés es propiciado en parte por una pujante industria electrónica que en el año 2017 tuvo un valor en el mercado de $444.70 billones de dólares como se indica en el estudio publicado por Foster (2018).

Son varias las ventajas que presentan los SoC por las cuales han tenido una gran aceptación por parte de la comunidad que integra la industria del diseño electrónico. Por un lado, permiten el uso de bloques funcionales de hardware previamente diseñados, probados y verificados denominados *Núcleo de Propiedad Intelectual* (Intellectual Property Core (IPCore), en inglés). Estos núcleos son capaces de realizar tareas específicas como la compresión de imágenes o cifrado de información. Por otro lado, posibilitan un desarrollo acelerado de los productos de tal modo que pueden tener una presencia más pronta en el mercado. Además, la capacidad de integración que tienen los SoC permite desarrollar sistemas de menor tamaño, menor costo y menor consumo de energía, lo cual los hace muy atractivos para un gran número de aplicaciones que se consumen actualmente.

Sin embargo, la coexistencia de una gran cantidad de elementos de procesamiento al interior de un SoC, genera un nuevo problema: su rendimiento. Actualmente, el rendimiento de un SoC está principalmente determinado por el sistema de interconexión que comunica a los elementos de procesamiento, más que en la capacidad de procesamiento de los mismos. De tal modo que el desarrollo de modelos, arquitecturas e implementaciones de sistemas de interconexión, se ha convertido en una área principal de investigación en los SoC como es señalado por Karkar et al. (2016). Hoy en día, los principales retos que enfrentan los desarrolladores de SoC tienen que ver con tener sistemas de interconexión que: puedan escalar fácilmente; puedan tener la capacidad de controlar su consumo de energía; puedan regular el tráfico que existe en el sistema de interconexión; posibiliten la operación de múltiples aplicaciones en tiempo real; posibiliten que las aplicaciones puedan cumplir con los requerimientos de QoS que se les establecieron –latencia, rendimiento, consumo de energía, etc–. La solución a la mayoría de estos retos se relaciona directamente con los sistemas de interconexión y su administración. De tal modo que la industria, presiona constantemente a la comunidad científica para que se propongan nuevos y mejores modelos de sistemas de interconexión que les permitan mejorar sus condiciones de competencia en el mercado.

1.2. Justificación

Hoy en día, el desarrollo de un proyecto de diseño electrónico se encuentra presionado desde el mismo momento de su concepción. La competencia por presentar en el mercado nuevas soluciones electrónicas crece en intensidad cada día. El ganador es aquel que presente una solución en el menor tiempo y con las mayores prestaciones posibles. Las nuevas metodologías de diseño teniendo esto en mente, promueven prácticas que permiten a los ingenieros responder a estos retos con mayor eficiencia. Una de ellas es el reuso de IPCores. Con esto se evita iniciar un proyecto desde cero y los tiempos de desarrollo se reducen. Para hacer más sencilla la integración de los IPCores es común que éstos se comuniquen por medio de una interfaz estándar que les permite su fácil intercomunicación. Además, hoy en día el proceso de diseño se enfoca desde su primera etapa en cumplir con los requerimientos de operación que le fueron establecidos al producto. Cuando se realiza la implementación *hardware* de un algoritmo ésta se desarrolla teniendo en mente satisfacer los requerimientos que tienen que ver principalmente con: rendimiento, la implementación *hardware* debe tener la capacidad de procesar una cierta cantidad de datos por unidad de tiempo; consumo de energía, el sistema no debe de sobrepasar una cantidad de consumo especificado; tamaño, el circuito no debe exceder una cierta cantidad de área especificada; latencia, el tiempo para que un dato sea procesado no debe exceder un límite establecido. Este último requerimiento es especialmente crítico para sistemas

que operan en tiempo real. Lo anterior sitúa a los equipos de diseño en una aparente contradicción, ya que para cumplir con los requerimientos de diseño – generalmente ambiciosos– es necesario realizar implementaciones *ad hoc* lo que impide en principio el reuso de IPCores implicando tiempos de desarrollo mayores.

Con la finalidad de poder lograr tener una buena relación entre tiempo de desarrollo y cumplir con los requerimientos de operación, se han desarrollado al interior de los SoC, sistemas de interconexión de los IPCores que al mismo tiempo que permiten su fácil integración, posibilitan hasta cierto punto cumplir con los requerimientos de QoS que se le imponen al producto. Sin embargo, aunque los sistemas de interconexión se han convertido *de facto* en la tecnología de interconexión de elementos de procesamiento al interior de los SoC, diferentes autores apuntan que hoy en día, el principal cuello de botella que limita el rendimiento de un SoC, se encuentra principalmente en el sistema de interconexión más que en el rendimiento particular de cada IPCore Poletti et al. (2003); Ahmed et al. (2017); Ben Slimane et al. (2017); Dally and Towles (2003); Benini and Bertozzi (2005). Ante esto la comunidad científica que estudia los problemas relacionados con esta área, ha dirigido sus esfuerzos a estudiar y proveer sistemas de interconexión que coadyuven a incrementar el rendimiento de los SoC. Con esto, intentan mejorar las prestaciones de las aplicaciones para que puedan cumplir con los requerimientos de diseño establecidos.

Al convertirse los sistemas de interconexión en un elemento clave para que un SoC mejore sus prestaciones, existen varias características que son deseables que estos tengan:

- **Flexible:** Aunque la infraestructura de un sistema de interconexión se construya sobre una topología específica, el sistema de interconexión debe de proveer la capacidad de poder operar lógicamente bajo diferentes topologías.
- **Reconfigurable:** El sistema de interconexión debe de tener la capacidad de modificar sus parámetros de operación en cualquier momento. Debe de ser capaz de modificar su topología; establecer o eliminar flujos; cambiar políticas de arbitraje; debe de tener la capacidad de agregar o eliminar elementos de procesamiento del sistema de interconexión; etc.
- **Escalable:** Debe permitir modificar las dimensiones del sistema de interconexión sin impactar el rendimiento de las aplicaciones.
- **Reutilizable:** El sistema de interconexión debe de poder integrarse fácilmente a diferentes entornos de SoC.
- **Fácil Administración:** La lógica que permita realizar las anteriores características no debe de ser mayor que la misma infraestructura del sistema de interconexión.

Una alternativa de solución para la implementación y gestión de los sistemas de interconexión, que ha surgido en los últimos años, tiene su origen en el modelo SDN que es utilizado para la administración de redes de comunicación de datos. Este paradigma le ha provisto a la industria de las redes de comunicación de una herramienta que le posibilita un mejor control de sus sistemas. Un sistema que tuvo en sus inicios un fin académico McKeown et al. (2008), hoy es utilizado por algunos de los grandes competidores en el área de redes de comunicaciones como son Cisco (2020) y ONF (2014), para proveer herramientas que les permitan a sus usuarios: simplificar operaciones; desarrollar redes programables; centralizar y automatizar las configuraciones; implementar más rápido nuevas aplicaciones aprovechando el uso de una *Interfaz de Programación de Aplicaciones* (Application Programming Interface (API), en inglés) abierta.

Algunos autores han tomado los conceptos principales del paradigma SDN para aplicarlos en los sistemas de interconexión de los SoC. La separación del plano de retransmisión de datos, del plano de control que promueve SDN, encaja muy bien en los sistemas de interconexión tipo red, que son una tendencia generalizada en la comunidad desarrolladora de SoC. Este paradigma, posibilita crear estructuras de administración y control definidas por *software*, sin necesidad de hacer cambios en el *hardware* del sistema de interconexión. Además, por medio de la construcción de interfaces de comunicación estandarizadas se posibilita el reuso de los IPCores con lo que el tiempo de arribo al mercado de nuevas aplicaciones es más corto. Los problemas referentes a mapeo de aplicaciones, ruteo de paquetes, administración de la energía, programación, son ahora resueltos desde capas superiores de la red vía *software*. La aplicación de los conceptos SDN en los SoC resulta muy prometedora, sus principales objetivos: escalabilidad, control de tráfico, gestión dinámica del ancho de banda, gestión centralizada de los recursos, supervisión de recursos y usuarios, generación de estadística para la toma de decisiones, seguridad en las comunicaciones, son también objetivos que se desean obtener en el entorno de un sistema de interconexión para un SoC. Sin embargo, la tarea de desarrollar sistemas de interconexión basados en SDN apenas comienza, hay una gran cantidad de trabajo que realizar. Es necesario proponer modelos, arquitecturas, protocolos, interfaces, infraestructura que permitan alcanzar estos objetivos.

Los sistemas de interconexión basados en *bus* han estado vigentes por mucho años. Es la base de algunos sistemas de interconexión utilizados por varios de los principales competidores en la industria ARM (2010), Intel (2020a). Sin embargo, a pesar de que han sido ampliamente utilizados, existen elementos en su comportamiento que no han sido totalmente estudiados. Por ejemplo, las políticas de arbitraje que actualmente se utilizan en la mayoría de las implementaciones de este tipo de sistemas de interconexión, se ocupan más en asegurar un acceso justo al sistema, que en regular el uso del *bus* de acuerdo a las necesidades particulares de un ele-

mento de procesamiento. Teniendo en cuenta que esto es una característica deseable de un sistema de interconexión considero que es importante estudiar nuevas políticas de arbitraje que permitan hacer un uso diferenciado de un recurso compartido. También es deseable presentar mayor evidencia del comportamiento de las políticas de arbitraje en escenarios que son cada día mas comunes y que han sido poco estudiados como aquellos escenarios donde se ejecutan simultáneamente aplicaciones heterogéneas con dependencia en la ejecución de sus tareas.

Por lo anteriormente planteado, se decidió realizar un estudio sobre la aplicación de los conceptos del modelo SDN en un sistema de interconexión para un SoC. Poniendo atención principalmente en las características que deben de tener los elementos que forman parte de la capa de infraestructura —*hardware*— del sistema de interconexión tipo *bus*. Además, en el estudio se deberán mostrar los principales mecanismos de comunicación con las capas superiores del modelo seleccionado. Desde mi punto de vista, es útil realizar un estudio que proporcione evidencia que permita identificar las ventajas del uso del modelo SDN en un SoC en general y, en un sistema de interconexión tipo *bus* en lo particular.

1.3. Alcances

El estudio aquí realizado toma como marco de referencia la arquitectura para sistemas de interconexión bajo el enfoque SDN propuesta por Sandoval-Arechiga et al. (2015). El estudio se centra principalmente en el diseño de la infraestructura – *hardware* – necesaria para implementar un sistema de interconexión definido por software. En este trabajo se hace especial énfasis en las características de los elementos encargados de la transmisión de los datos. La forma en que opera la infraestructura es regida por elementos en las capas superiores de la arquitectura – *software* –. El estudio no implementa en su totalidad estas capas, pero sí proporciona las principales rutinas de servicio que se deben de ofrecer para que la infraestructura pueda operar en el modo SDN. Los algoritmos de optimización que permiten mejorar las prestaciones de uno o varios elementos del sistema se encuentran fuera del alcance de este estudio.

Tradicionalmente los sistemas de interconexión en un SoC se han dividido en tres categorías: sistemas de interconexión basados en *bus*, sistemas de interconexión tipo *crossbar* y las denominadas *Redes en Chip* (Network-on-Chip (NoC), en inglés). Varios autores han apuntado que estos sistemas de interconexión más que competir entre sí, se complementan Ben Achballah et al. (2017); Bjerregaard and Mahadevan (2006). En este estudio se ha decidido centrar la investigación en los sistemas de interconexión basados en *bus* por las siguientes razones:

- En la actualidad existen muchas aplicaciones que tienen como base el sistema

de interconexión basado en *bus*. Estas aplicaciones pueden verse beneficiadas con las posibilidades que ofrece una gestión basada en SDN como: la posibilidad de operar por medio de flujos; la posibilidad de asignar un ancho de banda específico a una aplicación en el sistema; control energético del sistema; reconfiguración de flujos, aplicaciones, y de los parámetros de operacion de los IPCores en *tiempo de ejecución*; conocer la estadística del sistema de interconexión para la toma de decisiones, entre otros.

- Algunos elementos del sistema de interconexión basado en *bus* son comunes a los otros sistemas de interconexión, de tal modo que los resultados aquí obtenidos pueden extrapolarse a los otros sistemas de interconexión.

- Los principios básicos de funcionamiento del sistema de interconexión basado en *bus* permiten aislar puntualmente problemas que se presentan en los sistemas de interconexión. Uno de estos problemas, que causa el principal cuello de botella en un SoC, es el uso de recursos compartidos. Hoy en día, no es suficiente tener un uso justo del *bus*, sino que hay que racionalizarlo de acuerdo a las necesidades de los usuarios – aplicaciones –. La evaluación de árbitros y sistemas de apoyo que permitan el uso racional de estos recursos es parte fundamental de este estudio y el sistema de interconexión tipo *bus* es el medio ideal para su estudio.

En este trabajo se presentarán las características de los elementos de la infraestructura de un sistema de interconexión basado en *bus* bajo el paradigma SDN. Se propondrá una arquitectura que permita el transporte de los datos que se encuentre aislada de los elementos que procesan los datos. Se presentará una estrategia para la integración de los elementos de procesamiento por medio de una interfaz bien conocida procurando el reuso de los IPCores. Se mostrará un estudio de los principales sistemas de arbitraje utilizados en SoC y se hará una propuesta de una política que permite el uso proporcional de recursos compartidos. Además, se definirán las principales rutinas de servicios que deben estar integradas en los planos superiores de la arquitectura de interconexión definida por *software*.

1.4. Planteamiento del problema

Una de las principales medidas de QoS que se le impone a una aplicación es el rendimiento. Es decir, una aplicación debe de cumplir con una cierta cuota de procesamiento en una unidad de tiempo. Esta medida puede resultar difícil de obtener cuando en un SoC se ejecutan simultáneamente diferentes aplicaciones que compiten por el uso de un recurso – sistema de interconexión, controlador de memoria, etc.–. Cuando esto sucede es común que una aplicación, la que genera las transacciones

más grandes, acapare el uso del recurso compartido. Lo anterior es debido principalmente a que, los árbitros que tradicionalmente se han encargado de permitir el acceso al recurso, se rigen bajo una política que se enfoca más, en permitir un acceso justo al recurso, que en controlar el tiempo de uso del mismo. En los actuales SoC, donde se ejecutan aplicaciones muy diversas, es deseable que durante ciertos periodos una de las aplicaciones pueda acceder al recurso de manera preferencial. Pero en otro periodo es necesario que la preferencia la tenga otra aplicación. Lo anterior plantea la necesidad de contar con un sistema de arbitraje, que no solo controle el acceso a un recurso compartido, sino que adicionalmente regule el tiempo que este puede ser usado. Además, la regulación debe de poder llevarse a cabo sin sacrificar el rendimiento global del sistema.

Por otro lado, una de las principales necesidades de la industria electrónica, es poder llevar productos al mercado en el menor tiempo posible. Por esto, durante mucho tiempo se ha insistido en la posibilidad del reuso de los IPCores, ya sean propios o de terceros. En este sentido se han tenido buenos resultados al respecto debido a que ya existen interfaces de comunicación bien definidas en el mercado. Sin embargo, las interfaces trabajan principalmente bajo el modelo de *mapeo de memoria* o de *streaming*. Estas interfaces no son lo suficientemente flexibles y escalables como lo requieren los SoC en la actualidad. En este sentido con la finalidad por un lado de poder aprovechar la gran cantidad de IPCores que ya existen en el mercado; y por otro lado integrarlos a un sistema de interconexión más flexible, es necesario el desarrollo de circuitos convertidores de protocolo que permitan a los IPCores existentes, integrarse fácil y rápidamente en un sistema de interconexión de conmutación de paquetes. En este caso, basado en una topología *bus*.

Existen muchas aplicaciones cuyo resultado se deriva de efectuar una serie de transformaciones en forma secuencial a un conjunto de datos. Es decir, los datos que son generados por una fuente "fluyen río abajo" a un primer IPCore que les transforma. Los datos, resultado de esta transformación continúan su flujo "río abajo" a un segundo IPCore para continuar su procesamiento, y así sucesivamente hasta que los datos terminan su transformación. Cuando este tipo de aplicaciones son implementadas en un SoC con un sistema de interconexión tipo *bus* tradicional, existe un *overhead* en la transmisión de los datos. Esto es debido a que el maestro encargado del proceso envía los datos al primer IPCore en el flujo y espera por el resultado de esta transformación. Posteriormente envía los datos que recibió transformados al segundo IPCore para continuar su transformación y vuelve a esperar por el resultado antes de transmitirlo nuevamente. Este comportamiento implica el transporte de los mismos datos en un par de ocasiones por el *bus*. Lo que además de retardar la operación de los datos, genera un mayor uso del *bus*. Este comportamiento no solo impacta en el rendimiento de la aplicación, ya que está generando tráfico redundante, sino que en general afecta el tráfico de las demás aplicaciones, perjudicando el

rendimiento global del sistema. Por lo anteriormente planteado es deseable el desarrollo de una nueva infraestructura para un sistema de interconexión tipo *bus* que permita mayor flexibilidad en el transporte de los datos.

La seguridad es un elemento que hoy en día es parte fundamental de cualquier sistema de comunicación. Un sistema de interconexión en un SoC no está exento de problemas de intrusión, por lo que se deben proveer mecanismos de seguridad que controlen el acceso al mismo. Es decir, debe de ser capaz de permitir que en el sistema solo viajen paquetes pertenecientes a nodos y/o flujos autorizados. Además, debe de ser capaz alertar a los elementos de gestión del intento de intrusión.

Para dar solución a los problemas anteriormente planteados se propone el diseño e implementación de una arquitectura de un sistema de interconexión basada en los conceptos de SDN que sea: flexible, reconfigurable, reusable, segura y de fácil administración, características que se esperan de los actuales sistemas de interconexión.

1.5. Objetivos

El objetivo principal de esta investigación es: el desarrollo de la infraestructura para un sistema de interconexión tipo *bus* que opere bajo el paradigma SDN en un SoC, de tal modo que se puedan incrementar las prestaciones otorgadas por los modelos tradicionales de interconexión tipo *bus*.

Los objetivos específicos del estudio son:

- Diseñar una política de arbitraje que permita el uso diferenciado del *bus* poniendo especial atención en escenarios donde se ejecuten simultáneamente aplicaciones con dependencia de tareas.
- Diseñar un sistema que permita integrar los IPCores con interfaces con *mapeo de memoria*, a un sistema de interconexión basado en el transporte de paquetes.
- Diseñar la infraestructura necesaria para que un sistema de interconexión tipo *bus* pueda trabajar bajo el modelo SDN.
- Identificar y detallar las capacidades que deben de tener los controladores de *software* que deben de existir en los planos superiores de la arquitectura SDN para que exista una gestión adecuada de los recursos de *hardware* del sistema.

1.6. Metodología

Para el desarrollo de este estudio se siguió la siguiente metodología:

1. Se realizó un estudio detallado del estado del arte respecto al tópico de sistemas de interconexión en SoC. Se detectaron qué ventajas y/o desventajas se presentan en las propuestas encontradas.

2. Se detectaron los principales problemas que presentan los sistemas de interconexión utilizados en un entorno SoC.

3. Con el objetivo de mejorar las prestaciones de los sistemas de interconexión en un SoC, se estudiaron nuevos modelos de gestión y se identificó cómo estos pueden impactar en la mejora de dichas prestaciones. Se identificó una arquitectura basada en SDN para sistemas de interconexión, la cual fue tomada como ancla para el desarrollo de los elementos de la capa de infraestructura de este trabajo. Se identificó qué elementos de la arquitectura pueden ser aplicados al sistema de interconexión tipo *bus* con la finalidad de mejorar el rendimiento general del sistema y aumentar sus prestaciones, con respecto a un sistema de interconexión tipo *bus* tradicional.

4. Se identificaron un conjunto de escenarios de uso común en SoC sobre los cuales los modelos de sistemas de interconexión tipo *bus* tradicionales limitan su rendimiento. Además se identificó qué elementos de la infraestructura son los que limitan el rendimiento.

5. Se diseñó e implementó una arquitectura hardware de un sistema de interconexión basado en *bus* tradicional. Esta arquitectura incluye: controlador del bus, árbitro, interfaces de red, maestros, esclavos y acopladores.

6. Uno de los principales problemas que enfrentan los diferentes tipos de sistemas de interconexión, es el poder permitir el uso diferenciado de un recurso compartido a diferentes elementos de procesamiento que lo requieran. Para poder conocer en qué grado diferentes políticas de arbitraje pueden resolver este problema, se desarrollaron e implementaron estas políticas, y se integraron al sistema de interconexión tipo *bus* construido para su evaluación. Del conjunto de simulaciones realizadas se efectuó un estudio comparativo de su comportamiento.

7. Se desarrolló y probó una nueva política de arbitraje que permite el uso diferenciado del *bus* y que presenta el mejor comportamiento cuando el sistema de interconexión se encuentra ejecutando múltiples aplicaciones con tareas dependientes.

8. En base a las características detectadas en el modelo de referencia, las cuales pueden contribuir a proveer un mejor comportamiento del sistema de interconexión tipo *bus*, se procedió a modificar la arquitectura *hardware* del sistema

de interconexión. Esta infraestructura involucra los elementos necesarios para proveer un control del *bus* bajo una filosofía SDN. Además se detectaron y desarrollaron los controladores de *software* que habilitan el uso de esta infraestructura.

9. Se procedió a evaluar el comportamiento de la arquitectura desarrollada y se contrastó con la arquitectura tradicional. Se evaluó que la nueva arquitectura tenga las posibilidades de trabajar bajo el modo de flujos de datos, que tenga la capacidad de reconfiguración en *tiempo de ejecución*, que permita diferentes tipos de operación, que permita conocer el estado de cada uno de los elementos en cualquier momento, y que genere estadística que pueda ser consultada en cualquier momento por el controlador del sistema, para que en caso de ser necesario, se pueda modificar el comportamiento del sistema de interconexión.

1.7. Contribuciones

Las principales contribuciones de este trabajo son:

- La realización de un estudio, con precisión de ciclo de reloj, del comportamiento de las principales políticas de arbitraje utilizadas en los sistemas de interconexión para SoC.
- La propuesta de una nueva política de arbitraje basada en presupuesto que permite el uso diferenciado de un recurso compartido.
- El desarrollo de una infraestructura de hardware para que un sistema de interconexión tipo *bus* pueda ser gestionado bajo el modelo SDN.
- Se presenta evidencia que muestra que las prestaciones de un sistema de interconexión tipo *bus* pueden ser incrementadas cuando éste trabaja bajo el paradigma SDN.
- Se presenta evidencia que muestra que los sistemas de interconexión para SoC basados en SDN permiten cumplir con las características deseables de los sistemas de interconexión actuales.

1.8. Publicaciones Científicas Relacionadas

Parte de los resultados que se han derivado del proceso de investigación y que se exponen en esta Tesis Doctoral han aparecido previamente en algunas publicaciones:

- Gómez-Rodríguez J.R., Sandoval-Arechiga R., Ibarra-Delgado S., Rodriguez-Abdala V.I., Vazquez-Avila J.L. and Parra-Michel R. A Survey of Software-Defined Networks-on-Chip: Motivations, Challenges and Opportunities. *Micromachines* **2021**, 12(2), 183.
- Ibarra-Delgado S., Sandoval-Arechiga R.,Gómez-Rodríguez J.R., Ortíz-López M. and Brox M. A Bandwidth Control Arbitration for SoC Interconnections Performing Applications with Task Dependencies. *Micromachines* **2020**, 11(12), 1063.
- Ibarra-Delgado S., Sandoval-Arechiga R., Brox M. and Ortíz-López M. Throughput Unfairness in Fair Arbitration Interconnection-Buses for Aerospace Embedded Systems. *IEEE Latin America Transactions* **2020**, 18(9), 1606
- Ibarra-Delgado S., Sandoval-Arechiga R., Brox M., Ortíz-López M. Arquitecturas Definidas por Software, una alternativa flexible para la interconexión de Sistemas en Chip. *Libro de Actas XXVI Seminario Anual de Automática, Electrónica Industrial e Instrumentación*, Córdoba, España, 2019, pp. 108-113.
- Ibarra-Delgado S., Sandoval-Arechiga R., Brox M., Ortíz-López M. Software defined network controller: A neat solution administration for reconfigurable multi-core NoC. *International Conference on ReConfigurable Computing and FPGAs*, Cancun, México, 2017, pp. 1-4.
- Sandoval-Arechiga R., Ibarra-Delgado S., Flores-Troncoso J., A Software Defined Interconnection Architecture for Systems on Chip. *Difu100ci@ Revista de Ingeniería y Tecnología* **2017**, 10(2).
- Sandoval-Arechiga R., Vazquez-Avila J.L., Parra-Michel R., Flores-Troncoso J., Ibarra-Delgado S. Software Defined Networks-on-Chip for multy/many-core systems: A performance evaluation, *ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, Santa Clara, USA, 2016, pp. 129-130.
- Sandoval-Arechiga R., Vazquez-Avila J.L., Parra-Michel R., Flores-Troncoso J., Ibarra-Delgado S. Shifting the Network-on-Chip Paradigm towards a Software Defined Network Architectura. *International Conference on Computational Science and Computational Intelligence*, Las Vegas, USA, 2015, pp.869-870.

1.9. Estructura general de la tesis

El resto del presente documento está estructurado de la siguiente forma: en el capítulo dos, se muestra el marco teórico que presenta los principales conceptos que

deben de ser tomados en cuenta en un sistema de interconexión para un SoC. Al mismo tiempo se integra un estudio de los principales trabajos que la comunidad científica ha presentado al respecto. En el capítulo tres se presenta el modelo del sistema, donde se mencionan las herramientas, modelos, topologías y suposiciones hechas para el desarrollo del estudio. La arquitectura de la infraestructura de interconexión propuesta con un enfoque SDN es presentada en el capítulo cuatro. Los resultados obtenidos así como la discusión de los mismos son presentadas en el capítulo cinco. Finalmente las conclusiones de este estudio se presentan en el capítulo seis.

Capítulo 2

Marco Teórico

La industria informática y electrónica busca constantemente dar soluciones a problemas complejos con aplicaciones que se encuentren integradas en circuitos cada vez más pequeños, baratos y que consumen menos energía. El desarrollo de soluciones basadas en SoC es una alternativa común hoy en día. Sin embargo, los retos a los que se enfrentan los desarrolladores son diferentes y variados. Uno de ellos, quizá el más importante, tiene que ver con el sistema de intercomunicación que existe entre los elementos de procesamiento que hay en el interior de un SoC. Antes de ofrecer una alternativa de solución a este problema, en este capítulo se describirán algunos conceptos fundamentales de los SoC y de sus sistemas de interconexión. Inicialmente se describirán las principales características de los SoC. Posteriormente se describirán los principales tipos de sistemas de interconexión utilizados en los SoC así como sus ventajas y desventajas. A continuación se presentará la necesidad de realizar arbitraje y los principales tipos de árbitros que se utilizan en los SoC. El trabajo aquí presentado toma como marco de referencia el paradigma SDN, los principales conceptos de este modelo son presentados en este capítulo. Enseguida se muestra el enfoque SDN aplicado a los SoC. Se indican las principales diferencias con los sistemas de interconexión tradicionales y se delinean las características de la infraestructura necesaria para implementar sistemas de interconexión con el enfoque SDN. El desarrollo de SoC está ligado al cumplimiento de requerimientos de operación que comúnmente son englobados en lo que se denomina Calidad de Servicio. En este capítulo se describen las principales medidas que se usan en este sentido. Finalmente, se menciona la necesidad de mapear aplicaciones sobre un conjunto de procesadores en un SoC.

2.1. Sistemas-en-Chip

Una constante que se observa en la industria del diseño electrónico, es su capacidad ininterrumpida de crecimiento. La cantidad de elementos electrónicos que pueden ser integrados en una sola pastilla es cada día mayor. Esto ha permitido no solo contar con circuitos integrados que puedan realizar operaciones más complejas, sino integrar, en un solo chip, múltiples elementos de procesamiento, que en la industria del diseño electrónico se les conoce como SoC. Lee and Bergmann (2003), señalan que inicialmente esta tecnología se convirtió en la metodología predominante para el diseño de los denominados *Circuitos Integrados para Aplicaciones Específicas* (Application Specific Integrated Circuits (ASIC), en inglés). Sin embargo, hoy en día resulta también común encontrarlos en sistemas en chip reconfigurables como el denominado Zynq-7000, de Xilinx (2018) o el AGILEX, de Intel (2020b). Cuando a los SoC se les pudieron integrar cientos de elementos de procesamiento, fue necesario que a éstos se les adicionaran nuevas capacidades, las cuales permitan que la industria provea productos con mejores prestaciones, al mismo tiempo que estén en el mercado en el menor tiempo posible. Dos de las características más importantes que se busca en la industria hoy día, con respecto a los SoC, son: la reusabilidad y su capacidad de integración.

2.1.1. Reusabilidad

Desde hace más de una década, algunos autores, Saleh et al. (2006); Keating and Bricaud (2012), han señalado que una de las principales características que se busca que tengan los SoC es la reusabilidad. Es decir que los denominados IPCores, puedan ser utilizados en diferentes proyectos, en diferentes momentos, por diferentes equipos y por diferentes compañías. Esto es posible hoy en día porque los IPCores son implementados en base a código desarrollado en un *Lenguaje de Transferencia de Registros* (Register Transfer Level (RTL), en inglés) que puede ser transportado a cualquier tecnología. Los IPCores pueden ser propietarios o provenir de terceros. Existe una gran variedad de IPCores: procesadores de propósito general, procesadores de propósito específico, controladores de memoria, controladores de buses, interfaces, etc. Y son utilizados en diferentes áreas: comunicaciones, procesamiento de imagen, procesamiento de vídeo; y por varios sectores: defensa, tecnología aeroespacial, automotriz, medicina, industria, entre otras. Cada día son más las aplicaciones que son puestas al mercado en cada una de las áreas anteriormente mencionadas y existe mucha presión por parte del mercado para obtener más rápido, nuevos y mejores productos. Por lo que nuevas técnicas deben de ser utilizadas para el desarrollo de productos. Algunas de los razones por las cuales se considera necesario la reusabilidad de los IPCores son identificadas por Keating and Bricaud (2012):

- Presiones por reducir el tiempo de salida al mercado de los productos.
- La calidad de los resultados, en términos del rendimiento en área y energía, son claves para el éxito en el mercado.
- El incremento en la complejidad del *chip* hace el proceso de verificación más difícil.
- La operación a nivel *submicron* hace más difícil la sincronización.
- Los equipos de desarrollo tienen diferentes niveles y áreas de *expertise*, y a veces se encuentran dispersos.
- Los diseños basados en SoC frecuentemente incluyen procesadores embebidos, y por lo tanto un componente de *software* significativo que conduce a desafíos adicionales de metodología, proceso y organización.

Sin embargo, es importante mencionar que el reuso de componentes implica una metodología de diseño con cambios significativos respecto a las metodologías tradicionales. Según Keating and Bricaud (2012) esta nueva metodología debe de estar basada en los siguientes principios:

- Creación en todas las etapas del desarrollo, desde las especificaciones hasta la integración en el silicio, con el pensamiento que el diseño puede ser modificado y reusado en otros proyectos y por otros equipos de diseño.
- El uso de herramientas y procesos que capturen la información de diseño de una forma consistente y fácil de entender.
- El uso de herramientas y procesos que hagan fácil la integración de módulos dentro de un diseño, cuando el diseñador original no es parte del proyecto.

Además de las técnicas tradicionales de diseño: buena documentación, buen código, comentarios minuciosos, buenos ambientes de verificación y el desarrollo de *scripts* robustos, existen requerimientos adicionales que el IPCore debe de cumplir cuando es diseñado bajo el modelo de reusabilidad:

- Debe ser diseñado para resolver un problema general - Lo cual significa que debe de ser fácilmente configurable para ser utilizado en diferentes aplicaciones.
- Debe ser diseñado para ser usado con múltiples tecnologías - Esto significa que se deben de tener resultados satisfactorios cuando son sintetizados con diferentes tecnologías.

- Diseñado para simulación con una variedad de simuladores - Un IPCore o *testbench* de verificación que trabaje sólo con un simulador no es portable. Una buena práctica de reuso indica que deben de estar disponibles las versiones del modelo y *testbench*es para los principales *Lenguajes de Descripción de Hardware* (Hardware Description Language (HDL), en inglés) y deben de trabajar en la mayoría de los simuladores comerciales.

- Los diseños deben de poder ser verificados independientemente del chip en el cual van a ser usados - Los diseños reusables deben tener *testbench* completos e independientes que les permiten diferentes niveles de cobertura.

- Deben de ser verificados con un alto nivel de confianza - Esto significa, rigurosa verificación y construcción de un prototipo físico que sea probado en un sistema corriendo aplicaciones reales.

- Altamente documentado en términos de las aplicaciones apropiadas y sus restricciones - Deben de ser documentados a detalle los parámetros y configuraciones válidas, cualquier restricción de los valores en los parámetros debe de ser claramente especificada. Los requerimientos de la interfaz y restricciones de como debe de ser usado el IPCore deben de estar documentados.

2.1.2. Integración

La idea de poder integrar en un SoC diferentes IPCores, ya sean de manufactura propia o de terceros, es muy atractiva. Sin embargo, hay un conjunto de situaciones que deben de ser tomadas en cuenta. Entre otras: los IPCores que integran al sistema pueden estar diseñados para operar a diferente velocidad de reloj; la unidad de transferencia de datos de cada IPCore puede tener diferente tamaño; la interfaz de comunicación de cada IPCore puede ser diferente. En Saleh et al. (2006) se menciona que existen aproximaciones que permiten abordar este tipo de problemas, con el uso de modelos transaccionales, como los provistos por SystemC (2020) y SystemVerilog (2020), en el que los IPCores se comportan como islas aisladas con sus propios dominios de reloj y que se comunican entre sí por medio de un sistema con reloj global. Esta aproximación se denomina Globally Asynchronuos, Locally Synchronous la cual fue propuesta inicialmente por Shapiro (1984).

Los sistemas de interconexión son más que un conjunto de cables que permiten la conexión de los diferentes elementos que se encuentran dentro de un SoC. Es la infraestructura que permite la integración ordenada de los mismos. Esta infraestructura se ha convertido en una parte fundamental y crítica en el diseño de soluciones de procesamiento basadas en SoC. Sus características impactan directamente en el

rendimiento del sistema. Diferentes alternativas de solución para construir la infraestructura de comunicación en el interior de un SoC han sido propuestas. Las principales son:

- Sistemas de interconexión basados en *Bus*.
- Sistemas de interconexión basados en *Crossbar* .
- Sistemas de interconexión basados en *Network-on-Chip*.

Ben Achballah et al. (2017) y Bjerregaard and Mahadevan (2006), apuntan en sus respectivos estudios que no existe una solución única al problema de interconexión de los SoC. De tal modo que las arquitecturas más que competir entre sí, son adecuadas, en mayor o menor grado, a la aplicaciones específicas que estarán soportando. Los sistemas basados en *bus* se han convertido en la alternativa para los SoC que integran un número pequeño de Elementos de Procesamiento (EP). Benini and De Micheli (2002) apuntan que éstos se encuentran bien situados para sistemas que integran de cuatro a ocho EP. Por otro lado los denominados *crossbar* se convierten en la opción adecuada cuando el número de EP se encuentra en el orden de las decenas como se indica en Ahmed et al. (2017). Las NoC son la alternativa comúnmente utilizada cuando el número de EP se encuentra en el orden de los cientos o miles como se apunta en Wang et al. (2009).

2.1.3. Calidad del Servicio

QoS es un término muy utilizado en los sistemas de comunicaciones, tiene que ver principalmente con cumplir con el conjunto de especificaciones de operación establecidas por el usuario, las cuales deben de ser satisfechas para que el servicio sea considerado de calidad. En términos de SoC las métricas más estudiadas en este sentido son:

- **Acceso:** Se refiere a que se le garantice el acceso al medio a todos los elementos que lo tengan permitido y que lo soliciten. Se debe evitar a toda costa que un elemento entre en *starvation* por no poder acceder a algún recurso.
- **Latencia:** Esta es una medida de QoS principalmente utilizada para aplicaciones de tiempo real. Se debe de garantizar que una aplicación se ejecute dentro de una restricción de tiempo especificada.
- **Utilización:** Se enfoca en maximizar la utilización del medio compartido incrementando el rendimiento global del sistema.

- **Igualdad de Servicio:** Se refiere al servicio general experimentado por cada elemento de procesamiento dentro de un sistema de interconexión. Esta medida tiene un impacto más significativo en una NoC debido a la forma en que los diversos patrones de tráfico impactan las diferentes secciones de la NoC.
- **Ancho de Banda:** Se refiere al tiempo de uso, de un recurso compartido, que le debe de ser garantizado a un elemento de procesamiento para que opere satisfactoriamente.

2.1.4. Mapeo de Aplicaciones

El conjunto de elementos de procesamiento que se encuentran en el interior de un SoC, puede ser aprovechado para ejecutar algoritmos complejos divididos en un conjunto de tareas simples. Con esto existe la posibilidad de incrementar el rendimiento global de la aplicación. Una aplicación es dividida en tareas utilizando el denominado Grafo de Comunicación de Tareas (GCT) el cual expresa las tareas y sus relaciones. Un GCT consiste en un conjunto de nodos enlazados - las tareas - y enlaces de comunicación a través de los cuales los datos fluyen entre las tareas. La figura 2.1 muestra el gráfico de tareas de una aplicación. Cada nodo ejecuta una tarea en un momento específico (los valores en rojo), y cuando concluye, transmite una cantidad de datos (los valores en negro) a sus nodos hijos. Un nodo no puede ejecutar su tarea hasta que no reciba datos de los nodos padres. Una aplicación comienza su ejecución en el nodo raíz (nodo 0) y termina cuando todos los nodos que salen han terminado de operar (nodos 8,9,10).

Una aplicación puede explotar la potencia de cálculo dentro de un SoC utilizando un conjunto de elementos de procesamiento para ejecutar las tareas que lo constituyen. Para ello es necesario utilizar una técnica que permita mapear el gráfico de tareas de una aplicación dentro de los elementos de procesamiento adjuntos al sistema de interconexión del SoC. La figura 2.1 muestra el resultado del mapeo de la red de tareas de una aplicación utilizando dos técnicas diferentes García Morales et al. (2019). La distribución realizada por cada técnica puede afectar significativamente al tráfico generado en el sistema de interconexión.

2.2. Sistemas de Interconexión en Sistemas en Chip

Actualmente el desarrollo de IPCores es un segmento de la industria que se encuentra bastante consolidado. Una de las razones de este éxito es la capacidad que han tenido los desarrolladores de SoC de adoptar interfaces de comunicación estandarizadas para la intercomunicación de los mismos. Un ejemplo de esto es la interfaz de comunicación AXI desarrollada por ARM (2010).

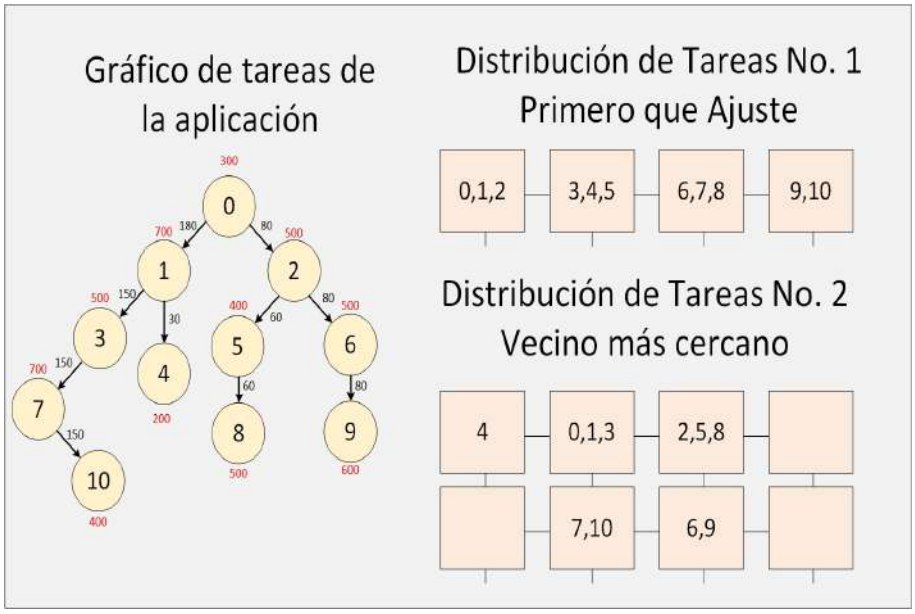


Figura 2.1: Ejemplo de mapeo de una aplicación utilizando dos técnicas diferentes García Morales et al. (2019)

Para que un desarrollo de *hardware* pueda ser considerado como un IPCore debe estar integrado por dos módulos: uno, el *motor* que ejecuta el algoritmo para el que fue específicamente desarrollado; dos, la *interfaz* de comunicación con otros EP. Los elementos que permiten la intercomunicación entre la *interfaz* y el *motor* son: uno, memorias de almacenamiento intermedias que sirven como *buffers* para obtener los datos y almacenar los resultados; y dos, un *controlador* que le indica al *motor* cuando la *interfaz* le solicita una acción, y en sentido contrario, cuando el *motor* le indique a la *interfaz* que ha concluido una acción. Adicionalmente el IPCore puede tener otro tipo de comunicación hacia el exterior del SoC con otros circuitos por medio de interfaces bien conocidas como *UART*, *IIC*, *SPI*, *WiFi*, *Bluetooth*, etc. La estructura general de un IPCore se puede observar en la Figura 2.2.

El uso de sistemas de interconexión bien definidos, permite resolver algunos problemas fundamentales que se presentan en la integración de los SoC. Al aislar el procesamiento de los datos, del transporte de los mismos, es posible el uso de varios dominios de reloj. La diferencia en el tamaño y formato de los datos también puede ser resuelta por los elementos del sistema de interconexión. Además, al dejar de ser el transporte de los datos un problema para los desarrolladores de IPCores, estos se pueden concentrar más en los problemas inherentes al algoritmo que están desarrollando como: rendimiento del sistema, consumo de energía y cantidad de área ocupada.

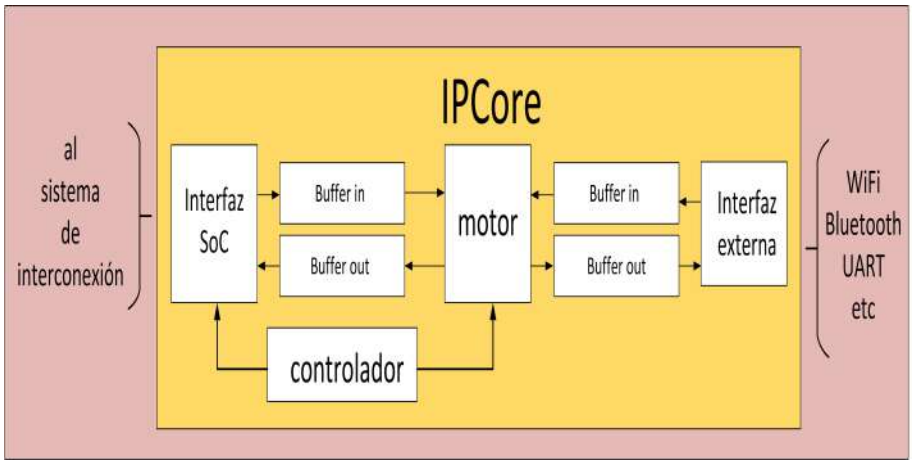


Figura 2.2: Arquitectura de un IPCore

La arquitectura básica de un SoC queda definida por dos planos: uno, el Plano de Reenvío de Datos, que es el encargado de la interconexión de los elementos de procesamiento; dos, el Plano de Procesamiento de Datos, encargado de la transformación de los datos. En la Figura 2.3 se muestra esta arquitectura.

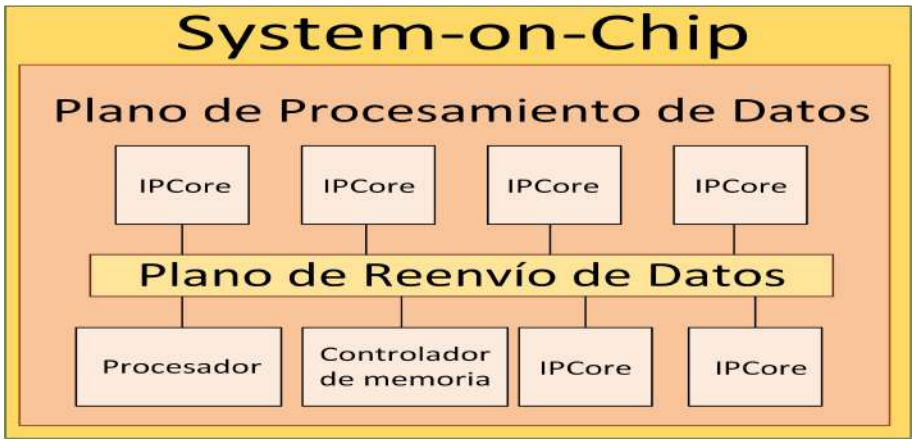


Figura 2.3: Arquitectura básica de un SoC

Los principales sistemas de interconexión utilizados en los SoC se describen a continuación.

2.2.1. Bus

Un *bus*, Figura 2.4, es una infraestructura utilizada para conectar un conjunto de EP que forman parte de un SoC. Cada uno de los EP se conecta con el *bus* por

medio de una Interfaz de Red (IR). La IR es el elemento que separa el transporte de los datos de la operación de los mismos. Un EP ejecuta una o varias tareas y transmite los resultados a través de la IR. El EP no tiene control sobre el origen y/o destino de los datos, su labor consiste en procesar la información que obtuvo por medio de la IR y entregarle a ella misma el resultado.

La IR es la encargada de recibir o transmitir los datos a otro elemento en el sistema de interconexión siguiendo el protocolo del *bus*. Primero, solicitando el acceso al mismo por medio de la señal de *solicitud*, pudiendo acceder al sistema de interconexión sólo cuando se le otorga el permiso por medio de la señal de *concesión*. Una vez que le es permitido el acceso al sistema de interconexión, la IR mantiene el control del *bus* mediante la señal de *sostener*, la cual debe de mantener activa durante todo el tiempo que esté realizando la transacción. Cuando la IR termina la transacción libera el *bus* desactivando la señal de *sostener*.

Este tipo de sistema de interconexión solo permite el acceso al *bus* a un solicitante a la vez. Cuando varios EP solicitan simultáneamente el acceso al *bus* el denominado *árbitro* es el encargado, bajo una política de arbitraje, de decidir a quién le otorga el acceso al sistema de interconexión. Una vez decidido a cuál EP se le otorgará el acceso, se le hace saber por medio de la señal de *concesión* asociada a este EP. El camino que siguen los datos de un EP a otro es designado por el *árbitro* por medio del denominado Controlador de Datos del Bus (CDB).

2.2.2. Crossbar

Un *Crossbar* es un elemento de interconexión para SoC que permite la comunicación simultánea de m entradas con n salidas, siempre y cuando dos o más entradas no se encuentren solicitando una misma salida. Los *Crossbar* son los elementos de conmutación que con mayor frecuencia son utilizados en la industria de los SoC reconfigurables. Por ejemplo, la compañía Xilinx utiliza para conectar los diferentes IPCores que se integran en un SoC el denominado *AXI SmartConnect*, Xilinx (2020). Este dispositivo tiene la capacidad de conectar uno o más IPCores del tipo *maestro* con interfaz AXI, con uno o más IPCores del tipo *esclavo* con la misma interfaz. El *AXI SmartConnect* es altamente reconfigurable, entre otras capacidades puede realizar la conversión del ancho de datos entre *maestros* y *esclavos*; puede operar IPCores que se encuentren en diferentes dominios de reloj; permite la conversión de protocolos. El núcleo de un *AXI SmartConnect* es un *Crossbar*. Su arquitectura está basada principalmente en elementos de conmutación y un sistema de arbitraje. Otro ejemplo de un sistema de interconexión tipo *Crossbar* es la denominada *Avalon Interface* de Intel, la cual es la arquitectura utilizada para conectar los IPCores que integran en su serie de circuitos basados en *Matrices de Compuertas Lógicas Programables en Campo* (*Field Programmable Gate Array (FPGA)*, en inglés) o en sus

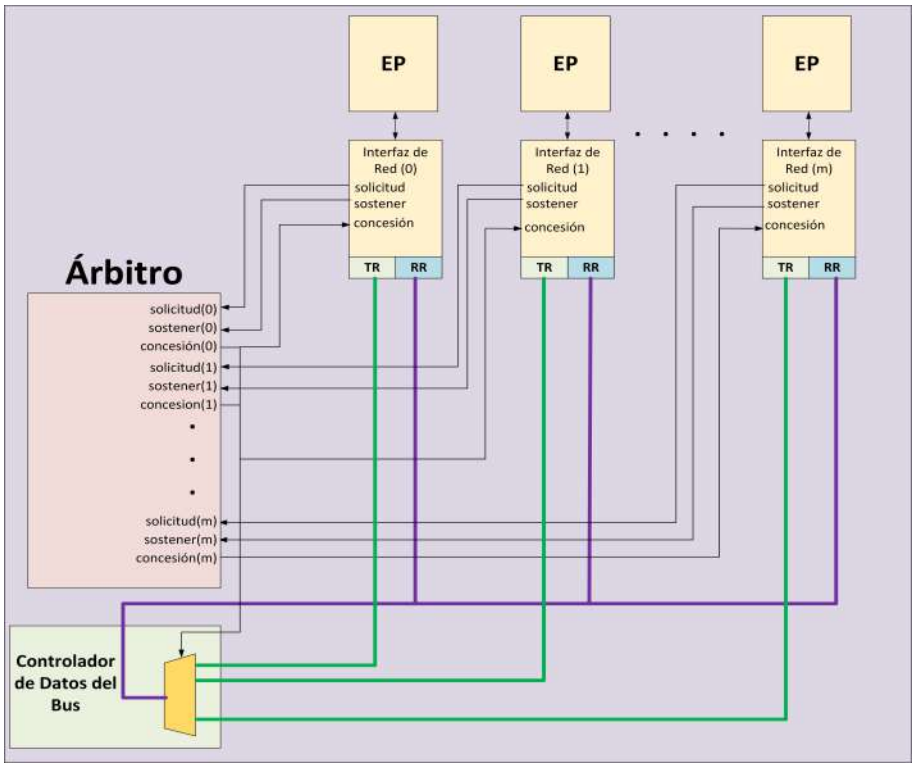


Figura 2.4: Infraestructura básica de un Sistema de Interconexión basado en Bus

circuitos basados en SoC. Esta infraestructura añade un *árbitro* y un *conmutador* por cada *esclavo* que se suma al sistema, permitiendo al final una conectividad total entre cualquier *maestro* y *esclavo* en el sistema.

Una arquitectura simple tipo *Crossbar* para cuatro *maestros* y cuatro *esclavos* es presentada en la Figura 2.5. Esta se encuentra constituida por dos elementos principales: uno, el *conmutador crossbar* formado por un conjunto de *multiplexores* que enrutan una de las n entradas hacia la salida $m(destino)$; dos, un *asignador* que permite al acceso único de uno de los *maestros* a uno de los *esclavos*. En el interior del *asignador* existen una serie de n *árbitros*, uno por cada *esclavo* en el sistema, que otorga de manera independiente el acceso a cada *esclavo*.

2.2.3. Network-on-Chip

Como es mencionado por Dally and Towles (2001) y Benini and De Micheli (2002), las NoC son sistemas de interconexión utilizados en los SoC como una respuesta a la cada vez mayor cantidad de EP que pueden ser integrados en un solo circuito. La necesidad de comunicación simultánea entre cientos o miles de EP en un SoC obliga a proveer estrategias de comunicación flexibles y robustas. Las NoC

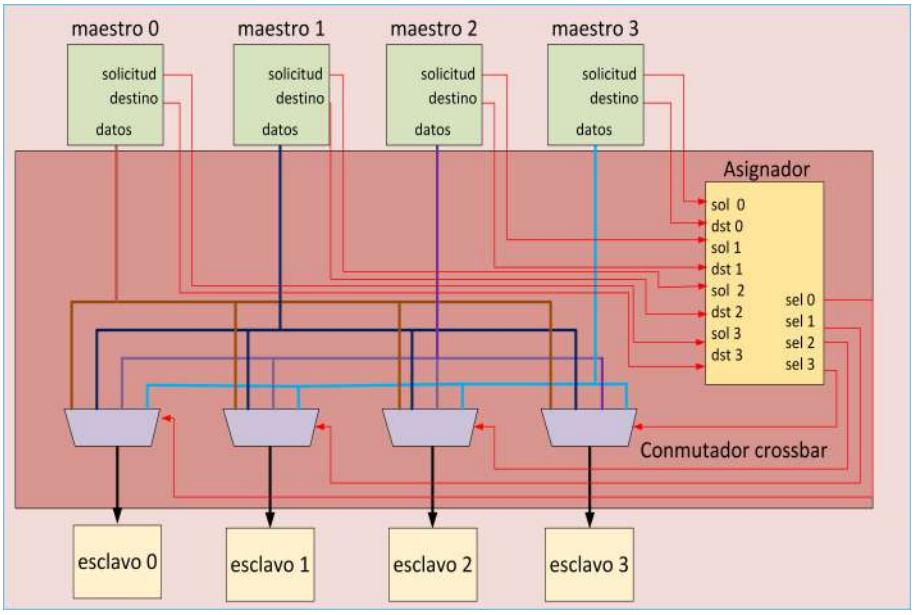


Figura 2.5: Infraestructura básica de un Sistema de Interconexión tipo Crossbar

surgen con el objetivo de proveer una infraestructura de comunicación general entre los elementos de un SoC, en lugar de tener una estructura de cableado particular para cada una de las aplicaciones que hay en ellos. Las principales ventajas que se obtienen al integrar una NoC en un SoC son: estructura, rendimiento y modularidad. Las NoC toman los conceptos, modelos y técnicas utilizados en el campo del diseño de redes de comunicación, y los adecúan y aplican al interior de un SoC. El modelo establecido por Benini and De Micheli (2002) propone una pila con tres capas:

- Capa Física, encargada del cableado estructurado de la red.
- Capa de Arquitectura y Control, encargada del enlace y el transporte de los datos en la red. La arquitectura especifica la topología de la red de interconexión y la organización física. El protocolo especifica cómo utilizar los recursos de la red durante la operación del sistema.
- Capa de *Software*, es la encargada de proveer los servicios *end-to-end* entre aplicaciones, las librerías y facilidades para que el sistema pueda ser soportado por programas estándares de programación.

En la Figura 2.6 se puede observar la arquitectura de una NoC con topología tipo malla (solamente tomada como ejemplo, en la literatura se presentan diversas topologías donde se muestran las ventajas y desventajas de cada una de ellas Salminen et al. (2008); Bjerregaard and Mahadevan (2006); Pande et al. (2005)). Se

puede observar que los EP se comunican entre ellos por medio de una red constituida por un conjunto de *ruteadores*. Estos tienen como función principal encaminar un paquete de datos desde un EP inicial hasta un EP destino. La ruta que seguirá el paquete dependerá de la política de ruteo establecida. Los EP se conectan a la red por medio de una IR. Los *ruteadores* constan de cinco puertos de acceso que sirven como entrada para un paquete. Este paquete es dirigido a una de las otras cuatro salidas de acuerdo a la información de cabecera que tiene el paquete que viaja por la red. La unidad de lógica y conmutación del *ruteador* es la encargada de direccionar el paquete entrante hacia una de las salidas. La estructura de un *ruteador* en una NoC se muestra en la Figura 2.7

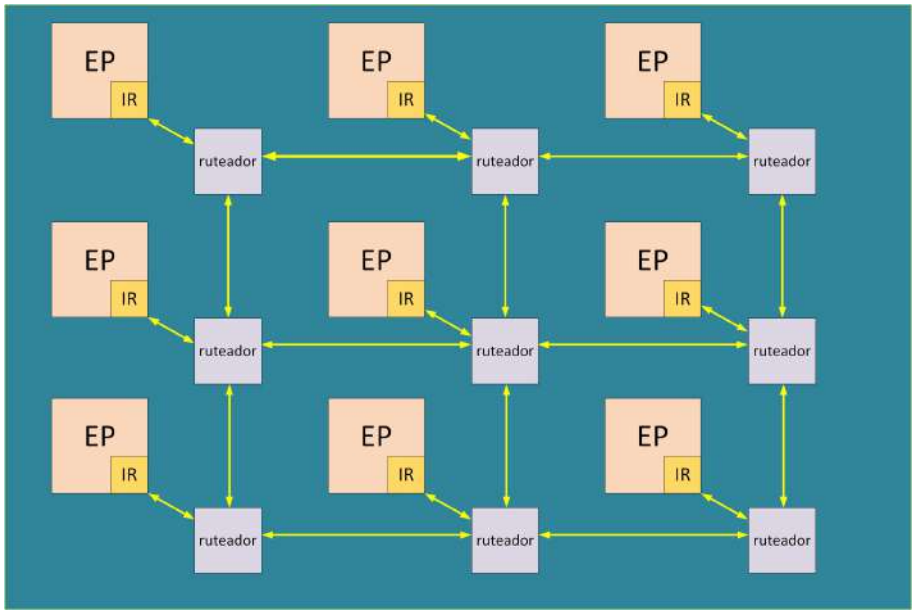


Figura 2.6: Infraestructura básica de un Sistema de Interconexión basado en una NoC

2.3. Ventajas/Desventajas de los Sistemas de Interconexión para Sistemas en Chip

Anteriormente se ha mencionado que los sistemas de interconexión más que competir se complementan. Cada uno de ellos presenta una serie de ventajas y/o desventajas de acuerdo a la situación en la que serán usados. Algunos autores como Guerrier and Greiner (2000); Ben Achballah et al. (2017); Kundu and Chattopadhyay (2015) indican algunas de ellas.

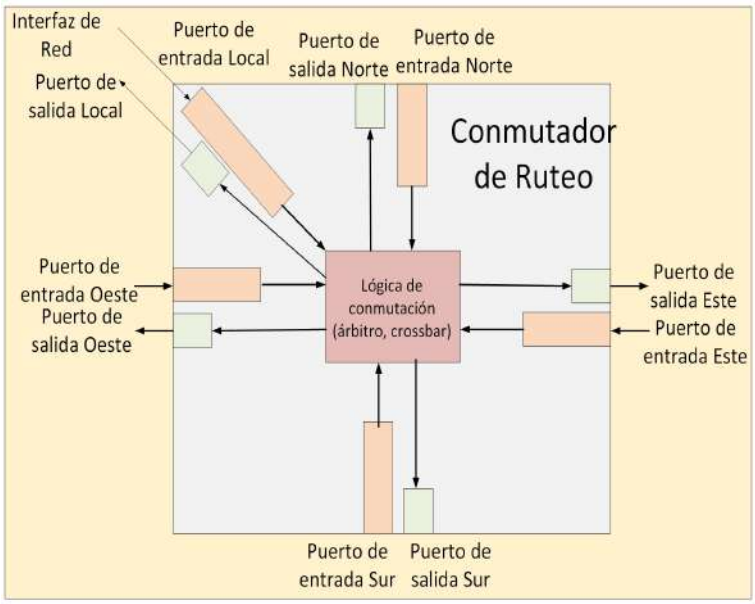


Figura 2.7: Arquitectura básica de un *ruteador* para una NoC

2.3.1. Bus

Las principales ventajas que presenta este tipo de sistema de interconexión son:

- La latencia del *bus* es cero una vez que el *árbitro* le otorgó el control a un EP.
- El costo del silicio de un *bus* es cercano a cero.
- Casi cualquier *bus* es directamente compatible con los IPCores disponibles en el mercado, incluyendo el *software* corriendo en los EP.
- El concepto es simple y bien entendido.
- Las transmisiones tipo *broadcast* son muy sencillas.

Las desventajas que presenta este tipo de sistema de interconexión son:

- Cada unidad incorporada al sistema, agrega capacitancia parásita.
- La sincronización del *bus* es difícil en procesos submicrónicos.
- Las pruebas de funcionamiento son problemáticas y lentas.
- Los retardos en el otorgamiento del acceso se incrementan conforme el número de EP aumenta.

- Solamente es posible un dominio de reloj.
- Solo un EP puede ocupar el *bus* en un tiempo.

2.3.2. Crossbar

Las principales ventajas que presenta este tipo de sistema de interconexión son:

- Varios *maestros* pueden hacer uso simultáneo del sistema de interconexión siempre y cuando soliciten la atención de diferentes *esclavos*.
- Puede haber diferentes dominios de reloj.
- Flexibilidad en la comunicación.
- Pueden ser acoplados elementos con diferente ancho de datos.

Las desventajas que presenta este tipo de sistema de interconexión son:

- Verificación e integración difícil.
- Necesita ser adecuado a nuevos sistemas lo cual introduce tiempo adicional de diseño.

2.3.3. Network-on-Chip

Las principales ventajas que presenta este tipo de sistema de interconexión son:

- La separación entre la ejecución y el transporte permite la independencia del sistema, múltiples dominios de reloj pueden ser utilizados.
- Múltiples EP pueden utilizar la red simultáneamente.
- Las decisiones de ruteo son distribuidas y el mismo *ruteador* puede ser replicado independientemente del tamaño y topología de la red.
- El ancho de banda escala con el tamaño de la red.
- Sólo un tipo de cable es usado para toda la red, los tiempos de propagación pueden ser claramente identificados.
- Los cables en la red pueden estar en una estructura tipo *pipeline* porque el protocolo de la red es globalmente asíncrono.

Las desventajas que presenta este tipo de sistema de interconexión son:

- La NoC consume un espacio significativo de área del silicio.
- La contención al interior de los *ruteadores* aumenta la latencia.
- Es necesario la elaboración de mejores y más complicados contenedores de los IPCores que se integran en el SoC.
- Es necesario contar con diseñadores que entiendan conceptos más avanzados de redes.
- No todas las aplicaciones permiten aprovechar la distribución del trabajo en paralelo.
- Se requiere hacer un esfuerzo adicional en el mapeo y programación de las tareas.

2.4. Arbitraje

2.4.1. Necesidad del Arbitraje

En todos los sistemas de interconexión se hace necesaria la existencia de un elemento de arbitraje que permita el acceso a los recursos compartidos. En el caso de un sistema de interconexión basado en *bus* los EP compiten por ganar el control del *bus*. En los *Crossbars* dos o más *maestros* compiten por obtener los servicios de un *esclavo*. En el caso de las NoC dos o más *flujos* compiten por tener el acceso a un puerto de comunicación. Es entonces cuando las políticas de arbitraje se convierten en parte fundamental en cualquier sistema de interconexión dentro de un SoC. Sin embargo, existen una serie de problemas a los que se enfrentan las políticas de arbitraje y que de acuerdo a la forma en que los resuelven, muestran su eficacia. Jain et al. (2015) mencionan los siguientes:

- *Congestion* - Se da cuando muchos puertos de entrada se encuentran solicitando un mismo puerto de salida.
- *Starvation* - Es un tipo de injusticia en la cual no todos los puertos de entrada tienen las mismas oportunidades de acceder al puerto de salida.
- *Deadlock* - Existe una dependencia cíclica entre puertos, un puerto de salida no puede ser alcanzado por un puerto de entrada debido a que éste se encuentra esperando por otro puerto de entrada para liberar el recurso.
- *Livelock* - Los paquetes de un puerto de entrada están en movimiento pero no pueden alcanzar el puerto deseado de salida.

- *Header Blocking* - Este problema se presenta principalmente en los *ruteadores*. Consiste en que un puerto de entrada de un *ruteador* se encuentra con el *buffer* lleno debido a que el sistema de arbitraje le está dando preferencia a otro puerto. Si en el *ruteador* anterior a éste hay un paquete que viene a este puerto, el paquete no puede avanzar porque el *buffer* de entrada está lleno y retiene a los paquetes que vienen detrás de él aunque vayan dirigidos a otro puerto.

El grado en el que una política de arbitraje pueda resolver los problemas anteriormente mencionados será uno de los factores a tomar en cuenta para su selección. Otros factores que influyen en la selección de una política de arbitraje son: el grado en que permite el uso diferenciado de los recursos, el área que ocupa el árbitro en el SoC, y la frecuencia máxima a la que puede operar al árbitro.

2.4.2. Políticas de Arbitraje

En la literatura se han propuesto diversas políticas de arbitraje, cada una de ellas con diferentes tipos de implementación. El enfoque inicial de estas políticas, fue el que permitieran lograr el acceso justo a un recurso. Es decir, que todos los elementos que solicitaran acceso, tuvieran la misma posibilidad de obtener el recurso en un momento determinado. Básicamente lo que se trataba de evitar era el problema de *starvation*. Posteriormente, con la necesidad que hoy en día se tiene de que las aplicaciones cumplan con sus requerimientos de QoS, se convierte en algo muy importante tomar en cuenta que el uso de un recurso puede variar dependiendo de la aplicación que se esté ejecutando. En muchas ocasiones es necesario permitir un acceso diferenciado al recurso para que los EP puedan cumplir con sus cuotas de operación. Ante esto, dos tipos de soluciones han sido planteadas: una, árbitros que permitan el uso diferenciado del recurso; dos, sistemas de apoyo que coadyuven en regular el uso del medio. A continuación se describen las políticas e implementaciones de los principales sistemas de arbitraje que se utilizan en los SoC.

Árbitro de Prioridad Fija

Ésta es la forma más simple de arbitraje, una implementación de este arbitro propuesta por Dally and Towles (2003) es mostrada en la Figura 2.8. En el Árbitro de Prioridad Fija, el orden en que son servidas las peticiones sigue un estricto orden de prioridad. En una ronda de arbitraje, el árbitro detecta la señal de *solicitud* del elemento con mayor prioridad; si la señal se encuentra activa, el *bus* le es otorgado lo cual se le hace saber activando la correspondiente señal de *concesión* asociada a este elemento. Posteriormente se propaga una señal a todos los demás elementos en el árbitro indicándoles que el *bus* ha sido asignado. Si la señal de *solicitud* del

elemento de mayor prioridad no está activa, se propaga la señal de *solicitud* invertida indicando al siguiente elemento en orden de prioridad, que el *bus* no ha sido asignado y que si él lo requiere puede obtenerlo. Este proceso se repite en cascada, por orden de prioridad para todos los elementos en el sistema. El problema de *starvation* puede ser muy pronunciado en este tipo de arbitraje. Si los elementos con mayor prioridad se encuentran constantemente solicitando el acceso al *bus*, impiden que los elementos de menor prioridad puedan obtenerlo. Otro problema a tener en cuenta, es el tiempo de respuesta del circuito, debido a la lógica en cascada con que se implementa. Este tiempo es linealmente proporcional al número de entradas en el circuito.

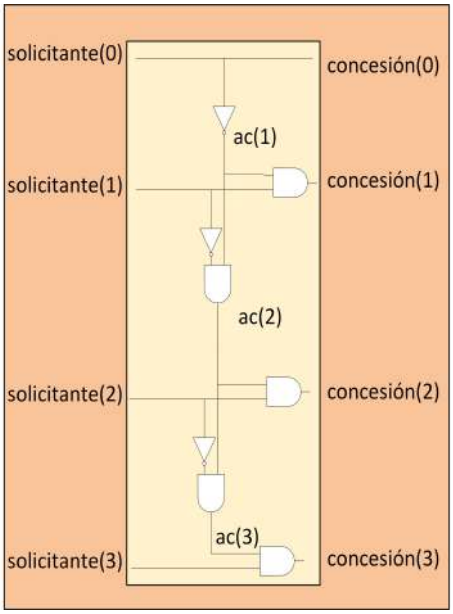


Figura 2.8: Arquitectura de un Árbitro de Prioridad Fija Dally and Towles (2001)

Árbitro de Prioridad Variable

Los Árbitros de Prioridad Variable, tienen la capacidad de poder modificar la prioridad de los *solicitantes* de acuerdo a una política específica. Normalmente están constituidos por dos unidades: la primera unidad, genera las señales de prioridad. En una ronda de competencia la prioridad solo le es asignada a un *solicitante*; la segunda unidad, es encargada de seleccionar al elemento ganador activando su señal de *concesión*, y además, propaga una señal de *acarreo* evitando que se active la señal de *concesión* de los demás solicitantes. Una implementación de este tipo de árbitro propuesta por Dally and Towles (2003) es mostrada en la Figura 2.9.

Uno de los principales problemas de este árbitro es que al no existir retroalimentación entre la sección que otorga el acceso al *bus* y la sección que otorga la

prioridad, no se tiene memoria de a quién se le ha otorgado el *bus* y éste puede ser otorgado a un solicitante con más frecuencia que a otros, lo que crea un problema de inequidad.

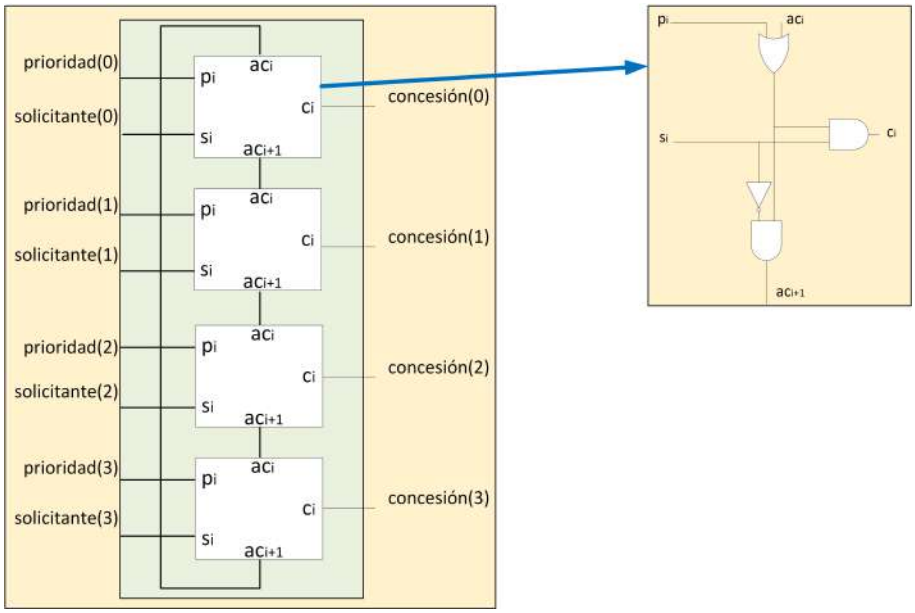


Figura 2.9: Arquitectura de un árbitro con prioridad Variable Dally and Towles (2001)

Árbitro Round-Robin (RR)

El Árbitro Round-Robin, es un árbitro con prioridad variable que tiene como objetivo proveer justicia en el acceso a un recurso compartido. La principal característica de este árbitro es que dada una ronda de competencia en el tiempo t_n toma en cuenta lo que sucedió en la ronda de competencia en el tiempo t_{n-1}. El solicitante que tuvo el acceso al medio en la ronda t_{n-1} se convierte en el solicitante con la menor prioridad en la ronda t_n. Una de las implementaciones más simples, propuesta por Dally and Towles (2003) se muestra en la Figura 2.10. En ésta se puede observar que la prioridad del solicitante p_i es otorgada por la señal de *concesión* del solicitante p_{n-1}.

Quizá por su simplicidad, los recursos que consume y las velocidad de operación que presenta, sea el árbitro del cual existen mayor cantidad de implementaciones. Gupta and McKeown (1999) presentan tres implementaciones basadas en un codificador de prioridad programable. Fatih Ugurdag and Baskirt (2012) muestran una versión optimizada de este tipo de árbitro a la que denominan *Árbitro de Prefijos*

Paralelos Termo-codificados (Thermo Coded - Parallel Prefix Arbiter (TC-PPA), en inglés). El arbitro denominado *Árbitro Ping-Pong* (Ping-Pong Arbiter (PPA), en inglés) propuesto por Chao et al. (1999) consta de un árbol de árbitros de dos elementos que permite conjuntar árbitros de mayor tamaño. Otro árbitro con estructura similar se muestra en Shin et al. (2002). Por su parte Zheng et al. (2002) presentan el árbitro denominado *Árbitro Round-Robin Paralelo* (Parallel Round-Robin Arbiter (PRRA), en inglés) basado en un árbol de búsqueda. Una versión mejorada de este árbitro es presentada en Zheng and Yang (2007). Dimitrakopoulos et al. (2008) proponen el árbitro que llaman *Árbitro Rápido Round-Robin* (Fast Round-Robin Arbiter (FRRA), en inglés) compuesto por dos tipos de nodos con lógica combinacional sencilla en un arreglo por capas con propagación de acarreo. Por otro lado Lee et al. (2009) presentan el *Árbitro Round-Robin de Alta Velocidad y Descentralizado* (High-Speed and Decentralized Round-Robin Arbiter (HDRA), en inglés) donde cada señal de *solicitud* es pasada por un filtro que consta de un *multiplexor* y un *flip-flop* que selecciona a qué solicitante se le otorgará el medio. Dimitrakopoulos et al. (2013) proponen una técnica que denominan *Árbitro y Multiplexor Fusionados* (Merged ARbiter and multipleXer (MARX), en inglés) donde a cada prioridad se le asigna un símbolo aritmético y posteriormente se asigna el medio al símbolo con el valor aritmético máximo. Por su parte Oveis-Gharan and Khan (2015) presentan el denominado *Árbitro Round-Robin basado en Índice* (Index Round-Robin Arbiter (IRRA), en inglés) donde la selección de la próxima concesión del medio se basa en el índice de la concesión actual más uno. Su implementación se efectúa en base a *multiplexores* y un registro que es la memoria del índice de la concesión actual.

La principal cualidad del árbitro RR es la equidad. Por otro lado, su principal desventaja consiste en no tener la posibilidad de otorgar el *bus* en base a un sistema de prioridades.

Árbitro de Matriz

El árbitro de Matriz es un árbitro del tipo RR pero que tiene la característica de que tiene la capacidad de otorgar el *bus* al último solicitante recientemente servido. Es decir, de los solicitantes que se encuentran en un momento solicitando el *bus*, éste le es otorgado al que tenga más tiempo esperando por él. En la Figura 2.11 se muestra la implementación propuesta por Dally and Towles (2003).

La complejidad de este árbitro es grande por lo que su costo en recursos se incrementa notablemente conforme la cantidad de solicitantes se incrementa.

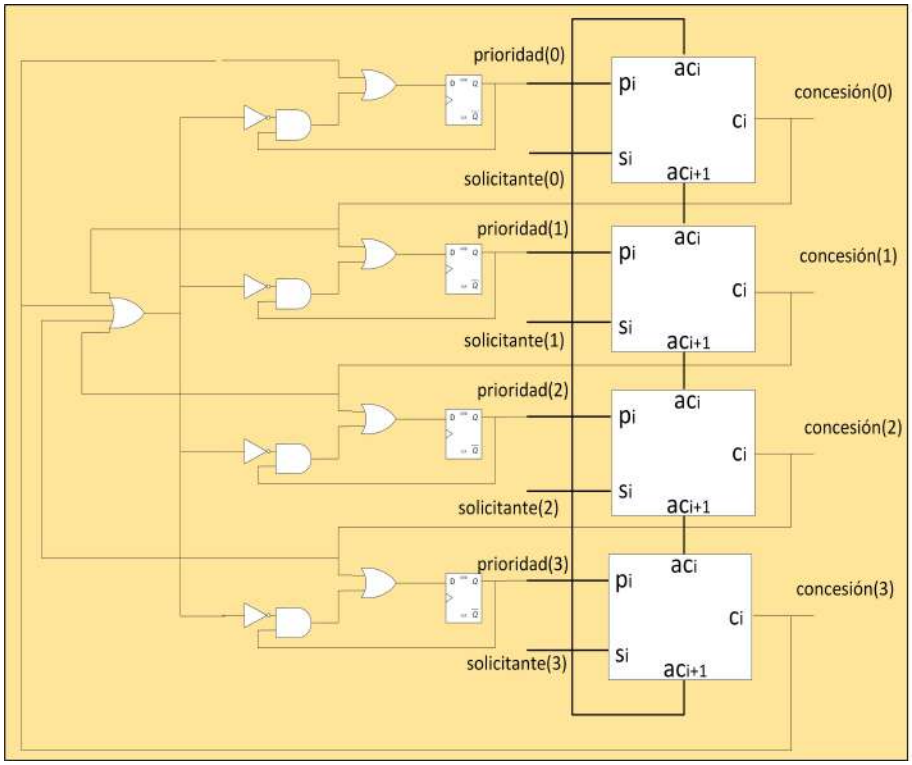


Figura 2.10: Arquitectura de un Árbitro Round-Robin Dally and Towles (2001)

Árbitro de Cola de Espera

Este árbitro otorga la prioridad a aquel solicitante que haya sido menos recientemente servido. Es un árbitro que provee un acceso al medio del tipo *Primero en Entrar Primero en Salir*. Cuando un solicitante desea obtener acceso al medio, genera una etiqueta que combina el índice del elemento que está solicitando el medio, con el tiempo en que lo está solicitando. Al momento de realizar una ronda de arbitraje a los solicitantes que no están solicitando el medio se les asigna un valor de tiempo mayor que el actual. Posteriormente por medio de un árbol de comparadores se selecciona a aquel solicitante que tenga el menor tiempo de arribo registrado. Dally and Towles (2003) proponen una arquitectura para este tipo de árbitro la cual se puede observar en la Figura 2.12.

Dependiendo del número de solicitantes que el árbitro deba atender será la profundidad del árbol de comparadores. Esto condiciona el tiempo de respuesta del mismo y por lo tanto la frecuencia de operación. A mayor cantidad de solicitantes la frecuencia máxima de operación se verá disminuida.

Figura 2.11: Arquitectura de un Árbitro de Matriz Dally and Towles (2001)

Árbitro de Acceso Múltiple por División de Tiempo, (Time Division Multiple Access (TDMA), en inglés)

El árbitro TDMA, distribuye el uso de un recurso en base a *ranuras* de tiempo que se le asignan de forma específica a cada uno de los solicitantes. Un EP solo tiene oportunidad de acceder al recurso durante el tiempo que dure la *ranura* que se le asignó. Si durante el tiempo que se le asigna la *ranura* de tiempo el EP no hace uso del recurso, éste queda ocioso. Dentro de las principales ventajas que tiene este esquema está la de poder asignar de manera diferenciada la cantidad de *ranuras* de tiempo con que cada solicitante puede acceder al recurso. Esto permite poder controlar la proporción del uso del mismo de acuerdo a las necesidades particulares de cada EP. Otra de las ventajas de la política es que su implementación es muy sencilla. La cantidad de recursos necesarios para su implementación es reducida y se tienen frecuencias de operación altas. La principal desventaja de este árbitro es el bajo rendimiento general que se logra. Lo anterior es debido principalmente a los tiempos de ocio que se producen cuando el medio es asignado a un elemento y éste no lo utiliza. Una implementación de un árbitro TDMA se muestra en la Figura 2.13. La implementación consta de un contador circular que sirve como generador de las direcciones –índice– de una memoria pre-programada con el número de *solicitante* que podrá usar el medio en una *ranura* específica y un decodificador que selecciona el solicitante que puede utilizar el recurso.

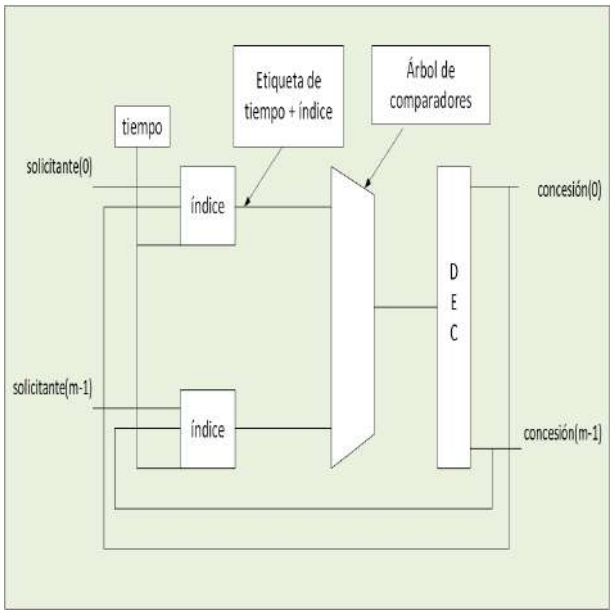


Figura 2.12: Arquitectura de un árbitro de Cola de Espera Dally and Towles (2001)

Árbitro de Lotería, (Lottery (LTY), en inglés)

El árbitro tipo LTY presentado por Lahiri et al. (2006) se fundamenta en un algoritmo de arbitraje probabilístico. En su implementación un "administrador de la lotería" solicita a los elementos que deseen participar en una ronda de arbitraje sus *boletos*. A cada uno de estos participantes se le ha asignado con anterioridad, ya sea de forma estática o dinámica, una cierta cantidad de *boletos* con los que participará en la lotería. No existe un número de *boleto* que sea asignado a dos o más participantes. En cada ronda de arbitraje, el administrador suma la totalidad de *boletos* correspondientes a los participantes que en esa ronda quieren concursar, y genera un número aleatorio entre uno y la suma obtenida. El número aleatorio generado pertenecerá de forma única a un participante al cual le será otorgado el recurso. Mientras más *boletos* se le asignen a un participante más probabilidades tendrá de obtener el acceso al recurso. La implementación de un arbitro con asignación dinámica de *boletos* puede observarse en la Figura 2.14. Las principales ventajas que provee este árbitro son: la primera, cuando el recurso está saturado, se puede tener control sobre la proporción de uso del recurso por parte de los participantes de acuerdo al número de *boletos* asignado; la segunda, provee una latencia promedio baja para comunicaciones de alta prioridad. Por otro lado es importante poner atención en la cantidad de *boletos* que se le asignan a cada participante. Si algunos de ellos tiene poca cantidad de *boletos* comparados con los otros, sus probabilidades de acceder al recurso serán limitadas lo que condicionará severamente su rendimiento individual.

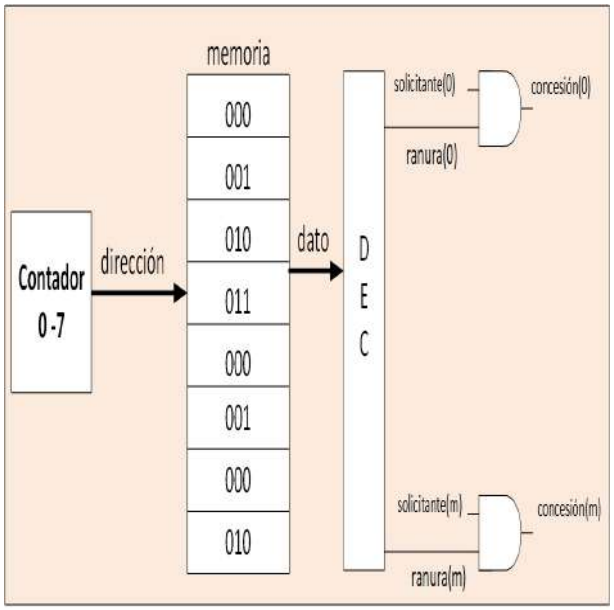


Figura 2.13: Arquitectura de un árbitro TDMA

Árbitro Round-Robin con Pesos, (Weighted Round-Robin (WRR), en inglés)

El árbitro WRR, como el que se muestra en la Figura 2.15, tiene como propósito principal controlar el grado en que un solicitante puede ser servido con respecto a otros. A cada uno de los solicitantes le es asignado un peso $peso_i$ que indica la fracción de uso del recurso que le será otorgado. El peso acumulado de todos los recursos está dado por $Peso = \sum_{j=0}^{n-1} peso_i$ de tal modo que $f_i = \frac{peso_i}{Peso}$. Mientras mayor sea el valor de $peso_i$ mayor sera la fracción de tiempo que se le asignará al $solicitante_i$. Una implementación de un árbitro WRR es presentada por Dally and Towles (2003). En ésta se le asigna un peso inicial a cada solicitante, solamente los solicitantes que tengan un peso diferente de cero serán los que puedan entrar a la ronda de arbitraje. Una vez que un solicitante obtuvo el acceso al recurso, durante cada ciclo de reloj que mantenga el uso del mismo le será descontado una unidad de su peso. Cuando este peso llegue a cero, ya no podrá contender por el recurso hasta que el peso le sea recargado. El peso le es recargado a todos los solicitantes simultáneamente cuando el peso de todos ha llegado a cero.

La principal ventaja de este tipo de árbitro consiste en la posibilidad de controlar el porcentaje de uso de un recurso de acuerdo al peso asignado. La principal desventaja es que cuando un solicitante agotó su peso ya no puede tener acceso al recurso aún y cuando el recurso se encuentre desocupado, lo que impacta directamente en el rendimiento del sistema.

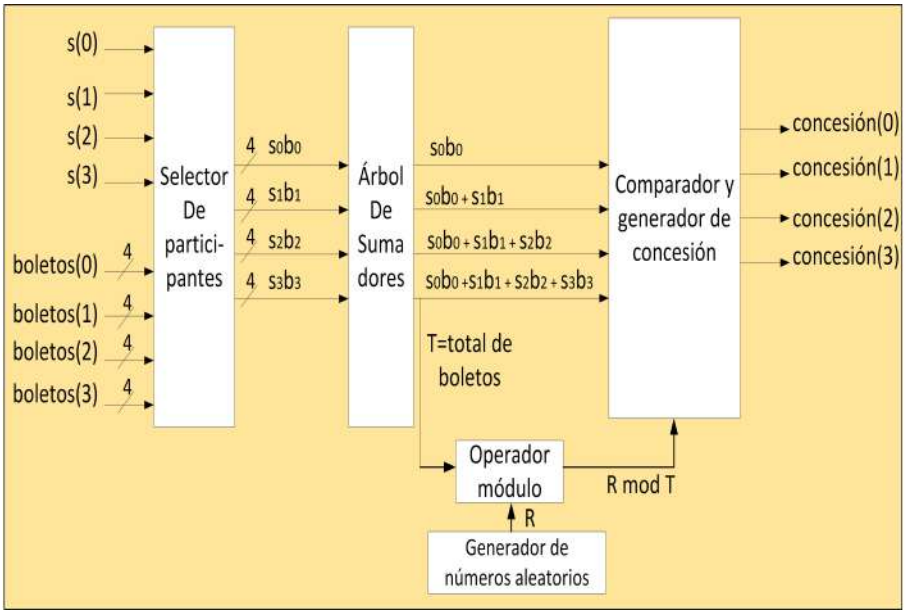


Figura 2.14: Arquitectura de un Árbitro de Lotería Dinámica Lahiri et al. (2006)

Árbitros Compuestos

Existen una serie de propuestas realizadas por diferentes autores donde se combinan diferentes políticas de arbitraje, o se adicionan elementos de control o programación, con el objetivo mejorar las prestaciones que proveen los árbitros de forma individual. A continuación se presentan una serie de trabajos propuestos divididos de acuerdo a la medida de QoS que pretenden cumplir.

Árbitros Compuestos con enfoque en la medida de latencia

El árbitro denominado *Round-Robin Clasificado* (Classified Round-Robin (CRR), en inglés), es presentado por Rahmati and Sarbazi-Azad (2017). Su trabajo tiene como objetivo proveer QoS proporcionando *Servicio Garantizado* en redes del tipo *Mejor Esfuerzo*. Para lograrlo hacen una diferenciación entre flujos de alta prioridad y flujos de baja prioridad. Los flujos de alta prioridad compiten entre sí en una cola de alta prioridad. Cuando el árbitro ha dado una ronda completa a los *maestros* de alta prioridad, permite que se le dé servicio a un flujo que se encuentra en la cola de baja prioridad. Los flujos de datos son clasificados de alta prioridad si son del tipo *tiempo-real* y de baja prioridad si son del tipo *no tiempo-real*. Los resultados muestran que en escenarios del tipo *peor de los casos*, CRR supera a RR disminuyendo la latencia y aumentando el ancho de banda.

Otra propuesta basada en TDMA se encuentra en Lara et al. (2019). Aquí los

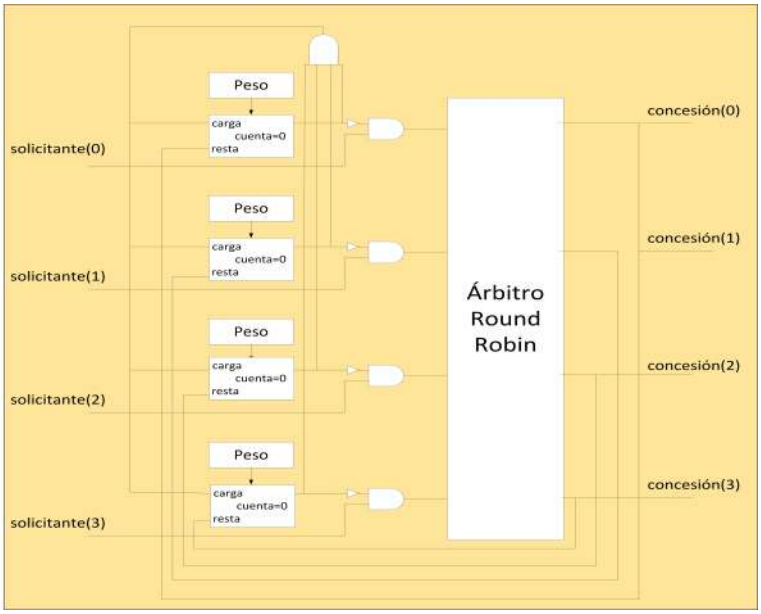


Figura 2.15: Arquitectura de un árbitro WRR Dally and Towles (2001)

autores presentan una arquitectura que tiene como objetivo ejecutar cargas de trabajo críticas, permitiendo en la medida de lo posible, que otros núcleos ejecuten tareas menos críticas. La propuesta está basada en el uso de una infraestructura denominada Verificador de Cumplimiento de Plazos (VCP). El VCP permite que tareas concurrentes que no son críticas se ejecuten mientras el *deadline* de la tarea crítica no sea violado. Si el *deadline* es violado el VCP inhibe todas las tareas que no son críticas. El VCP le indica al *Controlador del Bus* (que trabaja bajo una arquitectura TDMA) el modo en que debe operar que puede ser: TDMA tradicional (compartido); o TDMA máxima prioridad (independiente). Los resultados muestran que tareas críticas pueden ser ejecutadas concurrentemente con tareas que no son críticas mientras el *deadline* no sea violado.

LTY es un esquema de arbitraje que también ha sido utilizado para controlar el *bus*, ya sea para garantizar el uso del ancho de banda o la capacidad de ejecución en tiempo real. Chen et al. (2006) presentan una arquitectura con el objetivo de cumplir con estos dos requerimientos. Para resolver el problema proponen una arquitectura denominada *Lotería en Tiempo Real* (Real-Time lottery (RT_lottery), en inglés). La arquitectura está diseñada en dos niveles: el primero, el *Manejador de Tiempo-Real* (MTR) que tiene como función manejar los requerimientos de tiempo real; el segundo, *Lotería con Ajuste de Pesos* (LAP) que está diseñado para manejar los requerimientos de ancho de banda. MTR asigna un *contador de tiempo real* a cada *maestro* con requerimientos de tiempo real. Este contador es decrementado por uno

en cada ciclo. Cuando supera una *zona de advertencia* previamente establecida, el *maestro* toma la más alta prioridad. Cuando dos o más *maestros* superan la *zona de advertencia* el acceso es otorgado al *maestro* que tenga el valor más pequeño en el *contador de tiempo real*. El ajuste de los pesos para asignar los *boletos* al árbitro LTY es dado por un algoritmo que toma en cuenta la información de tráfico documentada por los diseñadores. El requerimiento de ancho de banda de cada *maestro* debe de ser menor que el máximo ancho de banda que éste pudiera tener cuando trabaja en modo *independiente*. El algoritmo intenta mover el ancho de banda de aquellos *maestros* que tienen una mayor cantidad del ancho de banda asignado a aquellos que tienen una menor cantidad del ancho de banda asignado. Los resultados muestran que con respecto a la distribución de ancho de banda, RT_lottery lo distribuye correctamente.

Lin et al. (2007) proponen un algoritmo de arbitraje denominado *Lotería de Ancho de Banda y Tiempo Real* (Real-Time Bandwidth Lottery (RB_Lottery), en inglés) que tiene como objetivo garantizar los requerimientos de tiempo real al mismo tiempo que asegura un control preciso de la asignación del ancho de banda. Basados en la propuesta hecha en Chen et al. (2006), los autores integran el denominado *Regulador de Ancho de Banda* (RAB) el cual supervisa el comportamiento del *bus* y provee control sobre la asignación de ancho de banda, de acuerdo al comportamiento dinámico del *bus*. El comportamiento de cada *maestro* es supervisado por un periodo de tiempo denominado *ventana de observación*. El RAB detecta la porción de ancho de banda utilizada por un *maestro* en la *ventana de observación* la cual es acumulada en un registro asociado. Cuando este registro alcanza su cuota, el *maestro* es temporalmente bloqueado hasta que la próxima ventana de observación inicie. Se generaron diferentes patrones de prueba con requerimientos tanto de tiempo real como de asignación de ancho de banda. Los resultados muestran que RB_Lottery tiene un porcentaje de fallo relativamente bajo aún con cargas de trabajo pesadas (cercanas al 85 %).

Con el fin de poder cumplir con las restricciones impuestas por los sistemas de tiempo real dentro de los sistemas embebidos, Jun et al. (2007) presentan un sistema de arbitraje integrando un *programador* y un *árbitro*. Su trabajo se enfoca principalmente en el desarrollo del *programador*, cuya función principal es calcular la holgura de cada solicitud, que se define como el número máximo de ciclos de reloj para completar el servicio sin violar la restricción de latencia. El *programador* necesita la longitud de la ráfaga de cada solicitud, así como la información de restricción de latencia para el cálculo de la holgura. El *programador* participa en el arbitraje solo cuando observa que una solicitud no tiene la holgura suficiente para cumplir con la restricción de tiempo dada. Los resultados muestran que en términos de latencia, con la arquitectura propuesta, la mayoría de los *maestros* alcanzan su cuota de tiempo y los que no la alcanzan, no lo logran por poco. Con esto superan a otras alternativas como Round-Robin y Prioridad Fija. En términos de los requerimientos

de ancho de banda el sistema no se ve degradado. Supera a un árbitro convencional Round-Robin en más del 100 %.

Un conjunto de propuestas que tienen como objetivo mejorar el rendimiento para cumplir con las restricciones de QoS, están basadas en combinar diferentes tipos de arbitraje. En Hwang et al. (2010) los autores presentan el denominado *Árbitro Auto Motivado* (Self-Motivated (SM), en inglés) el cual es un árbitro que puede trabajar en 9 esquemas diferentes de arbitraje basado en el nivel de prioridad y la longitud de la transferencia. El sistema tiene la capacidad de ajustar la unidad de datos procesados, cambiar la política de arbitraje en *tiempo de ejecución* y ajustar el esquema de arbitraje. Los esquemas de arbitraje que puede utilizar son: Prioridad Fija; Round-Robin; y Prioridad Variable. Los esquemas que puede utilizar son: basado en transacciones; basado en longitud; y basado en transacciones con prioridad fija. Sus resultados muestran una mejora entre el 14 % y 64 % en el rendimiento comparado con esquemas como Prioridad Fija; Round-Robin; y Prioridad Variable.

Peng and Lin (2010) presentan un arquitectura que tiene como objetivo garantizar la ejecución de aplicaciones en tiempo real al mismo tiempo que soporte los criterios de QoS. Para lograr esto proponen un algoritmo que conmuta dinámicamente entre los modos de *tiempo-real* (TR) y *no-tiempo-real* (NTR). Su objetivo es proveer un mecanismo que permita en la medida de lo posible evitar usar el modo TR, permitiendo que el sistema se enfoque más en cumplir con las restricciones de QoS. Para lograrlo se apoyan en las denominadas *Zonas de Advertencia por Longitud* (ZAL) las cuales sirven como advertencia para que una petición sea servida y lograr cumplir con sus requerimiento de TR. Los autores advierten que para lograr el objetivo de evitar entrar al modo de TR la selección de la ZAL es fundamental, por lo que en su propuesta desarrollan un modelo para hacer esta asignación. Sus resultados muestran que utilizando su técnica la razón de uso del modo TR puede ser reducida hasta un 27 % cuando es aplicada a controladores DRAM comerciales. Las pruebas realizadas con el decodificador de vídeo QFHD H.263 indican una reducción promedio del tiempo de decodificación de 10.4 %.

Nguyen and Tran (2018) muestran la arquitectura de un *ruteador* híbrido reconfigurable que combina el esquema de *conmutación* con un mecanismo de arbitraje con prioridades. El objetivo es soportar *Servicio Garantizado* (SG) y *Mejor Esfuerzo* (ME) sin la necesidad de tener un mecanismo de reservación. Lo anterior lo logran identificando el tipo de paquete que llega al *conmutador*. En el *flit* de cabecera se definen dos campos: uno, el denominado SG/ME; dos, el campo de *Prioridad*. El árbitro soporta dos modos de operación: uno, Round-Robin que provee QoS del tipo ME; dos, *Prioridad* que ofrece QoS del tipo SG. A los paquetes tipo SG se les asigna una prioridad que sirve para programar su atención. Si un paquete es etiquetado como ME entra en el modo Round-Robin el cual asegura que todos los *Canales Virtuales* (CV) puedan transmitir sus datos, aunque no garantiza el *rendimiento*.

Una vez garantizado un canal el CV lo ocupa hasta que éste es transmitido en su totalidad. Si un paquete es etiquetado como SG, a cada paquete se le asigna una prioridad que determina su acceso al CV en el cual el paquete es almacenado. La propuesta comparada con un *router* genérico muestra tener un rendimiento mayor mientras su latencia promedio disminuye.

Árbitros Compuestos con enfoque en la medida de utilización

Richardson et al. (2006) presentan una arquitectura de *bus* híbrida denominada *TDMA-dinámico* (dynamic-TDMA (dTDMA), en inglés). Su objetivo es generar un sistema que varíe la cantidad de *ranuras-de-tiempo* en un TDMA tomando solo en cuenta a los EP que requieren transmitir en una ventana de tiempo. Con esto en mente, solo son generadas las *ranuras-de-tiempo* justas para el número de solicitantes del *bus*. Cuando un EP no necesita más del medio, el sistema elimina su *ranura-de-tiempo* asignada y reorganiza la cantidad de espacio entre los EP que quedan activos. Esta técnica permite que el *bus* tenga un porcentaje de utilización mayor. Sin embargo, la distribución del ancho de banda queda sujeta a la razón con las que las peticiones son hechas al sistema.

Por su parte Burgio et al. (2010) proponen un esquema que trabaja en base a lo que ellos denominan *Programación Elástica* (Elastic Scheduling (ES), en inglés) y un *bus* con política de arbitraje TDMA adaptable. Su objetivo es que ES y TDMA trabajen juntos para asegurar la máxima utilización de los procesadores incluso en el caso de variaciones dinámicas en las cargas de trabajo en *tiempo de ejecución*. El algoritmo que asigna el ancho de banda del *bus* es implementado en *software*. Si un núcleo requiere incrementar su ancho de banda en tiempo de ejecución, cambia su nivel de servicio. El nivel de servicio es proporcional al número de *ranuras* otorgadas en TDMA. Esta técnica permite que el *bus* tenga un porcentaje de utilización mayor. Al igual que dTDMA, la distribución del ancho de banda queda sujeta a la razón con las que las peticiones son hechas al sistema.

En el trabajo desarrollado por Shah et al. (2011) los autores proponen un arbitraje de *bus* para memorias compartidas denominado *División de Prioridades* (Priority Division (PD), en inglés). Su trabajo tiene como objetivo lograr un elevado uso del *bus* al mismo tiempo que se garantiza un ancho de banda. Lo anterior lo logran combinando una política de arbitraje basada en prioridades fijas con TDMA. A diferencia de TDMA en lugar de declarar a un *maestro* como el propietario de una *ranura*, cada *maestro* tiene una prioridad para usar una *ranura*. Para garantizar el ancho de banda a todos los *maestros*, cada uno de ellos debe de tener la máxima prioridad en al menos una *ranura*. El *maestro* con mayor prioridad cuando requiere la *ranura* la toma y sólo la presta cuando está inactivo. Los resultados muestran que PD distribuye el ancho de banda de manera justa y es menos sensible a variaciones

en los patrones de tráfico, al mismo tiempo que proporciona una alta utilización del *bus*. Comparado con RR tiene una menor latencia y un mayor ancho de banda.

Un árbitro tipo LTY con asignación dinámica de *boletos* es presentado por Xu et al. (2014). Este árbitro es utilizado en los *ruteadores* de una NoC con topología *malla*. Su objetivo es poder mejorar el rendimiento cuando se tiene una comunicación desbalanceada entre los múltiples núcleos en la red. Dado que resulta difícil la asignación previa de pesos que sean consistentes con el tráfico que en un momento dado hay en el *ruteador*, se propone la asignación de pesos en *tiempo de ejecución*, por medio del denominado *Asignador Dinámico* (AD). El AD ajusta los pesos de cada dirección en razón de la congestión actual en los puertos. Cuando el AD detecta que una dirección está congestionada su peso es incrementado. Cuando la congestión es eliminada el árbitro recupera los pesos originalmente asignados a cada puerto. El árbitro es evaluado en términos de la latencia promedio, la cual es menor en un 27.7 % con respecto a un árbitro con política de Prioridad Fija y un 18.7 % con respecto a un árbitro con política Round-Robin.

Un árbitro con prioridad dinámica es presentado por Wang et al. (2009). La propuesta busca mejorar el rendimiento de una NoC tanto cuando el tráfico en la red es uniforme como cuando no es uniforme. El árbitro está compuesto por un *Registro* y un *Generador de Lotería* (GL) asociado a cada puerto de entrada en el *ruteador*. El *Registro* almacena el número de paquetes correspondientes al puerto, el cual es actualizado en cada ciclo de reloj. El GL genera la cantidad de *boletos* a utilizar por cada entrada en el árbitro de *Lotería*. La generación de *boletos* se efectúa dinámicamente en base al valor del *Registro*. La propuesta es comparada con Round-Robin, obteniéndose que si la comparación se efectúa inyectando tráfico uniforme (paquetes con características similares), el tiempo promedio de retardo es muy similar. El tamaño de paquete y el tamaño del *buffer* no impactan en la relación. Cuando la comparación se realiza utilizando tráfico que no es uniforme (paquetes con características diferentes), el rendimiento de la propuesta supera a Round-Robin. Además, los autores identifican que el esquema propuesto, reduce los requerimientos de almacenamiento en las IR del SoC cuando el tráfico no es uniforme.

Por otro lado Amin and Abdullah (2017) proponen un esquema de arbitraje denominado *Lotería Basada en la Edad* (Age-Based Lottery (ABL), en inglés). Este esquema combina el algoritmo de *Lotería* con el algoritmo *Basado-en-Edad* con el objetivo de mejorar el rendimiento del sistema. Lo anterior lo logran maximizando la utilización compartida del *bus*, además balancean la distribución del mismo con lo que obtienen una latencia aceptable. Al esquema tradicional de arbitraje de *Lotería* se le agrega el denominado *Generador Basado en Edad*. *ABL* asigna un valor máximo de *boletos* al *maestro* que ha ganado recientemente el *bus*. Cada *maestro* tiene un valor de *boletos* que varía entre uno y *máxima-edad* el cual es un parámetro fijo. Una

bandera es utilizada en cada *maestro* para balancear el valor de los *boletos*. Cuando un maestro llega a *máxima-edad* esta bandera es activada.

Árbitros Compuestos con enfoque en la medida de igualdad de servicio

En el trabajo que presentan Park and Choi (2015) se muestra una arquitectura denominada *Round-Robin con Ponderación Adaptativa* (Adaptively Weighted Round-Robin (AWRR), en inglés). Su objetivo es lograr *Igualdad de Servicio* en una NoC con múltiples núcleos. Ellos intentan distribuir uniformemente el tráfico entre los nodos de la red para balancear la carga. La idea es alcanzar justicia global en términos del servicio provisto cuando existe tráfico del tipo *hot-spot*. Para lograrlo en cada *ruteador* se implementa un sistema de arbitraje WRR, donde los pesos son dinámicamente adaptados por medio de paquetes de control que son dirigidos a cada *ruteador* indicándole el ajuste que se tiene que realizar en ese puerto, ya sea incrementar el peso cuando más tareas están utilizando ese puerto o decrementar el peso cuando una tarea concluye y deja de utilizar el puerto. Sus resultados muestran que AWRR presenta una distribución de tráfico más uniforme que WRR.

Una técnica denominada *Arbitraje Basado en la Edad* (Age-Based Arbitration (ABA), en inglés), que toma en cuenta la edad de un paquete en la red para proveer *Igualdad de Servicio* es presentada por Lee et al. (2010). La edad es determinada por la *cantidad de saltos* que ha dado un paquete en la red. Los autores proponen un esquema que combina el arbitraje probabilístico con una asignación de pesos basados en la distancia. Sus resultados muestran que cuando existe tráfico uniforme sobre un *hot-spot* el tráfico se distribuye equitativamente superando a técnicas como Round-Robin y Arbitraje Probabilístico con pesos lineales. Además se obtienen resultados muy similares a Arbitraje Probabilístico con pesos no lineales.

Árbitros Compuestos con enfoque en la medida de ancho de banda

Por su parte Yang et al. (2015) presentan un árbitro asíncrono denominado *Round-Robin Asíncrono de Prioridad Adaptativa* (Asynchronous Adaptive Priority Round-Robin (AAPRR), in inglés), basado en el protocolo *four-phase dual-rail*. Su objetivo es proveer el ancho de banda proporcionalmente requerido por los *maestros* en el sistema. Además del árbitro RR, el sistema cuenta con una unidad denominada *Módulo de Prioridad*. Esta unidad asigna a cada *maestro* un peso correspondiente a las solicitudes teóricas que necesita para cumplir con la asignación de ancho de banda. Cuando un *maestro* obtiene el control del *bus*, su peso es decrementado en uno. Si este valor llega a cero, indica que ha cumplido la cuota necesaria para alcanzar su requerimiento de ancho de banda, y ya no entra más a contender por obtener el control del *bus*. A partir de este momento el acceso solo es permitido a aquellos contendientes que no han cumplido su cuota. Este árbitro, comparado con

otros árbitros que utilizan políticas de arbitraje como: Prioridad Fija, Round-Robin y Lottery, puede distribuir más adecuadamente el ancho de banda de acuerdo a los pesos asignados. La distribución del ancho de banda es independientemente del tráfico en la red y de la razón de peticiones de los *maestros*. Sin embargo, al igual que otras políticas, los mejores resultados solo los presenta cuando los patrones de tráfico son muy agresivos, es decir, cuando el sistema está saturado.

Un esquema de reservación de ancho de banda que trabaja sobre el estándar AMBA-AXI es presentado en Pagani et al. (2019). El objetivo de los autores es proveer un mecanismo para que los *aceleradores de hardware* puedan ajustar sus requerimientos de ancho de banda. Esto se lleva a cabo por medio de la denominada *Unidad de Presupuesto AXI (UPA)*. Esta unidad implementa un mecanismo de reservación de ancho de banda, dando seguimiento al número de transacciones emitidas por los *maestros*, ajustándose al presupuesto máximo para cada transacción dentro de ventanas de tiempo periódicas. Cada *maestro* tiene una unidad UPA y existe a nivel global el denominado *Controlador UPA*. A cada UPA se le asigna un presupuesto máximo correspondiente al número total de transacciones que puede realizar. Además, se le asigna un periodo en el cual el presupuesto es recargado. Cuando un *maestro* efectúa una transacción su presupuesto es decrementado hasta que es cero. Mientras la UPA no tenga presupuesto no realiza transacciones en el *bus*. El presupuesto es recargado periódicamente de forma síncrona.

Un árbitro que implementa la compartición justa del ancho de banda basado en créditos es presentado por Slijepcevic et al. (2017). En su estudio, los autores abordan el problema que existe para asignar el ancho de banda proporcionalmente cuando un elemento recibe peticiones con transacciones de diferente tamaño. Los autores proponen un esquema de arbitraje denominado *Arbitraje Basado en Créditos* (Credit-Based Arbitration (CBA), en inglés) el cual tiene como objetivo que los EP tengan un servicio justo en términos de número de ciclos de reloj en los que tienen garantizado el acceso al recurso. Para lograrlo, asignan un crédito máximo que cada contendiente puede tener. Este crédito es inicializado en 0 y es incrementado proporcionalmente en cada ciclo de reloj de acuerdo a la relación de ancho de banda que cada núcleo desea obtener. Sólo podrán contender los núcleos que hayan alcanzado el crédito máximo. Cuando un núcleo obtiene el acceso su crédito es decrementado igualmente de forma proporcional. Al igual que otras implementaciones los autores indican el hecho de que los mejores resultados son alcanzados cuando el sistema se encuentra saturado.

Con el objetivo de proveer una asignación proporcional del ancho de banda, Li et al. (2007) presentan un algoritmo de arbitraje con prioridades adaptables dinámicamente. El árbitro registra la cantidad de solicitudes que genera un *maestro* en particular y la totalidad de solicitudes en el sistema. El peso asignado a cada *maestro* está dado por su cantidad de solicitudes entre la totalidad de solicitudes

en el sistema. Este peso es calculado periódicamente y vuelto a reasignar a cada *maestro*. En la propuesta, los autores presentan un esquema con dos niveles de prioridad: el primer nivel, adopta el algoritmo de arbitraje adaptable; el segundo nivel, adopta un esquema de prioridad fija con el objetivo de prevenir *starvation*. Los resultados muestran que cuando el ancho de banda a distribuir es igual en los *maestros*, Round-Robin tiene una mejor distribución. La propuesta se acerca más a la distribución adecuada cuando el ancho de banda a distribuir es proporcional.

Akesson et al. (2009), presentan un sistema denominado *Arbitraje de Prioridad Estática Controlado por el Crédito* (Credit-Controller Static-Priority Arbitration (CCSPA), en inglés) que tiene como objetivo programar y regular el acceso a los recursos compartidos de un SoC, evitando la sobre asignación a *maestros* que generan una gran cantidad de solicitudes. El árbitro consta de un regulador, que impone restricciones en la rapidez del servicio, y de un planificador de prioridad estática. El regulador proporciona contabilidad y cumplimiento, y determina qué solicitudes son elegibles para su programación en un momento particular, considerando la cantidad de servicios que se le han asignado. El servicio asignado a un solicitante consta de dos parámetros: uno, la ráfaga asignada y dos, la tasa de servicio asignada. Tres restricciones se deben de cumplir para que una configuración sea válida: 1) la tasa de servicio asignada debe ser al menos igual a la tasa de solicitud promedio, 2) no es posible asignar más servicio a los solicitantes que lo que ofrece el recurso, y 3) la ráfaga asignada debe ser lo suficientemente grande como para acomodar una unidad de servicio. En las pruebas llevadas a cabo se evaluaron las estrategias de programación *Tasa de Aproximación más Cercana* (TAC) y *Aproximación por Ráfaga* (AR), experimentalmente se demostró que TAC reduce la tasa de sobre asignación promedio en un factor de tres a uno, con un costo del 25 % más de generación de ráfagas sobre AR.

2.5. Principios de Redes Definidas por Software

SDN es un paradigma de gestión de redes de comunicación de datos que integra: protocolos, métodos y procedimientos. Nace para dar una respuesta a la cada vez más compleja administración de las redes dada la amplia adopción y diseminación de las mismas. Un estudio muy extenso sobre los principales tópicos de este paradigma de administración de redes es presentado por Kreutz et al. (2015). De aquí se muestran los conceptos principales que posteriormente serán utilizados en el contexto de Sistemas de Interconexión Definidos por Software.

2.5.1. Introducción a Redes Definidas por Software

En las redes de comunicación de datos tradicionales, los administradores necesitan configurar cada red individualmente utilizando comandos de bajo nivel y en ocasiones comandos específicos del proveedor. Además las redes están verticalmente integradas. El Plano de Control (PC) — el que decide como manejar el tráfico de la red — y el Plano de Datos (PD) — que retransmite los datos de acuerdo a las decisiones tomadas por el PC — se encuentran integrados en el interior de los dispositivos de red. SDN separa el PC del PD. Con esta separación los *conmutadores* en la red se convierten en simples dispositivos de transporte. La lógica de control es implementada en un controlador centralizado. La separación entre los planos se logra por medio de APIs.

SDN se concibe como una arquitectura de red apoyada en cuatro pilares.

1. El PC y el PD están desacoplados. La funcionalidad de control es eliminada de los dispositivos de red por lo que la nueva función de estos es sólo de retransmisión de paquetes.

2. Las decisiones de retransmisión de datos están basadas en flujos en lugar de estar basadas en el destino. En SDN un flujo se define como una secuencia de paquetes entre una fuente y un destino. Todos los paquetes de un flujo actúan bajo las mismas políticas en los dispositivos de retransmisión, unificando el comportamiento de los diferentes tipos de dispositivos de red. La programación de flujos permite una gran flexibilidad, estando solamente limitada por la capacidad de las *Tablas de Flujo* en el interior de los *conmutadores*.

3. La lógica de control es movida a una entidad externa denominada Controlador-SDN (C-SDN) o Sistema Operativo de Red (SOR). El SOR es una plataforma de *software* corriendo sobre un servidor y provee los recursos esenciales para facilitar la programación de los dispositivos de retransmisión. Proporciona, desde un punto de vista centralizado, una abstracción de la red.

4. La red es programable por medio de aplicaciones de *software* ejecutándose sobre el SOR, el cual interactúa con los dispositivos del plano de datos abajo de él. Esta es la característica principal de SDN, considerada como su principal propuesta de valor.

En la Figura 2.16, se muestra la arquitectura básica de una red de comunicación de datos con enfoque *Definido por Software.*

Al tener en SDN una lógica de control centralizada , se obtienen un conjunto de beneficios adicionales:

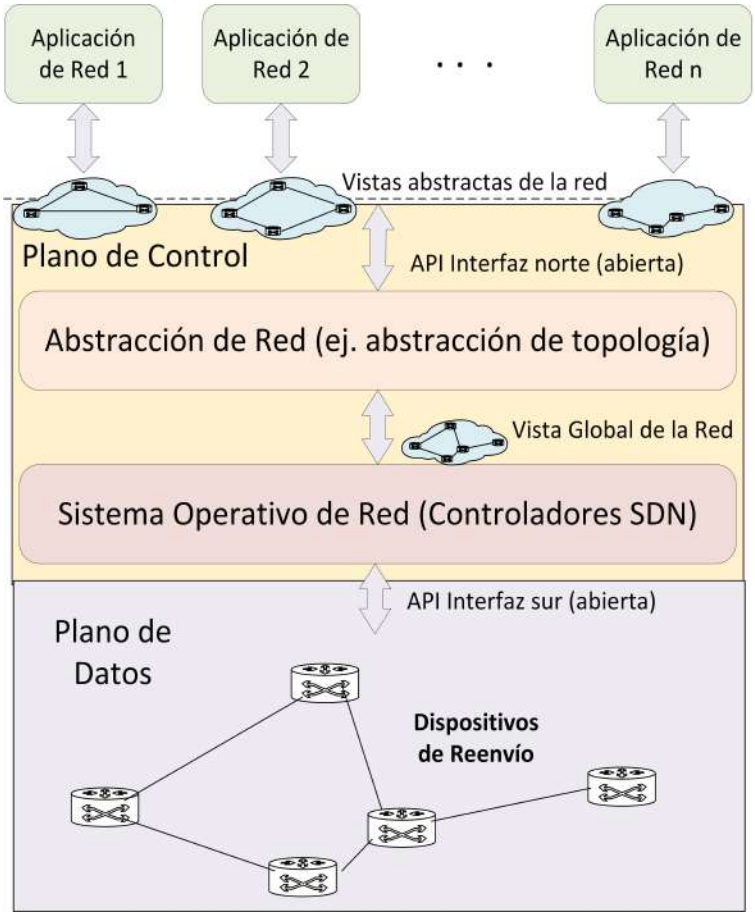


Figura 2.16: Arquitectura de Redes Definidas por Software y sus abstracciones fundamentales Kreutz et al. (2015)

1. Al poder utilizar *lenguajes de alto nivel* y *componentes de software*, resulta más sencillo y menos propenso a errores modificar las políticas de administración de la red.

2. Un programa de control puede reaccionar automáticamente a alteraciones en la red manteniendo las políticas de alto nivel intactas.

3. Tener el conocimiento global de la red simplifica el desarrollo de funciones, servicios y aplicaciones más sofisticadas.

2.5.2. Redes Convencionales Vs Redes Definidas por Software

El fuerte acoplamiento que existe entre los planos de control y de retransmisión de datos en las redes tradicionales hace que sea muy difícil agregar nuevas funcionalidades. En la Figura 2.17a se puede observar que los planos de control y de retransmisión de datos se encuentran físicamente embebidos en los elementos de la red. Implementar nuevas funcionalidades requeriría no solo la re/programación de los elementos de red, sino la posibilidad de realizar actualizaciones en el *firmware* de éstos, o inclusive la instalación de *hardware* nuevo. La introducción de nuevas facilidades en la red se realiza comúnmente introduciendo las denominadas *middleboxes* que normalmente son caras, especializadas y difíciles de configurar. Dentro de este tipo de elementos se pueden mencionar: balanceadores de carga, sistemas de detección de intrusiones, cortafuegos, etc. Estas *middleboxes* necesitan ser puestas estratégicamente en la red, haciendo difícil el poder realizar cambios posteriores en la topología, configuración y funcionalidad de la red.

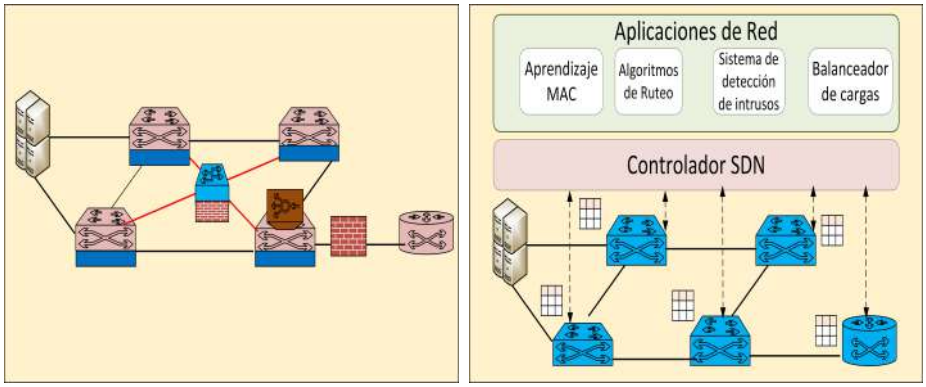
En contraste, SDN Figura 2.17b, desacopla el PC de los dispositivos de la red enviándolos a una entidad externa: el SOR o el C-SDN. Lo cual tiene varias ventajas:

- Resulta más sencillo por medio de lenguajes de programación, configurar y compartir las abstracciones de la red provistas por el PC.
- Todas las aplicaciones pueden tomar ventaja de la misma información de la red –la vista global de la red–, permitiendo el desarrollo de políticas más consistentes y efectivas.
- Estas aplicaciones pueden tomar acciones (por ejemplo, reconfigurar los dispositivos de retransmisión) desde cualquier lugar de la red. No es necesario generar una estrategia específica sobre la ubicación de la nueva funcionalidad.
- La integración de nuevas aplicaciones se vuelve más sencilla. Por ejemplo las aplicaciones de *balanceo de cargas* y *ruteo* pueden ser combinadas de acuerdo a las necesidades superiores de equilibrio de carga y políticas de *enrutamiento*.

2.5.3. Terminología de Redes Definidas por Software

Para identificar de manera precisa los elementos que componen la arquitectura SDN se define la siguiente terminología:

1. **Dispositivos de Reenvío (DR)**: son dispositivos de *hardware* o *software*, pertenecientes al plano de datos, que realizan un conjunto de operaciones



(a) Red Convencional (b) Red Definida por Software

Figura 2.17: Arquitectura Red Convencional Vs Red Definida por Software Kreutz et al. (2015)

elementales. Estos dispositivos tienen un conjunto de instrucciones bien definidas (por ejemplo, reglas de flujo) utilizadas para tomar una acción sobre los paquetes entrantes (por ejemplo, reenviar paquetes a un puerto específico, eliminar paquetes, reenviar al controlador, reescribir algún encabezado). Estas instrucciones son definidas por la Interfaz Sur, y son instaladas en los dispositivos de retransmisión por el C-SDN que implementa los protocolos *dirección sur*. Los principales protocolos existentes que son utilizados para realizar estas tareas son: OpenFlow, McKeown et al. (2008); ForCES, Doria et al. (2010); Protocol-Oblivious Forwarding, Song (2013).

2. **Plano de Datos (PD)**: los dispositivos de retransmisión están interconectados ya sea por medio de cables o de señales de radio inalámbricas. La infraestructura de red comprende los dispositivos de retransmisión interconectados, que representan el PD.

3. **Interfaz Sur (IS)**: el conjunto de instrucciones que van a seguir los dispositivos de retransmisión es definida por la API *dirección sur* la cual es parte de la IS. Además la IS también define los protocolos de comunicación entre los elementos del plano de control y los dispositivos de retransmisión. Formaliza la forma en que los elementos del plano de control y el de datos interactúan.

4. **Plano de Control (PC)**:los dispositivos de retransmisión son programados por medio de los elementos del PC que puede ser visto como el *cerebro de la red*. Toda la lógica de la red se encuentra en las aplicaciones y controladores pertenecientes a este plano.

5. **Interfaz Norte (IN)**: el SOR ofrece una API a los desarrolladores de aplicaciones. Esta API representa la IN. Típicamente esta interfaz abstrae las instrucciones de bajo nivel utilizadas en la IS para programar los dispositivos de retransmisión.

6. **Plano de Administración (PA)**: es el conjunto de aplicaciones que aprovechan las funciones ofrecidas por la IN para implementar la lógica de control y operación de la red. Incluye aplicaciones de enrutamiento, cortafuegos, balanceo de carga, supervisión, etc. Esencialmente, una aplicación de gestión define las políticas que finalmente se traducen en instrucciones específicas que serán enviadas a la IS para que se programen los dispositivos de retransmisión.

2.5.4. Arquitectura de Redes Definidas por Software

La arquitectura SDN puede ser representada en planos y capas con un conjunto específico de funciones cada una. En la Figura 2.18 se muestra su arquitectura.

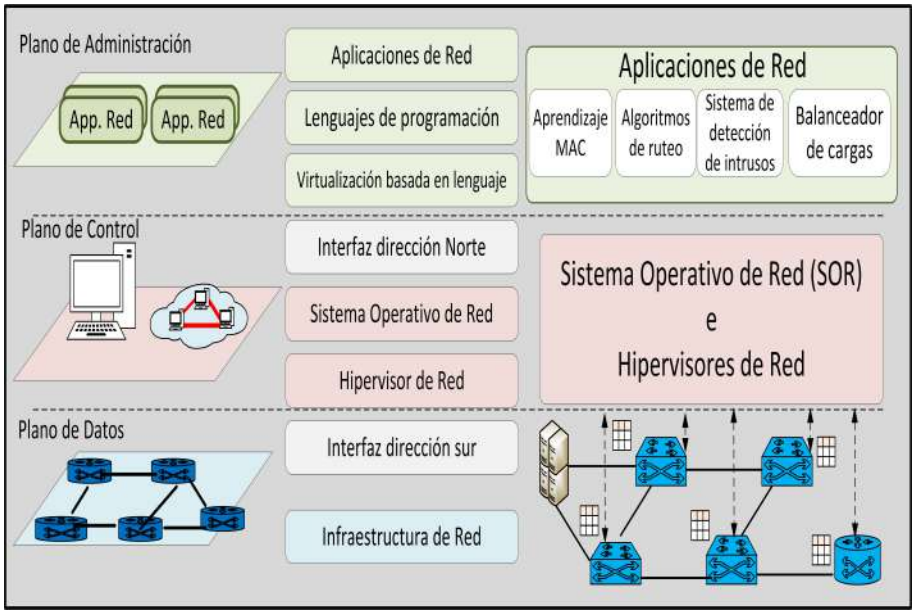


Figura 2.18: Planos y Niveles de la Arquitectura de Redes Definidas por Software Kreutz et al. (2015)

Capa 1: Infraestructura de Red

La infraestructura SDN está compuesta de un conjunto de elementos de red, principalmente *conmutadores*, *ruteadores* y *middleboxes*. La característica principal

reside en el hecho de que los dispositivos no contienen lógica de control embebida que les permita tomar decisiones autónomas. La inteligencia de la red es removida del Plano de Datos y enviada al Plano de Control. Las redes son construidas conceptualmente por medio de interfaces estándar como *OpenFlow*. Un dispositivo de retransmisión que esté habilitado para trabajar bajo el estándar *OpenFlow*, basa su operación en un conjunto de *tablas de flujo* donde cada entrada a la tabla está compuesta de tres partes: primera, una regla de coincidencia; segunda, las acciones que van a ser ejecutadas para los paquetes que cumplan con la regla de coincidencia; tercera, un conjunto de contadores que mantienen la estadística de los paquetes que cumplen con las reglas. En la Figura 2.19 se puede observar la estructura de un dispositivo habilitado para trabajar en un ambiente *RDS/OpenFlow*. La forma en que deben de ser manejados los paquetes al interior de un dispositivo *OpenFlow* es regido por el camino que tienen que seguir estos a través de las *tablas de flujo*. Cuando un nuevo paquete llega al dispositivo, el proceso de búsqueda se inicia en la primera tabla y termina cuando encuentra una coincidencia en una de las tablas del *pipeline* o en su caso cuando no encuentra ninguna coincidencia. Una regla se define por la combinación de la coincidencia de diferentes campos. Si no hay una regla por defecto el paquete puede ser descartado, aunque lo más común es que se genere una regla por defecto, que envíe el paquete que no tiene coincidencia al *controlador*. La prioridad de las reglas sigue el número natural en las tablas. Dentro de las posibles acciones que pueden ser tomadas se encuentran: 1) reenviar el paquete al puerto de salida; 2) encapsular el paquete y enviarlo al controlador; 3) desechar el paquete; 4) mandar el paquete al *pipeline* de procesamiento; 5) mandar el paquete a la siguiente *tabla de flujo* o a alguna *tabla especial*.

Capa 2: Interfaz Sur

Es el puente que realiza la conexión entre los elementos de control y los de retransmisión de datos. Son los instrumentos clave para separar la funcionalidad de los planos. Comúnmente las APIs Interfaz Sur representan una de las mayores barreras para la introducción y aceptación de una nueva tecnología de red. Con la llegada de propuestas de APIs SDN Interfaz Sur la aceptación de las nuevas tecnologías es más sencillo. Entre otras cosas porque estos nuevos estándares han demostrado que existe interoperabilidad en los equipos de diferentes proveedores.

Capa 3: Hypervisor de Red

Actualmente la *virtualización* es una tecnología consolidada. Hoy en día la cantidad de *servidores virtuales* superan el número de *servidores físicos*.

Los *Hypervisores* habilitan a distintas *máquinas virtuales* a compartir los mismos recursos *hardware*. Se habilita proveer a la infraestructura como un servicio,

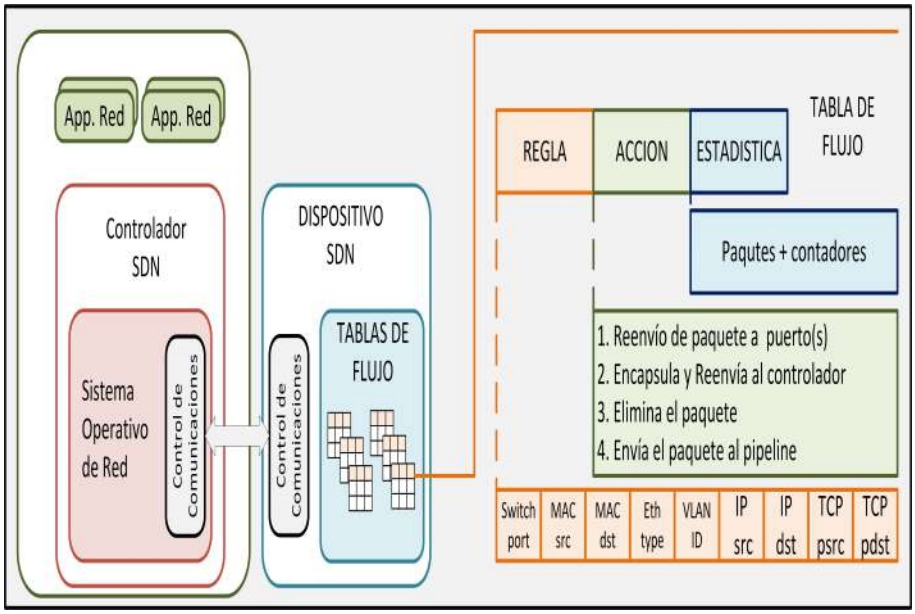


Figura 2.19: Proceso de comunicación entre un Controlador SDN y un Dispositivo SDN con OpenFlow Kreutz et al. (2015)

donde cada usuario puede tener sus propios recursos virtuales, desde cómputo hasta almacenamiento. Esto permite crear nuevos modelos de negocio, donde los recursos son asignados bajo demanda, a un costo relativamente bajo a los usuarios, utilizando una infraestructura física compartida. Al mismo tiempo los proveedores de los servicios hacen un mejor uso de la capacidad de la infraestructura instalada, generando mayores beneficios sin incrementar sus gastos. Para proveer una completa virtualización la red debe de proveer propiedades similares en la capa de cómputo. La infraestructura de red debe de estar habilitada para soportar topologías de red y esquemas de direccionamiento arbitrarios. Cada inquilino de la red debe de tener la habilidad de configurar tanto los nodos de cómputo como la red.

Capa 4: Sistema Operativo de Red

Los sistemas operativos tradicionales proveen abstracciones para acceder a los dispositivos en niveles inferiores, administrar el acceso concurrente a los recursos compartidos y proporcionar mecanismos de protección. Estas funcionalidades y recursos son factores clave para incrementar la productividad. El uso extendido de estos ha contribuido a la evolución de varios ecosistemas y al desarrollo de incontables aplicaciones. En contraste las redes han sido administradas y configuradas utilizando instrucciones de bajo nivel, específicas para cada dispositivo y muy dependientes del proveedor.

SDN promete facilitar la gestión de la red y aliviar la carga de resolver problemas, mediante un control centralizado ofrecido por un SOR. Al igual que los sistemas operativos tradicionales, el valor principal de un SOR es proporcionar abstracciones, servicios esenciales y APIs comunes a los desarrolladores. Además de las anteriores el SOR puede proporcionar funcionalidades genéricas como: conocer el estado de la red, conocer la topologías de la red, descubrir nuevos dispositivos en la red y distribuir la configuración de la red. Por ejemplo, con el SOR no es necesario conocer los detalles de bajo nivel como la distribución de los datos entre los elementos de *ruteo*, para definir las políticas de la red. Se puede decir que estos sistemas pueden crear un nuevo entorno capaz de fomentar la innovación a un ritmo más rápido al reducir la complejidad inherente de crear nuevos protocolos y aplicaciones de red.

Un SOR o un *controlador de red*, es un elemento crítico en la arquitectura SDN ya que es la pieza de soporte clave para la lógica de control que genera la configuración de la red en función de las políticas definidas por el operador de la red. Similar a un sistema operativo tradicional, la plataforma de control abstrae los detalles del nivel inferior para interconectarse e interactuar con los dispositivos de retransmisión. Es decir, materializar las políticas de la red. La plataforma de control debe de proveer servicios como: administración de dispositivos, gestión de estadística, administración de topologías, mecanismos de seguridad, administración de notificaciones, retransmisión por ruta más corta.

Capa 5: Interfaz Norte

La Interfaz Norte es un ecosistema de *software* que debe de permitir que capas superiores en la arquitectura no dependan de implementaciones específicas. Una API Interfaz Norte puede ser comparada con el estándar *POSIX* IEEE (2001) en los sistemas operativos. Éste representa una abstracción que garantiza independencia de los lenguajes de programación y los controladores. Un ejemplo de esta interfaz es *NOXIS* Yu et al. (2014), el cual define interfaces de aplicación de bajo nivel portables, haciendo que las APIs Interfaz Sur se vean como *Manejadores de Dispositivos*.

Capa 6: Virtualización Basada en Lenguaje

Dos características esenciales de las soluciones de virtualización son la capacidad de expresar modularidad y la de permitir diferentes niveles de abstracciones mientras se garantizan las propiedades deseadas, como la protección. Por ejemplo, las técnicas de virtualización pueden permitir diferentes vistas de una infraestructura física. Como ejemplo, un *conmutador virtual* de gran tamaño podría representar una combinación de varios dispositivos pequeños subyacentes. Esto simplifica intrínsecamente la tarea de los desarrolladores de aplicaciones, ya que no necesitan pensar sobre la secuencia de *conmutadores* donde las reglas de reenvío deben instalarse.

Este tipo de abstracción simplifica significativamente el desarrollo y la implementación de aplicaciones de red complejas, como servicios avanzados relacionados con la seguridad.

Capa 7: Lenguajes de Programación

El desarrollo de lenguajes de programación con filosofía SDN tiene como objetivo lograr un nivel de abstracción mayor. Los lenguajes de programación de alto nivel pueden ser herramientas poderosas como medio para implementar y proporcionar abstracciones para diferentes propiedades y funciones importantes de SDN, como estructuras de toda la red, actualizaciones distribuidas, composición modular, virtualización y verificación formal.

Capa 8: Aplicaciones de Red

Implementan la lógica de control que se traducirá en comandos que se instalarán en el plano de datos, dictando el comportamiento de los dispositivos de reenvío. Tomando como ejemplo una aplicación de enrutamiento, la lógica de esta aplicación es definir la ruta a través de la cual fluirán los paquetes desde el punto A al punto B. Para lograr este objetivo, una aplicación de enrutamiento debe, en función de la topología de entrada, decidir la ruta a utilizar e indicar al controlador que instale las reglas de reenvío respectivas en todos los dispositivos de reenvío en la ruta elegida, desde A hasta B.

2.6. Sistemas en Chip con enfoque de Redes Definidas por Software

2.6.1. Antecedentes

La administración de los SoC bajo el enfoque SDN es un área que ha empezado a desarrollarse en la última década. La idea principal detrás de este enfoque es aplicar, los fundamentos de SDN para redes de computadoras en los sistemas de interconexión de los SoC. El objetivo es poder tener sistemas de interconexión flexibles, que se adapten a los requerimientos impuestos por las aplicaciones corriendo en el SoC. Además SoC con SDN busca proporcionar un marco de referencia para poder realizar una integración más sencilla, con independencia de los proveedores y con tiempos de producción menores. SoC con SDN es un paradigma más que una implementación. El paradigma SoC con SDN es relativamente nuevo, se puede encontrar trabajo relacionado con este tema desde aproximadamente el año 2010.

Existen algunas aproximaciones que aunque no indican explícitamente el uso de un enfoque SDN, utilizan algunos de sus principios. Kobbe et al. (2011) presentan un esquema de administración de recursos basado en agentes para sistemas con múltiples núcleos en una NoC. En Singh et al. (2013) se presenta una técnica de mapeo en *tiempo de ejecución* para sistemas con múltiples núcleos en una NoC. Por su parte Kornaros and Pnevmatikatos (2014) muestran un sistema de gestión dinámico, de la energía y temperatura para sistemas de múltiples núcleos basados en NoC. Heisswolf et al. (2014) presentan un sistema invasivo que toma en cuenta el estado de la red para dinámicamente repartir el cómputo entre procesadores vecinos.

El trabajo realizado por Gorski et al. (2015), aunque no tiene el enfoque de SDN, sí permite por medio de *software* la reconfiguración de rutas en *tiempo de ejecución*. La propuesta la implementan a partir de dos elementos: uno, un sistema de supervisión de tráfico que ofrece la posibilidad de reconfiguración; dos, la posibilidad de interconectar múltiples interfaces de red independientes para conectar los recursos de cómputo. Esto posibilita tener diferentes caminos de punta a punta, dependiendo de la carga de la red.

Algunas propuestas que ya han tomado en cuenta explícitamente los conceptos de SDN son expuestas a continuación.

Cong et al. (2014) presentan un trabajo en el que proponen una NoC programable basada en SDN. Con esta programabilidad tienen la capacidad de atender a las aplicaciones de acuerdo a sus necesidades, ya sea asegurar un ancho de banda o asegurar QoS. Por ejemplo, si una aplicación requiere asegurar un ancho de banda, se le puede establecer una ruta exclusiva. Para lograr establecer esta ruta se manda un paquete con destino a todos los *ruteadores* involucrados y por medio de un bit en el paquete de cabecera se establece que el uso de ese canal será restringido solo a paquetes que provengan de esa aplicación. Con esto una parte de la red se encuentra trabajando en modo de conmutación de circuitos, mientras que el resto de la red se mantiene trabajando en modo de conmutación de paquetes. Cuando la aplicación envía el último paquete, libera el circuito y la red vuelve a su operación normal. En su implementación cuentan con un mecanismo en el que los *ruteadores* están al tanto de su capacidad, por lo tanto pueden conocer cuando se encuentran congestionados. Cuando esto sucede, pueden indicárselo directamente al *Controlador de Aplicaciones* para que éste a su vez pueda recalcular y programar una nueva ruta. Una de las ventajas de contar con un sistema de retroalimentación por parte de los *ruteadores*, es que el *Controlador de Aplicaciones*, al no recibir información de algún *ruteador* en particular, puede deducir la falla del mismo y eliminarlo de la topología de la red.

Wang et al. (2014) proponen una red en chip fotónica con enfoque SDN. Definen su arquitectura en base a dos planos: uno, el *plano de control*; dos, el *plano de conmutación*. Cuando el *plano de control* recibe una notificación por parte de

un EP para transmitir información, calcula la ruta y notifica a cada uno de los *ruteadores* fotónicos, distribuidos en una topología de malla, para que activen y/o desactiven sus *conmutadores*. Se establece una conexión basada en circuito para cada flujo que pueda existir en la red. Con esto aseguran transmisiones de alta velocidad para las aplicaciones a las que se les hubiera establecido un flujo. Para establecer una transmisión segura proponen utilizar un protocolo de comunicación orientado a conexión.

Por su parte Scionti et al. (2016) presentan un trabajo que aunque no se basa específicamente en algún marco de referencia SDN, sí trabaja bajo sus principios fundamentales. Ellos presentan una arquitectura que permite combinar diferentes tipos de topologías en una red tipo malla con el objetivo de mejorar el rendimiento del sistema y permitir una mejor escalabilidad. Añaden un conjunto de instrucciones específicas a los EP que están conectados a la red lo cual les permite directamente controlar la topología de la red en *tiempo de ejecución*. El sistema conoce en todo momento el uso que se le está dando a los elementos de la red, por lo que tiene la capacidad de desconectar aquellos *conmutadores* que no están siendo utilizados. Lo anterior le permite administrar el consumo de energía del sistema de interconexión.

En la propuesta que realizaron Ruaro et al. (2020) presentan un marco de referencia para el diseño de sistemas en chip con múltiples núcleos al que denominan *SDN Sistémico y Seguro* (SDN-SS). Su trabajo lo enfocan en dos problemas principales: uno, desarrollar una metodología que permita la administración segura de la red, para esto utilizan un método de autenticación que evita se falsifiquen paquetes; dos, presentan un protocolo que permite a las tareas cambiar su ruta de comunicación en *tiempo de ejecución* sin perder paquetes.

Tomando en cuenta que los SoC que trabajan bajo el paradigma SDN pueden ser programados en *tiempo de ejecución*, Sandoval-Arechiga et al. (2016) realizan una evaluación donde miden los tiempos promedio de configuración, el retardo y rendimiento global de un SoC . Para realizar sus pruebas los autores utilizan diferentes algoritmos de *ruteo* y varían las tasas de inyección de paquetes. Sus resultados muestran que el tiempo de configuración es dependiente del algoritmo de *ruteo* y en casos extremos el sistema puede colapsar.

El mapeo de tareas tiene un impacto significativo en el rendimiento de los SoC. Zhou and Zhu (2017), proponen un algoritmo dinámico de mapeo de tareas para SDNoC. El diseño jerárquico de SDNoC promueve la flexibilidad de las NoC y proporciona un entorno para la ejecución de un algoritmo de mapeo dinámico con técnicas de *software*. El objetivo del mapeo dinámico de tareas es minimizar el costo de comunicación de la ejecución de la aplicación y lograr el equilibrio de carga entre los *ruteadores*. Las tareas de las aplicaciones son asignadas a los nodos de la red mediante un mapeo que toma en cuenta un costo mínimo de comunicación y balanceo de carga entre los *ruteadores*.

En el trabajo presentado por Ellinidou et al. (2018, 2019) se muestra como la idea de SDN sobre SoC puede ir mas allá. Ellos presentan una tecnología que denominan *Red de Chips* (Network-of-Chips (NofC), en inglés), la cual está habilitada para modificar sus características como son: lógica de ruteo, rutas de transmisión, prioridades, agrupamiento de circuitos, etc. NofC se refiere a una gran cantidad de *Circuitos Integrados* (CI), que pueden tener diferentes velocidades de comunicación y niveles jerárquicos. Uno de los desafíos clave en el paradigma de NofC es la comunicación entre los grupos de núcleos y los CI. Por esa razón, los autores proponen adoptar los algoritmos y estrategias desarrollados dentro del campo de SDN. Otro trabajo que muestra las bondades de aprovechar las características de SDN en la interconexión de múltiples circuitos es presentado en Ellinidou et al. (2019), donde muestran un protocolo de comunicación para un sistema basado en *chiplet*s con una filosofía de administración SDN. El cual consiste en un protocolo *handshake*, supervisión de la red y *ruteo*. Además de probar que SDN funciona en el ambiente de *chiplet*s, los autores apuntan hacia la posibilidad de integrar algoritmos de *Machine Learning* los cuales en tiempo real podrían seleccionar los algoritmos de *ruteo* adecuados de acuerdo a las aplicaciones que se encuentran corriendo en el sistema.

La arquitectura NofC consiste en un SoC integrado con *Procesadores-Software* multinúcleo que permiten crear un Sistema-en-Chip con Múltiples Núcleos (SeCMN) en una PCI. Para diseñar una arquitectura de sistema heterogénea, se eligen núcleos de alto rendimiento y núcleos de bajo rendimiento para presentar diferentes perspectivas. Específicamente, los diferentes núcleos se clasifican de acuerdo con su rendimiento en *Grupos de Procesamiento*. La plataforma NofC se compone de una PCI de CI donde cada uno contiene *Grupos de Procesamiento* escalables. Para la comunicación en una PCI, los conmutadores SDN integrados se colocan en los límites de todos los circuitos integrados, para proporcionar una comunicación rápida entre los circuitos integrados y los *Grupos de Procesamiento* correspondientes. Para proporcionar comunicación entre los *Grupos de Procesamiento*, se adoptan conmutadores de *hardware* con funcionalidades SDN. Específicamente, un *conmutador SDN* en el plano de datos, que transporta el tráfico de paquetes. Además, en el plano de control, administra la comunicación entre los conmutadores y los controladores. La red en su conjunto seguirá una topología de malla donde la ruta de cada paquete se asigna dinámicamente.

En el trabajo presentado por Silva et al. (2019) se evalúa cuál es el comportamiento de diferentes algoritmos de ruteo cuando se crean circuitos virtuales bajo demanda en una NoC con enfoque SDN. Se evalúa la latencia de comunicación cuando se varía el tráfico en la red. Sus resultados muestran que el algoritmo adaptativo presenta mejores resultados con escenarios de bajo tráfico. Para escenarios con un tráfico intenso los dos algoritmos arrojan resultados similares.

2.6.2. Arquitecturas de Sistemas en Chip con enfoque de Redes Definidas por Software

En la literatura se han presentado diferentes trabajos que proponen arquitecturas para sistemas de interconexión trabajando bajo los principios de SDN. A continuación se describen estas arquitecturas.

2.6.2.1. Arquitectura de Interconexiones Definidas por Software

La arquitectura de *Interconexión Definida por Software* (Software Defined Interconnection (SDI), en inglés) presentada por Sandoval-Arechiga et al. (2015); Sandoval-Arechiga et al. (2017), es utilizada para interconectar múltiples núcleos de procesamiento en un SoC. Su objetivo es presentar una arquitectura con niveles y capas de abstracción bien definidos, que permita optimizar la interconexión de los elementos de procesamiento, de tal modo que las aplicaciones que se encuentran ejecutando, puedan cumplir con los requisitos de tiempo y QoS que se les demanden. Para elaborar su propuesta los autores se basan en la arquitectura SDN propuesta por Kreutz et al. (2015).

La arquitectura SDN se centra en el proceso de administración y operación de la red. En una interconexión de muchos núcleos, estos procesos serán de gran importancia a medida que los requisitos de potencia y rendimiento se vuelvan más difíciles. La arquitectura propuesta presenta tres capas y cinco planos. En la Figura 2.20 se puede observar la arquitectura. A continuación se describirán cada una de las capas y planos de la misma.

Capa del Sistema Operativo

La capa del Sistema Operativo normalmente reside en una *CPU* –un Procesador Elemental en el SoC–. Su tamaño puede variar dependiendo de los requisitos que impongan las aplicaciones, pudiendo ser tan pequeño como una sola aplicación ejecutándose en *firmware*. Esta capa es responsable de la interacción entre las aplicaciones y el sistema operativo de la red. Además, administra la asignación de recursos del sistema, como memoria, tiempo de CPU, E/S, entre otros.

Plano de Aplicación

Este plano está compuesto por las diferentes aplicaciones que se ejecutan en el SoC. Cada aplicación o algoritmo de procesamiento se presenta a modo de grafo donde los nodos son las tareas que se ejecutan en los EP, y las interconexiones son las dependencias de datos entre ellos. Cada aplicación genera su propio grafo y requerimientos y los presenta al Sistema Operativo de Red. El Sistema Operativo de

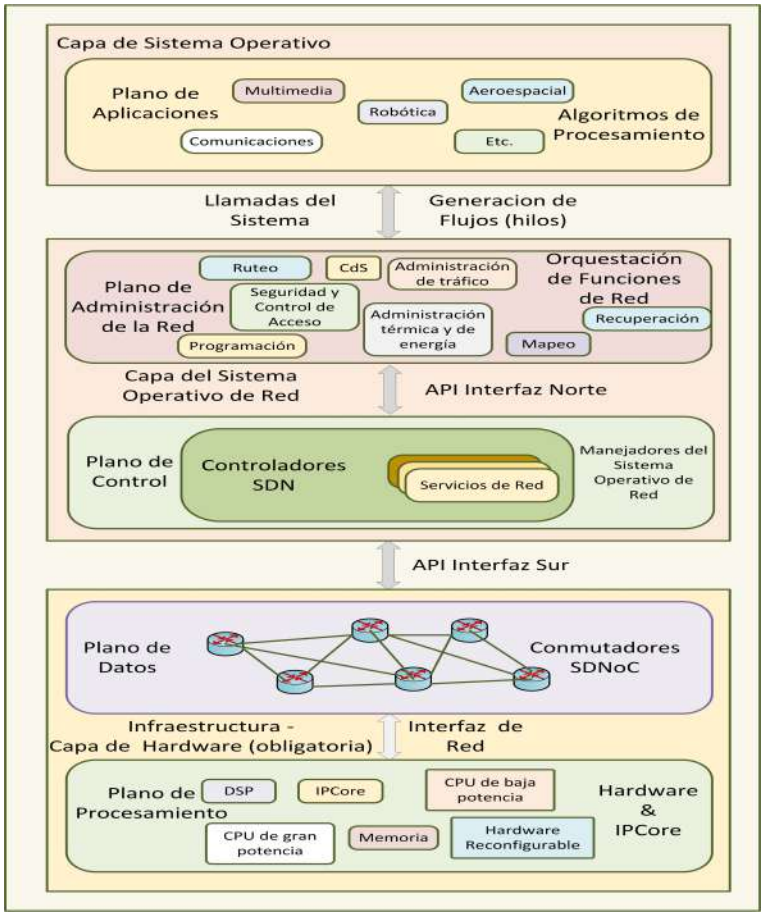


Figura 2.20: Arquitectura de Interconexiones Definidas por Software Sandoval-Arechiga et al. (2017)

Red organizará las funciones de la red para garantizar los requisitos de la aplicación y el presupuesto operativo del SoC, por ejemplo, restricciones de potencia y térmicas.

Capa del Sistema Operativo de Red

La capa Sistema Operativo de Red traduce los requisitos dados por la capa de aplicación en políticas y configuraciones de interconexión con las que debe de operar el SoC. Estas son enviadas en forma de *scripts* de configuración a la capa de infraestructura donde se ejecutan.

Plano de Administración de la Red

El Plano de Administración de la Red consta de varios motores de optimización que modifican la funcionalidad de la red, tales como: enrutamiento, mapeo, programación, gestión del tráfico, control y seguridad de acceso, QoS, gestión térmica y energética, recuperación, etc. Tales motores, llamados Funciones de Red, operan en base a una vista parcial o global de la red de interconexión que es proporcionada por el plano de control. Dichas funciones permiten programar las diferentes aplicaciones que se ejecutan en el SoC verificando que se cumplan con los requisitos de QoS en términos de rendimiento, retardos y potencia, entre otros. El Plano de Administración de la Red organiza la combinación requerida de los motores de optimización y organiza la ejecución de aplicaciones en la red. Además, el Plano de Administración de la Red genera políticas y configuraciones que son transmitidos a planos inferiores para su implementación. Los motores de optimización son modulares, de forma que si una Función de Red no está presente o debe agregarse/eliminarse, el cambio no tiene impacto en los otros motores. Este plano interactúa con las aplicaciones a través de llamadas del sistema operativo de la red que generan hilos o flujos en la red. Este plano ofrece, entre otros, los siguientes servicios: optimización del mapeo de una aplicación para que pueda cumplir con los requerimientos de QoS con los elementos de procesamiento que tenga disponibles; organización de las funciones de red para optimizar la interconexión en chip de manera global; organización de las aplicaciones que se ejecutan en la interconexión en chip; mapeo de aplicaciones a la red (en conjunto con las llamadas del Sistema Operativo de Red); asignación dinámica o estática de recursos en la interconexión en chip y la generación de abstracción de red.

Plano de Control

El Plano de Control determina los mecanismos que permiten la re/configuración de los dispositivos de reenvío de datos. Está compuesto por los controladores SDN que establecen una conexión basada en paquete o en flujo con el dispositivo de reenvío de datos. La comunicación se lleva a cabo por medio de la Interfaz Sur. Una vez que se establece una conexión, pueden enviar la configuración y recopilar el estado y los datos estadísticos de los dispositivos de reenvío. Una interconexión en chip puede tener uno o varios controladores SDN, dependiendo del tamaño de la red, el tráfico y la complejidad de las aplicaciones. Cada controlador SDN en la interconexión en chip puede tener una vista parcial de la red, pero una vez combinado, el plano de control puede producir una vista global de la red. Los controladores SDN pueden estar especializados por el dispositivo de reenvío que controlan: *enrutador* o *conmutador*; el modo de conmutación, como circuitos o paquetes; etc. Luego, para cada tipo de controlador SDN, se puede agregar un controlador en las implementaciones

del Sistema Operativo de Red. El plano de control se comunica con el plano superior utilizando la *Interfaz Norte*. Además, el plano de control ofrece los siguientes servicios: enviar configuración a un conjunto específico de nodos en la red; recopilar datos de estado y estadísticas de un conjunto específico de nodos en la red y generar una vista (estado) global o parcial de la red.

Capa de Infraestructura

La capa de infraestructura es responsable de las funciones de reenvío y procesamiento, comprende los planos de reenvío de datos y de procesamiento de datos. Debido a sus funciones, es una capa que obligatoriamente debe de ser implementada en *hardware*. La implementación puede variar de proveedores, aplicación, cliente o presupuesto. El único requisito indispensable para esta capa es la *Interfaz Sur* para comunicarse con la capa de Sistema Operativo de Red.

Plano de Reenvío de Datos

Se compone de *ruteadores*, *interfaces de red*, *conmutadores* y *buses* que componen la interconexión de la red en chip. Su función principal es la transmisión de datos desde su origen hasta su destino. En este plano, la comunicación con la capa superior se realiza por medio de la *Interfaz Sur*. Los servicios ofrecidos por este plano son: envío de datos desde el origen hasta el destino utilizando el modo de flujos, como lo indica el paradigma SDN; envío de datos desde el origen hasta el destino en modo normal utilizando paquetes; ejecución de la re/configuración en los dispositivos de retransmisión (para cada enlace) y recopilación del estado y las estadísticas de los dispositivos de retransmisión (para cada enlace).

Plano de Procesamiento de Datos

Se compone de los Elementos de Procesamiento, como *CPU*, *IPCores*, *hardware reconfigurable*, *memorias*, *controladores de acceso directo a memoria*, etc. Se conectan a través de las IR al sistema de interconexión. Su función principal es el procesamiento de datos especificado por las aplicaciones. Los servicios que ofrece este plano son: procesamiento de datos generales (en CPU); almacenamiento de datos para su posterior procesamiento (Memorias); procesamiento especializado de datos (en IPCores o hardware reconfigurable) y mejora de la latencia de acceso a datos (a través de DMA).

Los autores mencionan que una de las ventajas de tener una arquitectura modular es que los servicios pueden ser agregados al ritmo que la arquitectura madure.

2.6.2.2. Arquitectura Red en Chip Definida por Software

Berestizshevsky et al. (2017) presentan una arquitectura que denominan SDNoC, la cual está basada en un co-diseño *hardware/software*. El esquema que presentan contiene dos redes: una red de control y una red de datos. Estas redes son implementadas por redes físicas separadas. Los paquetes se envían a lo largo de rutas de enlace diferentes en la red de datos, avanzando a una velocidad de un salto por ciclo de reloj. El ancho de banda de los enlaces a lo largo de una ruta es completamente consumido por el enlace. Por lo tanto, cada enlace puede pertenecer como máximo a una ruta. Los módulos de la red de datos son IR y *conmutadores*. La ruta crítica en la red de datos depende solo de la longitud de un enlace único y del retraso de sus puntos finales (es decir, *conmutador* o IR). Esto implica que la red de datos puede ser escalada. Por ejemplo, la velocidad de reloj de una red de datos tipo malla no aumenta en función del tamaño de la red. La red de control contiene un único administrador de red centralizado denominado *Administrador de Red (AR)* que está conectado a todas las IR a través de la red de control. Los tres elementos de la infraestructura de la red son descritos a continuación.

Interfaz de Red

Cada IR está conectada a un núcleo y a un conmutador. Sirve como puerta de entrada a la red de datos. La IR solo trata con el tráfico de su núcleo. Cada IR está conectada a un *conmutador* a través de un solo puerto. Esto implica que una IR no puede recibir y enviar paquetes simultáneamente. Al recibir un paquete de su núcleo asociado, la IR envía una solicitud al *Administrador de Red* para la entrega del paquete. El *Administrador de Red* responde enviando instrucciones a las IR de origen y destino para comenzar a enviar/recibir el paquete. La IR comienza a enviar el paquete tan pronto como recibe una respuesta de inicio. La IR fuente agrega al paquete un *phit* de cola que indica el final del paquete. Al recibir el *phit* de fin de paquete, el destino IR envía un anuncio de fin de paquete al *Administrador de Red*. El *Administrador de Red* libera los recursos de la red (y envía instrucciones de configuración actualizadas a los *conmutadores*). En cada ciclo de reloj, la IR trata a lo sumo una solicitud.

Conmutador

Cada *conmutador* tiene puertos que pueden servir como entrada o salida (pero no ambos a la vez). Los puertos de los *conmutadores* vecinos están conectados por enlaces bidireccionales; el ancho de un enlace le permite entregar un *phit* por ciclo de reloj. Cada *conmutador* reenvía *phit*s desde sus puertos de entrada a sus puertos de salida, creando así una ruta en la red de datos. Cada *phit* entrante se reenvía

en el siguiente ciclo de reloj, ya sea a un *conmutador* vecino o a la IR destino. Por lo tanto, los *conmutadores* no almacenan *phit*s durante más de un ciclo de reloj y no se requiere administración del *búffer* en los *conmutadores*. El *Administrador de Red* envía instrucciones de configuración a los *conmutadores* que definen los puertos de entrada y salida activos y hacen coincidir los puertos de entrada activos con los puertos de salida activos de manera individual. Como la coincidencia es uno a uno, cada puerto de salida activo reenvía *phit*s desde un único puerto de entrada activo. Por lo tanto, no se crean conflictos entre los *phit*s. En SDNoC no es necesaria la información origen-destino en los encabezados de *flit*s (ni siquiera en el primer lanzamiento de un paquete) porque la configuración de un *conmutador* gobierna el reenvío de *phit*s. Dado que SDNoC emplea la conmutación de circuitos virtuales, los paquetes son enviados secuencialmente a lo largo de la ruta única reservada para el paquete por el *Administrador de Red*. Por lo tanto, no hay necesidad de encabezados ni números de secuencia. Dado que cada conmutador tiene un pequeño número constante de puertos, su configuración se puede codificar fácilmente con unos pocos bits.

Administrador de Red

El AR recibe por parte de las IR solicitudes para enrutar paquetes. El AR tiene una vista completa de la red de datos. Conoce las solicitudes activas actuales y las rutas asignadas a ellas. Además, conoce la configuración de las IR y los *conmutadores*. Conoce también las solicitudes que esperan la asignación de una ruta. El IR tiene una cola *FIFO* que almacena todas las solicitudes que esperan una ruta. La IR selecciona una ruta (si existe una ruta vacante) de acuerdo con su algoritmo y la asigna a la solicitud enviando mensajes de configuración a todos los *conmutadores* a lo largo de la ruta. Además, se envía un mensaje de inicio a las IR de origen y destino. Las solicitudes para las que no se encontró ninguna ruta se vuelven a ingresar a la cola *FIFO*. Al recibir un anuncio de fin de paquete de una IR de destino, el AR libera la ruta asignada al paquete para que se puedan atender nuevas solicitudes.

2.6.2.3. Arquitectura Scionti

En Scionti et al. (2018) se presenta una arquitectura para NoC basada en SDN. La propuesta se basa en una organización jerárquica: con anillos pequeños para comunicaciones locales más frecuentes entre grupos pequeños de EP, mientras que para comunicaciones entre EP distantes existe una interconexión global de malla. Esta jerarquía muestra una mejor relación rendimiento/potencia y escalabilidad en comparación con los diseños planos (es decir, un solo tipo de interconexión que sirve a todos los EP en el chip). La introducción de la capacidad de reconfiguración en la arquitectura NoC propuesta requiere componentes adicionales. Los autores

proponen dos tipos de *conmutadores*, uno para la topología de malla y otro para la topología de anillo. Uno de los EP ejecuta el *software* que administra la red, al cuál se le agregan un conjunto de instrucciones para manejar las fases de reconfiguración, supervisión y distribución de la carga.

2.6.2.4. Arquitectura Ruaro

Otra propuesta de arquitectura la presentan Ruaro et al. (2018). Ellos dividen la arquitectura en tres capas: *Aplicación*, *Middleware* y *Física*. La capa de *Aplicación* contiene las aplicaciones de los usuarios. Se puede describir una aplicación como un grafo donde los nodos representan tareas y las conexiones la comunicación entre las tareas. Las tareas intercambian datos utilizando un protocolo de comunicación como *MPI* u *open-MP*. La capa *middleware* contiene el Sistema Operativo y el *Controlador-NoC (CNoC)*. El sistema operativo se ejecuta en cada EP del sistema, abstrae los recursos físicos que necesitarán las tareas de las aplicaciones, proporcionando las primitivas de comunicación y la programación de las tareas. El CNoC implementa los servicios SDN en el sistema operativo. El sistema operativo puede solicitar al CNoC que defina una ruta de comunicación entre un EP de origen y de destino. El CNoC maneja las solicitudes de ruta del sistema operativo, busca la ruta de acuerdo con alguna política predefinida y notifica al sistema operativo el resultado del establecimiento de la ruta (éxito o falla). La capa inferior, *Física*, contiene los componentes físicos de la red.

2.7. Diferencias entre Sistemas de Interconexión Tradicionales y Sistemas de Interconexión Definidos por Software

La principal diferencia que existe entre los sistemas de interconexión tradicionales y los sistemas de interconexión definidos por software es que éstos últimos desacoplan la etapa de control de la etapa de transporte de los datos. Un elemento de *software* está a cargo de la lógica de control mientras que el hardware solo es responsable de la retransmisión de los datos. Si los diseñadores necesitan cambiar el esquema de comunicación, solamente la lógica de control es modificada por *software* mientras que la capa inferior de la red permanece intacta.

La infraestructura de los sistemas de interconexión tradicionales es más especializada que la infraestructura de los sistemas de interconexión con enfoque SDN.

En el trabajo que presentan Cong et al. (2014) indican que las implementaciones de NoC con enfoque SDN presentan ciertas ventajas: 1) no se restringen a un diseño específico; 2) es posible reconfigurar la red de acuerdo a la carga de trabajo que pue-

dan tener ciertos *conmutadores* en específico, 3) es posible el re-uso de procesadores; 4) existen menos problemas para manejar las restricciones de IPCores propietarios.

En los sistemas de interconexión con enfoque SDN, el ruteo de paquetes puede ser establecido en *tiempo de ejecución*. Ruaro et al. (2020) muestran el procedimiento: el C-SDN se encuentra corriendo en una parte específica del SoC. El controlador abstrae la administración de la red del Administrador de Red, que realiza la admisión de aplicaciones y el mapeo de tareas. Debido a dicha abstracción, cuando un *maestro* necesita una ruta entre una tarea *Fuente* F y una tarea *Destino* D, realiza una solicitud de ruta al C-SDN. El controlador busca la ruta guiado por el *estado* de la red y sus políticas (evitando por ejemplo *hot-spot*s y *ruteadores* defectuosos). Después de encontrar una ruta, el C-SDN efectúa su configuración física, mandando paquetes de configuración a los *ruteadores* SDN. Cuando la configuración finaliza, el C-SDN manda un paquete de reconocimiento al AR el cual configura a F y D para usar el camino. En este sentido el paradigma SDN difiere de los sistemas de interconexión tradicionales debido a que éstos se enfocan en cumplir metas específicas y no asumen que los *ruteadores* pueden ser dinámicamente modificados en *tiempo de ejecución* de acuerdo a las restricciones.

Los sistemas de interconexión con un enfoque basado en SDN proveen mecanismos de retroalimentación que permiten en todo momento conocer el estado de la red. Esto permite tomar acciones por parte de las capas superiores en la arquitectura para que las aplicaciones puedan cumplir con las restricciones de QoS que se les imponen. Por otro lado al conocer en todo momento el estado de la red le permite ahorrar energía dado que es posible apagar los *conmutadores* que no se estén utilizando.

2.8. Infraestructura para implementar Sistemas de Interconexión definidos por software en SoC

El paradigma SDN elimina la lógica de control de la capa de infraestructura. De tal modo que los elementos que componen esta infraestructura son más sencillos. Sin embargo existen ciertos elementos que deben de ser contemplados al momento de diseñarlos para que puedan cumplir correctamente con sus funciones. Los elementos fundamentales que puede tener una infraestructura con enfoque SDN son: *Interfaz de Red*, *Ruteadores*, *Buses*.

Diferentes autores Sandoval-Arechiga et al. (2017); Ruaro et al. (2018); Cong et al. (2014); Berestizshevsky et al. (2017) apuntan un conjunto de características generales que tienen que tener estos elementos:

- Los elementos que forman parte de la infraestructura de un sistema de interconexión con enfoque SDN deben de tener la capacidad de ser re/configurables en todo momento. Deben permitir la reconfiguración en *tiempo de ejecución* y al mismo tiempo evitar la pérdida de paquetes. Además de poder configurar trayectorias, rutas o flujos, deben de poder tener la capacidad de administrar el uso de energía, permitiendo poder encender o apagar puertos solo cuando estos se necesiten (bajo demanda). Deben tener la capacidad de regular el uso de los puertos de acuerdo al tipo de flujo que los requiere.
- El proceso de administración de las *tablas de flujo* debe de ser sencillo, las reglas para operar los flujos deben de ser pocas y sencillas.
- Deben de contar con una unidad que en todo momento conozca el estado del elemento, de tal modo que éste pueda ser reportado a los controladores en capas superiores y se tomen acciones basadas en este estado.
- Deben de poder llevar la estadística de uso de sus elementos: tiempo de uso, cantidad de paquetes que fluyen por cada puerto, tiempo promedio de espera de los paquetes en los *buffers* de entrada y salida.

Capítulo 3

Modelo del Sistema

En este capítulo se describen las principales consideraciones que se realizaron para el desarrollo del presente trabajo. En primer lugar se presentan las razones del uso de la topología seleccionada. A continuación se describe la suite para modelado de tráfico denominada MCSL-NoC cuyos patrones de tráfico fueron utilizados para desarrollar algunas pruebas en este trabajo. Enseguida se presentan otras aplicaciones que se utilizaron para la realización de otro conjunto de pruebas. Posteriormente se describe el modelo general del sistema tanto a nivel de *software* como de *hardware*. A continuación se presentan algunas herramientas adicionales que fueron desarrolladas para poder implementar las pruebas. Finalmente se mencionan algunas consideraciones generales que se hicieron para el desarrollo del trabajo.

3.1. Topología

La topología de red que se seleccionó para realizar el presente estudio fue tipo *bus*. Son varias las razones que motivaron esta decisión:

- Es una arquitectura de uso común en la industria del desarrollo de los SoC. En la actualidad los sistemas de interconexión desarrollados por algunos de los principales competidores en la industria, soportan la comunicación de sus elementos de procesamiento basados en una topología tipo *bus*: ARM (2010), Peterson and Herveille (2010), Intel (2020a). Por ejemplo, ARM en su especificación para la interconexión de circuitos en chip denominada *Advanced Microcontroller Bus Architecture AMBA*, presenta su interfaz *Advanced eXtensible Interface AXI* que es utilizada para interconectar elementos del tipo *maestro-esclavo* en topologías del tipo *bus* con diferentes configuraciones: 1) buses de datos y de direcciones compartidos, 2) múltiples buses de datos con direcciones compartidas, 3) múltiples buses de datos y direcciones en múltiples capas.

- A pesar que es un sistema de interconexión muy maduro y bien estudiado existen aún huecos por llenar sobre su comportamiento cuando es expuesto a escenarios con aplicaciones con dependencia de tareas.
- El uso proporcional del *bus* para asegurar que un elemento de procesamiento pueda cumplir con un rendimiento específico no ha sido completamente explorado.
- SDN introduce el concepto de operación basado en flujos. En otras arquitecturas como NoC la operación con flujos permite mejorar el rendimiento del sistema. Es de interés medir qué impacto tiene en el rendimiento el manejo de flujos en un sistema de interconexión tipo *bus*.
- Aunque finalmente todos los sistemas de interconexión cuentan en alguna parte de su infraestructura con un sistema de arbitraje, desarrollar pruebas en un sistema de interconexión tipo *bus*, permite aislar perfectamente el comportamiento de las políticas de arbitraje. Este comportamiento posteriormente puede ser extrapolado a otro tipo de sistemas de interconexión.
- Además, es de interés conocer cómo pueden ser aprovechadas otras características del paradigma SDN en sistemas de interconexión tipo *bus* como son: la reconfiguración en *tiempo de ejecución*, la generación de estadística, y la administración de la energía, entre otras..

3.2. Suite MCSL-NoC de modelado de tráfico para Redes en Chip

Una práctica común dentro de la comunidad científica es poder contrastar los modelos, algoritmos, desarrollos e implementaciones propuestas con los resultados obtenidos por las propuestas de otros autores. Esta comparación resulta muy complicada de llevarse a cabo si no se cuenta con las mismas métricas y patrones de prueba. Es por esto que parte del trabajo de la comunidad científica es el desarrollo de patrones de prueba que puedan servir como referencia para contrastar el trabajo realizado por otros autores. En el caso de los sistemas de interconexión, específicamente en las NoC, Liu et al. (2011) han desarrollado un conjunto de patrones de prueba que integran en una suite denominada *MCSL Network-on-Chip Traffic Suite*. Esta suite consta de dos tipos de patrones de tráfico: patrones de *tráfico sintético* y patrones de *tráfico real*. Debido a que es de interés conocer el comportamiento de las políticas de arbitraje cuando los EP ejecutan aplicaciones reales, para este estudio fueron utilizados los patrones de prueba de *tráfico real* que se generan en esta suite.

En la metodología utilizada en la suite para el desarrollo de los patrones de tráfico, los autores utilizan un mapeo centrado en el balanceo de carga, tienen una estrategia de programación centralizada que administra todos los recursos del circuito y coordina los EP. Las decisiones de control y programación son optimizadas de manera global. La idea básica es distribuir el procesamiento y las cargas de la transmisión de tal modo que se puedan alcanzar los mayores niveles de la utilización de los recursos de *hardware*. La estrategia de mapeo es asignar tareas a los EP una por una en orden topológico definido por la dependencia de las relaciones en el grafo. La programación en cada EP es determinada por la secuencia de tareas generadas en un mismo EP durante el mapeo. El objetivo es minimizar el tiempo de ejecución de principio a fin de una aplicación tomando en cuenta el *overhead* de la red de comunicaciones.

La suite está integrada por el tráfico generado para ocho aplicaciones de diferentes dominios. Las características de estas aplicaciones así como su número de tareas y enlaces de comunicación se muestran en la Tabla 3.1.

Esta suite no produce patrones de tráfico específicos para sistemas de interconexión basados en *bus*. Sin embargo, podemos utilizar este mapeo porque, en nuestro caso, el efecto que queremos medir es la capacidad de controlar el acceso al *bus* cuando las tareas interdependientes lo utilizan.

La suite tiene la capacidad de generar tráfico para tres topologías: *Mesh*, *Torus* y *Fat Tree*. El tamaño de las redes para cada topología puede observarse en la Tabla 3.2.

El tráfico generado por la suite para cada topología y tamaño de red es almacenado en un archivo binario que tiene la estructura de datos mostrada en la Tabla 3.3.

3.3. Otras aplicaciones utilizadas para generar tráfico en la red

Si bien en el estudio se utiliza tráfico sintético para realizar algunas pruebas de comportamiento de los árbitros, el interés principal es conocer el comportamiento del sistema de interconexión con aplicaciones que son de uso común. En este sentido se diseñó una aplicación utilizada en el ámbito satelital que efectúa las siguientes funciones: captura de imágenes espaciales, preprocesado de la imagen para eliminación de ruido, compresión de la imagen para que sea más ligera al momento de transmitirla y cifrado de la imagen para confidencialidad en la transmisión. Los IPCores utilizados se detallan a continuación:

- **Filtro**: para eliminación de ruido causado debido a las condiciones en la que la imagen fue adquirida. La implementación *hardware* de estos filtros normal-

Tabla 3.1: Aplicaciones incluidas en la Suite MCSL NoC

Aplicación	Descripción	# Tareas	# Enlaces de Comunicación
RS-32-28-8 cod	Codificador Reed-Solomon con formato de palabra RS(32,28,8)	262	348
RS-32-28-8 dec	Decodificador Reed-Solomon con formato de palabra RS(32,28,8)	182	392
H264-720p dec	Decodificador de vídeo H.264 con una resolución de 720p	2311	3461
H264-1080p dec	Decodificador de vídeo H.264 con una resolución de 1080p	5191	7781
ROBOT	Cálculo de control dinámico Newton-Euler para el manipulador Stanford de seis grados de libertad	88	131
FPPPP	SPEC95 Fpppp es un programa químico que realiza derivaciones multi-integrales	334	1145
FFT-1024	Transformada rápida de Fourier con 1024 entradas de números complejos	16384	25600
SPARSE	Solucionador de matrices dispersas aleatorias para simulaciones de circuitos electrónicos	96	67

Tabla 3.2: Configuraciones NoC incluidas en la Suite MCSL NoC

Topología	Tamaño (número de procesadores)
Mesh	2x2, 2x4, 3x3, 4x4, 5x5, 6x6, 7x7, 8x8, 9x9, 10x10, 11x11, 12x12, 13x13, 14x14, 15x15, 16x16
Torus	2x2, 2x4, 3x3, 4x4, 5x5, 6x6, 7x7, 8x8, 9x9, 10x10, 11x11, 12x12, 13x13, 14x14, 15x15, 16x16
Fat Tree	4, 8, 16, 32, 64, 128, 256

Tabla 3.3: Estructura de datos del tráfico generado por la Suite MCSL NoC

Enlace	
id	identificador del enlace
source_task	tarea origen del enlace
destination_task	tarea destino del enlace
lista de direcciones de memoria	lista de las direcciones de memoria
lista de tamaño de los mensajes	lista del tamaño de los mensajes
Tarea	
id	identificador de la tarea
procesador	procesador al que la tarea está asignada
lista de programación	número de secuencia en que las tareas serán ejecutadas
lista de tiempo de ejecución	lista de tiempo de ejecución de acuerdo a la secuencia de programación
enlace de entrada	cada entrada es un enlace de entrada
enlace de salida	cada entrada es un enlace de salida
Procesador	
id	identificador del procesador
renglón	renglón en la topología mesh/torus
columna	columna en la topología mesh/torus
tasks	lista de tareas que se ejecutaran en este PB
NoC Traffic	
topology	código de la topología
renglón	número de renglones en la topología
columna	número de columnas en la topología
iteraciones	número de veces que el grafo se ejecuta
procesadores	lista de procesadores
tareas	lista de tareas
enlaces	lista de enlaces
tareas iniciales	lista de tareas iniciales
tareas finales	lista de tareas finales

mente se realiza por medio de *arreglos sistólicos* que utilizan una máscara de convolución y realizan el proceso de filtrado a razón de un píxel por ciclo de reloj. El filtro empleado acepta como entrada una línea de la imagen original y genera como salida una línea de la imagen filtrada. Para las pruebas realizadas en este estudio se utilizó una implementación propia del algoritmo de suavizado de imagenes con una latencia de un ciclo de reloj por píxel. Esto quiere decir que si se manda procesar (filtrar) una línea de la imagen de n bits su tiempo de operación será de n bits.

- **Compresor**: este tiene como objetivo disminuir el tamaño de la imagen para facilitar su transporte. En este trabajo, se decidió utilizar el algoritmo sin pérdidas *Lossless Image Compression Algorithm, LOCO*, con la implementación realizada por Hernández-Calviño et al. (2018). El algoritmo en promedio puede comprimir hasta 50 % del tamaño original de la imagen. La implementación *hardware* de este algoritmo recibe como entrada una línea de la imagen, procede a realizar el proceso de compresión y entrega una línea comprimida. El IPCore tiene un tiempo de procesamiento de 16 ciclos/píxel.

- **Cifrador**: la protección de información es clave en los sistemas embebidos. Para proteger que la información pueda ser utilizada por terceros ésta es cifrada utilizando el esquema de cifrado por bloques denominado Advanced Encryption Standard (AES). La implementación utilizada es la descrita en Garcia-Luciano et al. (2017) la cual recibe 16 Bytes de datos originales y entrega 16 Bytes de datos cifrados. El tiempo de procesamiento es de doce ciclos de reloj.

3.4. Modelo del Sistema

El desarrollo práctico del presente estudio fue dividido en dos partes:

- Primera, la evaluación de los modelos fue llevada a cabo con la herramienta de programación *SystemC*. *SystemC* es un conjunto de clases C++ que permite la simulación de los modelos bajo el paradigma de activación por eventos. Es una herramienta útil para modelar el comportamiento de los sistemas. En el caso de este estudio, *SystemC* se convirtió en una herramienta fundamental porque gran parte del mismo tiene que ver con conocer el comportamiento de los diferentes elementos de las arquitecturas implementadas (árbitros, interfaces de red, sistemas de interconexión, etc.). Además, *SystemC* permite la evaluación de un sistema con precisión de ciclo de reloj.

- Segunda, la implementación hardware, que tiene como objetivo conocer las características físicas de la implementación (recursos consumidos, frecuencia

máxima de operación, consumo de energía). El desarrollo se realizó utilizando el lenguaje descriptor de hardware Verilog y su implementación se llevó a cabo sobre el SoC de Xilinx Zynq-7020.

Para poder operar la infraestructura de *hardware* del sistema de interconexión, se desarrolló un modelo en *SystemC* tal como se muestra en la Figura 3.1. El modelo toma como referencia la arquitectura presentada por Sandoval-Arechiga et al. (2015). Existen dos capas en el modelo: la primera, la capa de *Software* correspondiente a la capa del Sistema Operativo de Red del modelo de referencia. Ésta es la encargada de proveer las rutinas de servicio para re/configurar los elementos del sistema, generar la estadística del sistema y operar el sistema; la segunda, la capa de *hardware* que corresponde a la capa de *Infraestructura* del modelo de referencia. La capa está compuesta por dos planos: uno, el *Plano de Reenvío de Datos* (PRD), encargado de enviar los datos de un elemento de procesamiento a otro, está integrado por el *Controlador del Bus* (CB) y la Interfaz de Red (IR); dos, el *Plano de Procesamiento de Datos* (PPD), que contiene los elementos de procesamiento como: IPCores, *procesadores*, *controladores de memoria*, etc.

3.4.1. Capa de Software

La capa de *Software* es la que posibilita la operación de la infraestructura *hardware* del sistema de interconexión. Permite trabajar en dos modos: uno, sistema de interconexión tipo *bus tradicional*; dos, sistema de interconexión tipo *bus SDN* que de aquí en adelante se denominará *Software Defined Bus-on-Chip (SDBoC)*. Dependiendo del modo en que sea configurado el sistema, son las capacidades del mismo. En el modo *bus tradicional* el sistema no tiene capacidad de re/configuración, ni posibilidad de generar estadística para autocontrol. El sistema funciona únicamente con las capacidades propias de la política de arbitraje establecida. En el modo SDBoC se integra la capa del plano de control provista por el paradigma SDN y se tiene la capacidad de re/configurar y obtener estadística del sistema con el objetivo de poder controlar las funcionalidades del mismo.

En el modelo de referencia la capa del Sistema Operativo de Red se encuentra dividida en dos planos. El modelo aquí presentado pone principal atención en el Plano de Control ya que es el encargado de interactuar directamente con el hardware del sistema, que es el objetivo principal de este estudio. Las acciones de comunicación se llevan a cabo por la Unidad de Control (UC). Las funcionalidades de esta capa dependen del modo en que el sistema de interconexión vaya a operar.

Cuando el sistema se encuentra trabajando en el modo *bus tradicional*, la UC tiene dos funciones: primera, establecer la configuración inicial del sistema de interconexión, esto lo hace por medio de la *Unidad de Re/Configuración*; segunda,

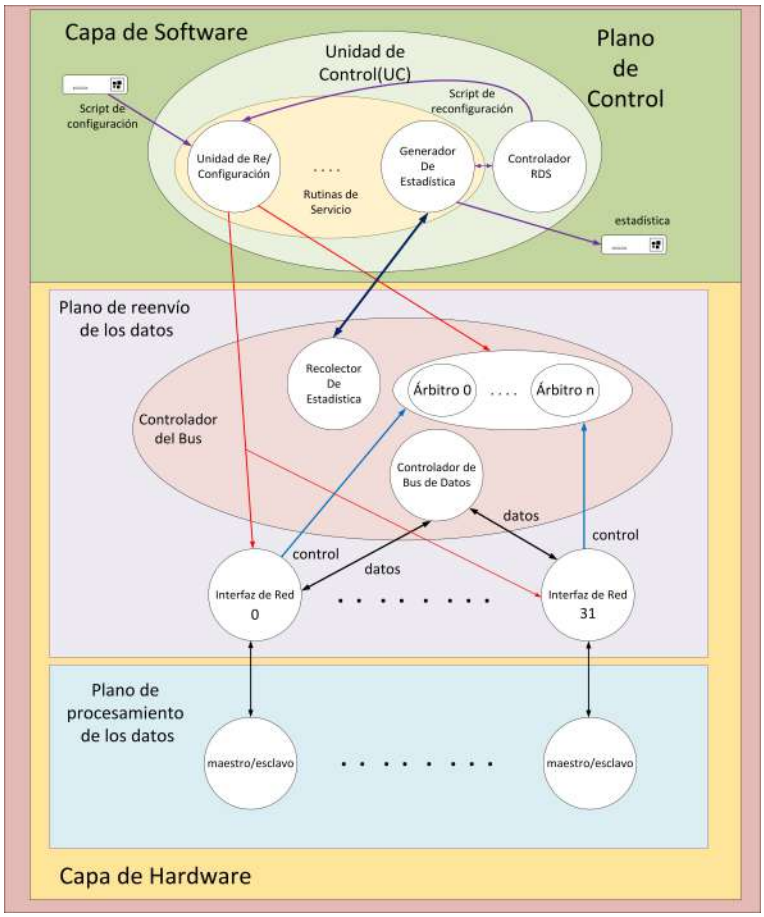


Figura 3.1: Modelo de la Arquitectura de Software de la aplicación

obtener y generar la estadística del comportamiento de la red, por medio del *Generador de Estadística* . La configuración inicial del sistema de interconexión se realiza por medio de un archivo de configuración. El sistema tiene la posibilidad de trabajar con seis políticas de arbitraje diferentes y puede operar con un sistema de interconexión de hasta treinta y dos nodos. La estructura de un archivo de configuración se puede observar en la Tabla 3.4.

Además cuando el escenario plantea el uso de tareas dependientes, en cada uno de los *maestros* se carga el archivo de tareas y dependencias que fue extraído de la *Suite MCSL NoC* de acuerdo a como se muestra en la Tabla 3.5.

Por otro lado, cuando el sistema funciona en modo SDBoC la re/configuración y generación de estadística se convierten en *rutinas de servicio* que son provistas por el C-SDN. Por ejemplo, el C-SDN puede pedir en *tiempo de ejecución* al *Generador de Estadística* del sistema, que solicite y reporte la proporción del uso del *bus* por

Tabla 3.4: Estructura de los Archivos de Configuración

Nombre	Descripción	Valores
Política	Es el tipo de política de arbitraje con el cual se va a ejecutar la simulación	0- RR, 1- Lottery, 2- TDMA, 3- WRR, 4- WRRM, 5- SuDO
Split	Indica si las transacciones del sistema serán del tipo split-transaction	0 - no 1 - sí
Maestros	número de maestros que formarán parte del sistema de interconexión	de 1 a 32 maestros en modo no-split-transaction de 1 a 16 maestros en modo split-transaction
Nombre	Descripción	Valores
tareas	Indica si el maestro trabajará en base a tareas dependientes o independientes	0 - independientes 1 - dependientes
archivo	si el maestro trabaja con tareas independientes es el nombre del archivo de dependencias para ese maestro	maestro(n)dep.dat
tipo	cada maestro se asocia con un tipo de IPCore	0 .. n (tipo de IPCore)
tamaño origen	tamaño de la carga útil del paquete del IPCore que inicia la transacción	1 - 2048 Phits
tiempo ejecución	tiempo promedio de ejecución de un IPCore esclavo	1 - 65536 ciclos de reloj
peso	peso de soporte de acuerdo a la política de arbitraje	RR - no aplica LTY - # de boletos TDMA - # de ventanas WRR - # de ciclos de reloj WRRM - # de ciclos de reloj SuDO - presupuesto

parte de cada uno de los *maestros*. En base a esta información y por medio de algoritmos de optimización, puede modificar las proporciones asignadas a los pesos de los *maestros*. Lo anterior permitirá cambiar la proporción de uso del *bus* por cada elemento con la finalidad de modificar su rendimiento. Finalmente, puede generar un nuevo *script* de reconfiguración y por medio de otra *rutina de servicio* puede indicar que se proceda a reconfigurar los elementos indicados en el *script*. Hay que recordar que este estudio se centra en proporcionar la arquitectura del sistema de interconexión y los servicios básicos para su operación. Los métodos de optimización de la red están fuera del alcance del estudio.

3.4.2. Capa de Hardware

La infraestructura de *hardware* básica que se desarrolló para este estudio se muestra en la Figura 3.2. En el PRD se encuentran dos elementos: uno, el CB que es el encargado en base a una política de arbitraje de otorgar el acceso al medio; dos, la IR que es la encargada de interactuar con los elementos de procesamiento y la transmisión/recepción de los datos que vayan desde/hacia ellos. Dentro del PPD se encuentran los *maestros* y/o *esclavos* encargados de la transformación de los datos.

El CB está dividido en dos elementos:

- **Controlador de datos del *bus***, en esta infraestructura se implementó un *bus* del tipo multiplexado. El *controlador* es el encargado de dirigir los datos provenientes de todas las IR y dirigirlo al canal de salida. La selección la hace a mandato expreso del árbitro del sistema.
- **Árbitro**: es el encargado de decidir a cuál de las IR que están solicitando acceso al medio le será otorgado. Esto lo hace en base a una política de arbitraje establecida. En el modelo de *SystemC* que se generó estas políticas de arbitraje son funciones que están encapsuladas lo que permite su fácil integración al sistema. Sólo una de ellas está activa en una simulación.

La IR es el elemento de interconexión entre el PPD y el PRD. Toman la información proveniente de un *maestro/esclavo* y solicitan el acceso a la red y cuando éste se les otorga, transmiten un paquete con la información. Las IR, si así es necesario, tienen la posibilidad de ser configuradas con una *Interfaz AXI*, esto las habilita para poder operar con IPCores propios o de terceros que a su vez cuenten con la misma interfaz.

Los *maestros* son elementos del PPD encargados del control de la ejecución de una aplicación o parte de la misma. Son los iniciadores de las transacciones que se transportarán en el sistema de interconexión. Los *maestros* son los encargados de

generar el tráfico que fluirá en el sistema de interconexión de acuerdo al escenario que se plantee. Tienen la posibilidad de comunicarse con la IR por medio de una *interfaz directa* o por medio de una *Interfaz AXI*.

Los *esclavos* son elementos del PPD que responden a la ejecución de una operación solicitada por un *maestro*. Dependiendo del modo en que sean configurados el resultado puede ser transferido fuera del sistema de interconexión, puede ser transferido al *maestro* que inició la transacción, ó puede ser transferido a otro elemento dentro del sistema de interconexión. Al igual que los *maestros*, los *esclavos* pueden comunicarse con la IR por medio de una *interfaz directa* o por medio de una *Interfaz AXI*.

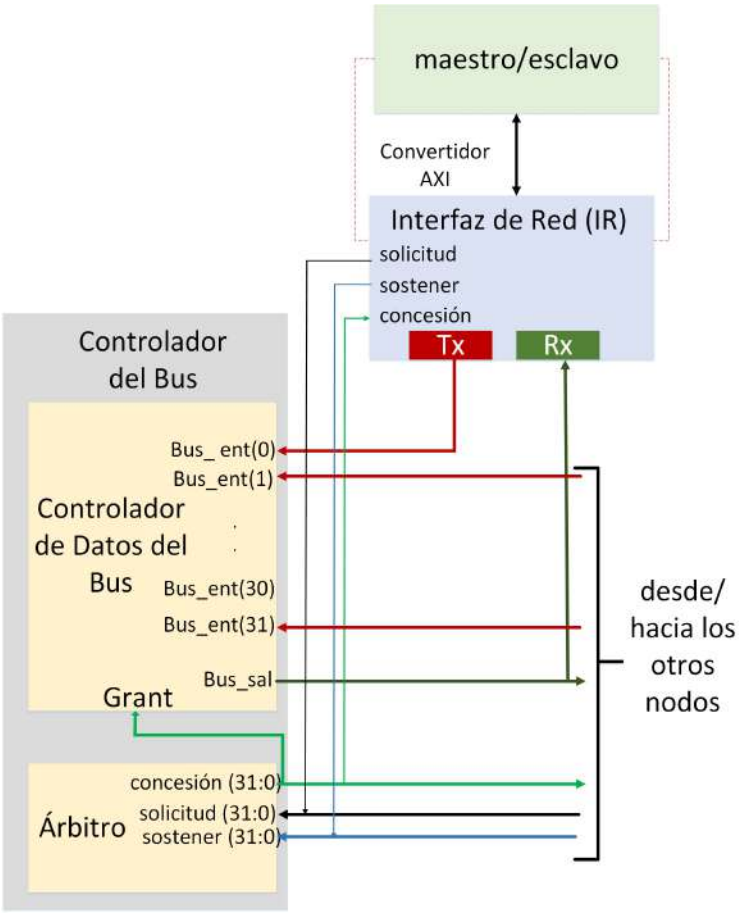


Figura 3.2: Modelo de la Arquitectura de Hardware de la aplicación

Tabla 3.5: Estructura del archivo de dependencias

Por cada tarea	
número	número de la tarea en el grafo original
tareas entrada	apuntador a la lista de tareas de las que es dependiente esta tarea
tareas salida	apuntador a la lista de tareas a las que se tendrá que comunicar esta tarea
Calendario de tareas (ordenado de primera a última tarea que se ejecutará en un maestro)	
tarea	número de tarea en el grafo original
tiempo	tiempo de ejecución de la tarea

3.5. Herramientas Adicionales

Como se ha mencionado con anterioridad, en este estudio se tiene el interés de observar el comportamiento del sistema de interconexión cuando los elementos de procesamiento ejecutan aplicaciones reales. Para probar este comportamiento se decidió utilizar la suite de tráfico generada por Liu et al. (2011). Cuando se ejecuta esta aplicación, se genera una estructura de datos que involucra todos los elementos de la red. Para poder operar el sistema de interconexión, se desarrolló una herramienta que extrae la información específica de la estructura original y genera una estructura para cada *maestro* en el sistema. Esta herramienta toma como fuente la estructura de datos original de la *Suite MCSL NoC* y genera una estructura de datos para cada maestro en el sistema, como la que se presenta en la Tabla 3.5.

3.6. Consideraciones

En el desarrollo de la presenta investigación se realizaron una serie de consideraciones.

La arquitectura de referencia que se toma para el desarrollo de este estudio es la presentada por Sandoval-Arechiga et al. (2017). La razón principal es que desde mi punto de vista, es el trabajo que presenta un marco de referencia integral del concepto SDN aplicado a los SoC. En esta arquitectura se indican los elementos que deben de integrar cada plano y sus relaciones. Sin embargo, el presente estudio se enfoca principalmente en el desarrollo de la infraestructura de *hardware* y los servicios de red que permiten su funcionamiento.

Cuando el uso de los recursos es compartido, el problema principal se presenta cuando este recurso es muy solicitado. Es por esta razón que en varias de las pruebas

que se realizaron en este estudio se consideró saturar el sistema. Esto se logró de dos formas: una, poniendo la *tasa de inyección de paquetes* en uno, lo cual indica que siempre hay un paquete listo para transmitir por parte de los nodos que intervienen en la simulación; dos, poniendo la cantidad suficiente de *maestros* y/o *esclavos* que permitan llevar el sistema a saturación.

Cuando se evaluó el comportamiento del sistema de interconexión para aplicaciones reales provenientes de la *Suite MCSL NoC*, se observó que para la mayoría de las aplicaciones, con las capacidades que tiene el sistema de interconexión, el sistema no llega a saturarse. Esto impide que se pueda observar de forma clara el comportamiento de las políticas de arbitraje con respecto a otorgar un uso diferenciado del *bus*. Solamente dos de las aplicaciones tuvieron la capacidad de saturar el *bus*, por esta razón, fueron las que se seleccionaron para realizar las simulaciones. Las aplicaciones que fueron seleccionadas son: FFT-1024 y FPPPP.

Cuando se evaluó el comportamiento del sistema de interconexión para aplicaciones reales independientes, el tráfico fue simulado de acuerdo a las características de operación del propio IPCore. Las siguientes consideraciones fueron tomadas en cuenta: primera, para la aplicación de *filtrado* se recibe una línea de datos de n bytes, su procesamiento tarda n ciclos de reloj y se genera una respuesta de n bytes; segunda, la aplicación de *cifrado* recibe 16 bytes de datos y genera 16 bytes de datos cifrados después de 12 ciclos de reloj; tercera, el resultado que genera el *compresor* cuando manda a comprimir una línea de n bytes, depende completamente del contexto que se ha ido generando de la imagen. Para este estudio se decidió poner una razón promedio de compresión de $n/2$ bytes, por lo que los paquetes que genera el IPCore que comprime los datos tienen la mitad del tamaño del paquete original que se recibió.

La posibilidad de poder integrar IPCores con una interfaz de comunicación bien conocida como lo es la *Interfaz AXI*, permite que el abanico de posibilidades de elementos de procesamiento que se pueden utilizar en un SoC sea muy amplio. Sin embargo, cuando se agrega esta capa intermedia entre el IPCore y la IR se crea un *overhead* tanto en el uso de recursos como en la operación del sistema. Es necesario hacer un estudio detallado de el rendimiento que se desea del sistema para considerar su utilización.

Capítulo 4

Arquitectura Software Defined Bus-on-Chip

En este capítulo se muestra la arquitectura del sistema propuesto, que trabaja bajo el enfoque de *Redes Definidas por Software*. Inicialmente se muestra la arquitectura general propuesta. Posteriormente se presentan las principales capas que la componen y la relación que hay entre ellas. En este trabajo se pone especial atención en la capa de *infraestructura* ya que es de principal interés mostrar los requerimientos de *hardware* necesarios para soportar un sistema de interconexión que trabaje bajo el modelo SDN. A continuación se detalla la operación del sistema de interconexión. Finalmente se presenta un análisis detallado de los recursos de *hardware* utilizados en la infraestructura.

4.1. Arquitectura General

En el trabajo presentado por Sandoval-Arechiga et al. (2017) se presenta una arquitectura definida por software para la interconexión de múltiples núcleos en un SoC, el propósito es tener una arquitectura que permita optimizar las interconexiones de diferentes aplicaciones ejecutándose simultáneamente. La arquitectura se presenta como un marco de referencia basado en planos, capas e interfaces donde se establecen los lineamientos generales para la administración de un sistema de interconexión en un SoC basado en el modelo SDN. Tomando como marco de referencia este modelo, en este trabajo se hace un estudio de las características y capacidades que deben de proveer los elementos de *hardware* que soportan la arquitectura. Además, se explora las capacidades del modelo para poder ser utilizado en otro tipo de sistema de interconexión, específicamente en uno de tipo *bus*, denominándose la arquitectura propuesta SDBoC. En este estudio se presta especial atención en las características de la infraestructura *hardware* necesaria para operar el sistema de interconexión bajo

la filosofía SDN. Sin embargo, para describir adecuadamente la infraestructura, es necesario tener una visión global del sistema y de las relaciones que existen entre los diferentes elementos de las capas y planos que la componen. Si bien no todas las funciones y servicios están desarrollados, sí se indica las características de los mismos y los elementos de soporte necesarios para su funcionamiento. En la Figura 4.1 se muestra la arquitectura propuesta. SDBoC consta de tres capas, cinco planos, dos *Interfaces de Programación de Aplicaciones* de comunicación y un conjunto de tablas de intercomunicación que a continuación serán descritas.

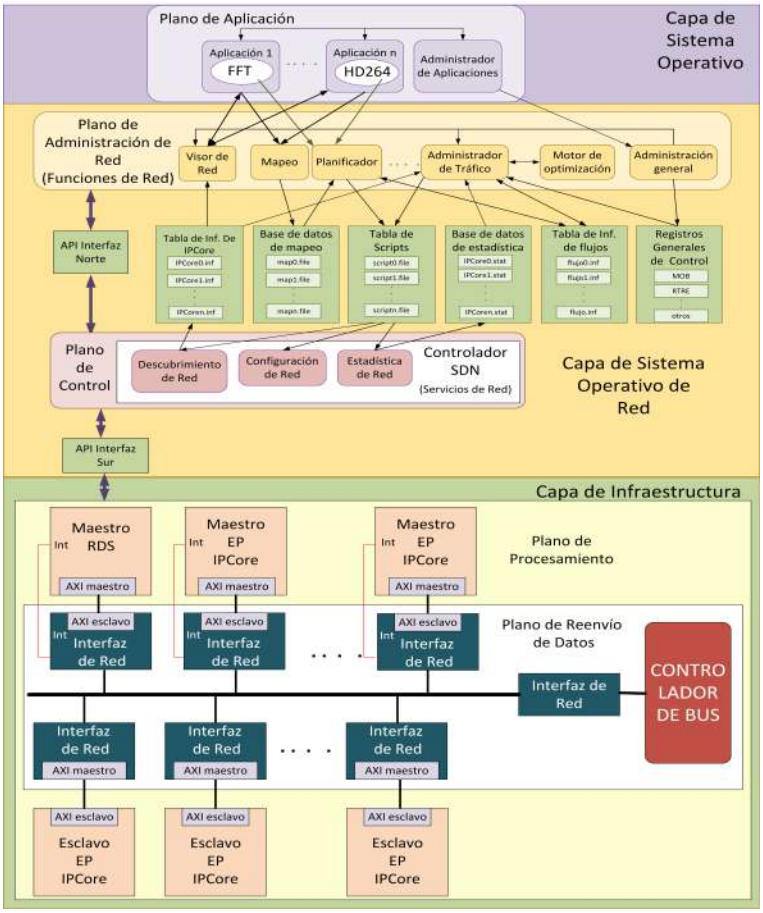


Figura 4.1: Arquitectura Bus-on-Chip bajo el paradigma Definido por Software

4.2. Capa de Sistema Operativo

En la parte alta de la arquitectura se encuentra la capa del Sistema Operativo. Dentro de ella se encuentran las aplicaciones que son ejecutadas por los elementos de

procesamiento en el interior del SoC. Cada una de las aplicaciones de esta capa tienen una abstracción propia del sistema de interconexión que hay en la parte inferior de la arquitectura, y en base a ella conoce las capacidades que tiene disponibles para su ejecución.

4.2.1. Plano de Aplicación

El Plano de Aplicación forma parte de la capa de Sistema Operativo, dentro de el se integran formalmente el conjunto de aplicaciones que pueden hacer uso de la infraestructura SDBoC. Los algoritmos y/o aplicaciones que forman parte de este plano, se comunican con el plano en el siguiente nivel en la arquitectura por medio de llamadas a funciones mediante las cuales pueden conocer y operar los elementos de la arquitectura SDBoC.

Administrador de Aplicaciones

El Administrador de Aplicaciones es el encargado de administrar el conjunto de aplicaciones que pueden operar el SDBoC. Además de las operaciones tradicionales de mantenimiento sobre las aplicaciones (altas, bajas, cambios de aplicaciones, etc), el Administrador de Aplicaciones está encargado de establecer los parámetros generales de la operación del SDBoC.

El paradigma SDN establece que una red deberá tener la capacidad de operar en modo *tradicional* o en modo *SDN*. El Administrador de Aplicaciones es el encargado de establecer el modo de operación del sistema de interconexión. Las aplicaciones en el Plano de Aplicación ajustan su comportamiento de acuerdo al modo de operación establecido. Cuando el *bus* trabaja en modo SDBoC, los parámetros generales de operación del sistema de interconexión, como por ejemplo, la frecuencia con que la estadística de la capa de infraestructura debe ser recolectada, son definidos por el Administrador de Aplicaciones.

Aplicaciones

Son las instancias operadas por los elementos de procesamiento del SDBoC. Cada aplicación tiene una vista propia de la red (topología, número de elementos en la red, características de estos elementos). Con esta visión se decide cómo se configurará los elementos en la red y sus relaciones teniendo en cuenta los requisitos de QoS que se le impongan.

4.3. Capa de Sistema Operativo de Red

La capa del Sistema Operativo de Red es el elemento intermedio en el modelo de referencia. Está compuesta por dos planos: el primero, el Plano de Administración de la Red, donde se concentran un conjunto de Funciones de Red que proveen a las aplicaciones corriendo en el Plano de Aplicación la información necesaria para que puedan hacer uso de los recursos de la red; el segundo, el Plano de Control, que por medio del Controlador-SDN es el encargado de generar los Servicios de Red que permiten la operación de la red. Las Funciones de Red y los Servicios de Red se comunican entre sí por medio de la denominada Interfaz Norte, y un conjunto de registros, tablas y archivos, que son utilizados como elementos intermedios para transferir la información.

Los servicios que las Funciones de Red del Plano de Administración de la Red le solicitan al Controlador-SDN, son realizadas por medio de la Interfaz Norte. Esta interfaz recibe las llamadas en formato *(función,servicio,script*)* y las transfiere al Controlador-SDN. Éste, identifica el Servicio de Red que se solicita y lo manda llamar indicándole el inicio del *script* a ejecutar por medio del apuntador a la denominada Tabla de Scripts. El *script* es ejecutado por el Servicio de Red y el resultado de la conclusión del mismo es indicado a la función origen a través de la misma Interfaz Norte en un formato *(función,estado)*.

Existen diferentes razones por las cuales un *script* tiene que ser ejecutado:

- Después del *reset* general del sistema el Sistema Operativo de Red necesita conocer cuál es el estado inicial del sistema de interconexión. Debe conocer la topología de la red, la cantidad de nodos que hay en el sistema de interconexión, el estado y modo de operación de las IR que hay en el sistema, el tipo de elementos de procesamiento que están atados a las IR, así como su estado y modo de operación inicial.
- Se desea realizar la configuración inicial del sistema de interconexión, donde se establece el modo de operación del SDBoC.
- Es necesaria la configuración de nuevas aplicaciones en el sistema.
- Se desea cambiar los parámetros de operación de una aplicación.
- Se desea obtener la estadística del sistema, ya sea por una tarea programada o porque una función en especial lo solicite.
- Se desea reconfigurar el sistema debido a que un proceso de optimización encontró cómo mejorar el rendimiento del sistema.

- Se detectó una falla de algún elemento de la red.

Los *scripts* de configuración son almacenados en archivos de datos. Las diferentes entradas que puede haber en un archivo de *script* tienen que ver directamente con los Servicios de Red que provee el Controlador-SDN. En la Tabla 4.1 se muestran los catorce servicios que se proveen en la actualidad. Sin embargo, es posible añadir otro tipo de entradas en el archivo de *script* siempre y cuando exista un Servicio de Red para esa entrada. Una de las ventajas de tener una arquitectura bien definida, es que es posible añadir Servicios de Red en el interior del Controlador-SDN en forma de módulos sin que se vea afectada la operación del sistema. La ejecución de un *script* comienza cuando el Controlador-SDN identifica el tipo de servicio que se le está solicitando e invoca a los Servicios de Red. Éste ejecuta el *script* que inicia con la entrada que se localiza en la dirección contenida en el apuntador que recibió por medio de la Interfaz Norte. La rutina de servicio toma en forma secuencial cada una de las entradas de la Tabla de Scripts, identifica la acción que se solicita y genera el paquete que será transmitido a la red para ejecutar dicha acción. La ejecución de un *script* termina cuando el Servicio de Red identifica la entrada *end script*.

Es importante mencionar que dado el sistema de prioridades con que trabaja el Controlador del Bus del sistema, el *maestro* a cargo del Controlador-SDN tiene la segunda prioridad más alta del sistema solo abajo del propio Controlador-SDN, por lo tanto todas las acciones referentes a configuración, control y obtención de estadística son atendidas de manera prioritaria.

El comportamiento general del sistema de interconexión se encuentra regido por un conjunto de registros denominados Registros Generales de Control, que están bajo el control de la función de Administración General en el Plano de Administración de la Red. El registro denominado Modo de Operación del Bus es una bandera donde se establece la forma de operación del bus. Su valor es establecido por el administrador general del SDBoC. Por medio de un Servicio de Red su valor es dado a conocer a todos los elementos de la capa de Infraestructura. Estos almacenan una copia de este valor y supeditan su comportamiento al valor de este registro.

Cuando la función de Administración General establece que el sistema trabaja en modo SDBoC, es posible solicitar la estadística de su comportamiento periódicamente. El registro denominado Registro de Tiempo de Recolección de Estadística almacena este valor en unidades de ciclos de reloj. Este registro será la base de tiempo de un *timer* que interrumpirá periódicamente al Sistema Operativo de Red para que éste, por medio de una Función de Red, solicite el servicio de recolección de estadística.

Para que las aplicaciones estén en capacidad de solicitar el mapeo, programación, y ejecución del algoritmo que las define, es necesario que tengan por lo menos una visión parcial del sistema de interconexión de la red. Esta vista la obtienen por medio de la función denominada Visor de Red, la cual a través de la IN solicita

Tabla 4.1: Entradas que pueden ser generadas a la tabla de scripts. C=Código, B=Byte DM=Dirección Multicast, DR=Dirección de Reenvío, P=peso, M=modo de operación

Entrada	C	B 7	B 6	B 5	B 4	B 3	B 2	B 1	B 0
Descubrimiento de Red	0	0	0	0	0	0	0	0	0
Coloca nodos activos	1	0	0	0	0	31-24	23-16	15-8	7-0
Coloca flujos activos	2	0	0	0	0	0	0	15-8	7-0
Coloca pesos de nodos	3	P 7	P 6	P 5	P 4	P 3	P 2	P 1	P 0
	3	P 15	P 14	P 13	P 12	P 11	P 10	P 9	P 8
	3	P 23	P 22	P 21	P 20	P 19	P 18	P 17	P 16
	3	P 31	P 30	P 29	P 28	P 27	P 26	P 25	P 24
Coloca peso de flujos	4	P 7	P 6	P 5	P 4	P 3	P 2	P 1	P 0
	4	P 15	P 14	P 13	P 12	P 11	P 10	P 9	P 8
Coloca peso de nodo	5	0	0	0	0	0	0	peso	nodo
Coloca peso de flujo	6	0	0	0	0	0	0	peso	flujo
Configura IR	7	0	0	0	0	0	0	0	nodo
	7	DM 3	DM 2	DM 1	DM 0	DR 3	DR 2	DR 1	DR 0
Configura maestro/esclavo	8	M 6	M 5	M 4	M 3	M 2	M 1	M 0	nodo
Lee estado de nodo	9	0	0	0	0	0	0	0	nodo
Lee estado de IR	10	0	0	0	0	0	0	0	nodo
Lee estadística de IR	11	0	0	0	0	0	0	0	nodo
Lee estadística maestro/esclavo	12	0	0	0	0	0	0	0	nodo
Borra Estadística	13	0	0	0	0	0	0	0	nodo
Fin de script	14	0	0	0	0	0	0	0	0

el servicio denominado Descubrimiento de Red. Este servicio ejecuta el *script* que permite conocer qué elementos existen en el sistema de interconexión. El servicio conoce por adelantado las capacidades máximas del sistema, en el caso del SDBoC la cantidad máxima de nodos. Esto le sirve como base para esperar el tiempo suficiente la respuesta de todos los nodos en el sistema.

La función Descubrimiento de Red generará un mensaje de *broadcast* que viajará por el sistema de interconexión solicitando que se identifiquen los elementos que lo recibieron. Como los paquetes responden con la misma prioridad, entonces el servicio Descubrimiento de Red espera un tiempo suficiente por la respuesta de todos los elementos en el sistema. Si no tiene respuesta de algún elemento se entenderá que está dañado o no se encuentra dado de alta (por ejemplo, el caso en que una FPGA en un proceso de reconfiguración parcial no tuvo la capacidad de cargar un nodo). En el *reset* inicial del sistema todos los nodos de la red se inicializan en modo activo.

En la implementación aquí realizada el Controlador-SDN establece un tiempo de espera máximo por la respuesta de todos los nodos de la red de 320 ciclos de reloj (diez ciclos por cada nodo). Este tiempo está holgado si se toma en cuenta que el mensaje inicial de *broadcast* ocupa tres *phit*s y cada nodo en el sistema responde con un mensaje de seis *phit*s donde además de transmitirse intrínsecamente su dirección de red, también transmite su configuración inicial (modo de operación, direcciones de reenvío, direcciones multicast). Además, se transmite el identificador del IPCore asociado al nodo, así como los modos de operación del IPCore.

La información que provee la función Descubrimiento de Red es almacenada en la denominada Tabla de Información de los IPCores. La estructura de cada una de las entradas de esta tabla se puede observar en la Tabla 4.2.

Uno de los pilares del paradigma SDN es su capacidad de trabajar con flujos de información. SDBoC permite este tipo de operación. En el sistema de interconexión que aquí se presenta, puede haber hasta dieciséis flujos operando simultáneamente. Sin embargo, cada IR sólo tiene la capacidad de operar un máximo de cuatro flujos. La ruta que siguen los flujos puede ser establecida por diferentes razones: por un lado, el flujo puede ser establecido cuando la Función de Programación del Plano de Administración de la Red programa un nuevo flujo; por otro lado, la ruta de un nuevo flujo puede ser establecida cuando como resultado de un proceso de optimización la Función de Administración de Tráfico encuentra una mejor forma de operar un flujo. Al igual que todas las operaciones de control que se realizan sobre la infraestructura SDBoC, la programación de flujos puede se llevada a cabo en *tiempo de ejecución*. Los flujos son lógicamente establecidos por las Funciones de Red en la Tabla de Información de Flujos, y físicamente establecidos en los elementos de *hardware* del sistema de interconexión por el servicio de Configuración de Red. La estructura que tiene cada una de las entradas en la Tabla de Información de Flujos se muestra en la Tabla 4.3.

Tabla 4.2: Estructura de la Tabla con Información de maestros/esclavos

Tabla de Información de IPCores (una entrada por cada nodo en la red)	
Nombre	Descripción
Dirección de Red	Es la dirección de la tarjeta de red asociada al IPCore conectado a ella (cada IR tiene una dirección única en la red).
Activo	Indica si el nodo está activo o inactivo (encendido/apagado).
IDIPCore	Identificación del IPCore asociado a la dirección (cada IDIPCore identifica de forma única a un tipo de IPCore).
Modo de Operación	En este campo se establece el modo de operación del IPCore asociado a la IR. Existen cuatro campos de este tipo porque el sistema SDBoC permite trabajar hasta con cuatro flujos independientes, cada uno con su propio modo de operación.
Peso	Es el presupuesto asignado a un nodo con el que compite por el medio.
Dirección de Reenvío	Cuando se está trabajando en modo SDBoC con flujos, ésta es la dirección siguiente en el flujo (a qué nodo se debe entregar el resultado). Existen cuatro campos de este tipo (misma razón que en el campo modo de operación).
Dirección Multicast	Cuando se está trabajando en modo SDBoC varios nodos pueden recibir datos de una misma fuente. Los nodos con una Dirección Multicast común pueden operar sobre el mismo conjunto de datos en forma *múltiples instrucciones, un dato.*

Una de las ventajas que presentan los SoC, es que tienen la posibilidad de que los elementos de procesamiento que se encuentran en su interior, puedan trabajar de forma colaborativa para ejecutar algún algoritmo. Cada uno de los EP puede trabajar en paralelo ejecutando diferentes partes del algoritmo. Ésta es una estrategia que ha sido especialmente aprovechada en el ámbito de las NoC. Un algoritmo o aplicación es dividido en un conjunto de tareas y relaciones que son establecidas para lograr su ejecución. Existen un conjunto de tareas que inician la ejecución del algoritmo. Cuando estas tareas terminan su ejecución, le indican a las tareas que dependen de ellas que han concluido su operación transmitiéndoles los datos que procesaron. Una tarea para su ejecución puede depender a su vez de varias tareas, una tarea no puede iniciar su proceso hasta que todas sus antecesoras hayan terminado y se lo notifiquen. Una tarea que no tiene a quien enviar el resultado de su proceso se dice que es una tarea de finalización.

Una NoC tiene una estructura idónea para el trabajo colaborativo. Cada uno de los EP conectados a ella puede ejecutar un conjunto de tareas mientras otros

Tabla 4.3: Estructura de la Tabla de Flujos

Tabla de Información de Flujos (una entrada por cada flujo en la red máximo 16 flujos)	
Nombre	Descripción
Número de flujo	Es el número con el que los elementos del sistema identifican al flujo (este número es único por flujo).
Activo	Indica si el flujo está activo o inactivo.
Tipo	El SDBoC puede operar dos tipos de flujo: el de reenvío y el multicast
Peso	Es el presupuesto asignado a un flujo con el que compite por el medio.
Primero	Es la Dirección de Red del primer nodo en el flujo, es el nodo que inicia la operación del flujo
Último	Es la dirección del último elemento en el flujo, donde se deposita el resultado de toda la transformación.
quantity	Cantidad de datos que van a ser procesados por el flujo
pointer*	Apuntador a la zona de memoria donde inician los datos con los que va a operar el flujo
Por cada elemento en el flujo (en estricto orden)	
Dirección de Red	Dirección de red del siguiente nodo en el flujo
Modo de Operación	Modo de operación del IPCore asociado a la IR

EP ejecutan otro conjunto de tareas. Las relaciones entre las tareas son establecidas a manera de paquetes de información viajando en la red. Sin embargo, el proceso de dividir las tareas entre los diferentes EP no es trivial. Este proceso denominado *mapeo* se ha convertido en una área muy importante de estudio por la comunidad dedicada al desarrollo de NoCs. La forma en que las tareas son distribuidas en los diferentes EP tiene un impacto significativo sobre el rendimiento de la aplicación. Como se ha mencionado con anterioridad, en este estudio se utilizan los archivos de tráfico generados por Liu et al. (2011) los cuales utilizan un proceso de mapeo basado en el balanceo de cargas. La función del Plano de Administración de la Red denominada *mapeo* toma como entrada el archivo de tráfico generado por Liu et al. (2011) y lo convierte a un conjunto de archivos individuales (uno por cada EP en el mapa) con la información de las tareas, relaciones y datos con los que va a operar. Esta información es almacenada en la denominada Base de Datos de Aplicaciones Mapeadas. Posteriormente, la aplicación solicita al *planificador* que cree un *script* de configuración en base a las entradas en la Base de Datos de Aplicaciones Mapeadas. El *planificador* realiza esta operación generando tantas entradas como *maestros* intervengan en el proceso de mapeo. La entrada en la Tabla de Scripts para realizar

Tabla 4.4: Estructura del Archivo de Dependencias de un elemento de Procesamiento en el SDBoC

Estructura del archivo de mapeo (una entrada por cada Elemento de Procesamiento en la Aplicación)	
Nombre	Descripción
Dirección de Red	dirección de red de la IR asociada al maestro
Modo de Operación	modo de operación del IPCore asociado a la IR (0 no hay dependencias / 1 existen dependencias)
Número de Tareas	cantidad de tareas que se ejecutan es este procesador
planificación*	apuntador a la tarea inicial en el calendario
Tareas (una entrada por cada tarea)	
Número de Tarea	número de la tarea en el grafo original
tareas_entrada*	apuntador a la lista de tareas de las que depende esta tarea para su ejecución
tareas_salida*	apuntador a la lista de tareas con las que se comunicará esta tarea cuando termine su ejecución
puntero_datos*	apuntador al área de datos con los que será operada la tarea, si es una tarea inicial (tareas_entrada*=null)

este mapeo es *configura maestro*. Una vez terminado este proceso, se solicita que el servicio Configuración de Red ejecute la configuración en los elementos de la infraestructura del sistema. La estructura que sigue cada archivo de mapa se muestra en la Tabla 4.4.

Otra de las características principales del paradigma SDN es la posibilidad de poder conocer en cualquier momento el comportamiento del sistema de interconexión. Esto se lleva a cabo por un proceso de recolección de estadística en cada elemento que pertenece a la red. La información que provee el proceso de recolección, sirve como base para que funciones de red como la denominada Administración de Tráfico, por medio de un algoritmo de optimización, pueda mejorar las prestaciones de las aplicaciones corriendo en el SoC. La estadística puede ser recolectada a petición expresa de una Función de Red, o puede ser programado el proceso de recolección periódico de la estadística por medio del registro Registro de Tiempo de Recolección de Estadística. En el caso de SDBoC se genera un archivo de estadística cada vez que el Registro de Tiempo de Recolección de Estadística se agota. El archivo contiene información de todos los elementos del sistema de interconexión, incluyendo controladores, flujos, interfaces y elementos de procesamiento. El tipo de estadística que es recolectado por cada elemento es mostrado en la Tabla 4.5.

Cuando un Servicio de Red es invocado, éste toma las entradas que hay en la Tabla de Scripts, las transforma en un formato tipo paquete y las envía por medio

Tabla 4.5: Estadística Generada por el SDBoC por Tipo de Elemento. Todos los tiempos son acumulados a partir del último *borrado* de la estadística

Estadística del Controlador del Bus	
Nombre	Descripción
tiempo de bus ocupado	tiempo que el bus ha permanecido ocupado (en ciclos de reloj)
tiempo de bus libre	tiempo que el bus ha permanecido desocupado (en ciclos de reloj)
flujos en operación	número de flujos corriendo en el SDBoC desde la última reconfiguración
Estadística de Flujos (una entrada por cada flujo)	
Nombre	Descripción
número	número de flujo
tiempo de uso del bus	tiempo que el flujo ha utilizado el bus (en ciclos de reloj)
número de paquetes procesados	cantidad de paquetes que han sido procesados por este flujo.
Estadística de Interfaz de Red (una entrada por cada IR)	
Nombre	Descripción
dirección de Red	dirección de la IR
tiempo de espera en buffer de entrada	tiempo de espera en el buffer de entrada para ser atendido por el IPCore (en ciclos de reloj, una entrada por cada flujo, solo es usado en modo SDBoC)
tiempo de espera en buffer de salida	tiempo de espera en el buffer de salida antes de poder tener acceso al medio (en ciclos de reloj, una entrada por cada flujo, solo es usado en modo SDBoC)
tiempo de uso del bus	tiempo que la NI ha utilizado el bus (en ciclos de reloj, una entrada por cada flujo, en modo tradicional solo reporta un valor)
número de paquetes	número de paquetes recibidos (una entrada por cada flujo, en modo tradicional solo reporta un valor)
Estadística de maestros/esclavos (una entrada por cada maestro/esclavo	
Nombre	Descripción
dirección de red	dirección de la IR a la que está asociado el IPCore
paquetes procesados	número de paquetes procesados (una entrada por cada flujo)
tiempo de ejecución	tiempo que ha pasado el IPCore procesando información (una entrada por cada flujo)

de la Interfaz Sur a la cola de espera para que las instrucciones, convertidas en paquetes, sean transmitidas por el sistema de interconexión.

4.4. Capa de Infraestructura

Los paquetes son la unidad de transferencia entre los elementos que conforman la infraestructura de hardware. Por lo que antes de comenzar con la descripción de los elementos propios de *hardware*, es importante conocer la estructura que estos tienen.

4.4.1. Estructura de los paquetes de datos

La estructura de los paquetes que viajan por el sistema de interconexión se muestra en la Figura 4.2. En esta implementación el tamaño de los *phits* y *flits* es el mismo, 66 bits. Todos los paquetes que fluyen en la infraestructura están compuestos por un *flit* de *cabecera*, al menos un *flit* de *carga útil* y un *flit* de *cola*. El *flit* de *cabecera* es sustituido por un *flit* de *reconocimiento* en el caso de trabajar con flujos encadenados.

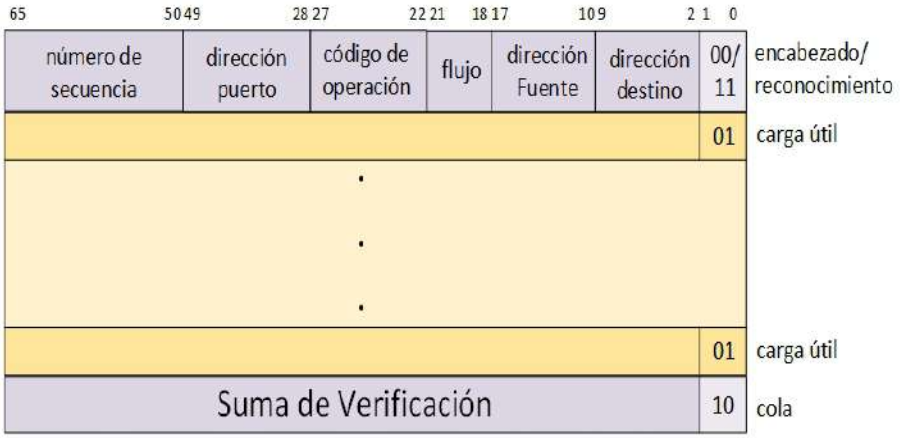


Figura 4.2: Estructura general de un paquete de datos

Los primeros dos bits de cada *flit* identifican su tipo. El *flit* de cabecera tiene un código 00b. Además, el *flit* de cabecera está compuesto por los siguientes campos:

- **dirección destino**: es la dirección de red, de la Interfaz de Red, a la que se le está enviando el paquete.
- **dirección fuente**: es la dirección de red, de la Interfaz de Red, donde se origina el paquete.

- **flujo**: cuando se está trabajando en modo SDBoC, indica el flujo al que pertenece el paquete. El flujo 0 siempre es considerado como el flujo general de configuración.
- **código de operación**: cuando se está trabajando en modo SDBoC, es el código de la operación que se debe de realizar como consecuencia de recibir este paquete.
- **dirección de puerto**: cuando se está trabajando en modo SDBoC, es la dirección de memoria de donde la interfaz AXI va leer o escribir la información.
- **número de secuencia**: cuando se está trabajando en modo SDBoC, indica el número de paquete, dentro de un flujo, que se está transmitiendo. Para indicar el paquete final en el flujo este campo es llenado con unos.

Después del *flit* de cabecera se encuentran los *flits* que corresponden a los datos que serán transmitidos, el conjunto de estos datos es denominado *carga útil*. Los *flits* de la *carga útil* se identifican con el código 01b. Finalmente, todos los paquetes terminan con un *flit* de *cola*. En este *flit* se incrusta el código de la suma de verificación que se ha calculado como la suma del contenido de todos los *flits* dentro del paquete y que servirá al receptor para verificar la integridad del paquete. El código para los paquetes de *cola* es 10b.

Adicionalmente se han generado otro conjunto de paquetes que se denominan paquetes de *reconocimiento*. Estos paquetes se utilizan cuando el sistema de interconexión trabaja en modo SDBoC. Los paquetes indican al emisor de un paquete de datos, que el envío que realizó por medio de un flujo, se ha procesado y que se tiene la posibilidad de procesar otro paquete de este flujo. A diferencia de los paquetes de cabecera normales, estos tienen un código 11b.

En lo que se refiere propiamente a la Capa de Infraestructura, ésta es una capa que está totalmente construida en *hardware*. Dentro de ella se encuentran tanto los elementos de procesamiento como los elementos de interconexión. La arquitectura que aquí se propone trabaja bajo el paradigma SDN, y se encuentra divida en dos planos:

4.4.2. Plano de Procesamiento de Datos

Dentro del Plano de Procesamiento de Datos se encuentran todos los elementos de procesamiento del SoC. Aunque cada uno de estos elementos tiene sus propias características, desde el punto de vista del sistema de interconexión son elementos que se comportan de forma similar: envían y reciben paquetes de información. Sin embargo, por las funciones que realizan y la relación que tienen con el sistema de interconexión, se identifican tres tipos de elementos de procesamiento:

1. **maestro**: es un elemento de procesamiento capaz de ejecutar algoritmos, con la particularidad que puede solicitar la ejecución de una o varias tareas a otros elementos que se encuentran en el sistema de interconexión. Principalmente integrado por procesadores capaces de realizar tareas de propósito general o específicas.

2. **maestro-SDN**: es un elemento del tipo *maestro* pero con la peculiaridad que es el elemento dentro del sistema de interconexión que es el encargado de la administración del mismo. Sus diferencias con respecto a un *maestro* se encuentran a nivel de *software*. Éste es el elemento de *hardware* que soporta las capas superiores de la arquitectura SDN. Como es el principal elemento de control del sistema se le asigna la máxima prioridad dentro del sistema de interconexión. Es decir, cuando requiere atención del medio, éste le es asignado dado que tiene la máxima prioridad estática (sólo después del mismo controlador del Bus).

3. **esclavo**: este tipo de elemento, generalmente especializado, ejecuta procesos a petición de un *maestro*. Es decir, solamente opera cuando un elemento externo se lo solicita. Dentro de este tipo de elementos principalmente se encuentran: IPCores, aceleradores de hardware, controladores de memoria.

Con anterioridad se ha mencionado que la industria constantemente presiona por tener nuevos desarrollos en el mercado. Para lograr esto, se plantean estrategias de reuso de dispositivos ya sean propios o de terceros. Sin embargo, es necesario que estos dispositivos cuenten con interfaces de comunicación comunes. En este trabajo se propone que la interfaz que una el plano de procesamiento de datos con el plano de reenvío de datos sea la interfaz AXI de ARM (2010). La decisión de adoptar esta interfaz, es tomada principalmente por la gran difusión que tiene a nivel global.

maestro-SDN/maestro con Interfaz AXI

La estructura de *hardware* de un elemento tipo *maestro* y la de un elemento tipo *maestro-SDN* es similar. Su principal diferencia radica en el *software* que ejecutan. Mientras el *maestro* se encarga de la ejecución de aplicaciones, el *maestro-SDN* tiene a su cargo la administración del sistema de interconexión. En la Figura 4.3 se muestra la arquitectura de este tipo de elementos. En un elemento tipo *maestro-SDN* se integra la denominada *Pila-SDN* donde residen las capas superiores del modelo SDBoC. El proceso que siguen los elementos tipo *maestro-SDN* y *maestro* para comunicarse con el sistema de interconexión es el mismo, solo habilitados por la ejecución de aplicaciones diferentes. A continuación se detallará el proceso que sigue un elemento del tipo *maestro-SDN* donde se incluyen los elementos de *software* de la *Pila-SDN*.

Desde el punto de vista del *maestro-SDN* una transacción comienza cuando una Función de Red solicita al Plano de Control la ejecución de un Servicio de Red de acuerdo a un *script* previamente almacenado en la Tabla de Scripts. El servicio, toma las entradas en la tabla y las prepara a modo de paquete: arma el encabezado del paquete, agrega los datos que serán transmitidos y calcula la totalidad de datos a transmitir. Los Servicios de Red utilizan la Interfaz Sur para comunicarse con la Capa de Infraestructura. La Interfaz Sur hace uso de los manejadores AXI-Write y AXI-Read para entablar comunicación con la IR a la que está unido. La interfaz AXI que se utiliza en este trabajo es la interfaz AXI-Full la cual es del tipo *mapeo de memoria*. Bajo esta perspectiva, cuando un *maestro* necesita transmitir información a la IR, realiza una operación de escritura a un conjunto de direcciones de memoria en la IR. Y cuando el *maestro* necesita obtener información de la IR realiza una operación de lectura. Los paquetes que viajan desde y hacia el *maestro-SDN* y que fluyen por la interfaz AXI contienen: datos, configuraciones, estados o estadísticas. Sin embargo, desde el punto de vista de la interfaz AXI el contenido de estos paquetes no tiene ningún significado, la interfaz tiene como única función el transporte de los datos entre el Plano de Procesamiento de Datos y el Plano de Reenvío de Datos.

El proceso de transmitir el paquete desde el *maestro-SDN* a la IR inicia con una operación de escritura utilizando el manejador AXI-Write. Este manejador tiene como finalidad preparar toda la información para que la interfaz pueda operar. Por medio de operaciones de escritura realizadas a registros de *hardware* específicos configura la operación de escritura. En el Registro de Dirección AXI, coloca la dirección destino de los datos en el *esclavo AXI*, en este caso la IR. En el Registro de Dirección de Almacenamiento, coloca la dirección de origen de los datos, es decir donde está almacenado el paquete en la memoria del *maestro-SDN*. En el Registro de Longitud, se indica la cantidad de datos que serán transmitidos por la operación de escritura. Posteriormente, por medio del Registro de control de memoria, el *maestro-SDN* transfiere el control de uso de la memoria al Maestro de escritura AXI-Full. Finalmente autoriza que se ejecute la operación de escritura activando el registro de Inicio AXI escritura. Cuando el Maestro de escritura AXI-Full termina de realizar la operación, avisa al *maestro-SDN* la conclusión de la operación por medio de la señal de *fin*. La señal *fin* se conecta por medio de una línea de interrupción a la Unidad Central de Procesamiento del *maestro-SDN*. Cuando el *maestro-SDN* detecta la señal de interrupción, identifica que la operación de escritura se ha efectuado y está en condiciones de preparar otra transmisión. En la Figura A.1 se muestra el diagrama de la Máquina de Estados Finitos de la interfaz Maestro de escritura AXI-Full de un elemento tipo *maestro-SDN/maestro*. Es importante mencionar que aunque la interfaz de comunicación entre el *maestro* y la IR es AXI, en el sistema de interconexión la información va dirigida a una IR destino, la dirección de la IR está contenida en el *flit* de cabecera que fue establecido por el Servicio de Red al

momento de formar el paquete.

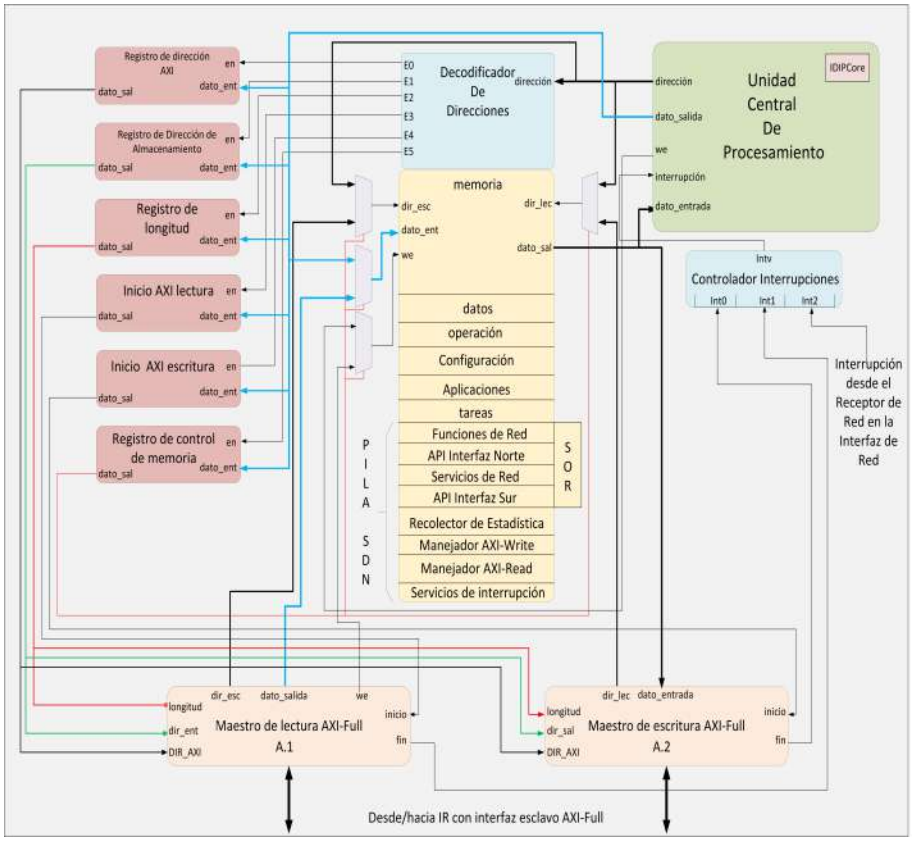


Figura 4.3: Arquitectura de un maestro con Interfaz AXI

En la filosofía *maestro/esclavo* una transacción sólo puede ser iniciada por un *maestro*. En la arquitectura que aquí se propone, los elementos del tipo *maestro* son los encargados de iniciar las transacciones ya que ellos son los que contienen la interfaz maestra AXI-Full. Entonces cuando llega un paquete a la IR atada a un *maestro*, ésta no puede avisarle por medio de la interfaz AXI-Full del arribo de la información. La forma en que le comunica el arribo de un paquete es por medio de una señal de interrupción. Cuando el *maestro-SDN* detecta la señal de interrupción proveniente de la IR entiende que ha llegado un paquete con destino a él. Sin embargo, el *maestro-SDN* no conoce el tamaño del paquete ni el flujo de trabajo al que éste llegó. Para conocer esta información el *maestro-SDN* efectúa una primera operación de lectura dirigida a una dirección AXI conocida por el Esclavo de lectura AXI-Full en la IR. Éste a su vez solicita a la Unidad de Control de la IR le suministre la información del flujo que se va a operar y la cantidad de datos que arribaron; estos datos son suministrados al *maestro-SDN* dando por concluida

las operación de lectura AXI. Cuando el *maestro-SDN* obtiene esta información, procede a realizar una segunda operación de lectura, pero ahora dirigida al flujo que se le indicó y con la cantidad de datos que necesita leer.

Cualesquiera que sea la razón por la que el *maestro-SDN* necesita realizar una operación del lectura sobre el puerto AXI, ésta la hace utilizando el Manejador AXI-Read. Para que el manejador pueda operar es necesario con antelación establecer ciertos valores en los registros asociados para su operación. En el Registro de dirección AXI se establece la dirección de donde se va a leer la información de la IR. Si la operación es de lectura del *estado*, en este registro se pone la dirección de memoria en la que la IR tiene reportado el estado de sus banderas de control. Si la operación es de lectura de información, en este registro se pone la dirección del *buffer* de entrada del cual se quiere leer la información del paquete que arribó. El número de datos que deben ser leídos se establece en el Registro de Longitud y la dirección donde serán almacenados los datos se indica en el Registro de Dirección de Almacenamiento. El *maestro-SDN* asigna el control de la memoria al Maestro de lectura AXI-Full por medio del Registro de control de memoria. La operación de lectura comienza cuando el *maestro-SDN* lo indica por medio del registro Inicio AXI lectura. Cuando la operación de lectura termina, el Maestro de lectura AXI-Full genera una interrupción a la Unidad Central de Procesamiento indicando su finalización. En la Figura A.2 se muestra la Máquina de Estados Finitos del Maestro de lectura AXI-Full de un elemento tipo *maestro-SDN/maestro*.

Como cualquier elemento dentro del sistema de interconexión, el *maestro-SDN* tiene la capacidad de recabar estadística, que en este caso consiste en reportar el número y tipo de servicios que ha ofrecido y el tiempo que ha permanecido ocupado generando estos servicios. La rutina encargada de realizar esta función es la denominada Recolector de Estadística, la cual es invocada por la Unidad Central de Procesamiento al inicio y al final de una transacción de lectura o escritura almacenando en contadores internos el tiempo que duró la transacción e incrementando el número de transacciones ejecutadas.

Como anteriormente se ha mencionado, en un modelo *maestro/esclavo* todas las transacciones son iniciadas por el *maestro*. En la arquitectura aquí presentada el *maestro-SDN* es el *maestro* de la transacción AXI y la IR es el *esclavo* de la transacción AXI. Entonces, cuando llega un paquete a la IR atada al *maestro-SDN*, la forma en que la IR indica al *maestro-SDN* del arribo del paquete es por medio de una señal de interrupción. La Unidad Central de Procesamiento del *maestro-SDN* tiene tres fuentes de interrupción: una, proveniente del Maestro de escritura AXI-Full, que le indica la finalización de una transacción de escritura AXI; dos, proveniente del Maestro de lectura AXI-Full, indicándole la finalización de una transacción de lectura AXI; tres, proveniente de la IR atada a él. Cuando esto último sucede, el *maestro-SDN* genera una primera operación de lectura por la interfaz AXI solicitando el

estado de la IR. La respuesta que obtiene le indica los flujos que están solicitando su atención y la cantidad de datos que llegaron por el flujo. Con esta información el *maestro-SDN* puede efectuar una segunda lectura a la IR solicitando los datos específicos del flujo que generó la interrupción. El proceso mencionado anteriormente es realizado por el Servicio de Interrupción asociado a la interrupción de la IR. Este servicio es capaz de almacenar servicios de interrupción, es decir, si al momento de estar atendiendo una interrupción por un flujo, arriba otra interrupción por otro flujo, la almacena y la atiende cuando termina con la transacción actual.

Bajo el esquema SDBoC un elemento del tipo *maestro* se comporta de forma muy similar al *maestro-SDN* en su proceso de comunicación con el sistema de interconexión, solamente que en este caso se eliminan los elementos correspondientes a la Capa de Sistema Operativo. La arquitectura de *hardware* al igual que los *drivers* AXI son los mismos. Los servicios de interrupción atienden a las mismas fuentes de manera similar. En cuanto a la estadística generada se adiciona un elemento que es el porcentaje de uso del *maestro*. Con esta estadística se puede conocer si el *maestro* tiene poca o excesiva carga de trabajo y una instancia superior en el modelo, puede tomar acción al respecto.

Una acción que se realiza al momento del *reset* inicial del sistema es que la IR atada a cada *maestro/esclavo* del sistema, necesita identificar el tipo de elemento que tiene atado – o no – a ella. Esto es necesario porque todos los nodos deben de tener esta información disponible en el momento que el *maestro-SDN* les solicite que se identifiquen. Entonces, cada una de las IRs además de indicarle su propia información, le indica el *ID* del elemento que tiene atado a ella. Para lograr lo anterior, el *maestro* realiza una operación inicial de escritura sobre la interfaz Maestro de escritura AXI-Full donde le indica a la IR que le está enviando su identificación.

Esclavo con Interfaz AXI

Un elemento del tipo *esclavo* se comunica con la IR por medio de operaciones de lectura y escritura a través de la interfaz AXI. Las operaciones de escritura que pueden ser dirigidas al *esclavo* son: configuración, operación, e inicialización de estadística. Las operaciones de lectura que puede ser realizadas son: identificación, obtención de resultado, y obtención de estadística.

Cuando un *esclavo* es inicializado, la primera acción que realiza el IPCore en su interior es colocar en una dirección de memoria específica su *ID* (esta dirección es conocida por la IR atada al *esclavo*). A continuación inicializa los registros correspondientes al control estadístico. El proceso estadístico en un IPCore principalmente tiene que ver con el porcentaje de uso del mismo. La estadística es actualizada cada ves que se hace uso del IPCore, y su resultado es almacenado en una dirección de memoria que debe de ser conocida por las capas superiores del SDBoC. Lo anterior

con el fin de que al momento que deseen conocer la estadística de este elemento sepan sobre cuál dirección se debe de hacer la operación de escritura. Así mismo, debe ser conocida la dirección donde un IPCore almacena los resultados de una operación, ya que después de operarlo la IR debe leer los resultados de esta dirección. Las direcciones necesarias para realizar el proceso de escritura están integradas en la cabecera del paquete de datos, siendo cada una de estas diferente para los tres tipos de operaciones que se realizan. Esta información debe estar disponible para los Servicios de Red después de haber solicitado el servicio a los Dispositivos de Reenvío y habérselo dado a conocer a las capas superiores del SDBoC. En la Figura 4.4 se puede observar la arquitectura de un elemento *esclavo* con interfaz AXI. La principal diferencia que se encuentra en la arquitectura general de un *esclavo* con respecto a la arquitectura de un *maestro* es que en este caso, el *maestro* de la interfaz AXI, se encuentra en la IR.

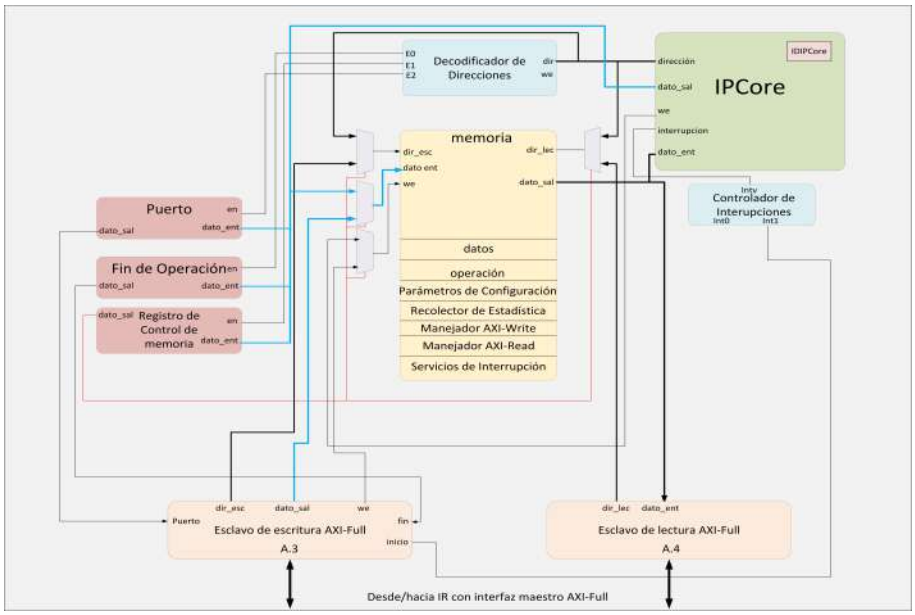


Figura 4.4: Arquitectura de un esclavo con Interfaz AXI

La forma en que el IPCore identifica qué operación tiene que realizar, está basada en la dirección AXI a la que va dirigida la escritura. Entonces antes de comenzar las operaciones propias de lectura/escritura, el IPCore establece una dirección de memoria donde almacenará la dirección inicial de la escritura que se está realizando. Esta dirección le es dada a conocer al Esclavo de escritura AXI-Full por medio del registro Puerto. Cuando llega una transacción de escritura el Esclavo de escritura AXI-Full además de almacenar el paquete que llegó, guarda en la dirección indicada en Puerto la dirección inicial destino del paquete. Posteriormente el Esclavo

de escritura AXI-Full le indica al IPCore, por medio de una interrupción, que ha llegado una transacción y que necesita ser atendida. El Esclavo de escritura AXI-Full mantiene la transacción pendiente de terminar hasta que el IPCore le responde que ha finalizado de procesar la transacción por medio de la señal de *fin*. Cuando esto sucede, manda la respuesta de transacción completada al Maestro de escritura AXI-Full en la IR. La Máquina de Estados Finitos de la interfaz Esclavo de escritura AXI-Full de un elemento tipo *esclavo* se muestra en la Figura A.3.

Por su parte cuando el IPCore detecta una interrupción, lo primero que hace es identificar el tipo de operación que se desea realizar. Para saber esto, lee el contenido de la dirección que está almacenada en el registro Puerto el cual le indica la dirección a la que fue dirigido el paquete. El mapa de memoria de cada IPCore es específico pero debe de contar con al menos las siguientes áreas: de configuración, el IPCore se debe de configurar con los parámetros establecidos a partir de esta dirección; de operación, el IPCore debe de realizar su operación con los datos que están almacenados a partir de esta dirección; de inicialización de estadística, el IPCore debe de borrar todos los contadores que tenga asociados a la estadística. Por las características propias de cada tipo de operación, cada una genera resultados de tamaño diferente. El tamaño del resultado, y la dirección donde se almacenó el resultado, deben de ser almacenados en una dirección fija conocida por la IR. Una operación no es dada por terminada por el IPCore hasta que ha sido completamente ejecutada, los resultados han sido almacenados y el tamaño de la respuesta ha sido grabado. Cuando esto sucede se le indica al Esclavo de escritura AXI- Full por medio de la señal de *fin*.

El Esclavo de lectura AXI-Full tiene como función realizar lecturas de la memoria del *esclavo* y transferirlas al Maestro de lectura AXI-Full en la IR. Cuando el IPCore termina cualquier operación que está realizando, transfiere el control de lectura y escritura a los puertos AXI. Cuando se le solicita al Esclavo de lectura AXI-Full que lea información, éste transfiere tantos datos como se le hayan indicado a partir de la dirección AXI de escritura. El diagrama de la Máquina de Estados Finitos que genera el Esclavo de lectura AXI-Full de un elemento tipo *esclavo* se muestra en la Figura A.4.

4.4.3. Plano de Reenvío de Datos

Este plano es el encargado de transportar los datos entre los diferentes elementos de procesamiento del SoC. Está compuesto de dos tipos de elementos: uno, las Interfaces de Red; dos, el Controlador del Bus. Debido a la forma en que se comunican con sus elementos de procesamiento, existen dos variaciones de las Interfaces de Red: primera, Interfaz de Red Maestra (IRM); segunda, Interfaz de Red Esclava (IRE). A continuación se dará una descripción detallada de los elementos del Plano

de Reenvío de Datos.

Interfaz de Red Maestra

La IRM es el elemento de *hardware* encargado de transmitir y recibir los paquetes dirigidos a un elemento del tipo *maestro* en el SoC. La comunicación de la IRM con el *maestro* se lleva acabo utilizando la interfaz AXI-Full. Debido a que un *maestro* siempre inicia la transacción, los elementos AXI en la IRM son del tipo *esclavo*. La comunicación con el sistema de interconexión se logra por medio de dos elementos: uno, el Receptor de Red (RR); dos, el Transmisor de Red (TR). En SDBoC la IR además de cumplir con el objetivo de mandar y recibir datos del *maestro*, debe de cumplir con otro conjunto de tareas. Para poder llevarlas a cabo integra un conjunto de registros, memorias y unidades especializadas de procesamiento. Los elementos que integran la Interfaz de Red Maestra se muestran en la Figura 4.5 y son detallados a continuación.

El Receptor de Red es el elemento encargado de recibir los paquetes que van dirigidos a la IR. Algunas de las capacidades que tiene el Receptor de Red son:

- Se encuentra habilitado para recibir paquetes de tipo *broadcast*, *multicast* o *unicast*.
- Permite operar a la IR en modo *tradicional* o en modo SDBoC.
- Solo permite recibir paquetes si el nodo está activo. En caso que el nodo no se encuentre activo solo recibe paquete si proviene del *maestro-SDN* y es de activación.
- Establece el tiempo de llegada de cada paquete, que es utilizado para la generación de la estadística.

Cuando el Receptor de Red identifica que un paquete que está en el *bus* va dirigido a él, extrae del *flit* de cabecera el flujo al que pertenecen los datos y almacena el paquete en el *buffer* asociado en el Buffer Múltiple de Entrada que se muestra en Figura 4.6. Posteriormente, el Receptor de Red activa la señal *ponRDPO* y establece el tamaño de la carga útil del paquete en su salida *datosPO*. Esta última acción tiene como objetivo indicarle a la Unidad de Control que ha llegado un paquete y se tiene que tomar acción sobre él. Posteriormente el Receptor de Red identifica si el paquete va dirigido a la propia IR o si va dirigido al *maestro* atado a ella. En caso de que sea lo último, genera una señal de interrupción dirigida al *maestro*. El diagrama de la Máquina de Estados Finitos del Receptor de Red se muestra en la Figura A.5.

El Transmisor de Red es el encargado de transmitir los datos al Sistema de Interconexión. Éste se encuentra verificando constantemente (cada ciclo de reloj) el

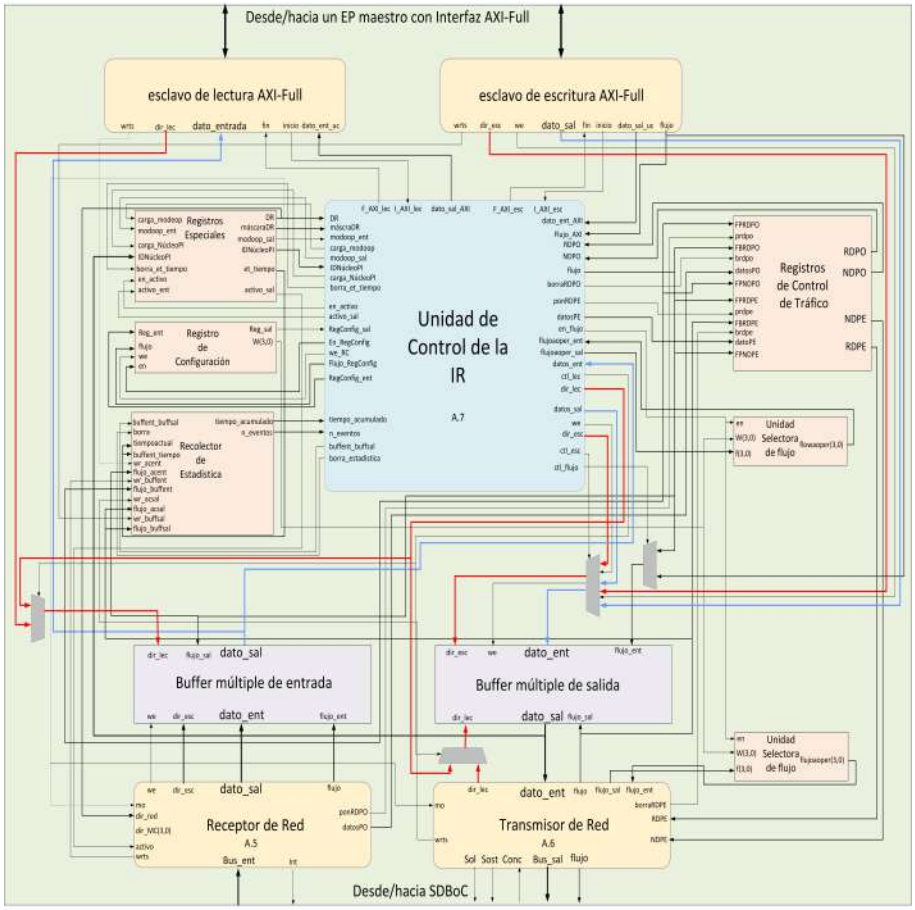


Figura 4.5: Arquitectura de la Interfaz de Red de un elemento de procesamiento tipo maestro

valor de la señal *Registro de Datos Por Enviar (RDPE)* proveniente de la unidad de Registros de Control de Tráfico (RCT). Cuando el Transmisor de Red identifica que en el vector RDPE hay uno o más flujos que solicitan la transmisión de un paquete, procede a solicitarle a la Unidad Selectora de Flujo que le indique cuál de los flujos que tienen que salir, es el que tiene más prioridad. Una vez conocido el flujo prioritario, el paquete que se va a transmitir lo selecciona del Buffer Múltiple de Salida que se muestra en la Figura 4.7 y al mismo tiempo por medio de la unidad de Registros de Control de Tráfico a través del vector de registros Número de Datos Por Enviar (NDPE) obtiene la cantidad de datos que deberán ser transmitidos. Con esta información, el Transmisor de Red está en condiciones de transmitir el paquete una vez que el acceso le haya sido permitido.

El Transmisor de Red se encuentra habilitado para operar ya sea en modo *tradi-*

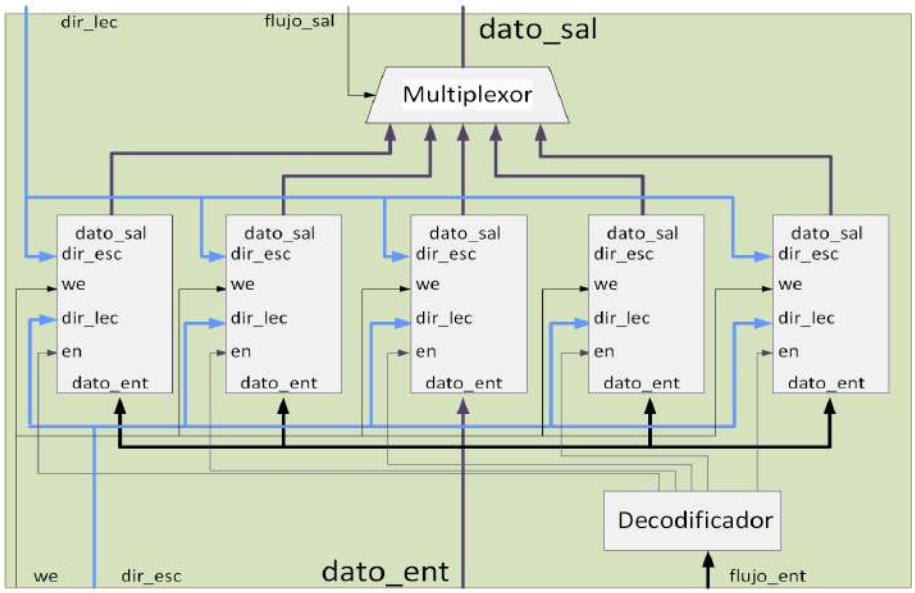


Figura 4.6: Diagrama del Buffer Múltiple de Entrada

cional o en modo SDBoC. El modo en que éste puede operar es establecido por un paquete de configuración proveniente del *maestro-SDN* a solicitud de una Función de Red. Además, con el fin de contabilizar el tiempo de espera del paquete en el *buffer* de salida, genera la señal *wrts* para que el Recolector de Estadística registre el tiempo y actualice la estadística. El diagrama de la Máquina de Estados Finitos que controla al Transmisor de Red se muestra en la Figura A.6.

La Unidad Selectora de Flujo asociada al Transmisor de Red tiene como función indicar a cuál de los flujos que están solicitando enviar un paquete por el *bus* se le otorga la posibilidad de transmitir. La decisión la toma un árbitro del tipo WRR, como el definido en la sección 2.4.3, en base a los pesos que previamente le fueron establecidos. El árbitro aquí diseñado presenta una modificación del árbitro original presentado por Dally and Towles (2003). Esta modificación permite el acceso a un *solicitante* cuando el peso de éste se encuentre en cero, siempre y cuando ningún otro *solicitante* con peso diferente de cero esté solicitando el acceso. Además la USF en lugar de proporcionar una *concesión* para el flujo seleccionado, proporciona el número de flujo que obtiene la posibilidad de transmitir. En la Figura 4.8 se muestra el diagrama de esta unidad.

La IR cuenta con un conjunto de registros de propósito específico, denominados Registros Especiales, que le permiten operar en modo SDBoC. Estos registros son mostrados en la Figura 4.9 y su finalidad es indicada a continuación:

- **Dirección de Red**, éste es un registro establecido en el proceso de síntesis

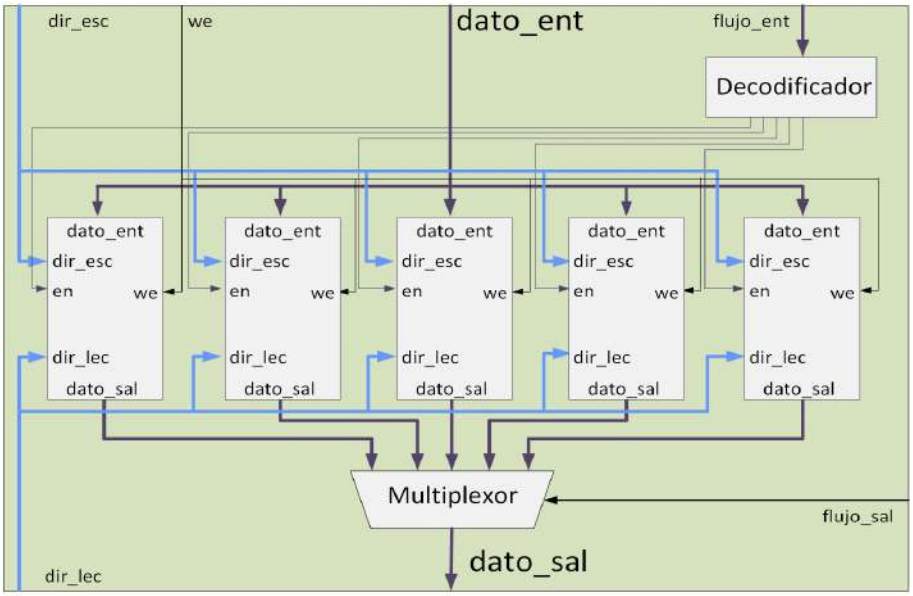


Figura 4.7: Diagrama del Buffer Múltiple de Salida

del circuito y es el que identifica de forma única a la IR.

- **Máscara de Dirección de Red**, es un registro de 32 bits en el que el bit que se encuentre encendido corresponde a la Dirección de Red.
- **ID-IPCore**, en este registro se almacena el identificador del IPCore asociado a esta IR.
- **contador de tiempo**, es un contador que lleva la cuenta global de ciclos de reloj que han transcurrido desde que el sistema fue inicializado. Es utilizado principalmente para el manejo de la estadística de la IR. Este registro puede ser re/inicializado por el *maestro-SDN* a través de una Función de Red.
- **Modo de Operación**, en este registro se establece el modo de operación del sistema, *tradicional* o SDBoC. Este registro puede ser configurado por el *maestro-SDN* a través de la Función de Red de Configuración.
- **Activo**, en este registro se configura si la IR está activa, es decir si puede operar. Puede ser configurado por medio de una operación de configuración desde el *maestro-SDN*.

La IR también cuenta con una serie de registros denominados Registros de Configuración que se pueden observar en la Figura 4.10. Hay un registro por cada flujo

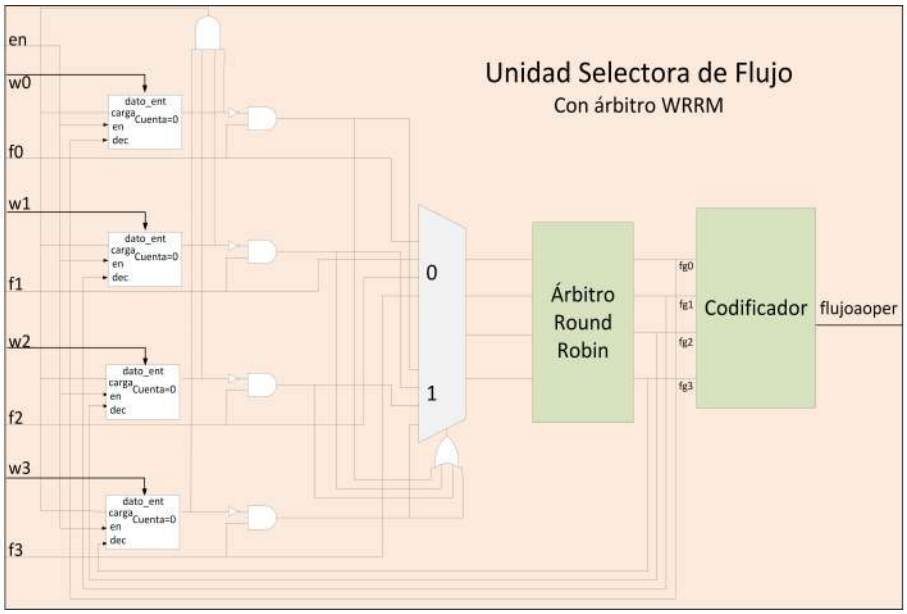


Figura 4.8: Arquitectura de la Unidad Selectora de Flujo

que soporta la IR. En estos registros se almacenan: por un lado, los pesos de soporte de cada flujo que serán utilizados por las Unidades Selectoras de Flujo para establecer la prioridad de los flujos; Por otro lado, se almacenan las direcciones de reenvío y las direcciones *multicast* para la operación de flujos en el modo SDBoC. En el caso de la IRM solo se hace uso de los registros de *peso* utilizados por la Unidad Selectora de Flujo para otorgar el permiso de trabajo a un flujo.

El Recolector de Estadística que se muestra en la Figura 4.11 es la unidad encargada de calcular la estadística de la IR. En este trabajo la estadística recolectada es: uno, el tiempo de espera en el *buffer* de entrada antes de que un flujo sea atendido; dos, el tiempo de espera en el *buffer* de salida antes de que un flujo pueda transmitir un paquete. El Recolector de Estadística está formado por dos unidades: la primera, el Recolector de Eventos que se puede observar en la Figura 4.12 y que está encargado de contabilizar, por cada flujo, la cantidad de eventos que se registran ya sea en el *buffer* de entrada o en *buffer* el de salida; la segunda, el Recolector de Tiempos de Almacenamiento, que se muestra en la Figura 4.13, el cual registra el tiempo que dura cada paquete en un *buffer*. Cuando un paquete llega a la IR, el Receptor de Red genera una señal para que se registre el tiempo de arribo. Cuando el paquete es atendido por el *maestro* el Esclavo de lectura AXI-Full registra el tiempo de salida. En el Recolector de Tiempos de Almacenamiento se registra y acumula la diferencia entre estos dos eventos. Un proceso similar sucede cuando el paquete es de salida, de forma que el Esclavo de escritura AXI-Full registra el tiempo de llegada del paquete

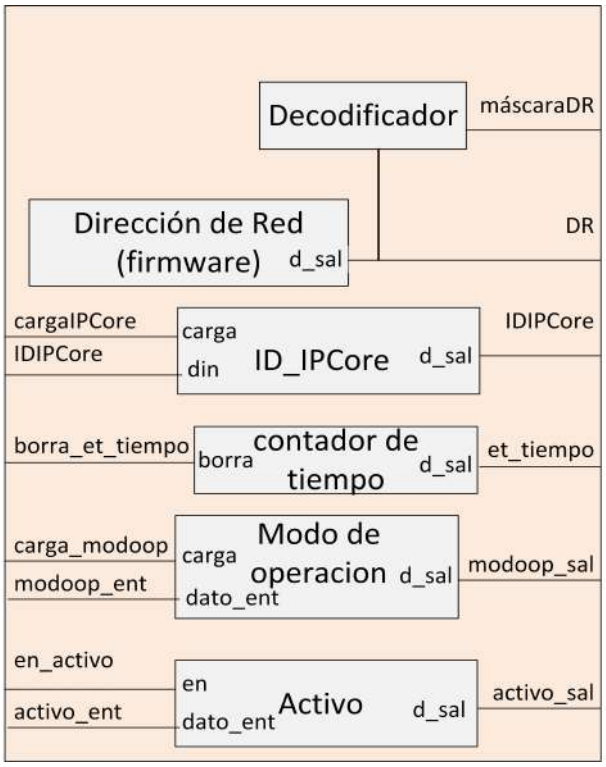


Figura 4.9: Arquitectura de la Unidad de Registros Especiales

y el Transmisor de Red registra el tiempo de salida del mismo.

La interfaz AXI es la encargada de establecer la comunicación con el *maestro* a la que la IR está atada. Responde a las operaciones de escritura iniciadas por el *maestro* por medio del Esclavo de escritura AXI-Full. Para conocer a qué flujo va dirigida la transacción de escritura, utiliza el primer dato proveniente del *maestro*, que es el *flit* de cabecera del paquete que se va a transmitir y donde está contenido el flujo. Con esta información puede seleccionar el *buffer* de salida. Cuando finaliza la operación de escritura, le indica a la Unidad de Control de la Interfaz de Red la cantidad de datos que se transfirieron. Finalmente genera una señal para el Recolector de Estadística donde le indica que un paquete acaba de ser almacenado en un *buffer* de la IR.

Por su parte el Esclavo de lectura AXI-Full es el encargado de leer la información que hay en los *buffers* de la IR y transmitirla al *maestro*. El proceso de lectura de un paquete es realizado en dos momentos. Esto es debido a que el *maestro* conoce que hay un paquete – por eso fue interrumpido – pero desconoce a qué flujo llegó y de qué tamaño es. Por lo que inicialmente se hace una lectura hacia una dirección específica conocida por el Esclavo de lectura AXI-Full con la que identifica que el

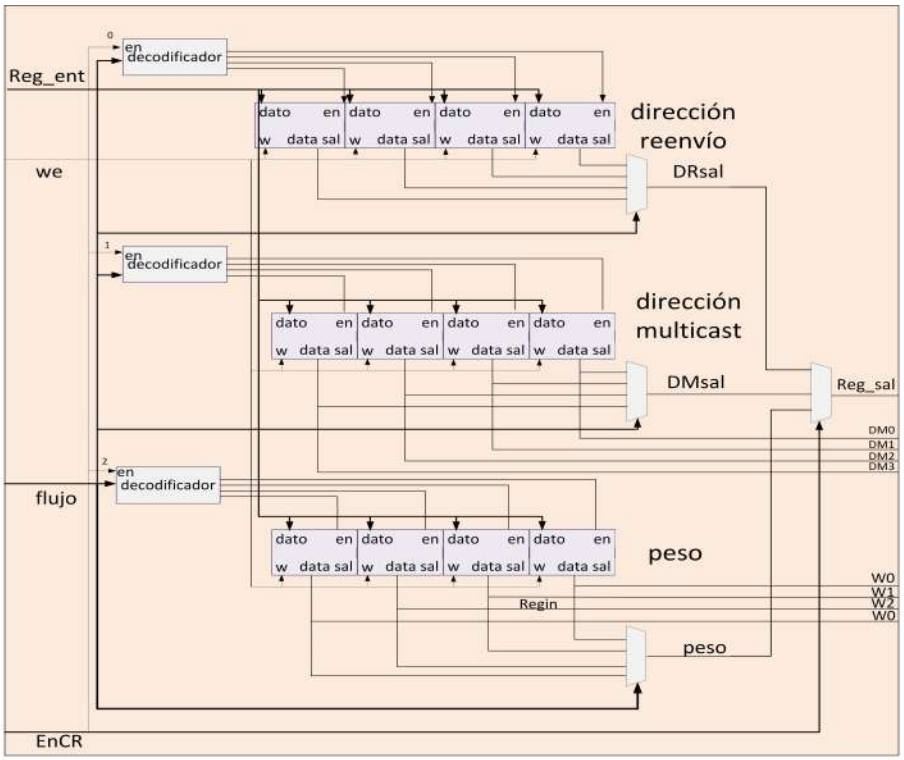


Figura 4.10: Arquitectura de la Unidad de Registros de Configuración

maestro quiere conocer la información antes mencionada (flujo, número de datos). Esta información es solicitada a la Unidad de Control la cual la toma de los Registros de Control de Tráfico y selecciona el flujo indicado apoyado de la Unidad Selectora de Flujos. El flujo y la cantidad de datos del paquete que llegó es transmitido al Esclavo de lectura AXI-Full el cual a su vez la envía al *maestro*. Con la información recibida, el *maestro* realiza una segunda operación de lectura en la cual ya puede solicitar los datos del paquete que arribó.

La Unidad de Control tiene a su cargo un conjunto de operaciones que permiten la operación de la IR ya sea en modo *tradicional* o en modo SDBoC. Es la encargada de recibir los paquetes de configuración en los que se establece el modo de operación de la IR. Es la encargada de controlar la estadística y de reportarla cuando así se le solicite. Es la encargada por medio de los Registros de Control de Tráfico de indicar al *maestro* qué flujo debe de operar y de indicarle al Transmisor de Red que hay nuevos paquetes por transmitir. También es la encargada de indicar qué elemento tiene el control de los *buffers* de entrada y salida. Responde a las solicitudes hechas tanto por el Esclavo de lectura AXI-Full como del Esclavo de escritura AXI-Full. En la Figura A.7 se muestra la Máquina de Estados Finitos que contiene a la Unidad

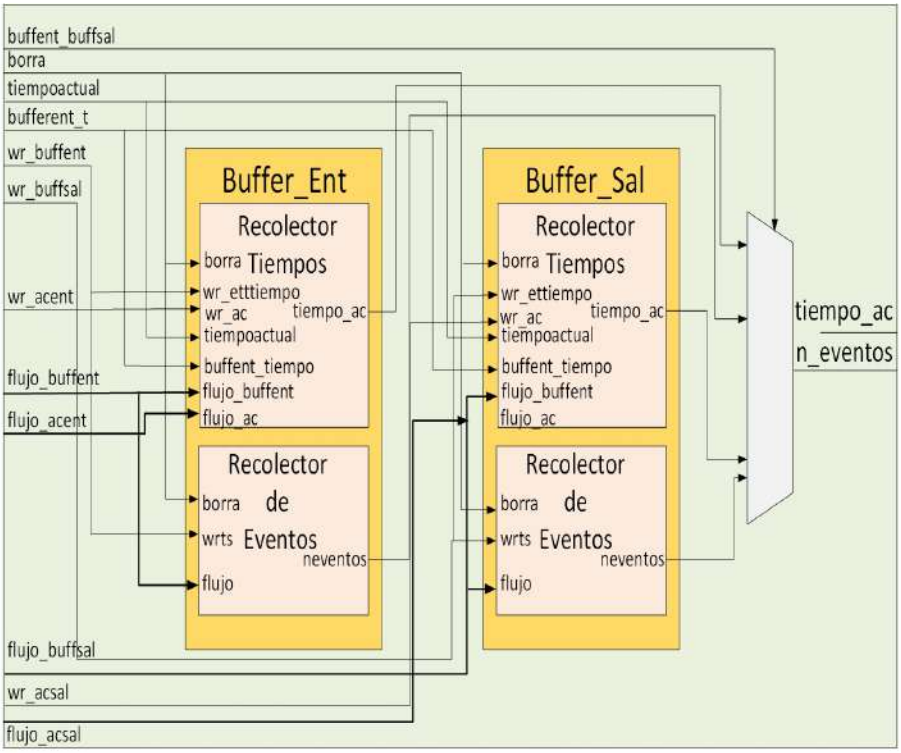


Figura 4.11: Diagrama general del Recolector de Estadística

de Control de la interfaz de Red Maestra.

Interfaz de Red Esclava

La Interfaz de Red Esclava es la encargada de unir el sistema de interconexión con los elementos de procesamiento esclavos como los IPCores. Al ser el soporte de comunicación del elemento tipo *esclavo*, es la que integra la mayor parte de las funcionalidades que se deben de proveer para trabajar en modo SDBoC. Por otro lado, es la encargada de comunicarse con los IPCores por medio de la interfaz AXI. Estos IPCores dentro de la interfaz se comportan como *esclavos* del sistema de tal modo que responden a operaciones de lectura y escritura a direcciones AXI provistas por la Interfaz de Red Esclava que se comporta como el *maestro* de la transacción AXI. En términos generales la Interfaz de Red Esclava contiene los mismos elementos que la Interfaz de Red Maestra pero con algunas funcionalidades añadidas.

Cuando se trabaja en modo SDBoC, el Receptor de Red tiene la posibilidad de integrar direcciones *multicast*. Se puede contar con una dirección *multicast* por cada flujo. Estas direcciones deben de ser establecidas por la Función de Red de configuración antes de que puedan ser operadas. Al momento del *reset* inicial, a las

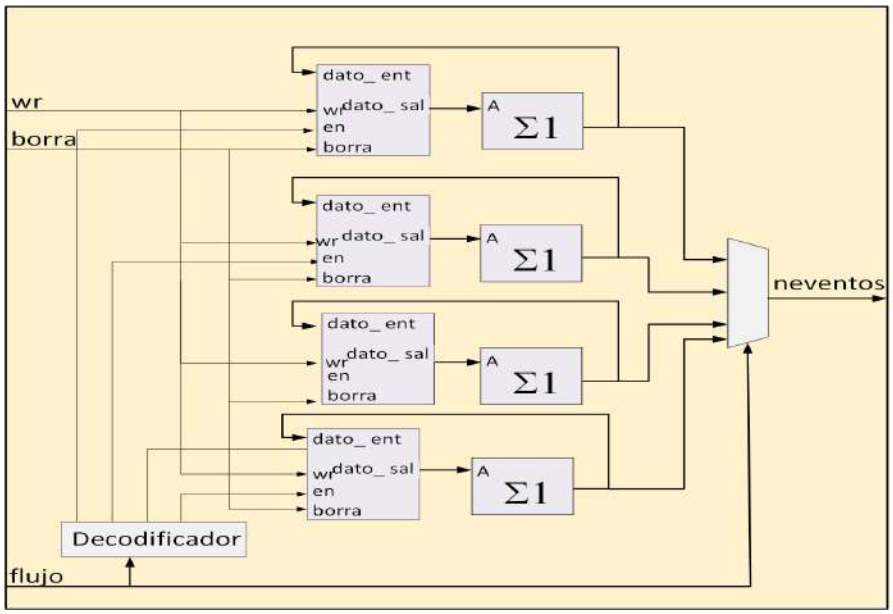


Figura 4.12: Diagrama del Recolector de Eventos

direcciones *multicast* se les establece el mismo valor de la dirección de *broadcast*. Otra funcionalidad que se agrega en el Receptor de Red es el control de los paquetes de *reconocimiento* de los flujos. Cuando el sistema de interconexión se encuentra operando con flujos, se forma una cadena de trabajo entre los IPCores que forman el flujo. Un IPCore no puede mandar el resultado de su operación al siguiente elemento en el flujo si éste no le ha confirmado que tiene posibilidad de atenderlo. El Receptor de Red verifica cada paquete que llega y si es de *reconocimiento* borra la bandera en el Registro de Espera Por Reconocimiento (REPR) asociado al flujo. Esta acción habilita la posibilidad de que el flujo continúe operando. El diagrama de la Máquina de Estados Finitos del Receptor de Red de la IRE se muestra en la Figura A.5.

En el Transmisor de Red también se debe de añadir la funcionalidad para poder operar en modo de flujo. Esto se efectúa en dos sentidos: primero, el Transmisor de Red no puede transmitir un paquete de un flujo si su bit asociado en el REPR de los Registros de Control de Tráfico no ha sido previamente borrado por el Receptor de Red; segundo, una vez que transmitió un paquete de datos por un flujo, enciende la bandera REPR del flujo asociado en los Registros de Control de Tráfico. El diagrama de la Máquina de Estados Finitos del Transmisor de Red de la IRE se muestra en la Figura A.6.

El Recolector de Estadística, las Unidades Selectoras de Flujos y los Registros Especiales contenidos en la Interfaz de Red Esclava tienen las mismas funcionalidades que en la Interfaz de Red Maestra. Con la particularidad que en la unidad de

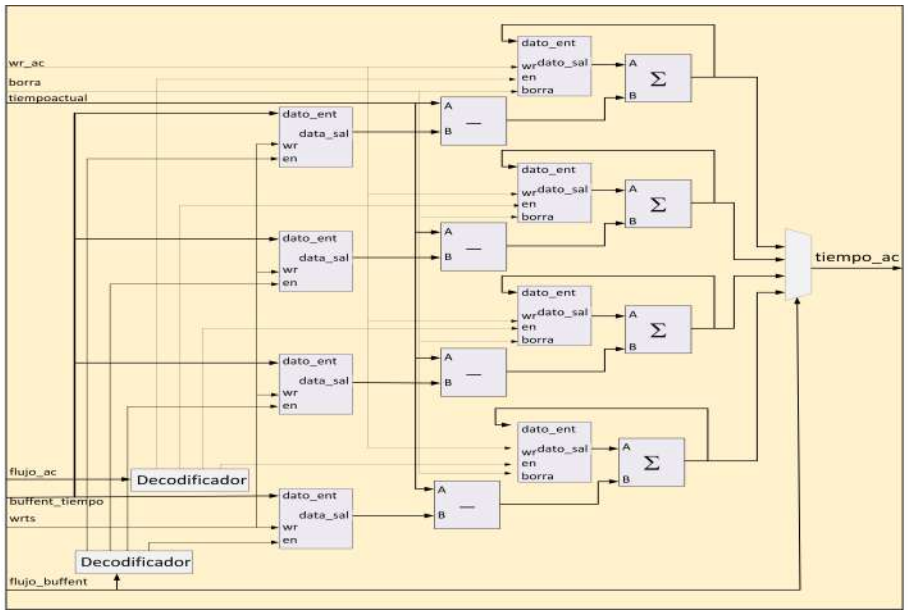


Figura 4.13: Diagrama del Recolector de Tiempos de Almacenamiento

Registros de Configuración ahora son tomadas en cuenta las direcciones *multicast* y las direcciones de reenvío de datos utilizadas cuando se trabaja en modo de flujo denominadas Direcciones de Reenvío.

El IPCore atado a la Interfaz de Red Esclava es operado por medio de operaciones de escritura y lectura realizadas por la interfaz maestra AXI-Full y controladas por la Unidad de Control. El Maestro de escritura AXI-Full transfiere la cantidad datos que le son indicados por la Unidad de Control a partir de la dirección que ésta le indica. El *buffer* asociado al flujo que se está operando también es establecido por la Unidad de Control. La máquina de estados del Maestro de escritura AXI-Full en la IR Esclava se puede observar en la Figura A.8. Por su parte el Maestro de lectura AXI-Full es el encargado de realizar la operación de lectura que le solicita la Unidad de Control, los datos obtenidos por la operación de lectura son almacenados en el *buffer* asociado al flujo que previamente fue establecido por la Unidad de Control. La máquina de estados del Maestro de lectura AXI-Full en la IR Esclava se puede observar en la Figura A.9.

La Unidad de Control además de encargarse de los procesos de configuración y reporte de estadística de la Interfaz de Red Esclava es la encargada del control del tráfico dirigido o proveniente del IPCore. Este proceso lo realiza apoyada en la información contenida en los Registros de Control de Tráfico que se muestran en la Figura 4.14. Los paquetes que arriban a la IR pueden tener dos destinos: uno, la propia IR; dos, el IPCore. Independientemente del destino que tenga el paquete,

inicialmente el *Receptor de Red* genera una señal dirigida a los Registros de Control de Tráfico donde señala en el RDPO el flujo al que llegó el paquete y en el *dataPO* la cantidad de datos que llegaron en el paquete.

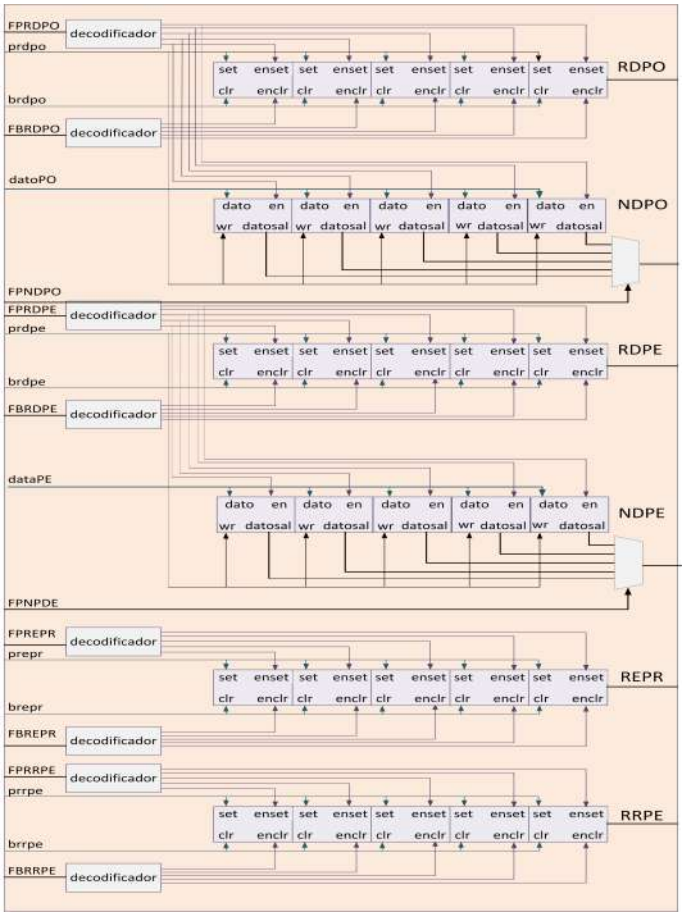


Figura 4.14: Diagrama de la Unidad de Registros Controladores de Tráfico

Los elementos contenidos en los Registros de Control de Tráfico representan banderas de control para cada flujo en la IR. Se ha establecido el flujo cero como el flujo de configuración ya sea que venga dirigido a la propia IR o que tenga como destino el IPCore. Los flujos restantes son de operación. La Unidad de Control se encuentra constantemente verificando el contenido del Registro de Datos Por Operar (RDPO) si una de sus banderas está encendida, indica que llegó un paquete a ese flujo y que requiere ser atendido. Si la Unidad de Control detecta que el paquete es de operación o en su caso es de configuración pero va dirigido al IPCore, primero verifica que el IPCore tenga posibilidad de atenderlo. En el modo de operación por flujo, SDBoC no puede operar un paquete hasta que no esté seguro que el *buffer* de

salida asociado a este flujo se encuentre libre; esto lo sabe verificando que la bandera REPR asociada al flujo esté apagada. Como cabe la posibilidad que existan paquetes de diferentes flujos esperando ser atendidos por el IPCore, la Unidad de Control se apoya de la Unidad Selectora de Flujo la cual le indica que flujo tiene la prioridad de ser atendido. Una vez que el paquete fue transferido al IPCore, por medio de una operación de escritura AXI, la Unidad de Control deshabilita la bandera RDPO asociada. Si la Unidad de Control detecta que la operación va dirigida a la propia IR quiere decir que se trata de ejecutar alguna de las siguientes acciones:

- El *maestro-SDN* desea conocer la identificación de la IR.
- El *maestro-SDN* desea activar o desactivar el nodo.
- El *maestro-SDN* desea establecer la configuración de la IR.
- El *maestro-SDN* desea conocer el estado de la IR.
- El *maestro-SDN* desea conocer la estadística de la IR.
- El *maestro-SDN* desea inicializar la estadística de la IR.

La Unidad de Control tiene el control de lectura de los resultados generados por el IPCore o cualquier otra información que de él se requiera. El proceso es llevado a cabo en tres momentos: primero, la Unidad de Control ordena una operación de escritura a la *Dirección AXI* establecida en el paquete origen (cada *Dirección AXI* representa una acción diferente que puede tomar el IPCore ya sea: de operación, de lectura de estadística, de lectura de estado, de configuración). Cuando el Maestro de escritura AXI-Full retorna el control a la Unidad de Control es por que el proceso de escritura culminó y la operación se ha ejecutado; segundo, la Unidad de Control realiza una operación de lectura de dos datos a una dirección previamente establecida y conocida por la IR, donde el IPCore almacenó la dirección de donde se pueden tomar los resultados y la cantidad de datos generados por la operación; tercero, con la información recabada la Unidad de Control efectúa una segunda operación de lectura, la información obtenida es almacenada en el Buffer Múltiple de Salida y se activa la bandera RDPE asociada al flujo. Además se almacena la cantidad de datos con que cuenta el paquete en el registro *datosPE* asociado al flujo. Finalmente la Unidad de Control genera un paquete de *reconocimiento* dirigido al nodo emisor del paquete original. Con esta acción le indica que está en condiciones de recibir otro paquete dirigido a este flujo. El control de los *reconocimientos* es llevado a cabo por medio del Registro de Reconocimiento Por Enviar (RRPE). La arquitectura de la Interfaz de Red Esclava se muestra en la Figura 4.15. Los diagramas de las máquinas de estados que integran la Unidad de Control se muestran en las Figuras A.10, A.11, A.12, A.13, A.14, A.15, A.16, A.17.

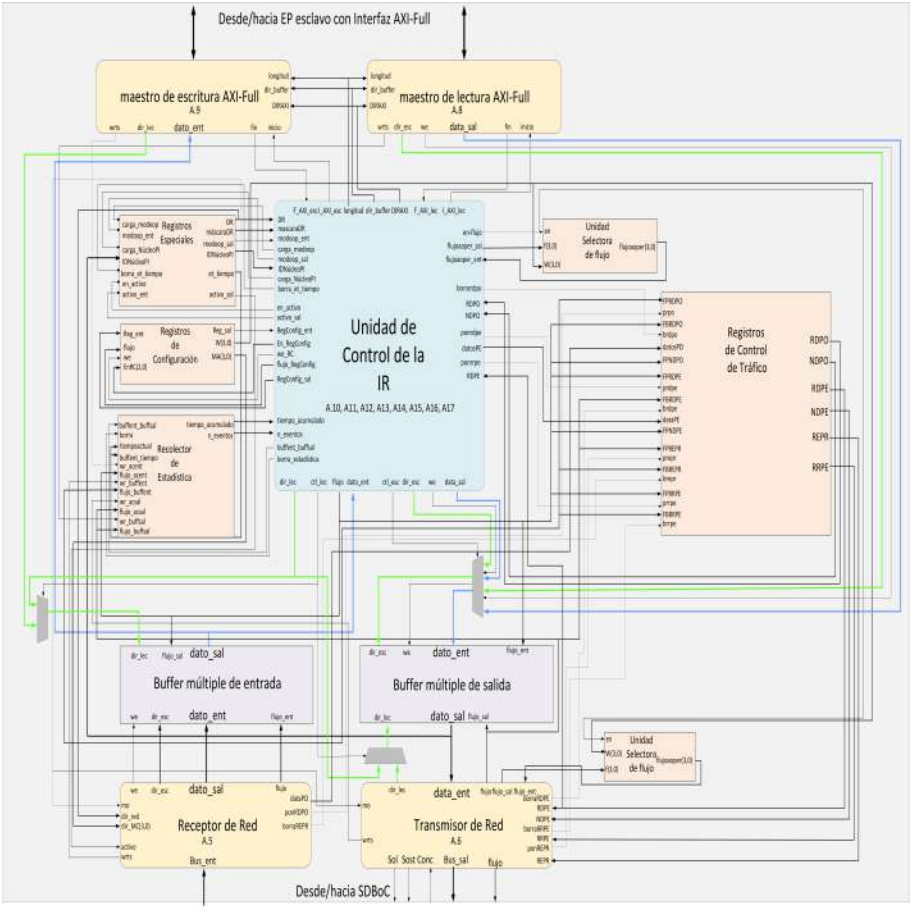


Figura 4.15: Arquitectura de la Interfaz de Red Esclava

Controlador del Bus

El Controlador del Bus es el elemento central del sistema de interconexión, bajo el paradigma SDBoC tiene bajo su control y supervisión las siguientes acciones:

- Es el encargado por medio del Motor de Arbitraje de decidir a quién de los elementos que están solicitando acceso al *bus* le será otorgado el medio.
- Por medio del Recolector de Estadística se encarga de recolectar la estadística referente al uso del *bus* ya sea de forma global, por flujo o por nodo.
- Por medio de la Unidad de Seguridad, está encargado de supervisar que en el sistema de interconexión no viajen paquetes espurios.

La arquitectura del Controlador del Bus se puede observar en la Figura 4.16. Para poder comunicarse con el *maestro-SDN* cuenta con una IR integrada que es capaz

de recibir y transmitir paquetes provenientes únicamente del *maestro-SDN*. Los paquetes que puede recibir son paquetes de configuración o solicitud de estadística y puede generar paquetes de reporte de fallo de seguridad o de envío de estadística.

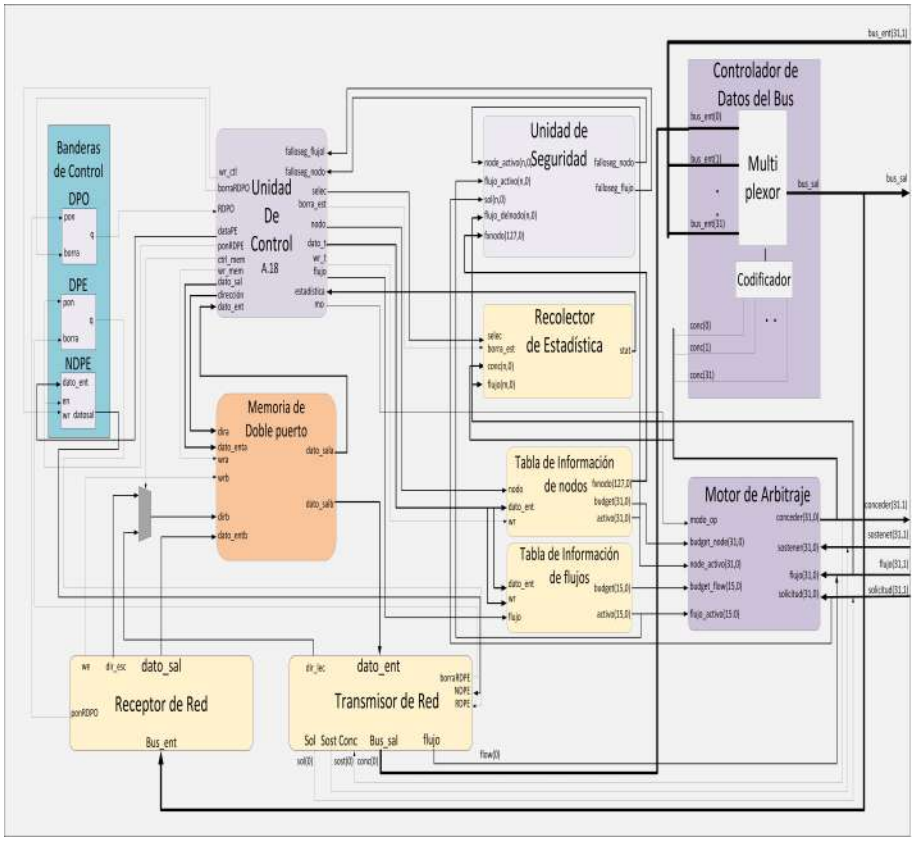


Figura 4.16: Arquitectura del Controlador del Bus

La IR integrada en el Controlador del Bus, al poder recibir paquetes provenientes sólo de un nodo y de un solo flujo, se convierte en una versión simplificada de la IR tipo *slave*. El Receptor de Red se encuentra constantemente verificando el estado del *bus* y cuando detecta un paquete proveniente del *maestro-SDN* lo almacena y activa la bandera *Dato por Operar* (DPO) en el Registro de Banderas de Control. Por su parte la Unidad de Control se encuentra verificando el estado de la bandera DPO y cuando detecta que ésta se encuentra encendida entiende que debe de atender alguna petición del *maestro-SDN*. Las peticiones pueden ser debidas a:

- El *maestro-SDN* le indica al Controlador del Bus cuáles son los nodos activos y el presupuesto que cada uno de ellos tiene. Esta información es leída por la Unidad de Control y almacenada en la Tabla de Información de Nodos para su posterior uso por parte del Motor de Arbitraje.

- El *maestro-SDN* le indica al Controlador del Bus cuáles son los flujos activos y el crédito que cada uno de ellos tiene. Esta información es leída por la Unidad de Control y almacenada en la Tabla de Información de Flujos para su posterior uso por parte del Motor de Arbitraje.
- El *maestro-SDN* le indica al Controlador del Bus el modo de operación, *tradicional* o *SDBoC*.
- El *maestro-SDN* le solicita al Controlador del Bus la estadística que ha generado.
- El *maestro-SDN* le solicita al Controlador del Bus que reinicie la estadística.

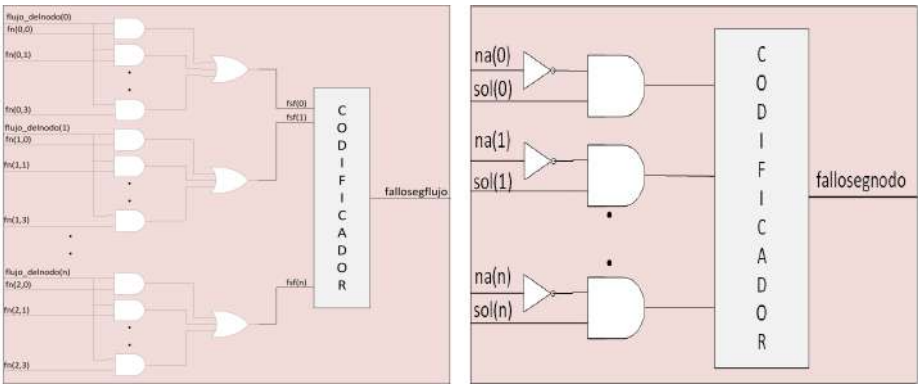
A cada una de estas peticiones la Unidad de Control responde generando un paquete ya sea con la información solicitada o con un paquete de *reconocimiento* indicando que la configuración se estableció. Para esto, además de crear el paquete y almacenarlo en la Memoria de Doble Puerto, borra la bandera DPO y establece la bandera *Dato por Enviar* (DPE) y el número de datos a transmitir en el registro *Número de Datos Por Enviar* (NDPE).

La Unidad de Control también se encuentra vigilando constantemente el estado reportado por la Unidad de Seguridad la cual le reporta si existe un flujo o un nodo no autorizado intentando hacer uso del sistema de interconexión. Ante esto, la Unidad de Control responde generando un paquete dirigido al *maestro-SDN* donde le informa del intento de intrusión. La Máquina de Estados Finitos de la Unidad de Control del Controlador del Bus se observa en la Figura A.18.

El Transmisor de Red se encuentra constantemente vigilando la bandera DPE, cuando detecta que ésta se encuentra activa, toma la cantidad de datos a enviar del NDPE y transmite el paquete. Cuando finaliza de transmitir el paquete, borra la bandera DPE.

La Unidad de Seguridad se encuentra formada por dos elementos: uno, el Detector de Intrusión por Flujo; dos, el Detector de Intrusión por Nodo los cuales se muestran en las Figuras 4.17a y 4.17b respectivamente. El Detector de Intrusión por Nodo verifica si todos los nodos que están solicitando acceso al sistema de interconexión se encuentran registrados como nodos activos. Si esto no es así, genera un código de acceso indebido, indicando el número de nodo "no activo" que está intentando acceder al sistema. En el caso del Detector de Intrusión por Flujo se verifica si el flujo que está solicitando el acceso se encuentra configurado en el nodo que está solicitando el medio.

El Recolector de Estadística que se muestra en la Figura 4.18 tiene la capacidad de poder recolectar el tiempo que el *bus* ha permanecido ocioso y el tiempo que ha permanecido ocupado desde la última vez que fue inicializado. Además por cada



(a) Detector de Intrusión por Flujo (b) Detector de Intrusión por Nodo

Figura 4.17: Elementos de la Unidad de Seguridad

nodo activo y/o flujo activo en el sistema de interconexión puede recolectar el tiempo que ese nodo/flujo ha permanecido utilizando el *bus* así como el número de veces que lo utilizó.

El Controlador de Datos del Bus es el encargado de redirigir una de las entradas provenientes de las Interfaces de Red del sistema al *bus* de salida. Esta acción la realiza en base a la indicación hecha por el Motor de Arbitraje.

La Tabla de Información de Nodos (Figura 4.19a) almacena información relacionada con cada uno de los nodos. Por cada uno de ellos almacena: su estado, activo o inactivo; el presupuesto que se le asigna para el arbitraje; los flujos que tiene asociados. La Tabla de Información de Flujos (Figura 4.19b) almacena por cada uno de los flujos: su estado, activo o inactivo; el presupuesto que se le asigna para el proceso de arbitraje.

El Motor de Arbitraje es el encargado, bajo una política de arbitraje, de otorgar el acceso al medio a uno de los elementos que lo esté solicitando. La política de arbitraje que se presenta aquí es *Arbitraje Oportunístico Con Deuda Supervisada* (Supervised Debt and Opportunistic (SuDO), en inglés) Ibarra-Delgado et al. (2020b), el cual es un resultado derivado del desarrollo de la presente tesis . El modo de operación establecido por el *maestro-SDN* es el que indica si el arbitraje se realizará tomando en cuenta el presupuesto de los flujos o el presupuesto de los nodos. Sin embargo, antes de realizar cualquier ronda de arbitraje, el Motor de Arbitraje filtra solo aquellas *solicitudes* de los nodos o flujos que se encuentren activos. De esta manera evita la posibilidad de uso del *bus* por algún elemento no permitido.

El comportamiento de la política de arbitraje SuDO es el siguiente: a cada uno de los contendientes por el *bus* se le asigna una cantidad inicial de presupuesto. Cuando se realiza una ronda de arbitraje, el filtro SuDO selecciona aquellos contendientes que tengan la mayor cantidad de presupuesto, estos son pasados por un árbitro *Round-*

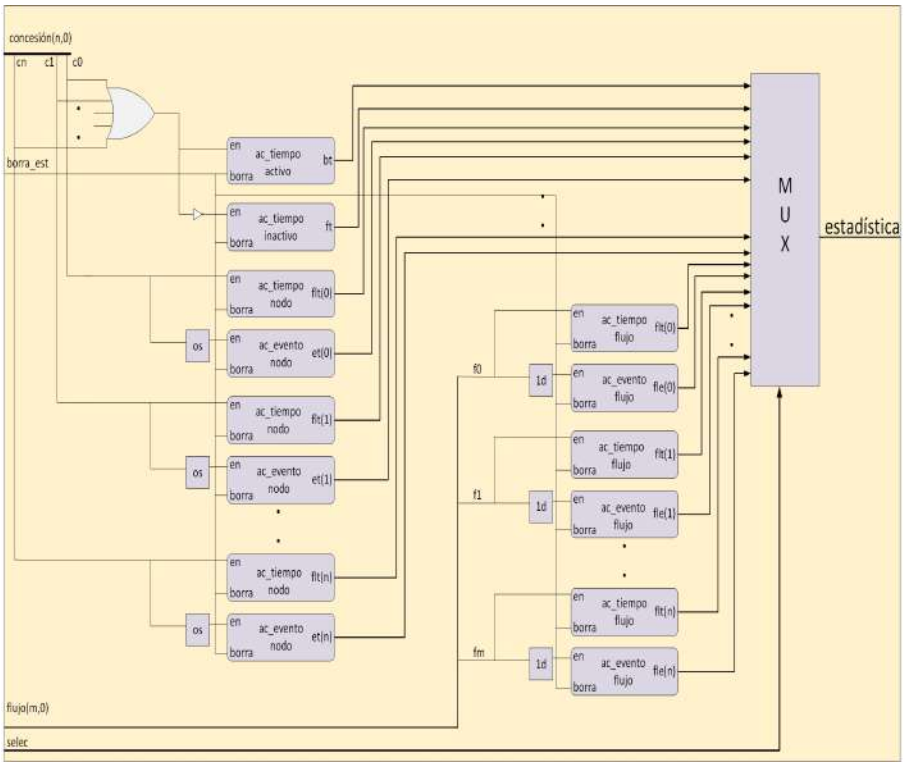
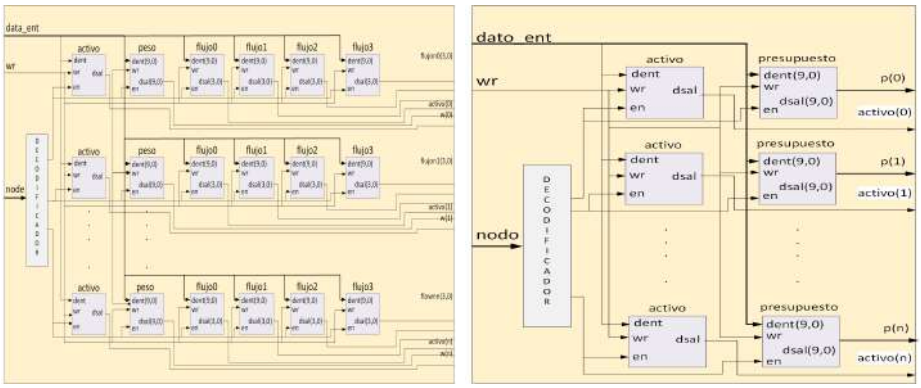


Figura 4.18: Diagrama del Recolector de Estadística

Robin el cual le asigna el medio a uno de los contendientes en caso de que exista un empate. Una vez que el medio fue asignado a un contendiente la señal de *concesión* es retro-alimentada al filtro SuDO y por cada ciclo de reloj que el elemento se mantenga utilizando el *bus* le será decrementado en uno su presupuesto. Cuando el elemento libera el *bus* una nueva ronda de arbitraje puede empezar. Existen dos elementos que por su importancia tienen las máximas prioridades y no entran en ninguna ronda de competencia: la máxima prioridad la tiene el propio Controlador del Bus; la segunda máxima prioridad la tiene el *maestro-SDN*. Si ninguno de estos elementos se encuentra solicitando el *bus* el acceso le es permitido al ganador proveniente del Filtro SuDO. La arquitectura general del Motor de Arbitraje se muestra en la Figura 4.20.

El diseño del filtro SuDO se presenta en la Figura 4.21. La principal función de esta unidad es filtrar el o los elementos que tienen el presupuesto más alto. En una ronda de arbitraje a la salida de la Unidad de Conteo Descendente, que se muestra en la Figura 4.22a, se tiene contabilizado el presupuesto y la deuda actual de cada uno de los participantes. Los presupuestos junto con las solicitudes de acceso son la entrada del Árbol de Presupuesto Mayor, mostrado en la Figura 4.22b. El



(a) Tabla de Información de Nodos (b) Tabla de Información de Flujos

Figura 4.19: Tablas de información

árbol está compuesto por un conjunto de elementos denominados Solicitante-Mayor, una unidad de estos elementos es mostrado en la Figura 4.23b, el cual compara el presupuesto de un par de contendientes. En caso de que un elemento no esté solicitando el medio la entrada al comparador es 0. A la salida de cada uno de estos elementos se encuentra el mayor de los dos valores que se compararon. El resto de comparadores en el árbol son comparadores tradicionales. Al final del árbol se obtiene el presupuesto máximo de los elementos que están contendiendo por el *bus*. Este presupuesto es ahora comparado con el presupuesto individual de cada uno de los elementos mediante el Comparador de Igualdad, que se muestra en la Figura 4.22c, siendo uno solo los *solicitantes* cuyo presupuesto coincida con el presupuesto máximo encontrado. En el Comparador de Igualdad, la salida de cada una de las comparaciones realizadas por los comparadores son multiplicadas por su señal de *solicitud* asociada dando como resultado aquellos *solicitantes* que solicitan el medio y que cuentan con el mayor presupuesto. Esta salida es enviada al árbitro *Round-Robin* el cual es el encargado de resolver un posible empate. A la salida del árbitro *Round-Robin* se tiene al *solicitante* ganador de la contienda y se genera la señal de *concesión* que además de avisarle a la *Interfaz de Red* asociada que tiene el uso del medio, se retro-alimenta a la entrada de la Unidad de Conteo Descendente en el filtro SuDO, para que éste, mientras la IR mantenga el control del *bus*, descuente en uno el presupuesto del *solicitante* por cada ciclo de reloj que mantenga el control del *bus*. El elemento encargado de descontar el crédito es el Contador-Descendente, mostrado en la Figura 4.23a. Este elemento, por cada ciclo de reloj que la señal de *enable* se encuentra activa, decrementa en uno su registro interno. Si el registro llega a cero y la señal de *enable* continua habilitada, satura la salida a cero e indica que su crédito se ha terminado. El registro en el Contador-Descendente es recargado con su presupuesto inicial hasta el momento en que todos los *solicitantes* han gastado

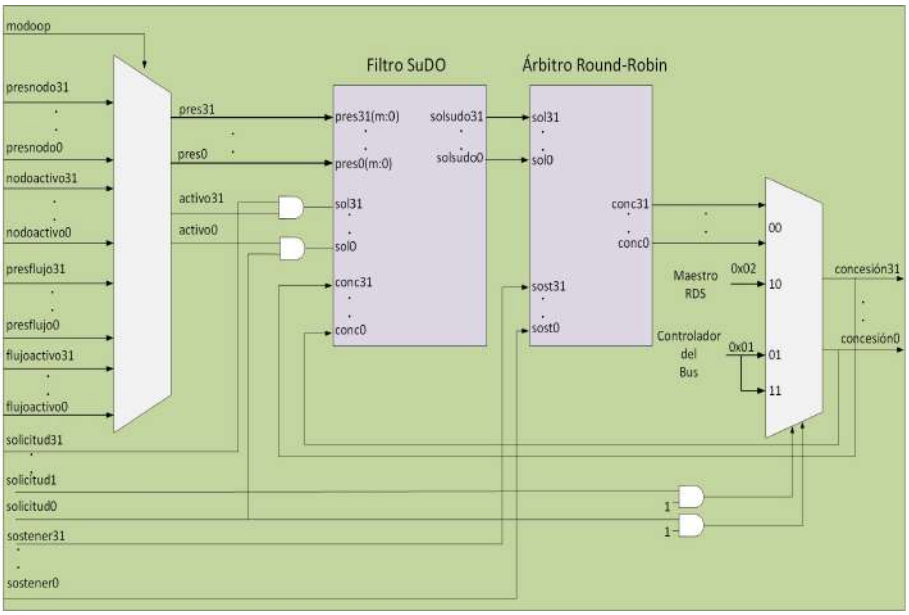


Figura 4.20: Arquitectura del Motor de Arbitraje con política basada en presupuestos

la totalidad de su presupuesto.

Una de las principales ventajas de SuDO es que permite el uso del *bus* a un elemento que haya gastado todo su presupuesto. La única condición es que no exista ningún *solicitante* con presupuesto por gastar. Si esta condición se cumple, entonces el acceso al medio se le otorga al *solicitante* que en ese momento tenga la menor deuda (de aquí el concepto de Deuda Supervisada). Para conocer quien de los solicitantes tiene la menor deuda se sigue un procedimiento similar al que se realizó para encontrar al elemento con presupuesto mayor, pero ahora utilizando un *Árbol de Deuda Menor*. El concepto de *Deuda Supervisada* obtiene importancia porque de esta forma el uso del *bus* no se sesga a un elemento que esté constantemente solicitando el medio aunque haya concluido con su presupuesto. En el trabajo realizado por Ibarra-Delgado et al. (2020a) se muestra que esta política de arbitraje es la que permite una mejor distribución del uso del *bus* de acuerdo a la relación de presupuestos originalmente asignados.

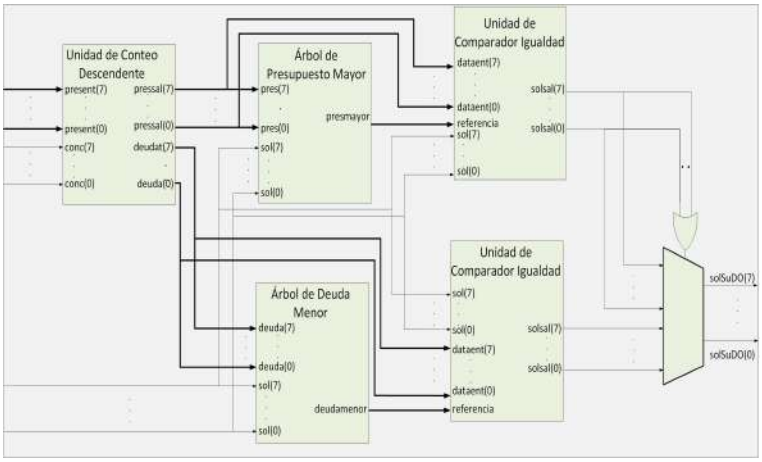


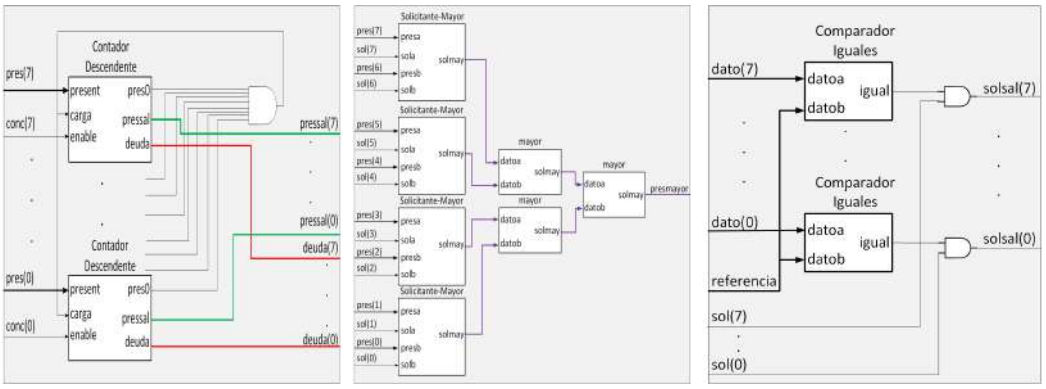
Figura 4.21: Arquitectura del Filtro SuDO

4.5. Operación del Bus-en-Chip Definido por Software

Una de las características del paradigma SDN es que la red pueda trabajar en modo tradicional si así es requerido. En la implementación SDBoC que aquí se presenta esto es posible debido a que el modo de operación puede ser controlado por el *maestro-SDN*. De hecho al inicializarse el sistema éste arranca en modo *tradicional* y no es modificada su operación hasta que lo establece el *maestro-SDN*. En modo tradicional las Interfaces de Red reciben paquetes, los reportan a su correspondiente elemento de procesamiento y envían paquetes a petición del mismo. En esta sección se dará una descripción de la forma de operación cuando se desea que el sistema trabaje en modo SDBoC.

Cuando el sistema es inicializado, cada IR que se encuentra en el sistema de interconexión procede a obtener el identificador de los elementos de procesamiento atados a ella. La primera función que debe de ser ejecutada desde el Sistema Operativo de Red es la de Visor de Red la cual solicita el servicio Descubrimiento de Red. Este servicio habilita al *maestro-SDN* a enviar un mensaje de *broadcast* para que todos los nodos existentes en la red se identifiquen. Cada uno de los nodos que recibió la solicitud del *maestro-SDN* responde con un paquete donde envía su Dirección de Red y el identificador del elemento que tiene atado a ella. El *maestro-SDN* reporta la información a la función Descubrimiento de Red, como resultado esta función actualiza la Tabla de Información de IPCores y le indica al Visor de Red que el proceso ha concluido. Con esta información el Visor de Red está en condiciones de reportar a quien así se lo solicite la vista global del sistema de interconexión.

Aunque cualquier servicio de red está habilitado para ser ejecutado en cualquier

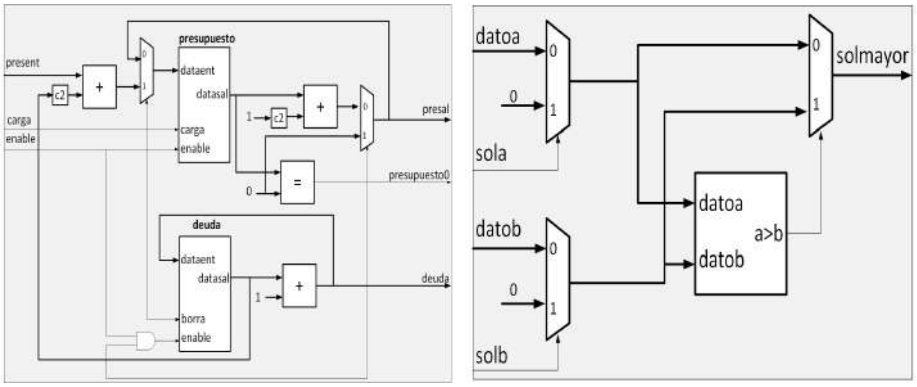


(a) Unidad de Conteo Descendente

(b) Unidad de Árbol de Presupuesto Mayor

(c) Unidad de Comparación de Igualdad

Figura 4.22: Principales elementos del filtro SuDO



(a) Unidad de Contador-Descendente

(b) Unidad de Solicitante-Mayor

Figura 4.23: Unidades de los elementos Contador-Descendente y Solicitante-Mayor

momento durante la operación, es común que las primeras acciones que se ejecuten tengan que ver con la configuración del SDBoC. El *maestro-SDN* envía un paquete de *broadcast* donde puede establecer el modo de operación del sistema de interconexión. Posteriormente debe de enviar un conjunto de paquetes de configuración dirigidos al Controlador del Bus y a las IR que se desea entren en operación.

En el caso del Controlador del Bus se deben de configurar los nodos y/o los flujos que estarán activos a partir de este momento. Además, a cada uno de ellos se le debe de asignar la cantidad de presupuesto que tendrá para competir en las rondas de arbitraje. Para tener un control de seguridad de los flujos y que un nodo solo pueda solicitar acceso por medio de un flujo que le haya sido asignado, a cada nodo se le configura los flujos que tiene asociados, esto también se le hace saber al Controlador

del Bus. Posteriormente el *maestro-SDN* manda un paquete al Controlador del Bus con el objetivo de inicializar los registros encargados de llevar la estadística.

En el caso de las IR se deben de configurar las Direcciones de Reenvío y las Direcciones Multicast para cada uno de los posibles flujos que puede operar un nodo. Además, se le debe de configurar un peso por cada flujo de forma proporcional al que tengan para acceder al *bus*. Este peso será utilizado por la Unidad de Control de la IR para controlar a quien de los posibles flujos que se encuentran solicitando el servicio del IPCore asociado se le dará la preferencia de operación. Los IPCores se encuentran preparados para operar una vez efectuada la operación de configuración.

Una vez que los elementos de procesamiento se encuentran operando, el *maestro-SDN* tiene la capacidad de reconfigurar el sistema de interconexión en cualquier momento. Por ejemplo, pudiera si así lo cree necesario cambiar los pesos de soporte de la política de arbitraje. Además, el *maestro-SDN* se encuentra habilitado para solicitar la estadística del sistema o de algún nodo en particular en cualquier momento. Del mismo modo, se encuentra habilitado para reiniciar la estadística de manera general o individual. Si por alguna razón el Controlador del Bus detecta que se está tratando de acceder al sistema de interconexión sin tener el permiso, genera un mensaje de alerta al *maestro-SDN* para que este a su vez tome las acciones necesarias al respecto.

Los *maestros* que operan en el sistema le indican al *maestro-SDN* la conclusión del proceso que les fue solicitado. Éste a su vez, reporta a las instancias superiores en la arquitectura SDN la finalización del proceso. Entonces el sistema puede ser reconfigurado generando nuevos pesos para los flujos aún activos en el sistema, o con nuevos flujos de trabajo.

4.6. Análisis de la Implementación Hardware del Bus-en-Chip Definido por Software

En esta sección se detallará los resultados obtenidos de la implementación de los elementos del sistema de interconexión del SDBoC. La implementación fue realizada utilizando el lenguaje de descripción de hardware Verilog y como referencia se seleccionó el SoC Zynq7020 de Xilinx.

La Interfaz de Red es el elemento que separa la capa de procesamiento de datos de la capa de transporte de datos. Se comunica con los maestros y/o esclavos del sistema mediante la interfaz de comunicación AXI-Full ARM (2010). La idea principal detrás de la comunicación por medio de esta interfaz, es que en el mercado existen una gran cantidad de IPCores con esta interfaz que pueden ser fácilmente acoplados a la infraestructura SDBoC. Con esto se logra cumplir con una de las exigencias actuales en el mercado, que es la disminución de tiempos de desarrollo de productos por medio

de la reutilización de IPCores ya diseñados. Sin embargo, cuando se tiene un IPCore que no cuenta con esta interfaz de comunicación, el núcleo debe de ser envuelto dentro de un contenedor que les permita entablar una comunicación AXI-Full como se especificó en la sección 4.4.2. En las Tablas 4.6 y 4.7 se muestran los recursos necesarios para desarrollar este contenedor tanto para un elemento de procesamiento del tipo *maestro* como de un *esclavo*. En ambos casos se puede observar que la mayor parte de los recursos necesarios se consumen en el desarrollo de la propia interfaz. Alguna lógica adicional debe de ser añadida para permitir la correcta operación de la interfaz AXI-Full. En términos porcentuales en caso de que sea necesario desarrollar un contenedor para un elemento del tipo *maestro* el consumo de *slice registers* es del 0.26 %, mientras que el de LUTs es del 0.73 %. Para un elemento del tipo *esclavo* el consumo de *slice registers* es del 0.20 %, mientras que el de LUTs es del 0.57 %.

Tabla 4.6: Consumo de recursos por cada elemento del contenedor AXI-Full para un elemento de procesamiento tipo maestro

Elemento	Slice Registers	Slice LUT
Interfaz de lectura AXI-Full	101	104
Interfaz de escritura AXI-Full	121	181
Registros de control y lógica de acoplamiento	51	96
Total	272	386
Porcentaje (Zynq 7020)	0.26 %	0.73 %

Tabla 4.7: Consumo de recursos por cada elemento del contenedor AXI-Full para un elemento de procesamiento tipo esclavo

Elemento	Slice Registers	Slice LUT
Interfaz de lectura AXI-Full	98	131
Interfaz de escritura AXI-Full	119	75
Registros de control y lógica de acoplamiento	18	98
Total	225	304
Porcentaje (Zynq 7020)	0.20 %	0.57 %

Aunque desde el punto de vista del sistema de interconexión todas las Interfaces de Red se comportan igual, el procedimiento que siguen para conectarse con el *maestro* o el *esclavo* es diferente desde el punto de vista de la interfaz AXI. Es decir una Interfaz de Red que debe conectarse a un *maestro* lo hace por medio de una interfaz AXI del tipo *esclavo* ya que el *maestro* es el que inicia las transacciones. Y una Interfaz de Red que se conecta con un elemento tipo *esclavo* tiene una interfaz del AXI tipo *maestro* ya que es la propia Interfaz de Red la que inicia la transacción. La

Tabla 4.8: Consumo de recursos para una Interfaz de Red Maestra

Elemento	Slice Registers	Slice LUT	Memory
Interfaz de lectura AXI-Full	101	104	
Interfaz de escritura AXI-Full	121	118	
Buffer múltiple de entrada		3	16kb/flujo
Buffer múltiple de salida		3	16kb/flujo
Receptor de Red	100	143	
Transmisor de Red	100	137	
Unidad Selectora de Flujo 1	34	59	
Unidad Selectora de Flujo 2	34	59	
Recolector de Estadística	144	169	
Registros de Configuración	29	16	
Registros Especiales	66	68	
Registros de Control de Tráfico	96	68	
Unidad de Control	160	194	
Lógica de Acoplamiento		48	
Total	985	1252	
Porcentaje (Zynq 7020)	0.92 %	2.35 %	

arquitectura de las Interfaces de Red es similar como se muestra en la sección 4.4.3., presentándose la principal diferencia en la interfaz AXI. El consumo de recursos es muy similar en ambas implementaciones como se puede observar en las Tablas 4.8 y 4.9.

Los recursos consumidos por el Controlador del Bus se pueden observar en la Tabla 4.10. Una buena parte de estos recursos es consumida por el propio *árbitro*. En este caso se implementó un árbitro con política de arbitraje SuDO. Es de hacer notar que para poder explotar las ventajas que provee la filosofía SDN la recolección de estadística es fundamental. Sin embargo, hay un costo en el uso de recursos necesarios para generar esta estadística. En la implementación que se realizó aquí para un total de ocho nodos la unidad denominada Recolector de Estadística consume más del 40 % de los *slice registers* del Controlador del Bus y aproximadamente el 25 % de los *slice LUT*. Otra consideración que hay que tomar en cuenta es que, el Controlador del Bus, al ser considerado un elemento más del sistema de interconexión, se le deben añadir un elemento del tipo Receptor de Red y un elemento del tipo Transmisor de Red, los cuales en conjunto consumen aproximadamente el 20 % de los *slice registers* utilizados por el Controlador del Bus.

Tabla 4.9: Consumo de recursos para una Interfaz de Red Esclava

Elemento	Slice Registers	Slice LUT	Memory
Interfaz de lectura AXI-Full	119	75	
Interfaz de escritura AXI-Full	98	131	
Buffer múltiple de entrada		3	16kb/flujo
Buffer múltiple de salida		3	16kb/flujo
Receptor de Red	100	143	
Transmisor de Red	100	137	
Unidad Selectora de Flujo 1	34	59	
Unidad Selectora de Flujo 2	34	59	
Recolector de Estadística	144	169	
Registros de Configuración	29	16	
Registros Especiales	64	68	
Registros de Control de Tráfico	96	68	
Unidad de Control	162	198	
Lógica de Acoplamiento		52	
Total	980	1181	
Porcentaje (Zynq 7020)	0.92 %	2.21 %	

Tabla 4.10: Consumo de recursos del Controlador del Bus

elemento	Slice Registers	Slice LUT	Memory
Verificador de Seguridad	0	55	
Recolector de Estadística	448	611	
Receptor de Red	100	143	
Transmisor de Red	100	137	
Tabla de información de flujos		4	
Tabla de Información de nodos		8	
Unidad de Control	53	62	
Motor de Arbitraje (SuDO)	360	1136	
Controlador de Datos del Bus		261	
Banderas de Control de Tráfico	10	10	
Memoria de Doble Puerto			1kb
Total	1071	2428	
Porcentaje (Zynq 7020)	1.00 %	4.5 %	

Capítulo 5

Resultados

En el presente capítulo se muestran los resultados de las pruebas realizadas al sistema de interconexión desarrollado en este estudio. En primer lugar se presentan algunas consideraciones que debieron de ser tomadas en cuenta para el desarrollo de las pruebas, permitiendo estas consideraciones generar un marco de referencia que permite identificar con claridad el escenario sobre el que las pruebas se están realizando. En segundo lugar se presenta una clasificación de las aplicaciones que se ejecutan sobre el SoC. En tercer lugar se presentan los resultados de un estudio que muestra que las políticas tradicionales si bien ofrecen justicia en términos del acceso al medio, no pueden regular el uso del *bus* lo que las imposibilita para ofrecer QoS. En cuarto lugar se muestra que la política de arbitraje que aquí se propone (SuDO) es la que mejores resultados presenta cuando se trabaja en escenarios que ejecutan aplicaciones con dependencia de tareas, escenarios de uso muy común hoy en día. Finalmente se presentan los resultados donde se comprueba que la arquitectura aquí propuesta tiene la capacidad de trabajar bajo el modelo SDN. Además se muestran las ventajas que esto supone. Las pruebas muestran la posibilidad que tiene de trabajar el sistema con múltiples flujos, procesos con direccionamiento *multicast*, recolección de estadística y reconfiguración de flujos en *tiempo de ejecución*.

5.1. Consideraciones Generales

Para evaluar el comportamiento del sistema SDBoC propuesto, se desarrolló en SystemC un sistema que simula un SoC con un sistema de interconexión basado en *bus*. El sistema trabaja con presición de ciclo de reloj. El desarrollo se centró principalmente en la implementación de la Capa de Infraestructura que es el objetivo de evaluación de este trabajo. Sin embargo, para poder mostrar las capacidades de la propuesta, se desarrollaron un conjunto de Servicios de Red pertenecientes al Plano de Control del Sistema Operativo de Red.

En esta sección se muestra que la implementación SDBoC es capaz de soportar las capacidades que debe de tener un SoC que trabaje bajo la filosofía SDN, tal como se estableció en la sección 2.8. Sin embargo, estas capacidades no tendrían mucho sentido si no contribuyeran a incrementar las prestaciones del sistema. Básicamente contribuir a lograr una mejora en la QoS en cualquiera de las acepciones que se tengan de éste como se menciona en la sección 2.1.3. En este sentido, se ha observado que en cualquier sistema de interconexión existen recursos que deben de ser compartidos, y que para regular su uso es necesario la existencia de políticas de arbitraje que lo permitan. Es por esta razón, que además de mostrar las capacidades de la Capa de Infraestructura para lograr su operación bajo la filosofía SDN, se incluyen los resultados de un estudio de políticas de arbitraje, en términos de su capacidad de poder regular el uso de un recurso compartido. Desde mi punto de vista, agregar un árbitro que coadyuve a lograr esta regulación complementa las capacidades de un sistema de interconexión que trabaje bajo el paradigma SDN. La posibilidad de tener un árbitro que regule el uso del *bus*, aunado a la posibilidad de recolectar estadística, utilizar motores de optimización, y poder reconfigurar la Capa de Infraestructura en *tiempo de ejecución*, habilita al SoC a cumplir los requerimientos de QoS que se le impusieron.

Antes de presentar los resultados obtenidos en el desarrollo del presente trabajo, es importante resaltar algunas consideraciones presentadas por parte de la comunidad dedicada al estudio de los sistemas de interconexión en los SoC. Lo anterior con la finalidad de mostrar la importancia de las políticas de arbitraje dentro de los sistemas de interconexión.

- El principal cuello de botella que existe actualmente en los SoC y que puede afectar notablemente el rendimiento de las aplicaciones que se ejecutan en su interior, se encuentra en el sistema de interconexión Poletti et al. (2003); Benini and Bertozzi (2005); Ahmed et al. (2017); Ben Slimane et al. (2017); Dally and Towles (2003).

- Una característica deseable de un sistema de arbitraje es el poder controlar el ancho de banda que se le asigne a cada uno de los contendientes por un recurso compartido Poletti et al. (2003).

- No existe una política de arbitraje predominante, diferentes políticas de arbitraje presentan mejores o peores resultados dependiendo del escenario al que estén expuestas Poletti et al. (2003).

5.2. Clasificación de Aplicaciones

Con el objetivo de poder hacer una evaluación justa de las políticas de arbitraje, en este trabajo se ha hecho una clasificación de las aplicaciones que se ejecutan en el SoC. Con esta clasificación es posible plantear diferentes escenarios y conocer el comportamiento de cada política. La principal característica que es de interés conocer, es la capacidad que tienen las políticas de arbitraje para permitir el uso diferenciado del *bus* por parte de los elementos que lo utilizan, y como este uso diferenciado impacta en el rendimiento global del sistema.

De acuerdo a como las aplicaciones se ejecutan en los elementos de procesamiento que integran un SoC se identifican dos tipos de aplicaciones:

- **Aplicaciones con tareas independientes:** en este tipo de aplicaciones cada uno de los elementos de procesamiento ejecuta tareas cuya operación no depende de la operación de otra tarea. Una vez que el elemento de procesamiento transmite un paquete con el resultado de una tarea, se encuentra en posibilidades de solicitar nuevamente el acceso el *bus* cuando termine la siguiente tarea.

- **Aplicaciones con tareas dependientes:** este tipo de aplicaciones tienen tareas que se comunican unas con otras para ejecutar un algoritmo. Las tareas son distribuidas en diferentes elementos de procesamiento para poder tomar ventaja de su habilidad de poder trabajar concurrentemente y que la aplicación pueda tener un mejor rendimiento. Las tareas se comunican entre ellas por medio del sistema de interconexión a modo de paquetes. Una tarea solo puede comenzar a ejecutarse si ha recibido la totalidad de los datos de las tareas de las cuales depende.

De acuerdo al tipo de tráfico generado por las aplicaciones que corren en los procesadores elementales del SoC, el tráfico se ha clasificado como:

- **Tráfico homogéneo:** todos los *maestros* generan transacciones con un tamaño de paquetes similar, es decir la diferencia en la cantidad de *phit*s en la carga útil de los paquetes generados solo varía por algunas unidades.

- **Tráfico Heterogéneo:** El tamaño de los paquetes transmitidos por los *mastros* varía significativamente entre ellos, en el orden de las decenas.

El tamaño de los paquetes transmitidos a través de un sistema de interconexión depende de la aplicación que se encuentre ejecutando. Tomando como referencia los patrones de tráfico generados por la suite que presentan Liu et al. (2011) se han identificado tres tipos de paquetes:

- **Paquetes de tamaño pequeño:** Son paquetes compuestos por *flits* en el orden de unidades. Como un ejemplo se encuentra el algoritmo FFT-1024_complex que genera una carga útil de entre cinco y siete *flits*.
- **Paquetes de tamaño mediano:** este tipo de paquetes tienen una carga útil en el orden de la decenas de *flits*. En la suite el algoritmo FPPPP genera este tipo de tráfico.
- **Paquetes de tamaño largo:** Son paquetes con carga útil en el orden de los cientos de *flits*. El algoritmo de codificación de video H264 de la suite produce este tipo de paquetes.

De acuerdo al origen del patrón de tráfico generado por los elementos de procesamiento que se encuentran en el SoC se identificaron los siguientes patrones:

- **Patrones sintéticos:** son patrones que no pertenecen a ninguna aplicación o algoritmo en específico, son patrones cuyo tamaño es generado en base a un modelo matemático. Son utilizados principalmente en simulaciones de larga duración. Y son útiles para para realizar una evaluación estadística a nivel de sistema.
- **Patrones Reales:** estos patrones contienen información detallada y precisa de las tareas que componen una aplicación real. Permiten observar el comportamiento de un sistema de interconexión en situaciones de tráfico real.

5.3. Selección de la política de arbitraje

5.3.1. Políticas con acceso justo/rendimiento injusto

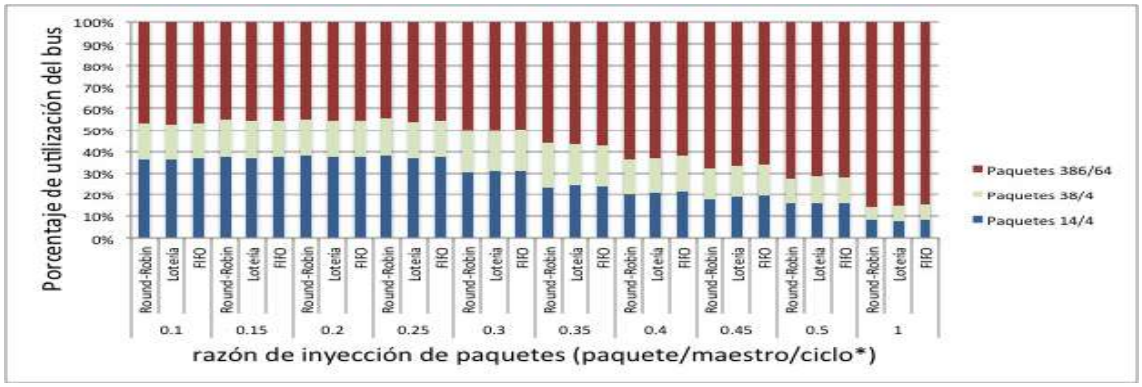
El interés inicial de este estudio se centró en identificar en qué grado las políticas de arbitraje comúnmente utilizadas en la interconexión de SoC, permiten el uso diferenciado del ancho de banda del bus. Es decir, identificar si alguna política ya desarrollada permite regular adecuadamente el uso del bus, y de nos ser así presentar una alternativa de solución. En el trabajo que se presentó en Ibarra-Delgado et al. (2020a), se muestra que las políticas tradicionales centran principalmente su atención en permitir un acceso justo al medio, no en controlar o regular el acceso al mismo. En este trabajo se probaron patrones de tráfico heterogéneo, reales y sintéticos. Con ambos se observó que los *maestros* que generan paquetes con carga útil más grande son los que acaparan el uso del *bus*. La técnica de comunicación que se utilizó entre los *maestros* y *esclavos* del sistema fue *split-transaction*. Con esta técnica, un *maestro* solicita que un *esclavo* ejecute una tarea con la característica que cuando

el *maestro* obtiene el acceso al *bus* transmite los datos al *esclavo* y libera el *bus* sin esperar el resultado. Con esto deja el medio libre para que pueda ser utilizado por otro elemento. Cuando el *esclavo* termina su operación, debe a su vez competir por el medio y cuando lo obtiene manda el resultado al *maestro*. Esta técnica permite que el *bus* quede libre mientras un *esclavo* se encuentra ejecutando una operación lo cual incrementa el rendimiento global del sistema.

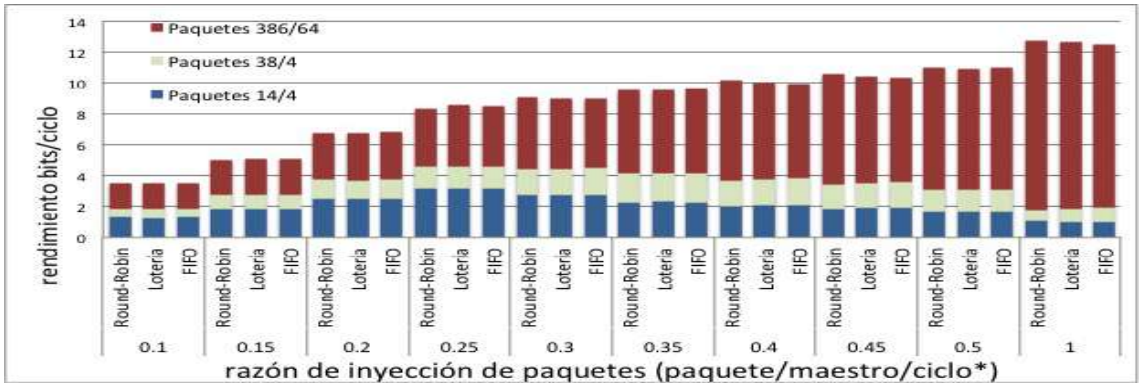
En el estudio se probaron tres políticas de arbitraje, *Round-Robin*, *Lotería* y *FIFO*. En el caso de *Lotería* la cantidad de *boletos* que se le asignaron a cada *maestro* fue el mismo.

En la primera prueba que se realizó, al sistema de interconexión se acoplaron tres *maestros* cada uno generando tráfico sintético de diferente tipo. El primer *maestro* genera paquetes con una carga útil de 64 *flits*, el *esclavo* tiene un tiempo de ejecución de 386 ciclos y responde con un paquete del mismo tamaño. El segundo *maestro* genera paquetes con carga útil de cuatro *flits*, el *esclavo* tiene un tiempo de ejecución de 38 ciclos y responde con un paquete del mismo tamaño. Finalmente el tercer *maestro* también genera paquetes con carga útil de cuatro *flits*, en este caso el *esclavo* tiene un tiempo de ejecución de 14 ciclos y responde con paquetes del mismo tamaño. Se varió la tasa de inyección de paquetes con el objetivo de observar el efecto que esta variación produce en cada política de arbitraje sobre el sistema de interconexión. En la Figura 5.1a se observa que las tres políticas presentan un comportamiento similar sin importar la razón en que cada *maestro* intenta inyectar los paquetes a la red. Se observa que cuando el *maestro* que genera los paquetes de mayor tamaño aumenta su tasa de inyección de paquetes hace un mayor uso del *bus* en detrimento de las otras dos aplicaciones. Esto se ve reflejado en el rendimiento individual de cada aplicación como se muestra en la Figura 5.1b. Se puede observar en la Figura 5.1c que la aplicación que genera los paquetes de mayor tamaño es la que menor cantidad de accesos tiene al medio y sin embargo es la que obtiene el mayor uso del *bus* y el mayor rendimiento.

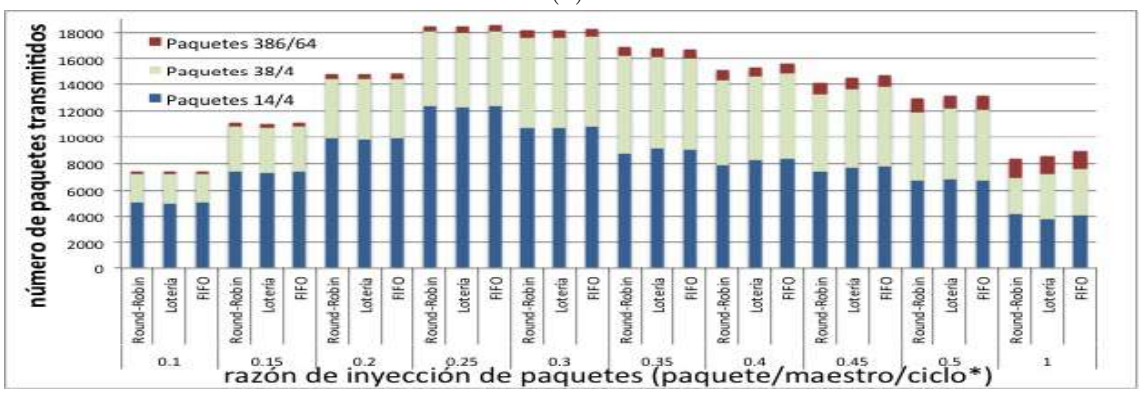
Una segunda prueba fue realizada utilizando una aplicación real correspondiente a la carga útil de un nano satélite. La aplicación genera imágenes cifradas y comprimidas para ser transmitidas a una estación terrena. Son tres los algoritmos que son procesados por la aplicación: primero, la imagen original, en formato de 640x480 píxeles, es pasada por un filtro de reducción de ruido para rectificar los posibles errores radio-métricos que hayan surgido durante su captura. El tamaño de paquetes que genera esta aplicación es de 640 *flits* y el tiempo de ejecución es de 640 ciclos de reloj; segundo, se utiliza un cifrador para proteger la información, el cifrador genera paquetes de dos *flits* y tiene un tiempo de ejecución de 16 ciclos de reloj; tercero, se usa un compresor para reducir el tamaño de la imagen original, el compresor genera paquetes de 640 *flits* y tiene un tiempo de ejecución de 6400 ciclos de reloj. La prueba fue realizada con los árbitros *Round-Robin*, *Lotteria* y *FIFO*. Una vez



(a)



(b)



(c)

Figura 5.1: Comportamiento de las políticas de arbitraje con aplicaciones heterogéneas generando tráfico sintético

más los resultados son muy similares. En la Figura 5.2 se muestra el rendimiento de la política *Lotería* cuando la tasa de inyección de paquetes se incrementa. Se puede observar que el rendimiento se ve más comprometido con el núcleo de cifrado cuando la tasa de inyección de paquetes se incrementa. La razón es que al ser el IPCore que genera los paquetes más pequeños, los otros dos algoritmos consumen la mayor parte del ancho de banda del *bus*.

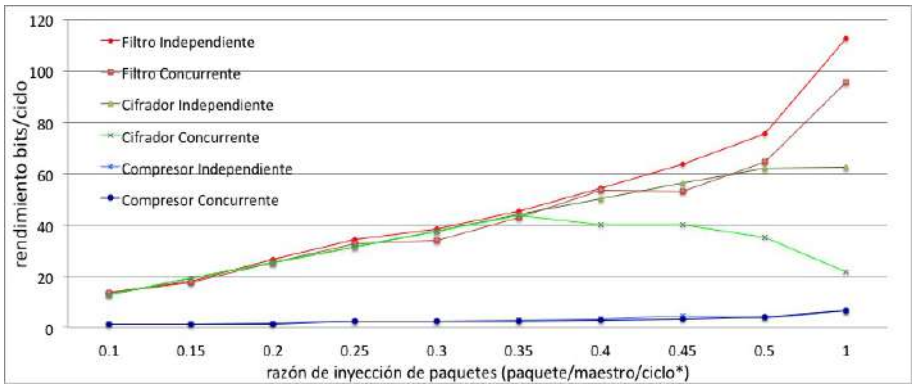


Figura 5.2: Comportamiento de la aplicación espacial cuando se varía la tasa de inyección de paquetes utilizando la política de arbitraje Lotería

Dado el carácter probabilístico de la política de arbitraje *Lotería*, se puede pensar en primera instancia que es un candidato para hacer un uso proporcional del *bus* de acuerdo al número de *boletos* que se le asignan a cada *maestro* en el sistema. Esto es cierto cuando se cumplen principalmente las siguientes condiciones: uno, los *maestros* que se encuentran compitiendo por el medio generan paquetes del mismo tamaño; dos, el sistema se encuentra saturado; tres, el tiempo de ejecución es similar. Es decir, cuando el sistema es homogéneo. Sin embargo, cuando los *maestros* en el sistema generan trafico heterogéneo esta política de arbitraje no es capaz de proporcionar un uso del *bus* proporcional al número de *boletos* que se le asignaron a cada *maestro*. Lo anterior es debido a que esta política regula la cantidad de ocasiones que un *maestro* puede obtener el medio, no el tiempo que lo puede usar. En la Figura 5.3 se puede observar que cuando existe tráfico heterogéneo, aún y cuando la relación de *boletos* se modifique a favor del *maestro* que genera paquetes de tamaño menor, esto no se ve reflejado en el porcentaje de uso del bus. El elemento que genera los paquetes de mayor tamaño continua teniendo el mayor uso del *bus* aún y cuando la cantidad de accesos al *bus* que obtuvo es mucho menor.

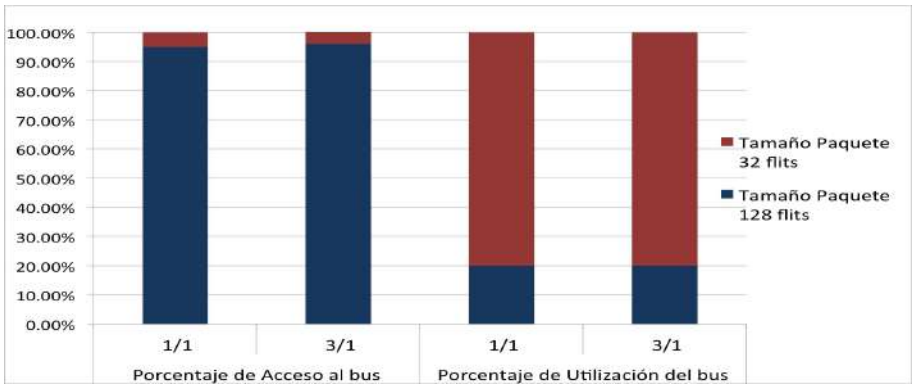


Figura 5.3: Comportamiento de la política de arbitraje Lotería con tráfico heterogéneo y pesos de soporte diferentes

5.3.2. Políticas con Regulación del uso del bus con aplicaciones con tareas dependientes

En la actualidad un uso que se le da a los SoC es dividir algoritmos complejos en un conjunto de tareas interdependientes que permitan aumentar su rendimiento. Esto se logra aprovechando el poder de cómputo al interior del SoC donde las tareas pueden ser ejecutadas concurrentemente. Teniendo esto en mente se procedió a evaluar qué política de arbitraje es capaz de soportar la ejecución de este tipo de tareas y que además permita el uso del *bus* de forma diferenciada para los elementos que lo requieren. Lo anterior con el fin de posibilitar alcanzar los requerimientos de QoS que se le impongan a las aplicaciones ejecutándose en el interior de un SoC. Para realizar la evaluación se establecieron una serie de parámetros y métricas que indicamos a continuación.

- **Política** P**:** en este parámetro se establece la política con que va a operar el Controlador del Bus en el sistema de interconexión.
- **Peso** W_i**:** indica la proporción deseada del ancho de banda de un *maestro*.
- **flitsize** fs**:** indica el tamaño en *bits* de la unidad de transmisión en un sistema de interconexión.
- **Utilización del Bus** U_b**:** esta variable acumula el total de ciclos de reloj que permanece el bus utilizado.
- **Bus Libre** I_b**:** esta variable acumula el total de ciclos de reloj que el bus no tiene actividad.

- **Tiempo de ejecución de una aplicación** $T_{appexec}(i)$**:** es el tiempo, en ciclos de reloj, que le toma a la *aplicación(i)* ejecutar todas las tareas que la componen.

- **Tiempo de ejecución de un *maestro*** $T_{masterexec}(i,j)$**:** es el tiempo en ciclos de reloj que le toma al $maestro_j$ transmitir los *flits* generados por las tareas de la *aplicación(i)*.

- **Tiempo de transmisión de un *maestro*** $T_{xflits}(i,j)$**:** es la cantidad de *flits* que son transmitidos por el $maestro_j$ cuando ejecuta las tareas corriendo en la *aplicación(i)*.

- **Rendimiento de un *maestro*:** es el rendimiento en bits/ciclo del $maestro_j$ ejecutando la *aplicación(i)*. Y está dado por:

$$Th_{master}(i,j) = \frac{T_{xflits}(i,j) \cdot fs}{T_{masterexec}(i,j)} \tag{5.1}$$

- **Porcentaje de utilización del *bus* por un *maestro*:** es el porcentaje de utilización del *bus* que el $maestro_j$ usa cuando ejecuta la *aplicación(i)*. Y está dado por:

$$U_{master}(i,j) = \frac{T_{xflits}(i,j)}{U_b + I_b} \cdot 100 \tag{5.2}$$

- **Tiempo total de ejecución:** es el tiempo total que le toma a las aplicaciones en un SoC ejecutar todas las tareas en que están divididas. Y está dado por:

$$T_{total} = \max_{i=1}^{n} T_{appexec}(i) \tag{5.3}$$

- **Rendimiento de una aplicación:** es el rendimiento en bits/ciclo de la $aplicacion_i$. Y está dado por:

$$Th_{app}(i) = \sum_{j=1}^{m_i} Th_{master}(i,j) \tag{5.4}$$

- **Porcentaje de utilización del bus de una aplicación:** es el porcentaje del *bus* que se utiliza cuando se ejecuta la *aplicación(i)*. Y está dado por:

$$U_{app}(i) = \sum_{j=1}^{m_i} U_{master}(i,j) \tag{5.5}$$

- **Rendimiento Global:** es el rendimiento total del sistema en bits/ciclo después que todas las aplicaciones han sido ejecutadas. Y está dado por:

$$Th_{overall} = \sum_{i=1}^{n} Th_{app}(i) \tag{5.6}$$

- **Porcentaje global de utilización del bus:** es el porcentaje total de utilización del *bus* cuando todas las aplicaciones han sido ejecutadas. Y está dado por:

$$U_{overall} = \sum_{i=1}^{n} U_{app}(i) \tag{5.7}$$

- m_i**:** es el número de *maestros* ejecutando la *aplicación(i).*
- n**:** es el número de aplicaciones ejecutándose en el SoC.

En la evaluación realizada se probaron las políticas de arbitraje RR, LTY, TDMA, WRR, WRRM y SuDO. Dependiendo de la política seleccionada a cada *maestro* en el sistema se le asigna un peso. El propósito de este peso y el origen de la política se puede observar en la Tabla 5.1.

Hoy en día, es común que en los SoC se ejecuten simultáneamente aplicaciones con dependencia de tareas y que además estas aplicaciones cumplan con requisitos de QoS. Esto implica que el sistema de arbitraje debe proporcionar dos capacidades: una, para operar, la capacidad de evitar *starvation* y *deadlock*; dos, para asegurar QoS, la capacidad de regular la utilización del *bus*, permitiendo a cada elemento de procesamiento un uso diferenciado del mismo.

El estudio del comportamiento de las políticas de arbitraje en escenarios que implican aplicaciones con dependencia de tareas ha sido poco estudiado. Desde mi perspectiva, es necesario evaluar el comportamiento de las políticas de arbitraje en cuanto a su capacidad para regular la utilización de los recursos. En este caso, el sistema de interconexión basado en *bus*. Esto debido a que dependiendo de la capacidad de controlar la utilización de un recurso, dependerá de la capacidad de cumplir con los requisitos de QoS establecidos para las aplicaciones.

En las pruebas que se realizaron en este trabajo, se utilizaron dos patrones de tráfico obtenidos de la suite presentada en Liu et al. (2011). El primer patrón de tráfico es el generado para la aplicación FFT-1024; esta aplicación tiene una gran cantidad de tareas (16.384) con (25.600) enlaces de comunicación; el tamaño de los mensajes que generan las tareas están en el orden de las unidades - entre 5 y 7 *flits* –. El segundo patrón de tráfico utilizado es el generado para la aplicación FPPPP; esta aplicación tiene un pequeño número de tareas (334) con (1.145) enlaces de comunicación; el tamaño de los mensajes que generan las tareas está en el orden

Tabla 5.1: Origen de la implementación y significado del peso para cada política

Política	Origen	Significado del peso asociado
RR	Williams and Towles Dally and Towles (2003)	En esta política el peso asociado no tiene ningún significado.
LTY	Lahiri et al. Lahiri et al. (2006)	Es el número de *boletos* asignados a cada *maestro*
TDMA	Implementación propia	Número de *slots* asignados a cada *maestro*
WRR	Williams and Towles Dally and Towles (2003)	Es el peso asignado a cada *maestro* en número de ciclos
WRRM	Williams and Towles Dally and Towles (2003)	Es el peso asignado a cada *maestro* en número de ciclos
SuDO	Basado en la arquitectura presentada en este documento	Presupuesto para cada *maestro* en número de ciclos

de las decenas - entre 50 y 60 *flits* –. Ambos patrones realizan un proceso de 20 iteraciones.

Los patrones fueron seleccionados por dos razones: la primera, al usar las dos aplicaciones se genera tráfico heterogéneo. La aplicación FFT-1024 genera una comunicación intensa con paquetes pequeños, mientras que la aplicación FPPPP genera una comunicación moderada con paquetes medianos; la segunda, en las pruebas que se realizaron, son los patrones que generan el mayor volumen de tráfico.

El proceso de mapeo que se realizó sobre las aplicaciones, se llevó a cabo en ocho elementos de procesamiento para ambos patrones. Lo anterior debido a que en las pruebas realizadas con este número de procesadores se tiene la mejor relación procesador/rendimiento. Cuando se ejecutaron las aplicaciones en modo autónomo, el mapeo FFT-1024 en ocho procesadores alcanzó un rendimiento de 29,13 bits/ciclo, y el mapeo FPPPP en ocho procesadores alcanzó un rendimiento de 15,95 bits/ciclo. Para realizar la evaluación, se utilizaron las métricas previamente establecidas: T_{total}, $Th_{app(i)}$, $Th_{overall}$, $U_{app(i)}$, y $U_{overall}$.

Se ha observado que algunas de las políticas que se han mencionado con anterioridad presentan dificultades cuando el escenario en el que trabajan contiene aplicaciones heterogéneas con dependencias de tareas. WRR es una política que puede entrar en *deadlock* en este escenario. Con LTY resulta complicado establecer

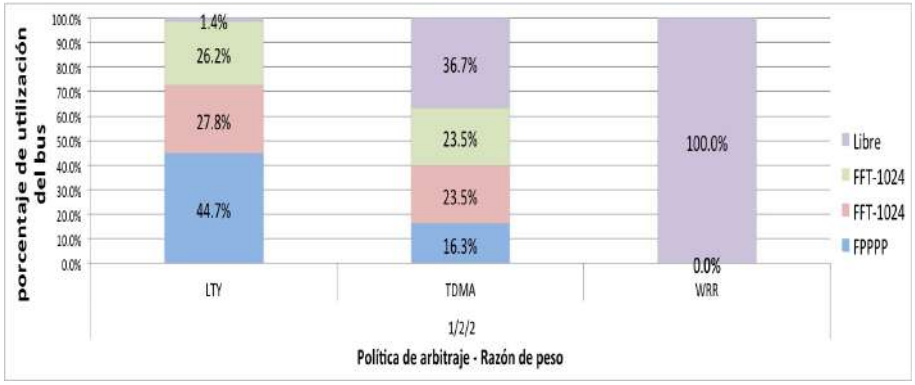
la cantidad de *boletos* que deben de asignarse a cada *maestro* para alcanzar el uso diferenciado del *bus*. TDMA es una política que permite el uso diferenciado del *bus* pero esto lo hace a costa del rendimiento global del sistema. Para corroborar lo anteriormente descrito se efectuó una simulación de las políticas de arbitraje en un escenario en el que se ejecuta una aplicación FPPPP y dos aplicaciones FFT-1024 con una relación de peso de 1/2/2. La Figura 5.4 muestra que la política WRR no genera resultados porque llega a un estado de bloqueo después de correr durante cierto período. La Figura 5.4a muestra que la política de arbitraje TDMA realiza una distribución de bus que tiende a obtener la razón de peso establecido. Sin embargo, el *bus* permanece ocioso por un tiempo prolongado (36,7 %), lo que afecta significativamente al rendimiento global del sistema $Th_{overall}$, como se muestra en la Figura 5.4b. La política de *Lotería* no puede controlar la utilización del *bus* para obtener la razón establecida en los pesos; en la Figura 5.4a, se puede ver que el mayor porcentaje de utilización está en FPPPP, que es la aplicación que genera transacciones de mayor tamaño. La simulación muestra que las dos aplicaciones FFT-1024 tuvieron mayor acceso al *bus* porque tenían un número más significativo de *boletos*, pero esto no refleja una mayor utilización del *bus*. En todas las pruebas que se realizaron con escenarios con aplicaciones con dependencia de tareas, el comportamiento de estas políticas es similar al que se muestra en la Figura 5.4. De aquí en adelante, para presentar resultados más concretos, se eliminan estas políticas.

A continuación, se proponen un par de escenarios que ejecutan aplicaciones con dependencia de tareas. El principal punto de interés es conocer hasta qué punto una política de arbitraje permite una utilización diferenciada del *bus*. También es importante observar cómo esta utilización diferenciada afecta al rendimiento global del sistema y al tiempo de ejecución de las aplicaciones.

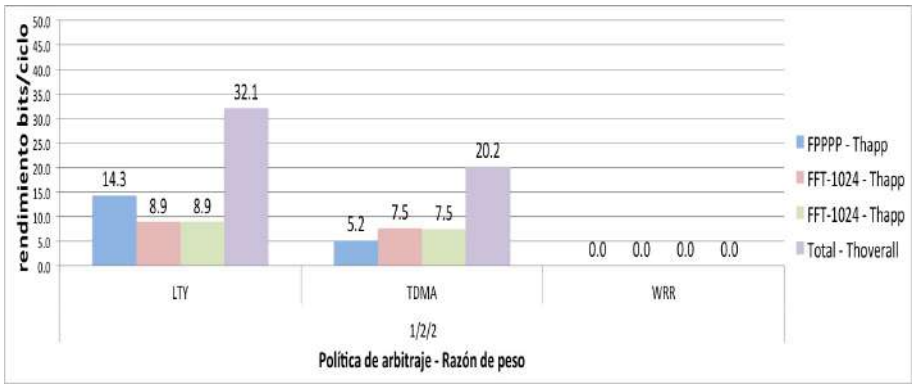
Escenario 1: *maestros* ejecutando simultáneamente aplicaciones con dependencia de tareas

En esta prueba, el escenario propuesto ejecuta tres aplicaciones con dependencia de tareas, una aplicación es FPPPP, y las otras dos aplicaciones son FFT-1024. El sistema ejecuta las aplicaciones utilizando en el Controlador del Bus las políticas de arbitraje RR, WRRM y SuDO. Para WRRM y SuDO, la relación de pesos establecida es de 1/1/1. Esta relación supone que los procesadores de las tres aplicaciones tienen la misma posibilidad de acceso al bus.

La Figura 5.5a muestra que las tres políticas distribuyen la utilización del bus de forma relativamente similar. La aplicación que obtiene la porción más considerable de utilización del bus U_{app} es FPPPP, esto es debido a que la aplicación FPPPP es la que genera las transacciones de mayor tamaño. El rendimiento general $Th_{overall}$, es similar en las tres políticas, como se ve en la Figura 5.5b siendo ligeramente



(a) Porcentaje de utilización del bus U_b

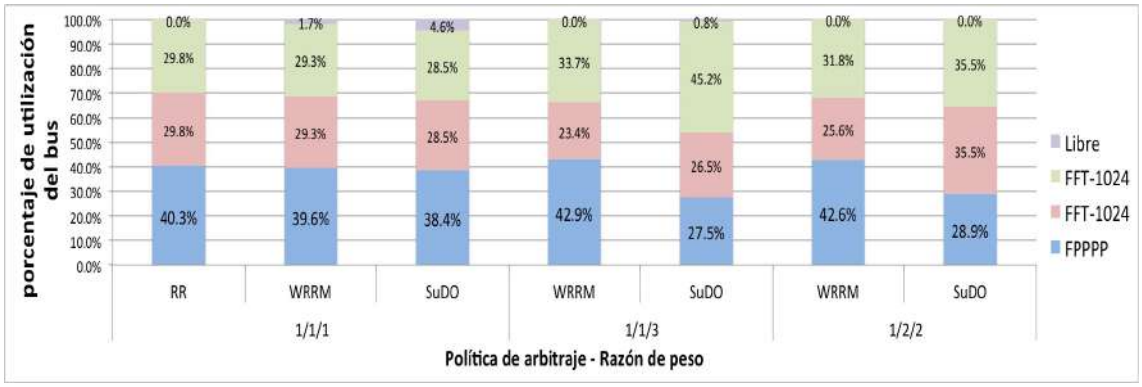


(b) Rendimiento por aplicación Th_{app} y rendimiento global $Th_{overall}$

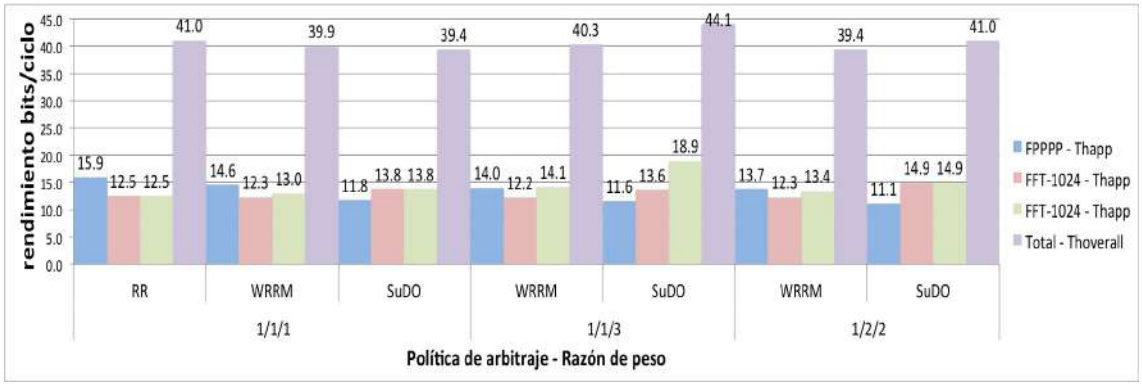
Figura 5.4: Algunos problemas detectados con políticas tradicionales corriendo en escenarios heterogéneos con aplicaciones dependientes. LTY tiene un control deficiente del ancho de banda. TDMA degrada la utilización del bus. WRR produce problemas de starvation y deadlock.

superior para Round-Robin. En cuanto al tiempo total de ejecución T_{total}, Round-Robin presenta un tiempo de ejecución ligeramente superior, como se muestra en la Figura 5.5c.

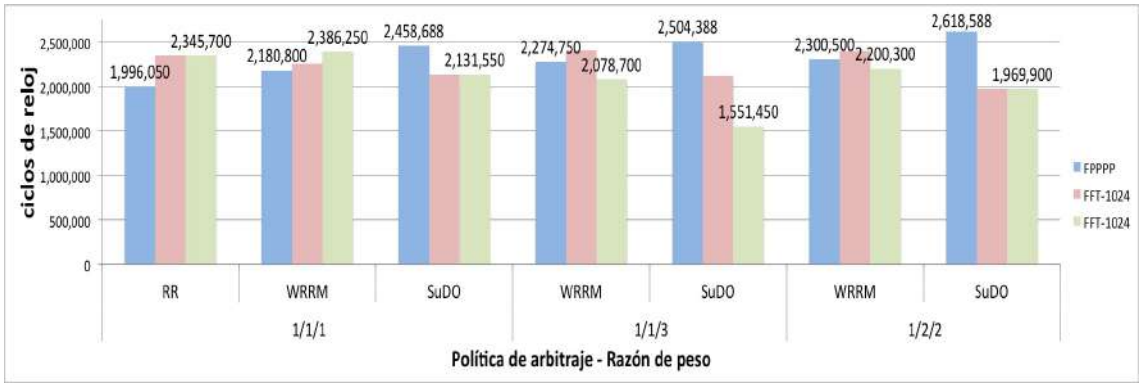
En la siguiente prueba, las aplicaciones fueron ejecutadas estableciendo en el Controlador del Bus las políticas de arbitraje WRRM y SuDO. En esta prueba, se quiere comprobar si una aplicación puede hacer una utilización diferenciada del *bus*. Para observar esto se fijó la relación de pesos en 1/1/3, lo que significaría una utilización teórica esperada del bus del 20 % para la aplicación FPPPP, el 20 % para una de las aplicaciones FFT-1024 y el 60 % para la otra aplicación FFT-1024. Se puede ver en la Figura 5.5a que WRRM no puede regular correctamente la utilización del *bus*. Aunque la aplicación FFT-1024 que tiene la mayor relación de peso ha aumentado su utilización del *bus*, U_{app} se encuentra muy lejos de la relación



(a) Porcentaje de utilización del bus U_b



(b) Rendimiento por aplicación Th_{app} y rendimiento general $Th_{overall}$



(c) Tiempo total de ejecución T_{total}

Figura 5.5: SuDO supera a las demás políticas de arbitraje en términos de control de ancho de banda en sistemas con *maestros* ejecutando diferentes aplicaciones con tareas dependientes.

deseada. Se puede ver que el porcentaje de utilización ganado por esta aplicación es a expensas de la otra aplicación FFT-1024. SuDO presenta una distribución más acorde con la proporción establecida de pesos; sin embargo, aún está lejos de la proporción deseada. Se puede observar que la ganancia se obtiene a expensas de la aplicación FPPPP, que es lo deseado. En cuanto al rendimiento, se muestra en la Figura 5.5b que el uso del árbitro SuDO no afecta al rendimiento global del sistema $Th_{overall}$, logrando que SuDO obtenga el rendimiento más alto. En cuanto al tiempo total de funcionamiento T_{total}, WRRM presenta el tiempo más corto, como se muestra en la Figura 5.5c.

La tercera prueba realizada en este escenario establece la relación de pesos en 1/2/2. Se espera un porcentaje de utilización del bus del 20 % para la aplicación FPPPP y del 40 % para cada aplicación FFT-1024_complex. Los resultados muestran que la mejor distribución de la utilización del *bus* T_{total}, según la relación establecida, se consigue con la política de arbitraje de SuDO, como se muestra en la Figura 5.5a. El rendimiento global, Th_{total}, es ligeramente superior para SuDO, como se ve en la Figura 5.5b. Sin embargo, el tiempo total de funcionamiento T_{total} es mayor en SuDO que en WRRM, como se muestra en la Figura 5.5c.

Escenario 2: *maestros* ejecutando aplicaciones iguales con dependencia de tareas

Otro escenario común en los SoC es tener diferentes instancias de la misma aplicación corriendo simultáneamente. En este caso, usamos tres instancias del patrón de tráfico FFT-1024 para ocho procesadores. La selección de esta aplicación se debe a que en las pruebas realizadas se muestra que el rendimiento del sistema se ve más comprometido cuando viajan paquetes de tamaño pequeño en el sistema de interconexión. Se realizaron las pruebas para las mismas políticas de arbitraje descritas en el escenario anterior. La relación peso/presupuesto establecida es de 1/1/1.

La Figura 5.6a muestra que independientemente de la política de arbitraje utilizada, cada una de las tres aplicaciones obtiene la misma utilización del bus U_{app}. El rendimiento general $Th_{overall}$ y el tiempo total de ejecución T_{total} es prácticamente el mismo en las tres políticas, como se muestra en las Figuras 5.6b y 5.6c.

La segunda prueba en este escenario se ejecuta con las políticas de arbitraje de WRRM y SuDO. Se fija la relación de pesos en 1/1/3. La Figura 5.6a muestra que WRRM no puede regular adecuadamente la utilización del *bus* U_{app}; se puede ver un porcentaje considerable de tiempo de inactividad del *bus*. SuDO presenta una mejor distribución de la utilización del *bus* de acuerdo a la razón de pesos establecida. Las aplicaciones que tienen la misma razón, obtienen el mismo porcentaje de utilización del *bus*. El rendimiento global del sistema $Th_{overall}$ es ligeramente superior en SuDO, como se muestra en la Figura 5.6b. La Figura 5.6c muestra que WRRM tiene un

tiempo total de funcionamiento menor (T_{total}).

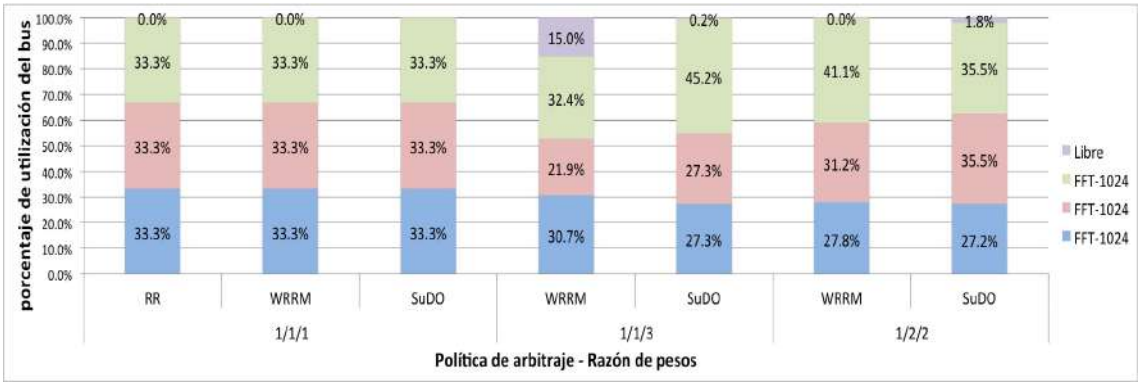
La última prueba establece una relación de peso de 1/2/2, lo que significa que una de las aplicaciones FFT-1024 tiene el 20 % del porcentaje de utilización del bus, mientras que las otras dos aplicaciones del FFT-1024 tienen el 40 % cada una. La Figura 5.6a muestra que la política de arbitraje de SuDO distribuye mejor la utilización del bus U_{app}. WRRM presenta un rendimiento global ligeramente superior, $Th_{overall}$, al de SuDO, como se muestra en la Figura 5.6b. El tiempo total de funcionamiento T_{total} es menor en WRRM, aproximadamente un 11 %, como se muestra en la Figura 5.6c.

5.4. Operación SDBoC

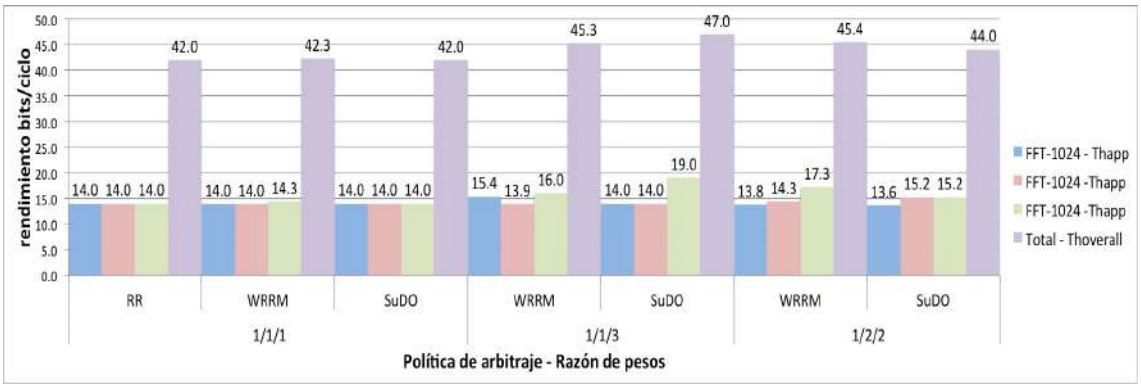
5.4.1. Ventajas del uso de flujos en un sistema de interconexión tipo *bus*

Una de las principales ventajas de trabajar bajo el esquema SDN es que se pueden establecer flujos de operación. En la implementación SDBoC que aquí se presenta esto supone una gran ventaja. La razón es que en aplicaciones con tareas encadenadas por medio de un flujo, el resultado de una operación no necesita ser enviado de regreso al emisor sino que puede ser directamente enviado al siguiente IPCore de procesamiento. Esto se logra estableciendo con anticipación las *Direcciones de Reenvío* de los datos. Son dos las ventajas que provee SDBoC al permitir el trabajo con flujos: la primera, se evita una sobrecarga innecesaria del *bus* con información redundante; la segunda, se obtienen tiempos de ejecución de las aplicaciones más cortos.

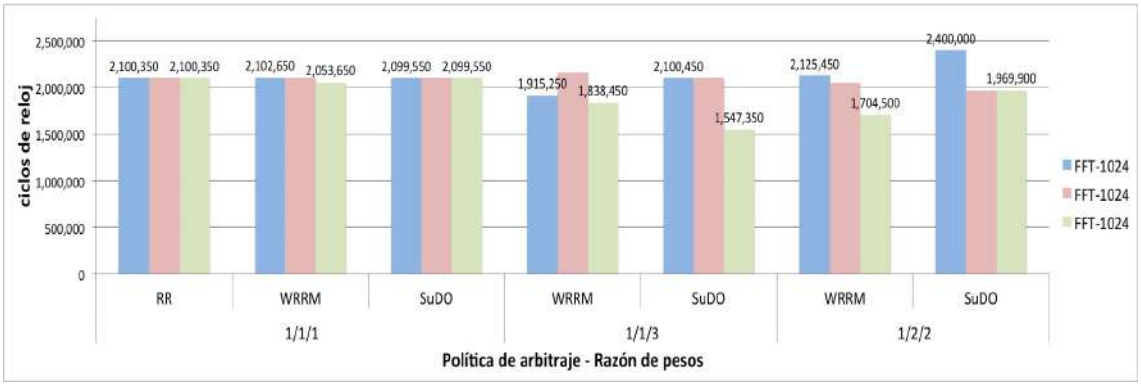
Se realizó una prueba sobre el sistema de procesamiento de imágenes satelitales establecido en Ibarra-Delgado et al. (2020a). La arquitectura del SoC es mostrada a la derecha de la Figura 5.7. Si el proceso se realiza por medio de un sistema de *bus* tradicional, el flujo que tendrían que seguir los datos es el siguiente (línea roja): una imagen es adquirida por un medio externo y es almacenada en la memoria principal (1). Un *maestro* –generalmente procesadores– toma esta imagen línea por línea y genera una imagen filtrada utilizando el IPCore de filtrado (2). Esta imagen es tomada por el siguiente *maestro* en el sistema (3), el cual por medio del IPCore de compresión, se encarga de reducir el tamaño de la imagen (4). La imagen comprimida es tomada por el último *maestro* en el flujo (5) y la cifra por medio del IPCore encargado de este proceso (6). Finalmente la imagen cifrada es transmitida por el Radio Tx (7). Cuando el sistema opera en modo SDBoC la operación anteriormente mencionada es modificada estableciendo un flujo Maestro 1 – IPCore Filtrado – IPCore Compresor – IPCore Cifrado – Maestro 3 como se muestra en la Figura 5.7 (línea azul).



(a) Porcentaje de utilización del *bus* U_b



(b) Rendimiento de la aplicación Th_{app} y rendimiento general $Th_{overall}$



(c) Tiempo total de ejecución T_{total}

Figura 5.6: *SuDO* supera a las demás políticas de arbitraje en términos de control de ancho de banda cuando en el sistema se ejecutan aplicaciones iguales con tareas dependientes.

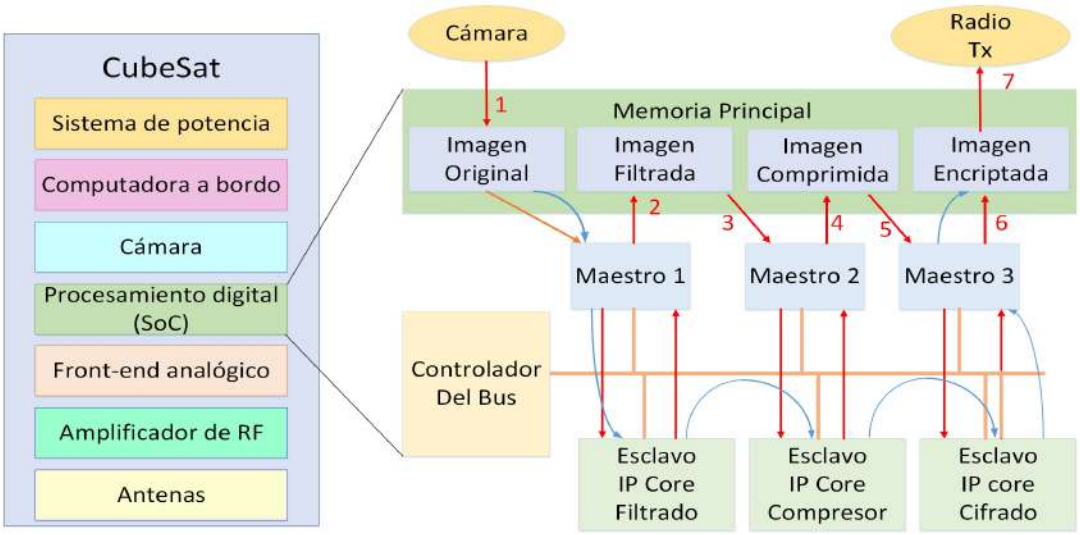


Figura 5.7: Arquitectura del sistema embebido CubeSat. Se muestra el flujo de datos de la aplicación empleada.

Los IPCores utilizados en esta prueba tienen diferentes características como tamaño de transacción y tiempo de ejecución. Para el sistema aeroespacial planteado en este estudio, se tienen los siguientes IPCores:

- **IPCore de Filtrado**, para eliminación de ruido causado debido a las condiciones en la que la imagen fue adquirida. La implementación hardware de estos filtros normalmente se realiza por medio de arreglos sistólicos que utilizan una máscara de convolución y realizan el proceso de filtrado a razón de un píxel por ciclo de reloj. El filtro empleado acepta como entrada una línea de una imagen original y genera como salida una línea de la imagen filtrada. Para la prueba se utilizó una implementación propia del algoritmo de suavizado de imagen con una latencia de un ciclo de reloj por píxel.
- **IPCore de Compresión**, disminuye el tamaño de la imagen para facilitar su transporte. En este trabajo, se decidió utilizar el algoritmo sin pérdidas *Lossless Image Compression Algorithm*, LOCO, con la implementación realizada en Hernández-Calviño et al. (2018) que logra una compresión de hasta 50 % del tamaño original. La implementación *hardware* de este algoritmo recibe como entrada una línea de la imagen, procede a realizar el proceso de compresión y entrega una línea comprimida. El IPCore tiene un tiempo de procesamiento de 2 ciclos/píxel.
- **IPCore de Cifrado**, la protección de información es clave en los sistemas embebidos. Se utilizó un cifrado AES el cual, una vez que se le ha dado la

llave de cifrado, recibe 16 Bytes de datos y entrega 16 Bytes de datos cifrados. La implementación AES-128 utilizada está descrita en Garcia-Luciano et al. (2017). El IPCore tiene un tiempo de procesamiento de doce ciclos de reloj.

En un esquema de *bus* tradicional, cuando un maestro toma el control del *bus*, manda un paquete de datos a un *esclavo* y queda a la espera de la respuesta manteniendo el control del *bus*, por lo que en este periodo el *bus* permanece inactivo. *Split-Transaction* es una técnica utilizada en sistemas de buses compartidos para evitar la inactividad del *bus*. Esta técnica divide la transacción en dos: primero, el *maestro* toma el control del *bus*, transmite el paquete, libera el *bus* y queda en espera de la respuesta por parte del *esclavo*; segundo, cuando el *esclavo* tiene un resultado disponible solicita acceso al *bus* y cuando lo tiene, transmite el resultado al *maestro* que solicitó la operación. El tiempo que tarda el *esclavo* en tener la respuesta, se denomina tiempo de respuesta t_r. Durante este tiempo el *bus* queda libre y a disposición de cualquier otro elemento en la red. La operación de la técnica *Split-Transaction* se puede observar en la Figura 5.8. Inicialmente, en la transición positiva del reloj en t0 (1), tres maestros solicitan el uso del *bus* (SolM(0) = 1, SolM(1) = 1, SolM(2) = 1). El *árbitro* otorga el *bus* al *maestro 0* (ConcM(0) = 1) el cual inicia con la transmisión de un paquete, conservando el uso del *bus* (SostM(0) = 1). Cuando termina de transmitir el paquete en t5, libera al *bus* (ConcM(0) = 0) (2) y queda a la espera de la respuesta por parte del *esclavo*. Al estar el *bus* libre, ahora el *árbitro* lo asigna al *maestro 1* en t5 (ConcM(1) = 1) (3), éste procede con la transmisión de datos desde t6 (SostM(1) = 1) (4) hasta t10 (SostM(1) = 0) (5), y queda a la espera de la respuesta por parte del *esclavo* al que transmitió el paquete. Dado que el *bus* vuelve a quedar libre, el *árbitro* concede su uso al *maestro 3*, y éste procede con la transmisión de un paquete de datos entre t11 (6) y t15 (7). En t16 (8) el *esclavo* asociado al *maestro 0* tiene disponible el resultado, solicita el uso del *bus*, lo obtiene y procede a la transmisión del resultado entre t17 (9) y t21 (10). En la Figura 5.8, se puede observar que el espacio de tiempo entre t5 y t16, es el tiempo que tarda el *esclavo 0* en procesar los datos que le fueron transmitidos por el *maestro 0*, espacio que fue utilizado por el *maestro 1* y el *maestro 2* para hacer uso del bus. Con lo que se comprueba que con esta técnica se tiene un mayor aprovechamiento del bus.

En la prueba realizada se midió el porcentaje de uso del *bus* cuando en el sistema se procesan imágenes de diferente tamaño. En la Figura 5.9 se puede observar que en todos los casos SDBoC tiene un uso del *bus* menor. Es de suponer que mientras la cadena de operaciones anidada sea mayor, esta diferencia podrá ser más notoria.

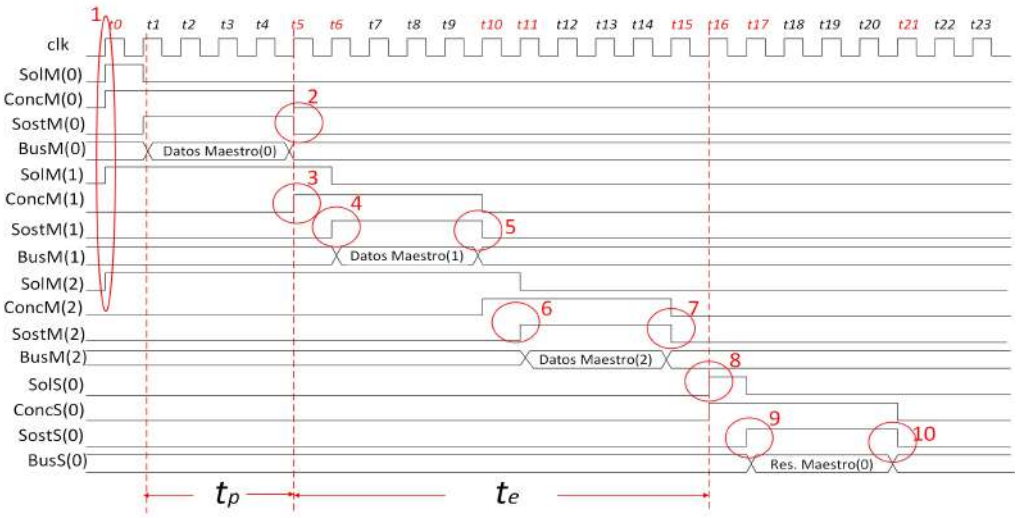


Figura 5.8: Diagrama temporal de las señales del bus con la técnica Split—Transaction. El tiempo disponible t_e entre transacciones del maestro 0 es aprovechado por transacciones de otros maestros, mejorando la utilización del bus.

5.4.2. Ejecución de procesos con direcciones *multicast*

SDBoC habilita la posibilidad de realizar operaciones con direcciones *multicast*. Un ejemplo del uso de este tipo de operaciones se encuentran en las aplicaciones satelitales. En este tipo de entornos, las aplicaciones deben de proveer sistemas tolerantes a fallos, como aquellos inducidos por la radiación existente en el espacio. Una técnica común utilizada en los sistemas satelitales es la denominada *triple redundancia*. En esta técnica tres sistemas independientes ejecutan el mismo proceso y su respuesta es tomada como válida si al menos dos de ellos generan el mismo resultado. SDBoC puede habilitar la ejecución de esta técnica configurando la misma dirección *multicast* a tres IPCores del mismo tipo.

Para observar el comportamiento de SDBoC operando con direcciones *multicast* se establecieron múltiples instancias de la aplicación satelital mencionada anteriormente. En la Figura 5.10 se puede observar que para los tres tipos de aplicaciones SDBoC presenta un menor porcentaje de uso del *bus*.

Los tiempos totales de operación resultaron similares en las aplicaciones que generan paquetes de respuesta del mismo tamaño que el paquetes origen. Pero cuando el tamaño del paquete de respuesta es menor, SDBoC tiene un tiempo total de ejecución menor. Tal es el caso del IPCore de compresión que en promedio genera paquetes de respuesta un 50 % menor que los paquetes originales. En las pruebas realizadas SDBoC se logró un tiempo de operación 10 % menor que el sistema tradicional.

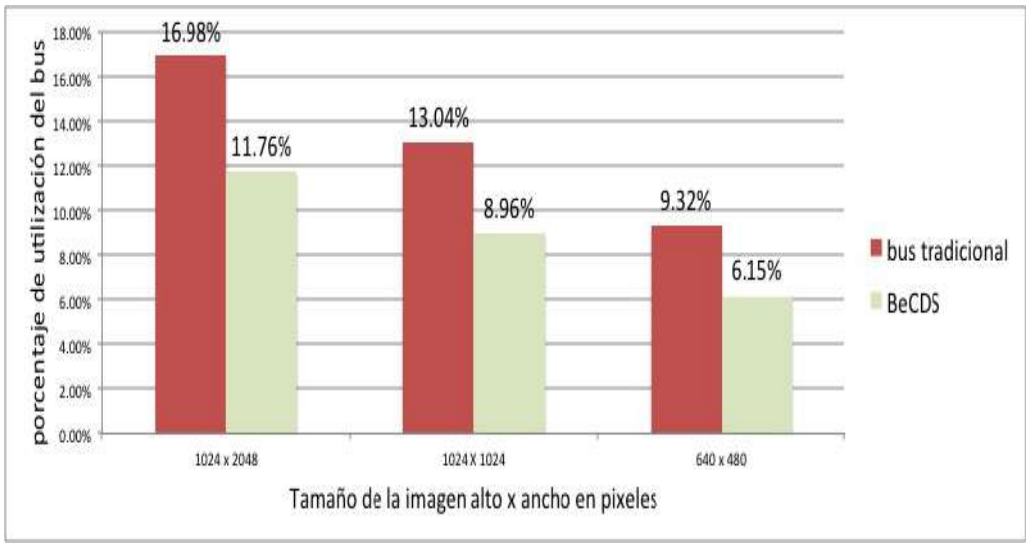


Figura 5.9: El uso de flujos en SDBoC le permite un menor porcentaje del uso del bus

5.4.3. Operación con múltiples flujos

SDBoC tiene la capacidad de poder operar simultáneamente más de un flujo. En la implementación realizada en este estudio cada IR tiene la capacidad de pertenecer a cuatro flujos diferentes. El IPCore que se encuentra atado a la IR puede operar en un modo diferente para cada flujo que se le establece. A cada uno de los flujos se le puede asignar un peso de soporte correspondiente al número de ciclos de reloj que podrá usar el bus dentro de una ventana de tiempo. En las simulaciones que se efectuaron para probar el trabajo con múltiples flujos la política de arbitraje utilizada fue SuDO de tal modo que los pesos de soporte corresponden al presupuesto que tiene cada una de los elementos que pertenecen a un flujo específico.

Se efectuaron un conjunto de pruebas para probar la capacidad que tiene SDBoC para operar con varios flujos simultáneamente. En las pruebas se establecieron cuatro flujos. Cada flujo consta de un *maestro* que inicia la operación del flujo y tres *esclavos* que operan el flujo, el resultado de la operación es transmitido del último *esclavo* en el flujo de regreso al *maestro* que inició la operación. Se establecieron paquetes de trabajo de 24 *flits* y el tiempo de operación por parte de los *esclavos* equivale al tamaño de los paquetes. El *bus* trabaja bajo la técnica de *split-transaction*. El sistema permanece en operación 2,000,000 de ciclos de reloj. El primer conjunto de pruebas se realizó con flujos independientes, es decir ninguno de los *esclavos* de un flujo es compartido con otro flujo. En la primera prueba se estableció un presupuesto igual para cada flujo, es decir se espera que los *esclavos* que pertenecen a cada uno de los flujos sean tratados equitativamente por el árbitro del sistema; en la segunda

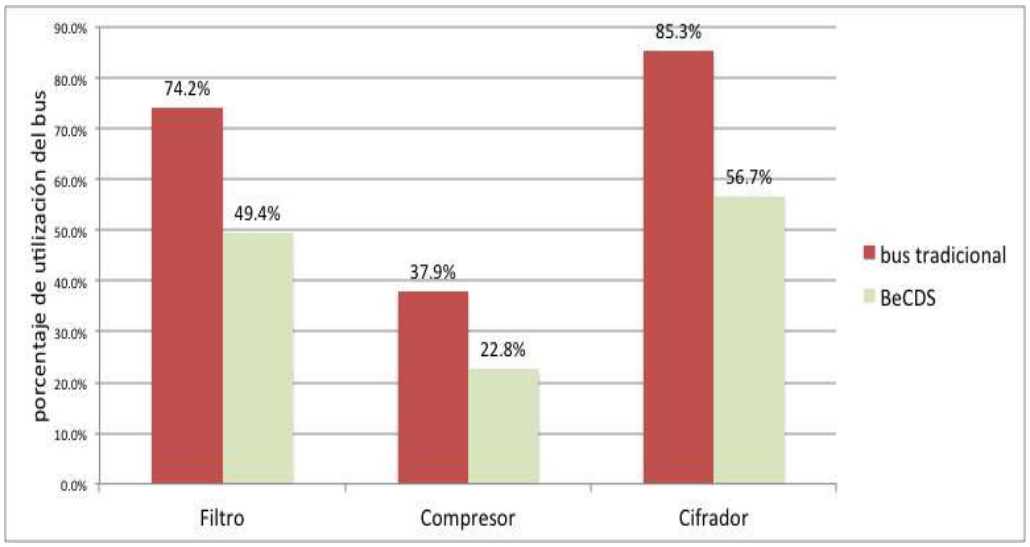
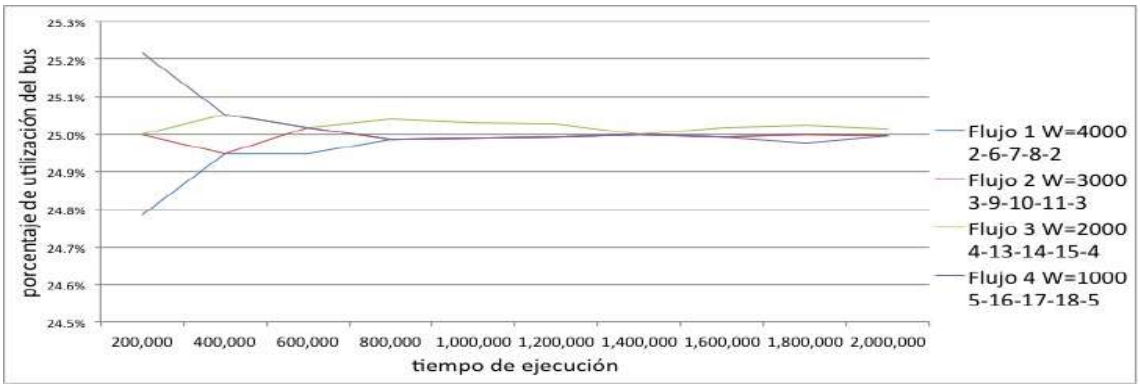


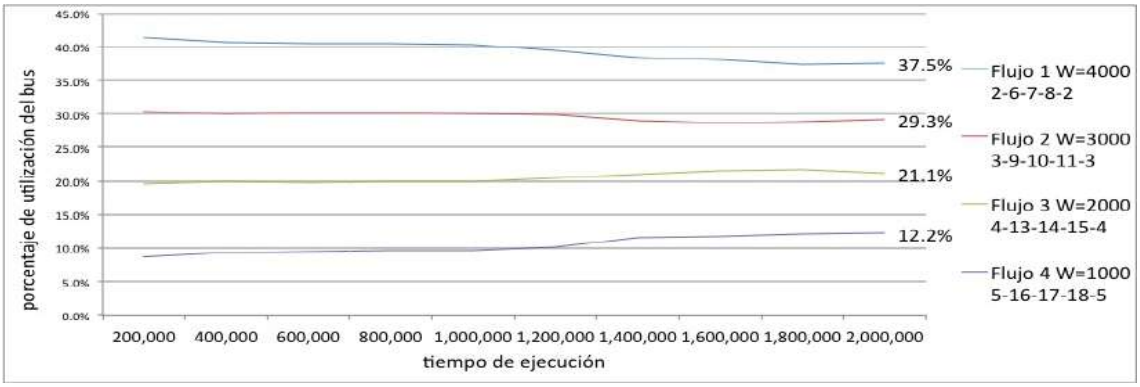
Figura 5.10: El uso de direcciones multicast para la ejecución en paralelo de múltiples instancias del mismo proceso en SDBoC, permite un menor porcentaje del uso del bus

prueba se estableció la relación de pesos en 4/3/2/1 esperando que el árbitro permita distribuir el uso del *bus* en un 40 % para el flujo 1, el 30 % para el flujo 2, el 20 % para el flujo 3 y el 10 % para el flujo 4. En la Figura 5.11a se puede observar que los cuatro flujos obtienen una distribución del *bus* similar muy cercana al 25 % cada uno. En la Figura 5.11b se muestra que los cuatro flujos tienden a obtener la relación que les fue establecida. Se puede observar que los flujos con mayor relación obtienen un uso del *bus* menor que el deseado mientras que los flujos con menor peso tienden a tener un uso mayor del *bus*. Una observación más a detalle del comportamiento de la política muestra que los elementos con mayor peso de soporte, cuando consumen su presupuesto piden prestado y el préstamo se les otorga. Sin embargo, al terminar la ventana de tiempo, al momento de recargar nuevamente el presupuesto, se les cobra el préstamo que se les hizo, esto implica que en la siguiente ventana la relación de pesos se vea ligeramente incrementada en los elementos con un peso de soporte menor, lo cual incrementa su porcentaje de uso del *bus*.

Una segunda prueba fue realizada con las mismas relaciones de pesos pero con otro escenario donde los flujos tienen elementos compartidos. Es decir un *esclavo* pertenece a dos flujos. En las Figuras 5.12a y 5.12b se puede observar que el comportamiento es muy similar a la prueba anterior.



(a)

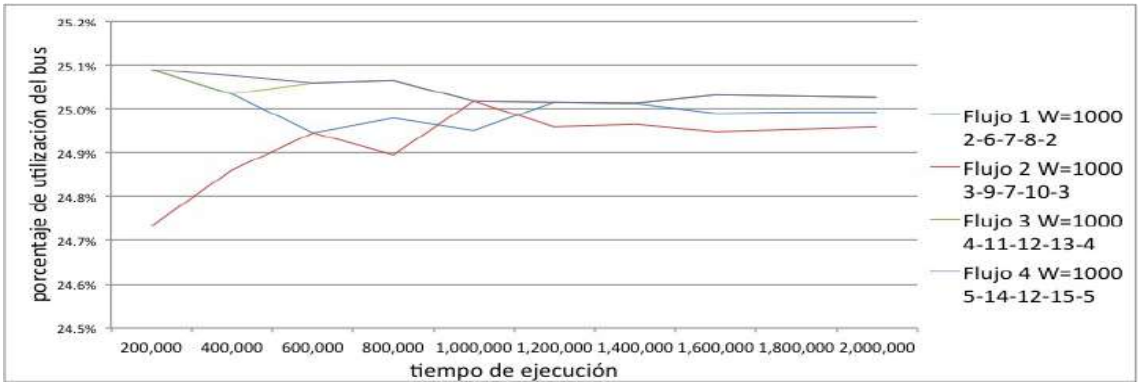


(b)

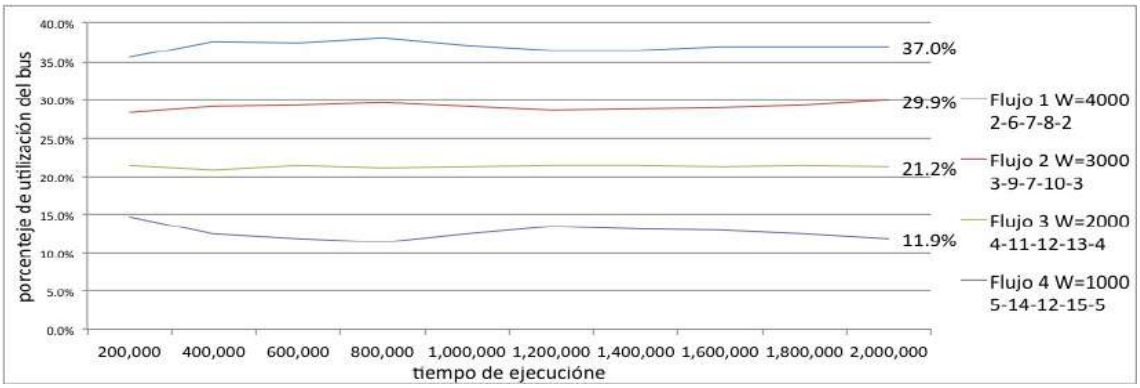
Figura 5.11: SDBoC operando cuatro flujos con nodos independientes. El porcentaje de utilización del bus corresponde con el peso establecido

5.4.4. Recolección de estadística

Una de las características principales de un sistema que trabaja bajo la filosofía SDN es que permite la posibilidad de conocer el estado del sistema en cualquier momento. En el caso de la implementación aquí realizada esto es posible con la generación de *scripts* que soliciten la estadística de uno o varios elementos del sistema de interconexión. En la Figura 5.13 se muestra un extracto de la simulación del ejemplo de simulación con múltiples flujos. En la gráfica el *maestro-SDN* (BousOut(1)) solicita la estadística de todos los elementos que se encuentran en el sistema cada 200,000 ciclos de reloj. Se puede observar que esta solicitud se realiza periódicamente. En la misma figura se muestra una vista ampliada de la misma traza en la que se pueda observar que realmente se hace una solicitud por cada uno de los nodos incluyendo al Controlador del Bus. El Controlador del Bus reporta estadística referente al tiempo del uso del bus en tres niveles: general, por flujo y por nodo.



(a)



(b)

Figura 5.12: SDBoC operando cuatro flujos con nodos compartidos. El porcentaje de utilización del bus corresponde con el peso establecido

Además, reporta por cada nodo y flujo el número de eventos (número de veces que les fue otorgado el *bus*). En el caso de los nodos, la estadística reporta el tiempo de espera en el *buffer* de entrada por cada flujo, es decir el tiempo que el paquete tiene que esperar antes que el IPCore lo atienda. También se reporta el tiempo de espera en el *buffer* de salida que es el tiempo que el paquete tiene que esperar para acceder al *bus*. Además, se reporta el tiempo de actividad del IPCore asociado al nodo. En todos los casos los nodos concluyen con la transacción actual, generan y transmiten la estadística, y continúan con su operación. El tiempo de respuesta depende del tiempo que el nodo tarde en transmitir la transacción actual. En la prueba aquí realizada se obtuvo un tiempo de respuesta promedio de 500 ciclos de reloj. Con respecto a la ventana de tiempo establecida para obtener la estadística el *maestro-SDN* tarda el 1.3 % de tiempo en conocer la estadística.

Una ventaja de la arquitectura aquí presentada, es que no es necesario conocer la estadística de todo el sistema de interconexión en un momento dado. El SDBoC

tiene la capacidad de decidir de cuáles elementos requiere la estadística. Por ejemplo, si el Controlador-SDN necesita saber cuál es el comportamiento de los flujos en términos del tiempo de uso del *bus* de cada uno, basta con solicitar esta estadística al *Controlador del Bus* lo que implica una sola respuesta que obtendrá inmediatamente después da haber hecho la solicitud.

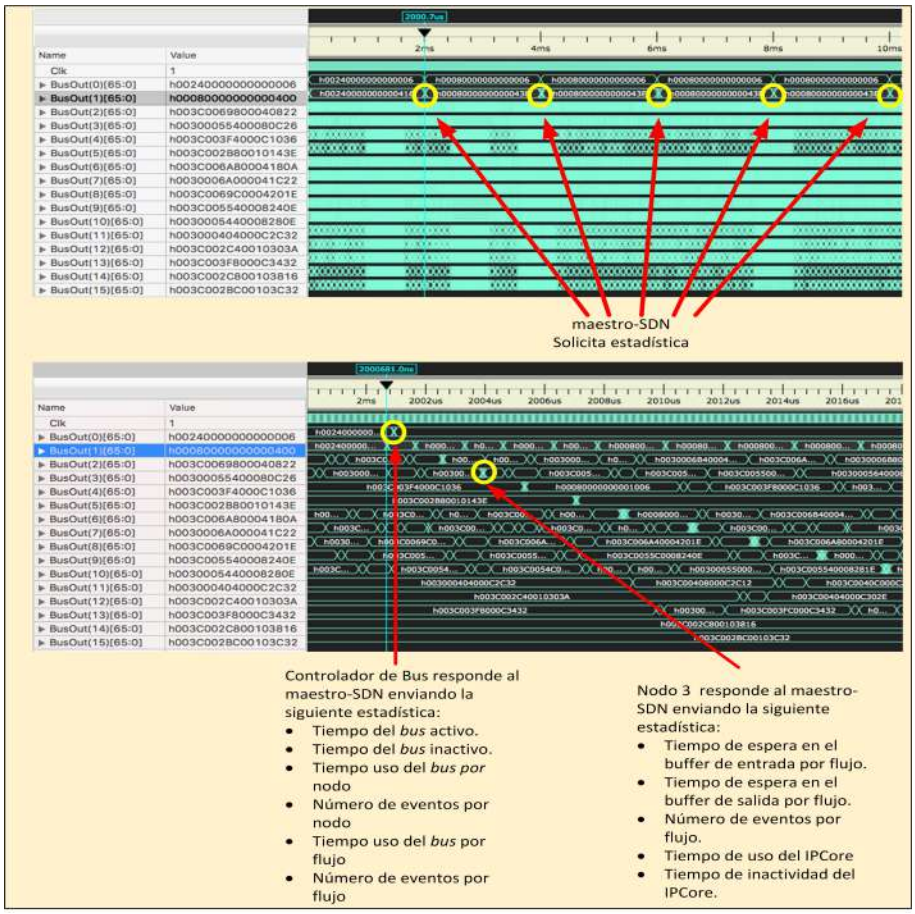


Figura 5.13: Extracto de traza de datos generada al correr una simulación donde el SDBoC solicita la estadística

5.4.5. Reconfiguración de flujos

Otra de las características fundamentales de SDBoC es su capacidad de reconfiguración en *tiempo de ejecución*. El proceso de reconfiguración puede ser ejecutado por diferentes razones. Por ejemplo: establecer nuevos flujos de operación; eliminar flujos de operación; cambiar de modo de operación; cambiar los pesos de soporte

para establecer la prioridad de los flujos o nodos, etc. El proceso de reconfiguración es llevado a cabo en el Plano de Control por medio del servicio Configuración de Red el cual ejecuta el *script* de configuración generado por alguna de las Funciones de Red.

Un proceso común que se lleva a cabo en SDBoC consiste en recolectar la estadística del sistema, suministrarla a un *motor de optimización* el cual establecerá qué cambios es posible hacer al sistema con el objetivo de que las aplicaciones que están corriendo en el SoC puedan cumplir con sus requerimientos de QoS. Como se ha mencionado anteriormente, el proceso de optimización queda afuera del ámbito de este estudio. Sin embargo, para ejemplificar el proceso de reconfiguración, se utiliza el escenario anteriormente planteado con cuatro flujos con nodos dependientes y relación de pesos 4/3/2/1. Esta relación es establecida por tres ventanas de tiempo y en la cuarta ventana como resultado de haber llamado al *motor de optimización* los pesos cambian a una relación 1/2/3/4 y el sistema es reconfigurado. Posteriormente en la séptima ventana de tiempo el *motor de optimización* indica que los pesos deben de ser reconfigurados nuevamente por lo que se genera un nuevo *script* de reconfiguración. El comportamiento del sistema se puede ver en la Figura 5.14 donde se observa cómo los flujos cambian su relación de porcentaje de utilización del *bus* de acuerdo a los nuevos pesos establecidos.

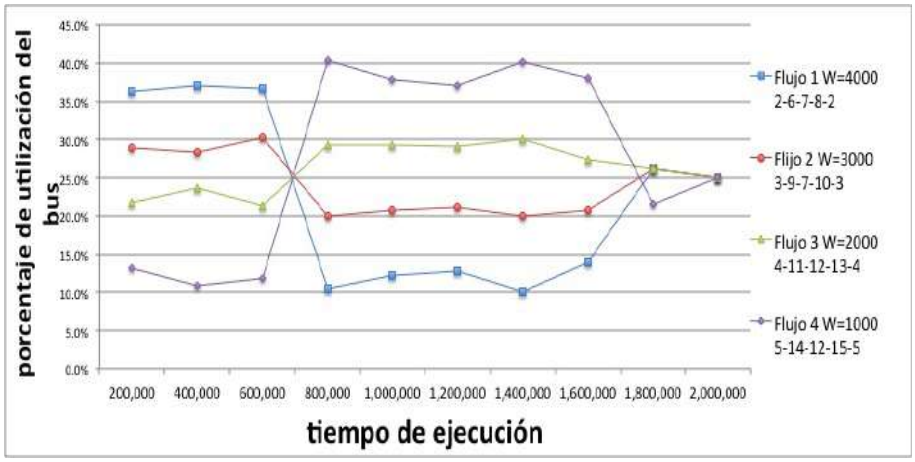


Figura 5.14: Proceso de reconfiguración de los pesos de los cuatro flujos inicia 4/3/2/1 luego se reconfigura a 1/2/3/4 y concluye en 1/1/1/1

En la Figura 5.15 se muestra otro ejemplo de reconfiguración. En este caso se tienen 3 flujos con nodos independientes. El flujo uno genera paquetes con carga útil de 48 *flits* mientras los flujos dos y tres generan paquetes con carga útil de 24 *flits*. Inicialmente, la relación de pesos de los flujos es la misma, por lo que se puede observar que el flujo uno, al tener paquetes del doble de tamaño que los otros dos

flujos, tiene un mayor uso del *bus*, cerca del 50 % contra el 25 % de cada uno de los otros dos flujos. En la cuarta ventana de tiempo la relación de tiempos para los flujos dos y tres se establece en el doble que el flujo uno. Al tener un mayor tiempo de uso del *bus* los flujos dos y tres pueden enviar más paquetes en una ventana de tiempo casi en una relación 2 a 1 con respecto al flujo uno, haciendo que el porcentaje de uso del *bus* casi se iguale para los tres flujos. Finalmente en la octava ventana de tiempo se vuelve a establecer una misma relación de tiempos lo cual hace que el comportamiento del sistema sea el mismo que al inicio de la simulación.

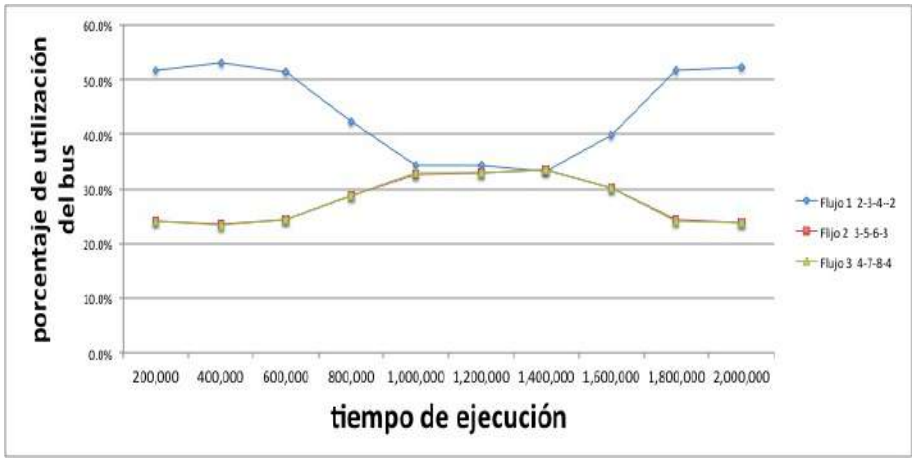


Figura 5.15: Proceso de reconfiguración con tres flujos inicia 1/1/1 luego se reconfigura a 1/2/2 y concluye en 1/1/1

Al igual que en el proceso de recolección de estadística, el elemento que es reconfigurado concluye la operación que se encuentra ejecutando e inmediatamente después establece su nueva configuración. Es decir el caso de un *esclavo* en el siguiente paquete que procese ya será realizado con la nueva configuración. En el caso del *Bus Controller* si son establecidos nuevos pesos de soporte, estos son almacenados en la *Tabla de Información de Nodos* y/o en la *Tabla de Información de Flujos*. Estos nuevos pesos serán tomados en cuenta hasta que el *motor de arbitraje* tenga la necesidad de recargar los pesos de soporte. En el caso de SuDO cuando todos los flujos o nodos hayan consumido su presupuesto. En este sentido, el peor escenario se presenta cuando el proceso de reconfiguración se solicita inmediatamente después de que el *motor de arbitraje* ha recargado los pesos de soporte, en este caso la latencia para recalibrar los pesos equivale a la suma de todos los pesos de soporte de los flujos o nodos activos.

Capítulo 6

Conclusiones

Como resultado del proceso de investigación y desarrollo que se llevó a cabo durante la realización del presente trabajo, se ha podido observar que establecer sistemas de interconexión que trabajen bajo el paradigma SDN permite multiplicar las capacidades de los SoC. A continuación se enumeran un conjunto de contribuciones, resultado de la presente investigación, que permiten incrementar estas capacidades.

- **Incremento en la flexibilidad del sistema de interconexión**:

  La arquitectura propuesta en este trabajo permite la posibilidad de reconfigurar en *tiempo de ejecución* la ruta de un flujo de trabajo por medio de la re-programación de la Dirección de Reenvío de los paquetes del flujo, incrementándose notablemente la flexibilidad del sistema de interconexión. Es posible establecer o eliminar una nueva característica en un flujo de trabajo modificando una sola Dirección de Reenvío. Por ejemplo en el caso de procesamiento de imágenes satelitales, la información podría ser transmitida a una estación terrena en modo cifrado o sin cifrar, haciendo o no haciendo al IPCore de cifrado parte del flujo de operación. Y esto se logra con la única modificación de la dirección de reenvío del IPCore que lo antecede.

- **Posibilitar la compartición adecuada de los recursos para asegurar QoS:**

  En este trabajo se ha mencionado que el principal cuello de botella de un SoC es su sistema de interconexión. En el caso de un sistema de interconexión tipo *bus* este problema es más pronunciado. Con la política de arbitraje que se propone en este trabajo se permite hacer un uso diferenciado del *bus* más acorde con las necesidades de ancho de banda de los algoritmos que se ejecutan en el sistema, lo cual coadyuva a lograr que las diferentes necesidades de ancho de banda de cada algoritmo puedan ser alcanzadas y con esto cumplir con sus requerimientos de QoS.

- **Obtención de estadística para la toma de decisiones:**

  El diseño que aquí se ha propuesto permite recolectar en *tiempo de ejecución* la estadística generada por el tráfico que existe en la red. El análisis de la estadística por elementos en capas superiores del modelo de referencia permite tomar acciones de control (reconfiguración) en *tiempo real* con la finalidad de mejorar el rendimiento global del sistema. Además, las acciones de control tomadas utilizan solo un fragmento del *bus* el cual es mínimo comparado con los beneficios que se pueden lograr por esta acción de control. En este punto es importante mencionar que el proceso de reconfiguración sobre un nodo o un flujo no impide la operación de los otros nodos y/o flujos en el sistema de interconexión.

- **Mejora del rendimiento del Sistema de Interconexión basado en *bus*:**

  La capacidad que presenta la arquitectura SDBoC de poder operar un sistema de interconexión tipo *bus* con flujos de operación permite un incremento notable en el rendimiento del sistema. Los paquetes que son utilizados para llevar el control de la secuencia de un flujo, utilizan un tiempo de *bus* mínimo comparado con la cantidad de tráfico que se genera en un sistema tradicional. Por otro lado el poder redirigir los paquetes de procesamiento en *tiempo real* hace que el *bus* sea muy flexible y posibilita su uso en escenarios complejos como por ejemplo en los sistemas satelitales. En este tipo de escenarios, donde no es posible hacer un cambio en el flujo de los datos físicamente, con el cambio de una Dirección de Reenvío es posible establecer un nuevo algoritmo de procesamiento.

- **Aseguramiento de Integración y Re-usabilidad de IPCores:**

  El paradigma SDN tiene como uno de sus pilares el poder aislar el transporte de los datos de su procesamiento. La Capa de Infraestructura de la arquitectura aquí presentada, tomando en cuenta este modelo, separa los elementos de transporte de los elementos de procesamiento con lo cual posibilita el uso de interfaces muy conocidas y utilizadas por la comunidad de desarrolladores de IPCores. Con esto se permite el re-uso de bloques funcionales que pueden ser fácilmente integrados en el sistema, reduciendo lo anterior los tiempos de desarrollo de un producto. Es importante hace notar que al aumentar las interfaces de comunicación entre diferentes elementos del sistema de interconexión, se tiene un *overhead* en términos de recursos utilizados y tiempo de procesamiento. Sin embargo, existen aplicaciones que se pueden permitir esta posibilidad en aras de lograr un menor tiempo de desarrollo.

- **Protección del Sistema de Interconexión de transacciones no autorizadas:**

Bajo el paradigma SDN el Sistema Operativo de Red tiene el conocimiento de qué nodos y qué flujos son los que pueden tener actividad en el Sistema de Interconexión. Bajo la arquitectura aquí presentada esta información es posible transferirla al Controlador del Bus de tal modo que éste no solo impida la posibilidad de acceso al *bus* de nodos y/o flujos no autorizados, sino que además reporte la intrusión al Sistema Operativo de Red para que se efectúen las acciones de control necesarias.

Aunque en este trabajo solo se realizaron pruebas con un sistema de interconexión tipo *bus* es posible visualizar que algunas de las contribuciones hechas pueden ser utilizadas en otro tipo de sistemas de interconexión que trabajen bajo la filosofía SDN.

La posibilidad de poder recolectar estadística en *tiempo de ejecución* en una SDNoC permite la posibilidad de identificar rutas saturadas de tal modo que los flujos pueden ser reconfigurados modificando las Direcciones de Reenvío de un nodo en particular.

Como se ha mencionado anteriormente cualquier sistema de interconexión tiene en un momento dado la necesidad de compartir recursos; la adición de un árbitro con una política de arbitraje que le permita regular el uso de los recursos, como la presentada aquí, puede ser de gran utilidad para que el SoC pueda cumplir con sus requerimientos de QoS.

En términos de la seguridad del sistema de interconexión, es posible el diseño de una unidad de seguridad reconfigurable controlada directamente por el Sistema Operativo de Red.

Aunque finalmente se puede decir que un sistema de interconexión para un SoC es estático, la arquitectura aquí presentada le da un grado de dinamismo que le permite una gran flexibilidad. Lo anterior puede ser de gran utilidad cuando en un SoC se deseen ejecutar aplicaciones que no estuvieron contempladas en el diseño original y que requieran la adición de nuevos IPCores. Con las nuevas tecnologías que existen en términos de dispositivos lógicos programables, es posible, por medio de SDN, establecer un sistema de interconexión fijo que permita la integración en *tiempo de ejecución* de IPCores que puedan ser intercambiados a voluntad utilizando la tecnología de reconfiguración parcial. Aunque esta posibilidad no fue estudiada en este trabajo, sí se establecen las bases del sistema de interconexión para que en un futuro pueda ser utilizado. La idea es contar con un sistema de interconexión SDBoC al que se le puedan integrar o quitar elementos de procesamiento por medio de reconfiguración parcial. Con esto, aunque se tenga un sistema de procesamiento que pueda operar pocos núcleos, es posible contar con un repositorio de IPCores en memoria secundaria listos para ser integrados al sistema de interconexión en cualquier momento.

Finalmente, se puede decir que con el desarrollo del presente trabajo se aportó evidencia que demuestra que las características que son deseables en un sistema de interconexión para SoCs, es posible obtenerlas desarrollando sistemas de interconexión utilizando el modelo SDN.

Apéndice A

Diagramas de Máquinas de Estados Finitos

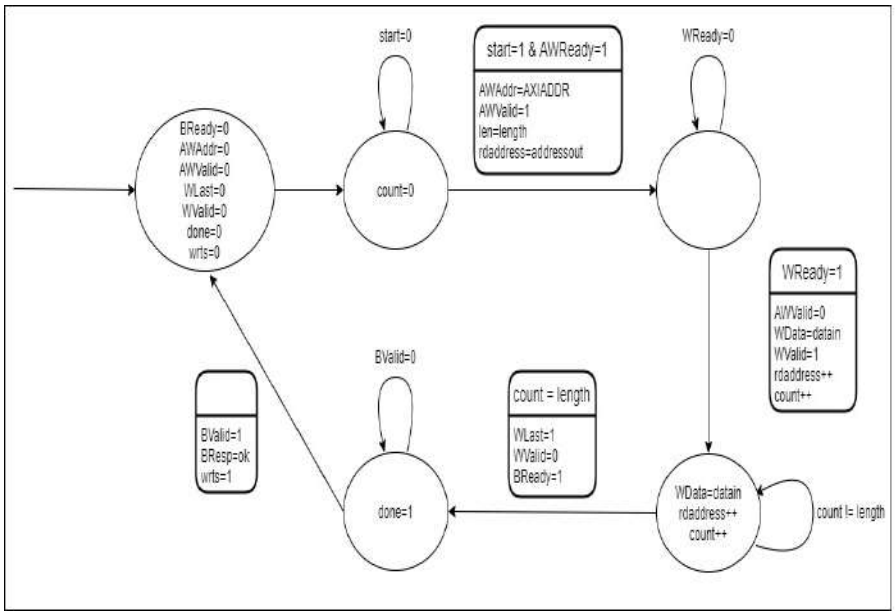


Figura A.1: Diagrama de la MEF de la Interfaz *Maestro de escritura AXI-Full* en un elemento tipo *maestro-SDN/maestro*

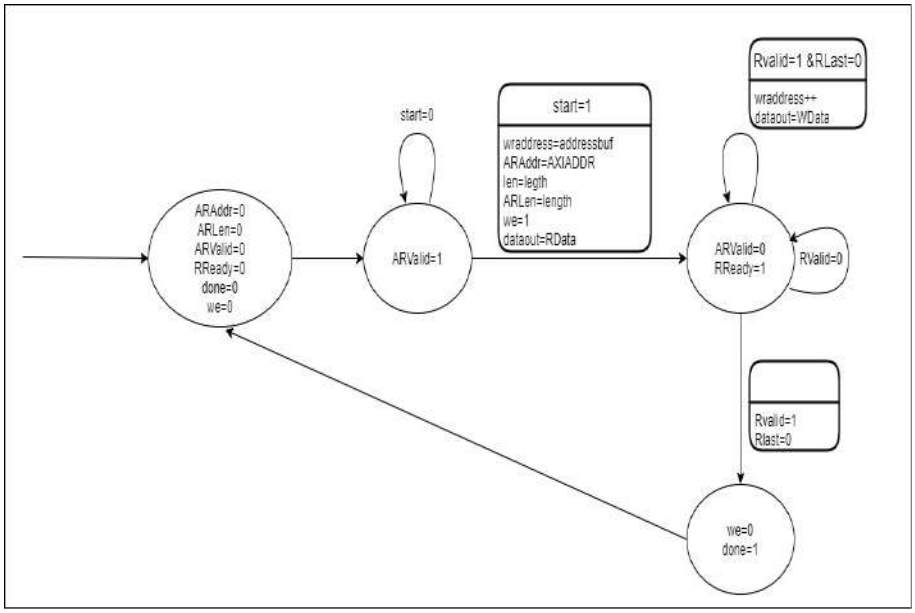


Figura A.2: Diagrama de la MEF de la Interfaz *Maestro de lectura AXI-Full* en un elemento tipo *maestro-SDN/maestro*

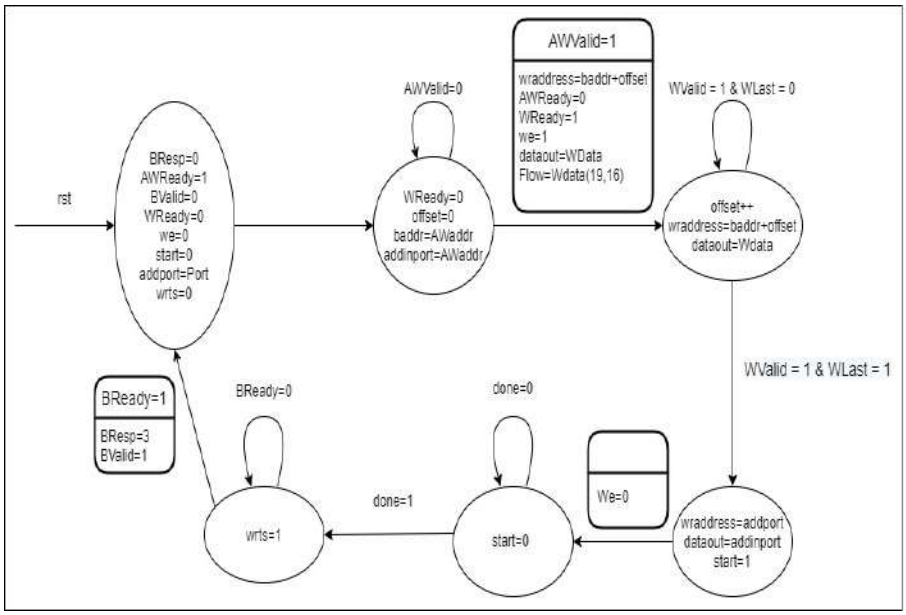


Figura A.3: Diagrama de la MEF de la Interfaz *Esclava de escritura AXI-Full* en un elemento tipo *esclavo*

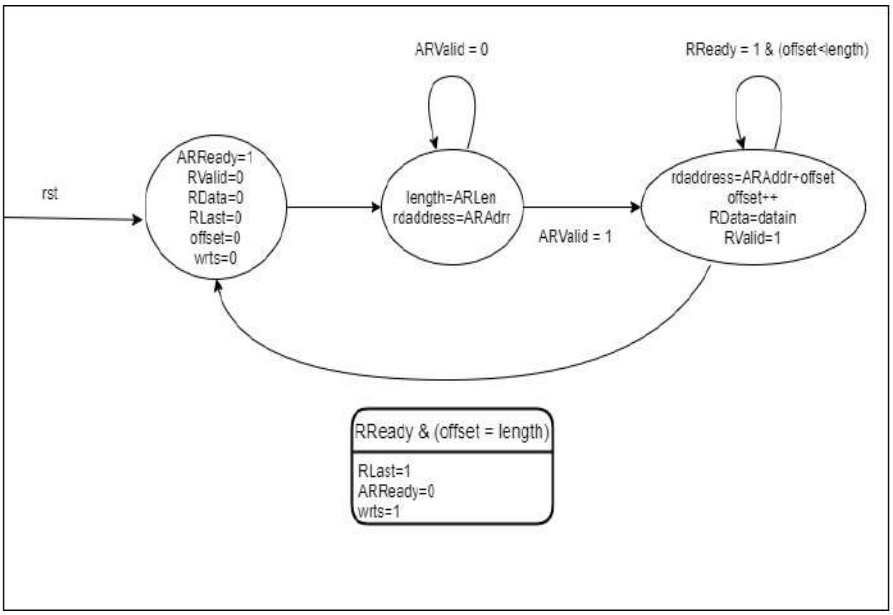


Figura A.4: Diagrama de la MEF de la Interfaz *Esclava de lectura AXI-Full* en un elemento tipo *esclavo*

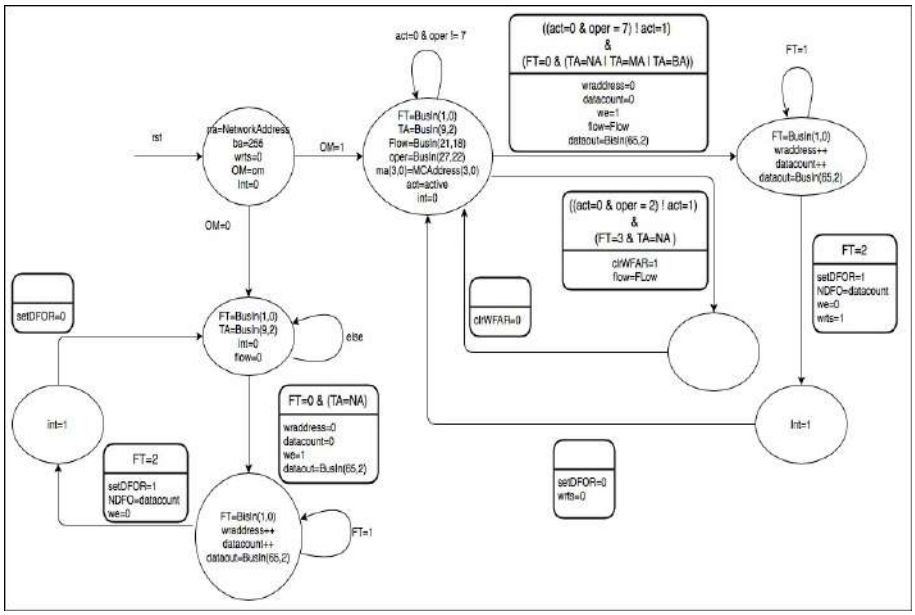


Figura A.5: Diagrama de la MEF del *Receptor de Red*

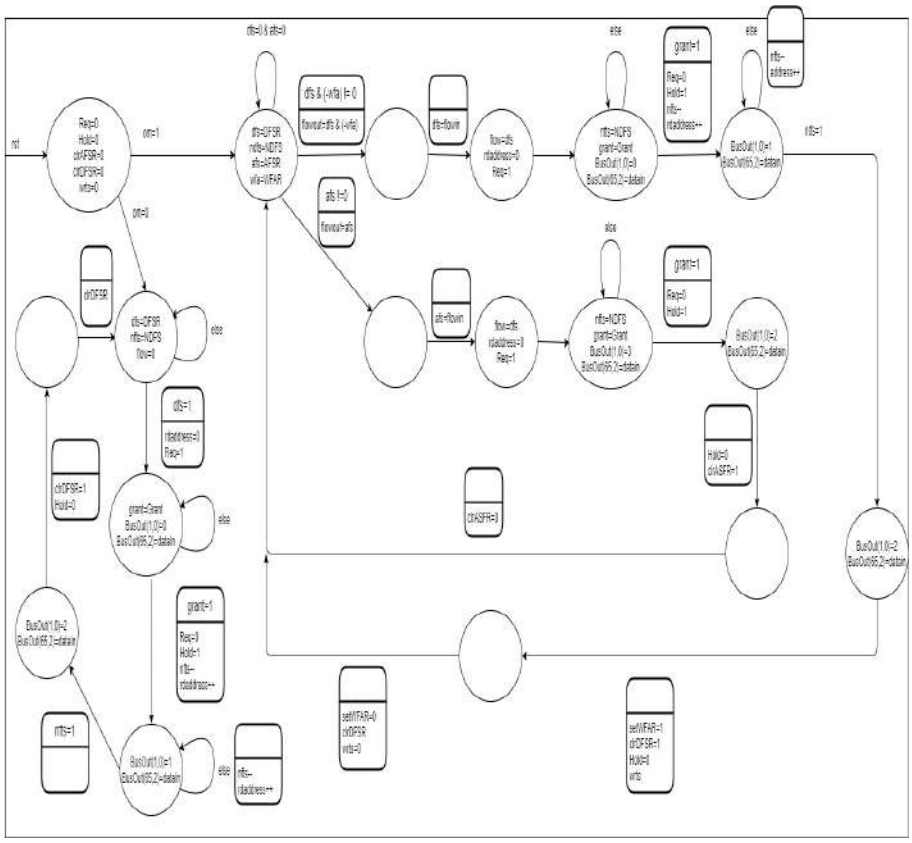


Figura A.6: Diagrama de la MEF del *Transmisor de Red*

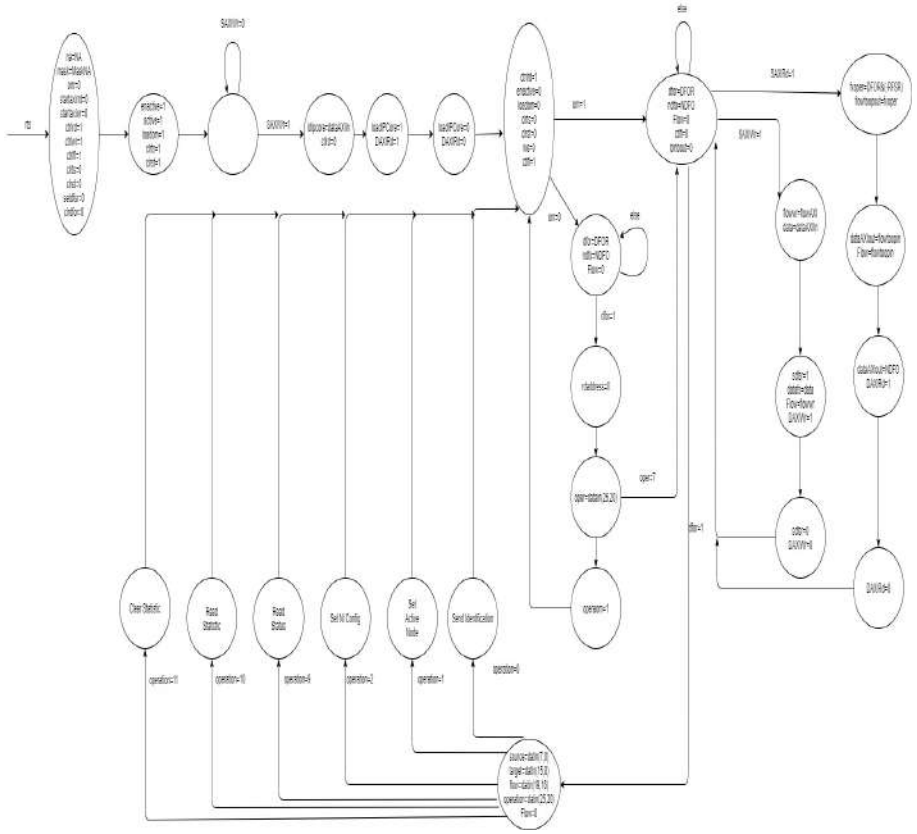


Figura A.7: Diagrama de la MEF de la *Unidad de Control* de la *Interfaz de Red Maestra*

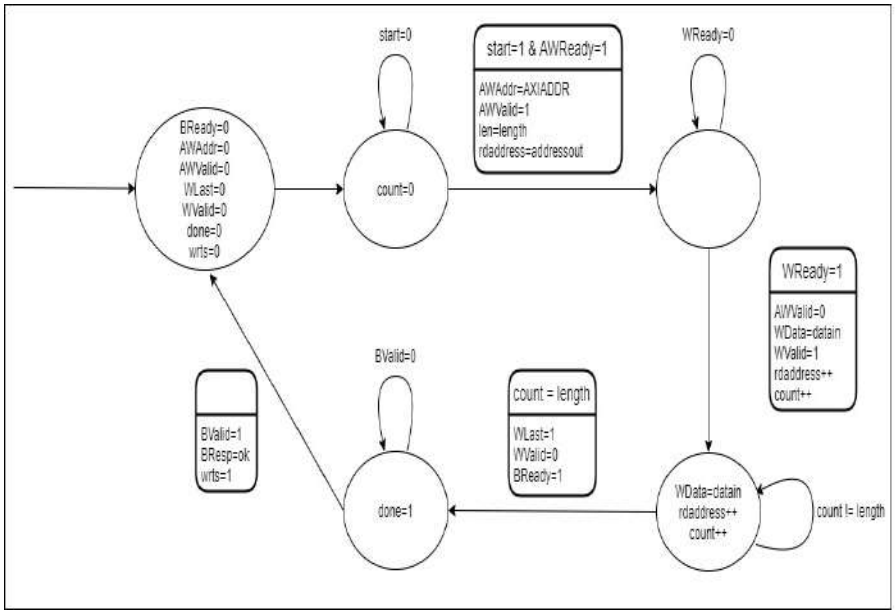


Figura A.8: Diagrama de la MEF de la Interfaz *Maestro de escritura AXI-Full* en la *Interfaz de Red Esclava*

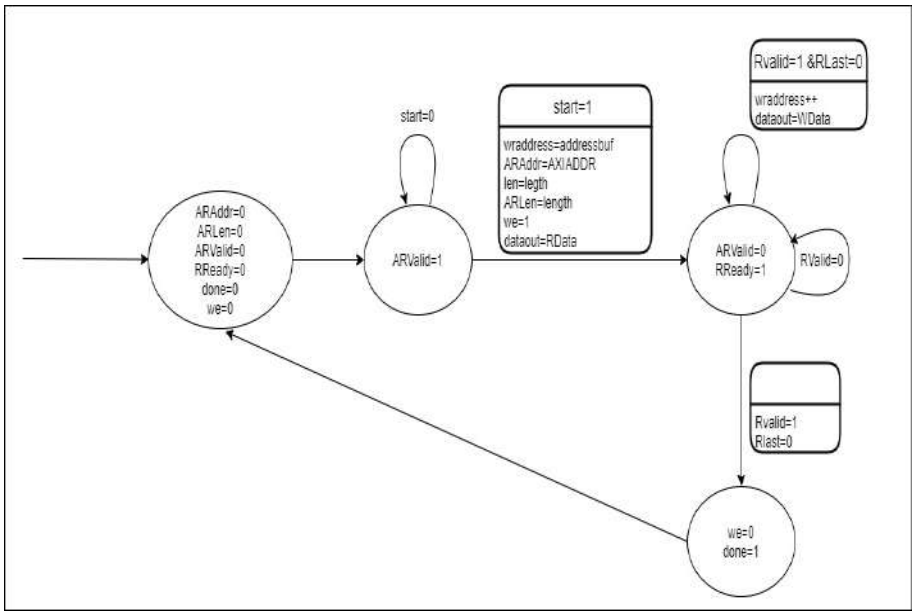


Figura A.9: Diagrama de la MEF de la Interfaz *Maestro de lectura AXI-Full* en la *Interfaz de Red Esclava*

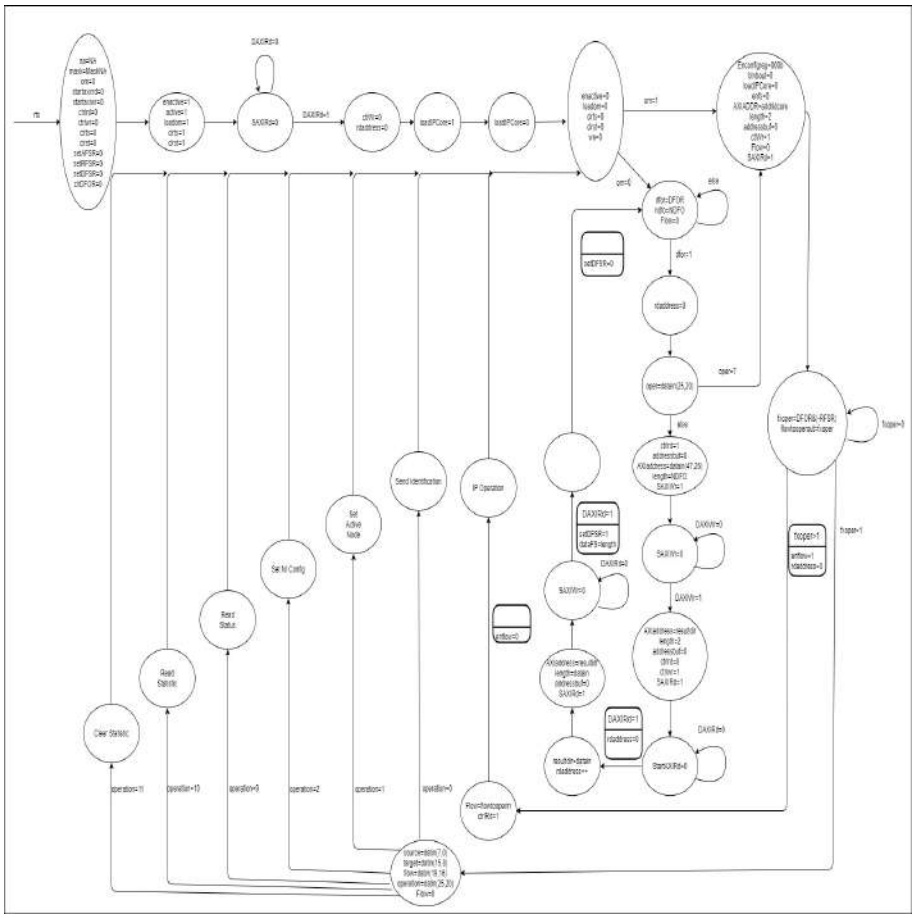


Figura A.10: Diagrama de la *Unidad de Control* de la *Interfaz de Red Esclava*

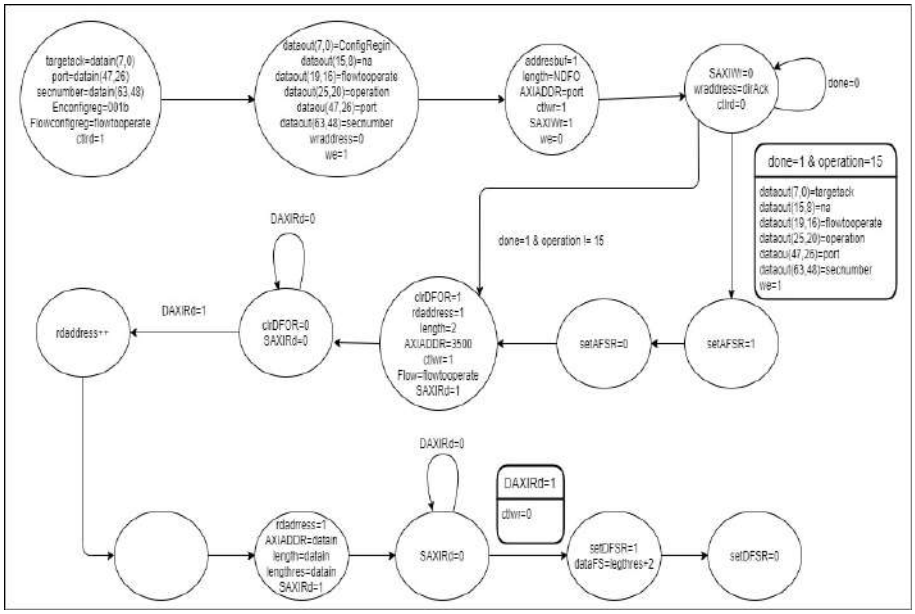


Figura A.11: Diagrama de la *Unidad de Control* (operación) de la *Interfaz de Red Esclava*

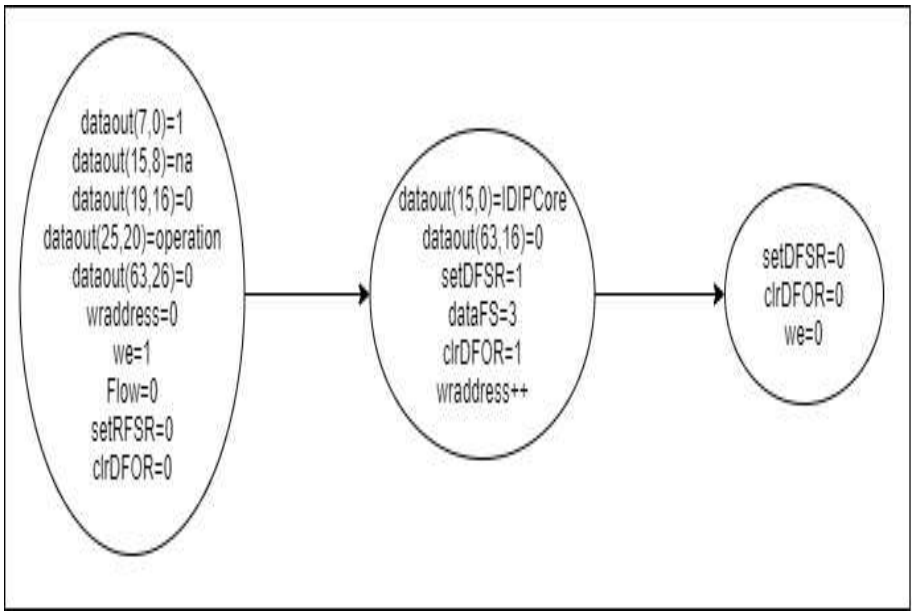


Figura A.12: Diagrama de la *Unidad de Control* (envía identificación) de la *Interfaz de Red Esclava*

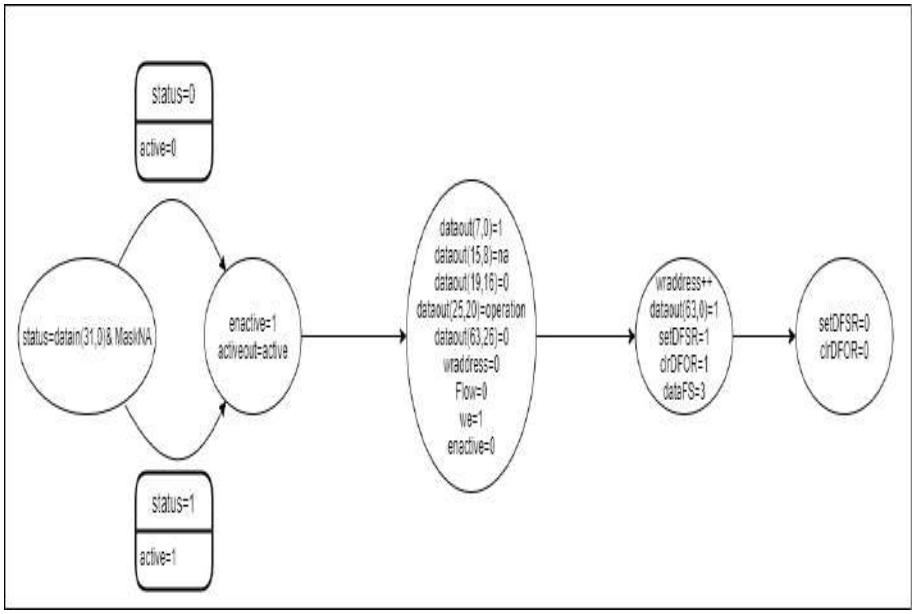


Figura A.13: Diagrama de la *Unidad de Control* (activa nodo) de la *Interfaz de Red Esclava*

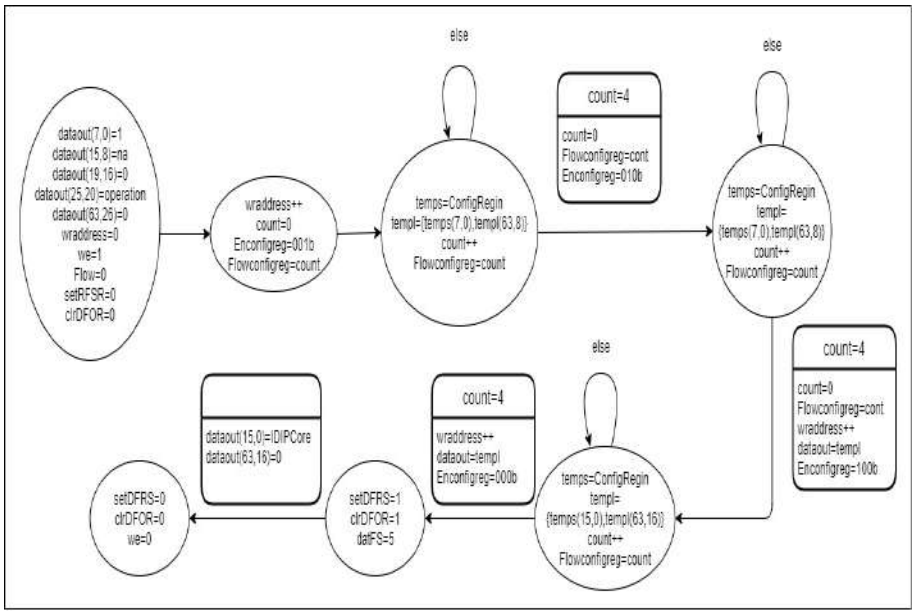


Figura A.14: Diagrama de la *Unidad de Control* (leer estado) de la *Interfaz de Red Esclava*

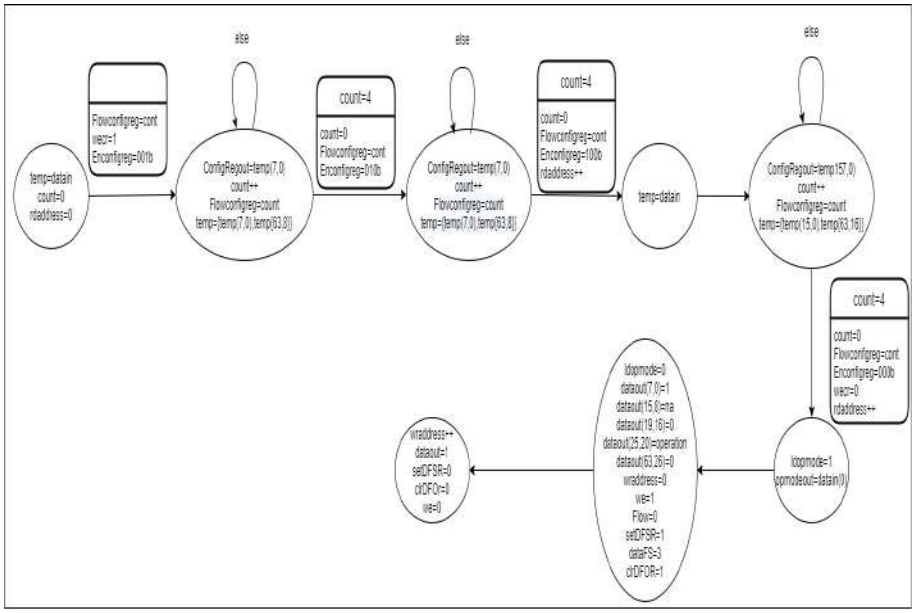


Figura A.15: Diagrama de la *Unidad de Control* (configura IR) de la *Interfaz de Red Esclava*

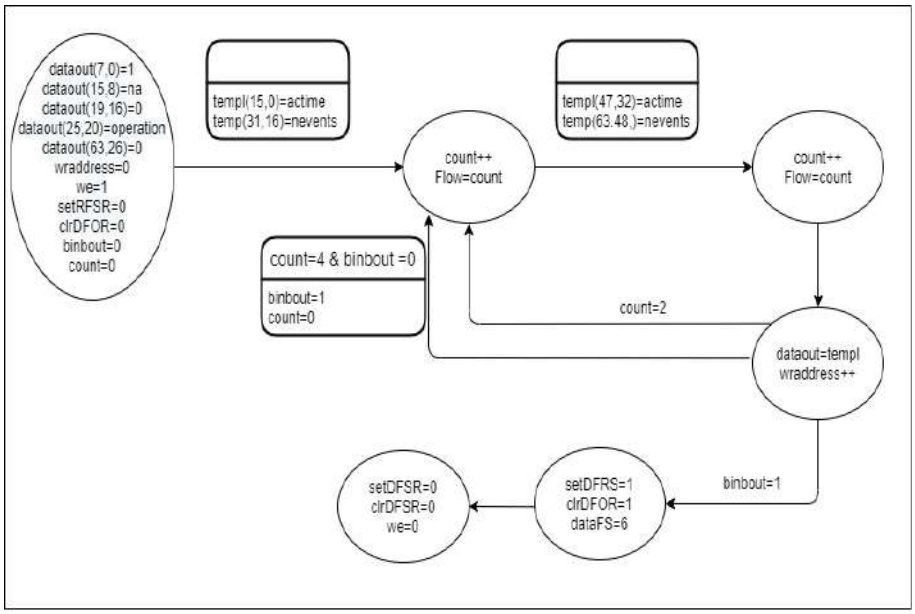


Figura A.16: Diagrama de la *Unidad de Control* (leer estadística) de la *Interfaz de Red Esclava*

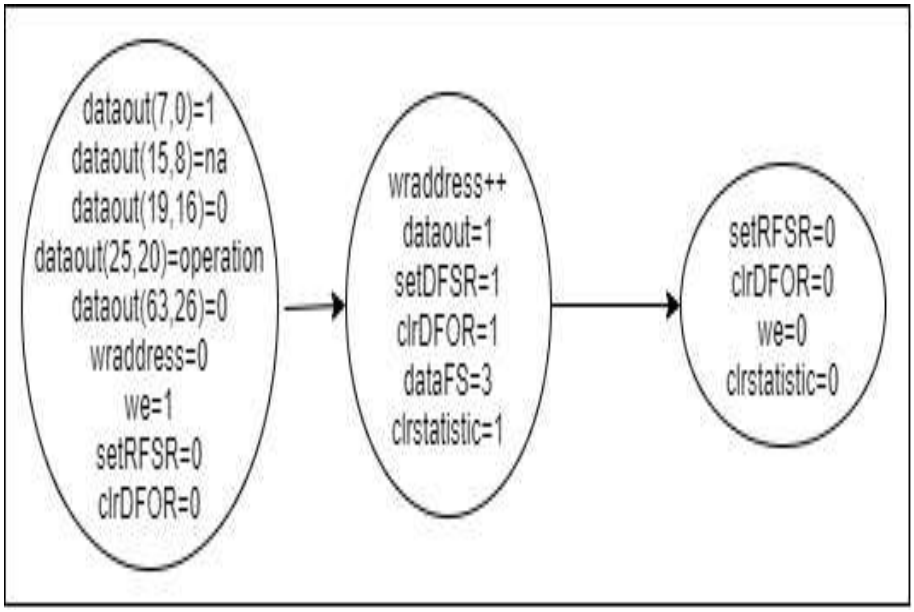


Figura A.17: Diagrama de la *Unidad de Control* (borrar estadística) de la *Interfaz de Red Esclava*

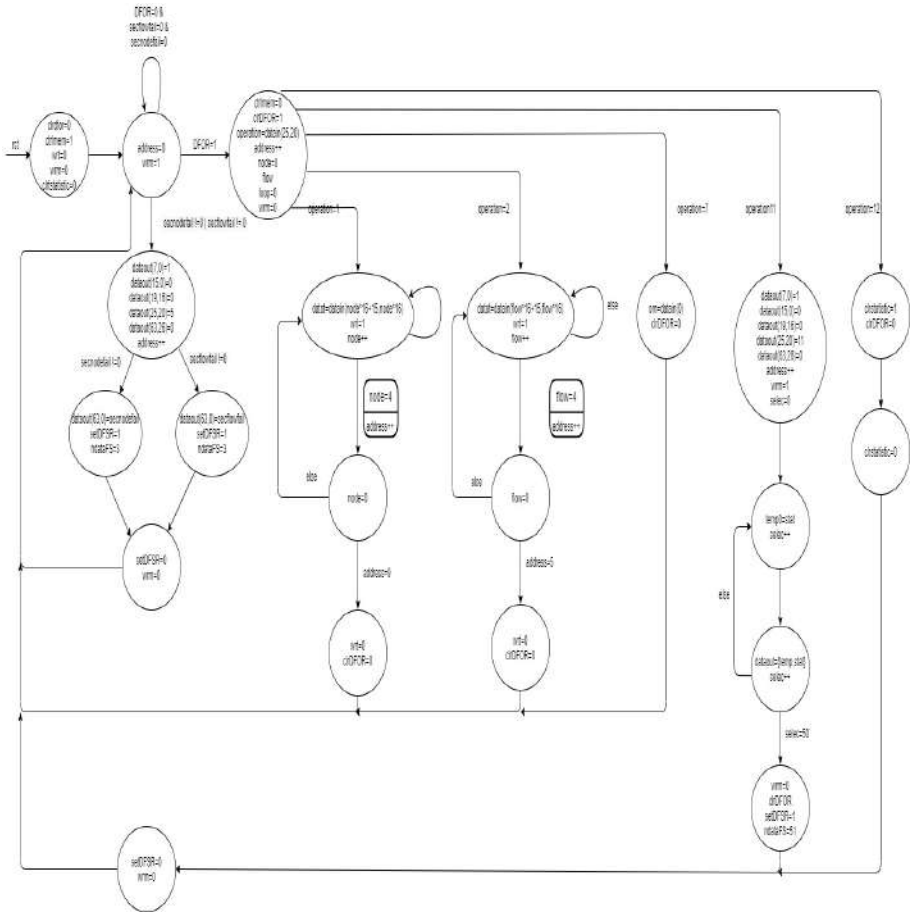


Figura A.18: Diagrama de la *Unidad de Control* del *Controlador del Bus*

Referencias

Ahmed, K. E., Rizk, M. R., and Farag, M. M. (2017). Overloaded CDMA crossbar for network-on-chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(6):1842–1855.

Akesson, B., Steffens, L., and Goossens, K. (2009). Efficient service allocation in hardware using credit-controlled static-priority arbitration. In *Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 59–68. IEEE.

Amin, M. and Abdullah, A. (2017). A Bus Arbitration Scheme with an Efficient Utilization and Distribution. *International Journal of Advanced Computer Science and Applications*, 8(3):113–118.

ARM, L. (2010). AMBA AXI Protocol Specification. *Arm*.

Ben Achballah, A., Ben Othman, S., and Ben Saoud, S. (2017). Problems and challenges of emerging technology networks-on-chip: A review. *Microprocessors and Microsystems*, 53:1–20.

Ben Slimane, M., Ben Hafaiedh, I., and Robbana, R. (2017). Formal-Based Design and Verification of SoC Arbitration Protocols: A Comparative Analysis of TDMA and Round-Robin. *IEEE Design and Test*, 34(5):54–62.

Benini, L. and Bertozzi, D. (2005). Network-on-chip architectures and design methods. *IEE Proceedings - Computers and Digital Techniques*, 152:261–272.

Benini, L. and De Micheli, G. (2002). Networks on chips: a new soc paradigm. *Computer*, 35(1):70–78.

Berestizshevsky, K., Even, G., Fais, Y., and Ostrometzky, J. (2017). SDNoC: Software defined network on a chip. *Microprocessors and Microsystems*, 50:138–153.

Bjerregaard, T. and Mahadevan, S. (2006). A survey of research and practices of network-on-chip. *ACM Computing Surveys*, 38(1):71–121.

Burgio, P., Ruggiero, M., Esposito, F., Marinoni, M., Buttazzo, G., and Benini, L. (2010). Adaptive tdma bus allocation and elastic scheduling: A unified approach for enhancing robustness in multi-core rt systems. In *Proceedings of the 2010 IEEE International Conference on Computer Design*, pages 187–194. IEEE.

Chao, H. J., Lam, C. H., and Guo, X. (1999). A fast arbitration scheme for tera-bit packet switches. In *Proceedings of the Global Telecommunications Conference, Seamless Interconnection for Universal Services*, volume 2, pages 1236–1243. IEEE.

Chen, C. H., Lee, G. W., Huang, J. D., and Jou, J. Y. (2006). A real-time and bandwidth guaranteed arbitration algorithm for SoC bus communication. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 600–605. IEEE.

Chen, W., Ray, S., Bhadra, J., Abadir, M., and Wang, L. C. (2017). Challenges and Trends in Modern SoC Design Verification. *IEEE Design and Test*, 34(5):7–22.

Cisco (2020). *[Online], Available: https://www.cisco.com/c/es$_m$x/solutions/software−defined − networking/overview.html.*

Cong, L., Wen, W., and Wang, Z. (2014). A configurable, programmable and software-defined network on chip. In *Proceedings of the IEEE Workshop on Advanced Research and Technology in Industry Applications*, pages 813–816. IEEE.

Dally, W. and Towles, B. (2003). *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Dally, W. J. and Towles, B. (2001). Route packets, not wires: on-chip interconnection networks. In *Proceedings of the 38th Design Automation Conference*, pages 684–689. ACM.

Dimitrakopoulos, G., Chrysos, N., and Galanopoulos, K. (2008). Fast arbiters for on-chip network switches. In *Proceedings of the IEEE International Conference on Computer Design*, pages 664–670. IEEE.

Dimitrakopoulos, G., Kalligeros, E., and Galanopoulos, K. (2013). Merged switch allocation and traversal in network-on-chip switches. *IEEE Transactions on Computers*, 62(10):2001–2012.

Doria, A., Salim, J. H., Haas, R., Khosravi, H., Wang, W., Dong, L., Gopal, R., and Halpern, J. (2010). Forwarding and Control Element Separation (ForCES) Protocol Specification, IETF.

Ellinidou, S., Sharma, G., Dricot, J. M., and Markowitch, O. (2018). A SDN solution for system-on-chip world. In *Proceedings of the International Conference on Software Defined Systems*, pages 14–19. IEEE.

Ellinidou, S., Sharma, G., Kontogiannis, S., Markowitch, O., Dricot, J. M., and Gogniat, G. (2019). MicroLET: A New SDNoC-Based Communication Protocol for ChipLET-Based Systems. In *Proceedings of the Euromicro Conference on Digital System Design*, pages 61–68. IEEE.

Elsevier (2020). [online], available: https://www.journals.elsevier.com/microprocessors-and-microsystems.

Fatih Ugurdag, H. and Baskirt, O. (2012). Fast parallel prefix logic circuits for n2n round-robin arbitration. *Microelectronics Journal*, 43(8):573 – 581.

Foster, H. (2018). The 2018 wilson research group functional verefication study [online], available: https://blogs.mentor.com/verificationhorizons/blog/2018/11/19/part-1-the-2018-wilson-research-group-functional-verification-study/.

Garcia-Luciano, L., Ibarra-Delgado, S., Gallegos-Ruiz, H., and Sandoval-Arechiga, R. (2017). Hardware implementation of a block cipher with AXI Stream Interface. In *Memorias del Congreso Internacional de Investigación Academia Journals Celaya 2017*, volume 9, pages 2245–2251. Academia Journals.

García Morales, L., Aedo Cobo, J., and Bagherzadeh, N. (2019). A new approach to the Population-Based Incremental Learning algorithm using virtual regions for task mapping on NoCs. *Journal of Systems Architecture*, 97:443–454.

Gorski, P., Wegner, T., and Timmermann, D. (2015). Centralized and software-based run-time traffic management inside configurable regions of interest in mesh-based networks-on-chip. In *Proceedings of the International Applied Reconfigurable Computing*, pages 179–190. Springer International Publisher.

Guerrier, P. and Greiner, A. (2000). A generic architecture for on-chip packet-switched interconnections. In *Proceedings of the Design Europe Conference and Exhibition, Automation and Test*, pages 250–256. IEEE.

Gupta, P. and McKeown, N. (1999). Designing and implementing a fast crossbar scheduler. *IEEE Micro*, 19(1):20–28.

Heisswolf, J., Zaib, A., Weichslgartner, A., Karle, M., Singh, M., Wild, T., Teich, J., Herkersdorf, A., and Becker, J. (2014). The Invasive Network on Chip - A Multi-Objective Many-Core Communication Infrastructure. In *Proceedings of the*

International Conference on Architecture of Computing Systems, pages 1–8. VDE VERLAG.

Hernández-Calviño, M., Ibarra-Delgado, S., Sandoval-Aréchiga, R., Flores-Troncoso, J., and García-Luciano, L. (2018). Image compressor ip-core based on loco algorithm for space photography application. In *Proceedings of the IEEE International Autumn Meeting on Power, Electronics and Computing*, pages 1–4. IEEE.

Hwang, S. Y., Kang, D. S., Park, H. J., and Jhang, K. S. (2010). Implementation of a self-motivated arbitration scheme for the multilayer AHB busmatrix. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(5):818–830.

Ibarra-Delgado, S., Sandoval-Arechiga, R., Brox, M., and Ortíz-López, M. (2020a). Throughput Unfairness in Fair Arbitration Interconnection-Buses for Aerospace Embedded Systems. *IEEE Latin America Transactions*, 18(9):1606–1613.

Ibarra-Delgado, S., Sandoval-Arechiga, R., Gómez-Rodríguez, J. R., Ortíz-López, M., and Brox, M. (2020b). A Bandwidth Control Arbitration for SoC Interconnections Performing Applications with Task Dependencies. *Micromachines*, 11(12):1063.

IEEE (2001). IEEE Standard for IEEE Information Technology - Portable Operating System Interface (POSIX(TM)), IEEE Std 1003.1-2001 (Revision of IEEE Std 1003.1-1996 and IEEE Std 1003.2-1992). pages 1–3678.

IEEEXplore (2020). [online], available: http://www.ieeexplore.ieee.org.

Intel (2020a). Avalon Interface Specifications, [online], Available: https://www.intel.com/content/www/us/en/programmable/documentation/nik1412467993397.html.

Intel (2020b). Intel Agilex FPGAs and SoCs Advanced Information Brief, [online], Available: https://www.intel.com/content/www/us/en/programmable/documentation/onc1551901789668.html.

Jain, K., Singh, S. K., Majumder, A., and Mondai, A. J. (2015). Problems encountered in various arbitration techniques used in NOC router: A survey. In *Proceedings of the International Conference on Electronic Design, Computer Networks and Automated Verification*, pages 62–67. IEEE.

Jun, M., Bang, K., Lee, H. J., Chang, N., and Chung, E. Y. (2007). Slack-based bus arbitration scheme for soft real-time constrained embedded systems. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 159–164. IEEE.

Karkar, A., Mak, T., Tong, K., and Yakovlev, A. (2016). A survey of emerging interconnects for on-chip efficient multicast and broadcast in many-cores. *IEEE Circuits and Systems Magazine*, 16(1):58–72.

Keating, M. and Bricaud, P. (2012). *Reuse Methodology Manual, For System-on-a-Chip-Designs*. Springer Science+Business Media.

Kobbe, S., Bauer, L., Lohmann, D., Schröder-Preikschat, W., and Henkel, J. (2011). DistRM: Distributed resource management for on-chip many-core systems. In *Proceedings of the 9th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 119–128. IEEE.

Kornaros, G. and Pnevmatikatos, D. (2014). Dynamic power and thermal management of noc-based heterogeneous mpsocs. *ACM Transactions on Reconfigurable Technology and Systems*, 7(1):1–26.

Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

Kundu, S. and Chattopadhyay, S. (2015). *Network-on-chip: The next generation of system-on-chip integration (1st ed.)*. CRC Press.

Lahiri, K., Raghunathan, A., and Lakshminarayana, G. (2006). The LOTTERYBUS on-chip communication architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(6):596–608.

Lara, E., Debon, G., Goerl, R., Villa, P., Schramm, D., Poehls, L. B., and Vargas, F. (2019). A new approach to guarantee critical task schedulability in tdma-based bus access of multicore architecture. In *Proceedings of the IEEE Latin American Test Symposium*, pages 1–6. IEEE.

Lee, A. S. and Bergmann, N. W. (2003). On-chip communication architectures for reconfigurable System-on-Chip. In *Proceedings of the 2003 IEEE International Conference on Field-Programmable Technology*, pages 332–335. IEEE.

Lee, M. M., Kim, J., Abts, D., and Lee, J. W. (2010). Approximating Age-Based Arbitration in On-Chip Networks. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, pages 575–576. ACM.

Lee, Y., Jou, J. M., and Chen, Y. (2009). A high-speed and decentralized arbiter design for noc. In *Proceedings of the IEEE/ACS International Conference on Computer Systems and Applications*, pages 350–353. IEEE.

Li, H., Zhang, M., Zheng, W., and Li, D. (2007). An adaptive arbitration algorithm for SoC bus. In *Proceedings of the Conference on Networking, Architecture, and Storage*, pages 245–246. IEEE.

Lin, B., Lee, G., Huang, J., and Jou, J. (2007). A precise bandwidth control arbitration algorithm for hard real-time soc buses. In *Proceedings of the 2007 Asia and South Pacific Design Automation Conference*, pages 165–170. IEEE.

Liu, W., Xu, J., Wu, X., Ye, Y., Wang, X., Zhang, W., Nikdast, M., and Wang, Z. (2011). A NoC Traffic Suite Based on Real Applications. In *Proceedings of the 2011 IEEE Computer Society Annual Symposium on VLSI*, pages 66–71. IEEE.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

Nguyen, H. K. and Tran, X. T. (2018). A novel priority-driven arbiter for the router in reconfigurable Network-on-Chips. In *Proceedings of the International Conference on IC Design and Technology*, pages 25–28. IEEE.

ONF (2014). Open Networking Fundation [Online], Available: https://www.opennetworking.org/.

Oveis-Gharan, M. and Khan, G. N. (2015). Index-based round-robin arbiter for NoC routers. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pages 62–67. IEEE.

Pagani, M., Rossi, E., Biondi, A., Marinoni, M., Lipari, G., and Buttazzo, G. (2019). A Bandwidth Reservation Mechanism for AXI-Based Hardware Accelerators on FPGAs. In *Proceedings of the Conference on Real-Time Systems*, pages 1–24. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Pande, P. P., Grecu, C., Jones, M., Ivanov, A., and Saleh, R. (2005). Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEEE Transactions on Computers*, 54(8):1025–1040.

Park, H. and Choi, K. (2015). Adaptively weighted round-robin arbitration for equality of service in a many-core network-on-chip. *IET Computers & Digital Techniques*, 10(1):37–44.

Peng, H. K. and Lin, Y. L. (2010). An optimal warning-zone-length assignment algorithm for real-time and multiple-QoS on-chip bus arbitration. *ACM Transactions on Embedded Computing Systems*, 9(4):1–39.

Peterson, W. D. and Herveille, R. (2010). WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores, rev. version: B4, Open Cores Organization, [online], Available: www.opencores.org.

Poletti, F., Bertozzi, D., Benini, L., and Bogliolo, A. (2003). Performance analysis of arbitration policies for soc communication architectures. *Des. Autom. Embedded Syst.*, 8(2):189–210.

Rahmati, D. and Sarbazi-Azad, H. (2017). Classified Round Robin: A Simple Prioritized Arbitration to Equip Best Effort NoCs With Effective Hard QoS. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(1):257–269.

Richardson, T. D., Nicopoulos, C., Park, D., Narayanan, V., Yuan Xie, Das, C., and Degalahal, V. (2006). A hybrid soc interconnect with dynamic tdma-based transaction-less buses and on-chip networks. In *Proceedings of the Conference on VLSI Design*, pages 1–8. IEEE.

Ruaro, M., Caimi, L. L., and Moraes, F. G. (2020). A Systemic and Secure SDN Framework for NoC-Based Many-Cores. *IEEE Access*, 8:105997–106008.

Ruaro, M., Medina, H. M., Amory, A. M., and Moraes, F. G. (2018). Software-Defined Networking Architecture for NoC-based Many-Cores. In *Proceedings of the International Symposium on Circuits and Systems*, pages 1–5. IEEE.

Saleh, R., Wilton, S., Mirabbasi, S., Hu, A., Greenstreet, M., Lemieux, G., Pande, P. P., Grecu, C., and Ivanov, A. (2006). System-on-chip: Reuse and integration. *Proceedings of the IEEE*, 94(6):1050–1068.

Salminen, E., Kulmala, A., and Timo, D. H. (2008). Survey of Network-on-chip Proposals. volume 1, page 13. White Paper, Open Core Protocol Int. Partnership.

Sandoval-Arechiga, R., Ibarra-Delgado, S., and Flores-Troncoso, J. (2017). A Software Defined Interconnection Architecture for Systems on Chip. *Difu100ci@ Revista en Ingeniería y Tecnología, UAZ*, 10(2).

Sandoval-Arechiga, R., Parra-Michel, R., Vazquez-Avila, J. L., Flores-Troncoso, J., and Ibarra-Delgado, S. (2016). Software Defined Networks-on-Chip for Multi/Many-Core Systems. In *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, pages 129–130. IEEE.

Sandoval-Arechiga, R., Vazquez-Avila, J. L., Parra-Michel, R., Flores-Troncoso, J., and Ibarra-Delgado, S. (2015). Shifting the network-on-chip paradigm towards

a software defined network architecture. In *Proceedings of the Conference on Computational Science and Computational Intelligence*, pages 869–870. IEEE.

Scionti, A., Mazumdar, S., and Portero, A. (2016). Software defined Network-on-Chip for scalable CMPs. In *Proceedings of the Conference on High Performance Computing and Simulation*, pages 112–115. IEEE.

Scionti, A., Mazumdar, S., and Portero, A. (2018). Towards a Scalable Software Defined Network-on-Chip for Next Generation Cloud. *Sensors (Switzerland)*, 18(7):2330.

Shah, H., Raabe, A., and Knoll, A. (2011). Priority division: A high-speed shared-memory bus arbitration with bounded latency. In *Proceedings of the Design, Automation Test in Europe*, pages 1497–1500. IEEE.

Shapiro, D. (1984). *Globally-asynchronous, locally-synchronous*. Ph.D. Thesis Dept. Comput. Sci., Standford Univ., Standford, CA.

Shin, E. S., Mooney, V. J., and Riley, G. F. (2002). Round-robin arbiter design and generation. In *Proceedings of the International Symposium on System Synthesis*, pages 243–248. IEEE.

Silva, R. S., Cruz, P. P., Kreutz, M. E., and Pereira, M. M. (2019). Communication Latency Evaluation on a Software-Defined Network-on-Chip. In *Proceedings of the Symposium on Computing System Engineering, SBESC*, pages 1–7. IEEE.

Singh, A. K., Shafique, M., Kumar, A., and Henkel, J. (2013). Mapping on multi/many-core systems: Survey of current and emerging trends. In *Proceedings of the ACM/EDA/IEEE Design Automation Conference*, pages 1–10. IEEE.

Slijepcevic, M., Hernandez, C., Abella, J., and Cazorla, F. J. (2017). Design and implementation of a fair credit-based bandwidth sharing scheme for buses. In *Proceedings of the 2017 Design, Automation and Test in Europe*, pages 926–929. IEEE.

Song, H. (2013). Protocol-Oblivious Forwarding: Unleash the Power of SDN through a Future-Proof Forwarding Plane. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pages 127–132. ACM.

SystemC (2020). [online], available: http://www.systemc.org.

SystemVerilog (2020). [online], available: http://www.systemverilog.org.

Wang, J., Li, Y., Peng, Q., and Tan, T. (2009). A dynamic priority arbiter for Network-an-Chip. In *Proceedings of the 2009 IEEE International Symposium on Industrial Embedded Systems*, pages 253–256. IEEE.

Wang, J., Zhu, M., Peng, C., Zhou, L., Qian, Y., and Dou, W. (2014). Software-defined photonic network-on-chip. In *Proceedings of the The Third International Conference on e-Technologies and Networks for Development*, pages 127–130. IEEE.

Xilinx (2018). *Zynq-7000 SoC Data Sheet: Overview, Product Specification.*

Xilinx (2020). *Smartconnect V1.0 LogiCORE IP Product Guide PG247.*

Xu, Z., Zhang, S., Ni, W., Yang, Y., and Bu, J. (2014). Design and implementation of a dynamic weight arbiter for networks-on-chip. In *Proceedings of the International Conference on Information Science and Technology*, number 2, pages 354–357. IEEE.

Yang, Y., Wu, R., Zhang, L., and Zhou, D. (2015). An asynchronous adaptive priority round-robin arbiter based on four-phase dual-rail protocol. *Chinese Journal of Electronics*, 24(1):1–7.

Yu, M., Wundsam, A., and Raju, M. (2014). NOSIX: a lightweight portability layer for the SDN OS. *ACM SIGCOMM Computer Communication Review*, 44:28–35.

Zheng, S. Q., Mei Yang, Blanton, J., Golla, P., and Verchere, D. (2002). A simple and fast parallel round-robin arbiter for high-speed switch control and scheduling. In *Proceedings of the 45th Midwest Symposium on Circuits and Systems*, volume 2, pages II–II.

Zheng, S. Q. and Yang, M. (2007). Algorithm-hardware codesign of fast parallel round-robin arbiters. *IEEE Transactions on Parallel and Distributed Systems*, 18(1):84–95.

Zhou, X. and Zhu, Z. (2017). A dynamic task mapping algorithm for SDNoC. *Microelectronics Journal*, 63:58 – 65.