ESCUELA POLITÉCNICA DEL EJÉRCITO

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA, REDES Y COMUNICACIÓN DE DATOS

PROYECTO DE GRADO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERÍA

DISEÑO E IMPLEMENTACIÓN DE UN SIMULADOR DE ARQUITECTURA NETWORKS-ON-CHIP (NOC)

EDWIN TUFIÑO VILLACÍS

SANGOLQUÍ – ECUADOR

2013

CERTIFICACIÓN

Certificamos que el presente proyecto de grado titulado: “DISEÑO E IMPLEMENTACIÓN DE UN SIMULADOR DE ARQUITECTURA NETWORKS-ON-CHIP (NoC)”, ha sido realizado en su totalidad por el señor EDWIN DARÍO TUFIÑO VILLACÍS con CC:1720539152, el cual ha sido guiado y revisado periódicamente.

Ing.Pablo Ramos	Ing. Darwin Aguilar
DIRECTOR	**CODIRECTOR**

AUTORÍA DE RESPONSABILIDAD

EDWIN DARIO TUFIÑO VILLACIS

DECLARO QUE:

El proyecto de grado denominado "DISEÑO E IMPLEMENTACIÓN DE UN SIMULADOR DE ARQUITECTURA NETWORKS-ON-CHIP (NoC)", ha sido desarrollado con base a una investigación exhaustiva, respetando derechos intelectuales de terceros, conforme las citas que constan al pie, de las páginas correspondientes, cuyas fuentes se incorporan en la bibliografía.

Consecuentemente este trabajo es de mi autoría.

En virtud de esta declaración, me responsabilizo del contenido, veracidad y alcance científico del proyecto de grado en mención.

Sangolquí, 03 de septiembre de 2013

Edwin Dario Tufiño Villacís
CC:1720539152

AUTORIZACIÓN

Yo, Edwin Darío Tufiño Villacís

Autorizo a la Escuela Politécnica del Ejército la publicación, en la biblioteca virtual de la Institución, del trabajo "DISEÑO E IMPLEMENTACIÓN DE UN SIMULADOR DE ARQUITECTURA NETWORKS-ON-CHIP (NoC)", cuyo contenido, ideas y criterios son de mi exclusiva responsabilidad y autoría.

Sangolquí, 03 de septiembre de 2013.

Edwin Darío Tufiño Villacís

CC:1720539152

DEDICATORIA

La culminación de este proyecto la dedico a mi Familia y a mi ángel Toto, solamente nosotros sabemos lo que hemos vivido.

AGRADECIMIENTO

Quiero agradecer a todas las personas que se vieron involucradas para que yo culminara el proyecto.

A Dios por ser el que me dio la oportunidad de vivir, a mi Familia por permitirme vivir, a Mí por decidir cómo vivir y a mi ángel Toto por enseñarme a vivir.

A mi director de tesis, Ing. Pablo Ramos, y codirector, Ing. Darwin Aguilar, por dedicar su tiempo y paciencia al desarrollo de este proyecto.

ÍNDICE DE CONTENIDO

CAPÍTULO 1..... 1

INTRODUCCIÓN..... 1

1.1 ANTECEDENTES 1

1.2 JUSTIFICACIÓN E IMPORTANCIA..... 2

1.3 OBJETIVOS 3

1.3.1 General 3

1.3.2 Específicos 3

CAPÍTULO 2..... 5

REDES DE DATOS 5

2.1 INTRODUCCIÓN 5

2.2 ELEMENTOS BÁSICOS DE UNA RED DE DATOS 6

2.2.1 Topologías de Red..... 6

2.2.2 Enrutador 8

2.2.3 Conmutador o Switch..... 10

2.2.4 Conmutación 11

2.2.5 Ancho de Banda 11

2.2.6 Latencia 12

2.2.7 Throughput 13

2.2.8 Buffer 13

2.2.9 Interfaz de Red 13

2.2.10 Enlace 14

2.2.11 Modelo OSI 14

2.2.12 Protocolo Data Unit (PDU) 17

CAPÍTULO 3 18

NETWORK-ON-CHIPS (NoC) 18

3.1 LA ARQUITECTURA DE UNA NoC 20

3.1.1 La Red NoC 20

3.2 TOPOLOGÍA 24

3.2.1 Rendimiento Teórico de las Topologías 25

3.2.2. Propiedades de 2d Mesh y Torus 26

3.3 MODELO OSI 27

3.3.1 Capa de Aplicación 28

3.3.2 Capa de Transporte 29

3.3.3 Capa de Red 29

3.3.4 Capa Física 31

3.4 CONMUTACIÓN DE PAQUETES Y CIRCUITOS 32

3.4.1 Latencia 33

3.4.2 Wormhole Routing 34

3.4.3 Deadlock 35

3.5 ALGORITMO DE RUTEO 37

CAPÍTULO 4 39

SIMULADORES DE NOCs 39

4.1 SIMULADORES 40

4.1.1 Nivel de Red 41
4.1.2 Nivel de Conexión 41
4.2 SIMULADORES DE NoC 42
4.2.1 NS-2 42
4.2.2 Noxim 43
4.2.3 SunFloor 43
4.2.4 Orión 2.0 H 43
4.2.5 BookSim 44
4.2.6 Worm_sim Simulator 44
4.2.7 VNOC 44
4.2.8 Nostrum 45
4.2.9 Nirgam 45
4.2.10 GPNoCsim 45
4.2.11 Tabla de Simuladores 46
4.3 PARÁMETROS A EVALUAR 47
4.3.1 Selección de Simulador Base 48

CAPÍTULO 5 57

SIMULADOR BASE NOXIM 57

5.1 PARÁMETROS DE ENTRADA Y SALIDA 58
5.2 INSTALACIÓN 59
5.3 MANUAL DE USUARIO 61
5.4 CÓDIGO BASE 64

CAPÍTULO 6..... 79

SIMULADOR EtNoC BASADO EN NOXIM 79

6.1 MODIFICACIONES 79

6.1.1 Topología de la Red 80

6.1.2 Algoritmo de Enrutamiento 83

6.1.3 Interfaz Gráfica 86

CAPÍTULO 7..... 89

ANÁLISIS DE RESULTADOS 89

7.1 OBTENCIÓN DE PARÁMETROS DEL ESCENARIO DE SIMULACIÓN 89

7.2 BUFFER 99

7.2.1 Algoritmo de Ruteo 100

7.3 ANÁLISIS DE RESULTADOS 102

7.3.1 Latencia 102

7.3.2 Troughtput 107

7.3.3 Energía Consumida 108

CAPÍTULO 8..... 111

CONCLUSIONES Y RECOMENDACIONES 111

8.1 Conclusiones 111

8.2 Recomendaciones 113

REFERENCIAS BIBLIOGRÁFICAS 115

ÍNDICE DE TABLAS

Tabla. 1. Comparación Enrutamiento Estático y Dinámico....10
Tabla. 2. Tecnología y Ancho de Banda....12
Tabla. 3. Ventajas(V) y desventajas(X) entre los buses de datos y las NoC....19
Tabla. 4. Modelo OSI cubierto en las NoC....28
Tabla. 5. Simuladores NoC encontrados....46
Tabla. 6. Resumen de los Parámetros entrada/salida de Simuladores NoC....48
Tabla. 7. Parámetros ingresados al simulador para evaluar el desempeño....76
Tabla. 8. Diferencia en términos de latencia....77
Tabla. 9. Parámetros de entrada ingresados al simulador....77
Tabla. 10. Promedio de los resultados....78
Tabla. 11. Número de Combinaciones Sin Repetir....90
Tabla. 12. Probabilidad que existe en cada una de las matrices....91
Tabla. 13. Número de Combinaciones repetidas y con orden....92
Tabla. 14. Número de saltos desde origen(O) destino(D) desde nodo A-B....95
Tabla. 15. Número de saltos desde origen(O) destino(D), desde nodo A-D....95
Tabla. 16. Promedio del número de saltos de las combinaciones sin repetir....96
Tabla. 17. Retraso Promedio....101
Tabla. 18. Parámetros a Evaluar....102
Tabla. 19. Latencia Máxima de las diferentes arquitecturas....109

ÍNDICE DE FIGURAS

Figura. 1. Topología tipo Bus 6
Figura. 2. Topología tipo Anillo 7
Figura. 3. Topología tipo Estrella 8
Figura. 4. Topología tipo Fat-tree 8
Figura. 5. Modelo OSI 15
Figura. 6. PDU Modelo OSI 17
Figura. 7. Arquitectura NoC en una topología Mesh 21
Figura. 8. Arquitectura de un conmutador 23
Figura. 9. Tipos de Elementos de procesamiento dentro de una NoC 24
Figura. 10. Rendimiento teórico de diferentes topologías de redes de N nodos 25
Figura. 11. Topología a) Mesh y b) Torus 27
Figura. 12. Conmutación de un paquete mediante wormhole 35
Figura. 13. Tráfico estancado por deadlock. 36
Figura. 14. Deadlock causado por tamaño de Buffer 36
Figura. 15. Algoritmo de ruteo para Topología 2D Mesh 38
Figura. 16. Lectura del Manual de Usuario 49
Figura. 17. Running Simulations 50
Figura. 18. Ejecución del Simulador 51
Figura. 19. Falla en los Simuladores 52
Figura. 20. Falla en los Simuladores 52
Figura. 21. Obtención de Resultados 53
Figura. 22. Revisión de Estructura y Código de Programación 54
Figura. 23. El código para comprender los Desarrolladores 54
Figura. 24. Los 5 Simuladores de NoC más populares 56
Figura. 25. Descripción de un tile 69
Figura. 26. Malla de tiles 70
Figura. 27. Topología TorusS 81
Figura. 28. Conexión entre dos conmutadores 82

Figura. 29. Conexiones Topología TorusS 82
Figura. 30. Red Torus 84
Figura. 31. Topología Torus con algoritmo de ruteo borde 84
Figura. 32. Topología Torus con algoritmo de ruteo completo 84
Figura. 33. Algoritmo de Ruteo TorusS.....85
Figura. 34. Interfaz Gráfica EtNoC.....86
Figura. 35. Parámetros de entrada y despliegue de Resultados 87
Figura. 36. Resultados entregados por EtNoC 88
Figura. 37. Latencia Promedio de N paquetes despachados 93
Figura. 38. Estructura de la NoC para el despacho de paquetes 94
Figura. 39. Resultados randómicos vs resultado teórico en una matriz 2x2.....96
Figura. 40. Resultados randómicos vs resultado teórico en una matriz 3x3.....97
Figura. 41. Resultados randómicos vs controlado en una matriz 3x3.....99
Figura. 42. Latencia promedio al incrementar el tamaño del buffer.....100
Figura. 43. Latencia promedio de diferentes algoritmos de ruteo.....101
Figura. 44. Latencia promedio de las diferentes arquitecturas.....103
Figura. 45. Latencia promedio de las diferentes arquitecturas Matriz 2x2.....104
Figura. 46. Latencia promedio de las diferentes arquitecturas Matriz 3x3.....105
Figura. 47. Latencia promedio de las diferentes arquitecturas Matriz 4x4.....105
Figura. 48. Latencia promedio de las diferentes arquitecturas Matriz 5x5.....106
Figura. 49. Troughtput promedio de las diferentes arquitecturas 107
Figura. 50. Energía Consumida de las diferentes arquitecturas.....108

RESUMEN

La NoC (Network-on-Chip) es el nuevo paradigma de comunicación intra chip que reemplaza la comunicación tradicional de los buses de datos debido a que, actualmente, en espacios milimétricos, se tienen millones de conexiones y tanto el diseño como la interconexión entre los elementos de procesamiento se complican. La NoC basa su arquitectura de manera similar a una red computacional (internet). Se pueden encontrar dentro de la arquitectura de una NoC los siguientes componentes: router, buffer, enlaces, algoritmos de enrutamiento, topología, entre otros; lo que ha llevado a realizar estudios y mejoras de las redes dentro de los chips, para así poder garantizar una óptima comunicación entre los elementos de procesamiento. Es importante conocer el desempeño de una arquitectura NoC previo a realizar su implementación, puesto que esto conlleva el uso de recursos como tiempo y dinero, por lo que evaluar nuevas propuestas de arquitecturas NoC en términos de latencia, troughtput y energía consumida en un ambiente de simulación es fundamental para analizar el desempeño de una nueva propuesta y determinar si es factible realizar la implementación de la misma.

CAPÍTULO 1

INTRODUCCIÓN

1.1 ANTECEDENTES

En la actualidad, los sistemas embebidos han llegado a escalas nanométricas, proporcionando la integración de varios sistemas dentro de un solo chip, conocido con el nombre de System On Chip (SoC). Estos sistemas intra-chip, por su alta complejidad computacional, contienen varios módulos internos con un grado de comunicación importante. De ahí que el aumento de las conexiones dentro de un Circuito Integrado (IC) está llegando a niveles en los cuales las soluciones tradicionales, como cables dedicados y buses de datos, ya no son viables, principalmente por el incremento en la disipación de potencia. Asimismo, de acuerdo a la International Technology Roadmap Semiconductor (ITRS), en el 2012, en espacios de 22nm y con frecuencias de reloj de 10GHZ, se obtuvieron billones de transistores y con ello billones de conexiones dentro de un chip. Por lo mencionado anteriormente, es de vital importancia encontrar una nueva alternativa de interconexión que supere los problemas de disipación de potencia y proporcione una mejora en el throughput, permitiendo reducir la latencia y pérdida de paquetes. La tendencia actual es emplear las Networks-On-Chip (NoC) para solventar la comunicación intra-chip. (Tsai, Lan, Hu, & Chen, 2012)

Por otra parte, la Escuela Politécnica del Ejército (ESPE), a través del Departamento de Eléctrica y Electrónica (DEEE) está incursionando desde hace 3 años en el diseño de sistemas embebidos basados en FPGAs. Dentro de este campo el estudio de las NoC es relevante para el diseño de sistemas más eficientes, Por ello, se propuso realizar el proyecto de investigación en Networks-On-Chip (NoC) orientado a Multi-Processor-Systems-On-Chip (MPSoC). Este proyecto pretende diseñar una infraestructura MPSoC con co-diseño hardware/software e implementarla en FPGAs para realizar investigaciones sobre NoC, evaluar arquitecturas existentes y analizar el desempeño de propuestas propias.

Cabe señalar que, previo a la implementación de una nueva propuesta de arquitectura NoC, es necesario realizar su diseño, simulación y emulación. De ahí que, como parte del proyecto, está considerado el diseño e implementación de un simulador NoC que permita analizar los parámetros característicos de una arquitectura NoC establecida, tales como la latencia, troughtput y energía consumida.

1.2 JUSTIFICACIÓN E IMPORTANCIA

A nivel mundial, los grupos de investigación buscan nuevas alternativas para solventar la comunicación intra-chip. Dentro de este ámbito, se han propuesto varias soluciones que aún no han alcanzado la respuesta esperada y por ello se continúan realizando estudios en esta área de conocimiento. Consecuentemente, el nuevo paradigma de comunicación intra-chip NoC abre un camino de oportunidades en el campo investigativo. (Achballah & Saoud, 2011)

El incursionar en la concepción de NoCs mediante la ejecución de este proyecto, permitirá comprender el funcionamiento de una NoC y caracterizarla de una manera

adecuada a fin de proponer nuevas soluciones que aporten en el desarrollo de futuras aplicaciones basadas en esta tecnología, disminuyendo el gasto de recursos, tiempo y dinero.

A través de la realización de este simulador, se logrará alcanzar las capacidades necesarias para plasmar nuevas propuestas dentro de las NoC. De esta manera, se aportará en la generación de nuevos conocimientos en esta área fundamental dentro del desarrollo de nuevas tecnologías en el Departamento de Eléctrica y Electrónica de la Escuela Politécnica del Ejército del Ecuador.

El aporte de la presente tesis de grado para el proyecto de investigación "Networks-On-Chip (NoC) orientado a Multi-Processor-Systems-On-Chip (MPSoC)" desarrollado por el Departamento de Eléctrica y Electrónica, ayudará a analizar el comportamiento de una propuesta de arquitectura NoC previo a su implementación sobre una FPGAs, para garantizar que la arquitectura a implementarse presente un desempeño adecuado en lo referente a throughput, mínima latencia y mínima energía consumida.

1.3 OBJETIVOS

1.3.1 General

- Implementar un simulador NoC que permita evaluar el desempeño de arquitecturas NoC existentes y de nuevas propuestas.

1.3.2 Específicos

- Caracterizar a las arquitecturas NoC.

- Evaluar distintos simuladores existentes y seleccionar un simulador de NoC base.
- Analizar y probar el desempeño del simulador de NoC base.
- Mejorar el simulador de NoC base.
- Implementar una nueva propuesta de arquitectura NoC.
- Evaluar el desempeño de las arquitecturas NoC implementadas en el simulador.
- Elaborar la documentación necesaria.

CAPÍTULO 2

REDES DE DATOS

2.1 INTRODUCCIÓN

En la actualidad, las redes de datos son de suma importancia en las comunicaciones por la facilidad con la que permiten a sus usuarios interconectarse. Es decir, cada usuario puede transmitir y recibir información de acuerdo al área o entorno en que éste se encuentre en la red.

Para el desarrollo de este proyecto es indispensable conocer, de forma general, los conceptos básicos de las redes de datos, puesto que las NoC basan su estructura en éstas.

No es necesario profundizar en detalle con la explicación de cada uno de los elementos de la red, sino simplemente saber que estos son parte de la red de datos para luego profundizar los conceptos en las NoC.

2.2 ELEMENTOS BÁSICOS DE UNA RED DE DATOS

2.2.1 Topologías de Red

La topología de Red es la disposición física en la cual se encuentran interconectados diferentes elementos. (Bicsi, 2002)

. Tipos de Topología

Existen diferentes tipos de topologías de red como: anillo, bus, fat-tree, estrella, etc. A continuación detallaremos su función y su correspondiente topología.

. Topología tipo Bus

En la topología tipo Bus todos los elementos están conectados a un único medio por el cual pasa la información que es transmitida. Una de las ventajas de esta topología es que si falla un elemento, no altera el funcionamiento de toda la Red y la inclusión de un elemento es fácil. La figura 1 nos muestra la topología de tipo Bus. (Bicsi, 2002)

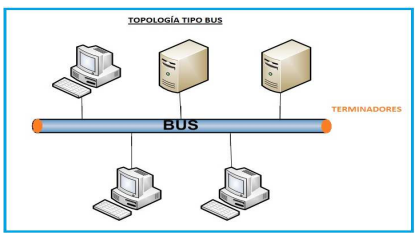


Figura. 1. Topología tipo Bus

. Topología tipo Anillo

En la topología tipo Anillo, cada elemento está conectado físicamente a un elemento anterior y a otro posterior. Cada elemento recibe la información que, si no pertenece a ese elemento específico, es enviada al siguiente sin alterar el paquete. Proporciona velocidades de transmisión altas con tasas de errores bajos; su inconveniente es que si falla un elemento afecta al funcionamiento de toda la Red, como se muestra en la Figura 2. (Bicsi, 2002)

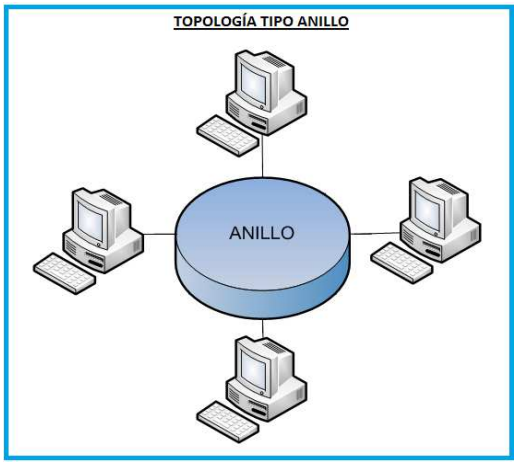


Figura. 2. Topología tipo Anillo

. Topología tipo Estrella

Todo el tráfico de la Red pasa a través de un conmutador que redirige el tráfico hacia los dispositivos de destino; la estructura lógica puede mantenerse al igual que la topología tipo Bus, esto permite que los ordenadores puedan acceder al mismo cable por medio del conmutador. La Figura 3 muestra la topología tipo Estrella. (Bicsi, 2002)

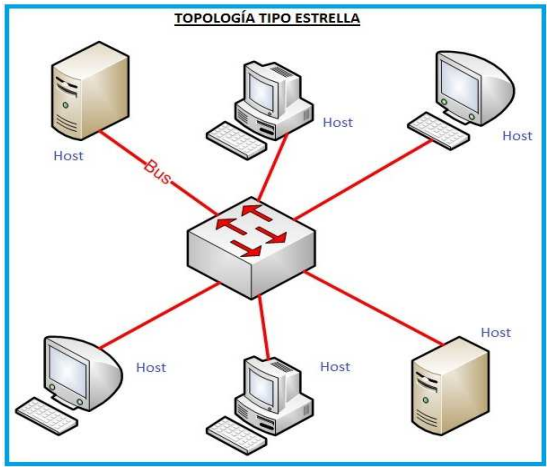


Figura. 3. Topología tipo Estrella

. Topología tipo Fat-tree

Esta topología se conoce también como topología jerárquica (Figura 4.) ya que divide a toda la red en 3 capas (Acceso, Núcleo y Distribución) que cumplen con una función específica dentro del funcionamiento de la Red en general. Una red Fat-tree nos brinda las ventajas de rendimiento, escalabilidad, seguridad, fácil administración y redundancia. (Bicsi, 2002)

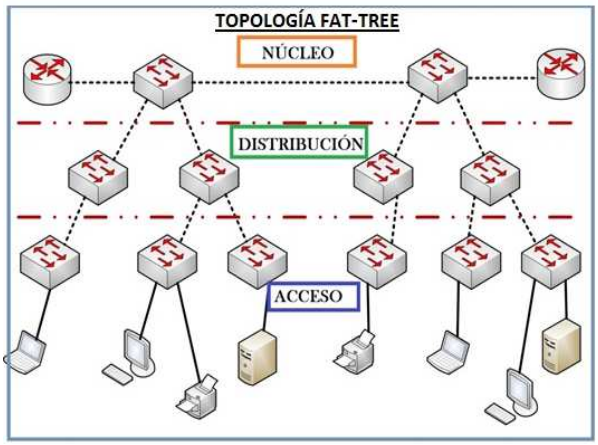


Figura. 4. Topología tipo Fat-tree

2.2.2 Enrutador

El Enrutador, o Router, es un dispositivo de red que trabaja a nivel de capa 3 del modelo OSI. El Router puede interconectar diferentes redes y envía los paquetes provenientes de una red hacia otra. Es el que se encarga de determinar hacia dónde debe ser despachada la información. (Craig Partridge, 2006)

. Algoritmo de Enrutamiento

El algoritmo de Enrutamiento se encarga de decidir una salida (Interfaz) por la que se transmitirá el paquete originado por el host. Tiene la capacidad de manejar los diferentes cambios que hay en una topología y tráfico, sin afectar las actividades en los host. Existen dos tipos de enrutamiento: Estático y Dinámico.

El Enrutamiento Estático es aquel en el que el administrador de Red debe configurar manualmente la ruta por la cual un paquete debe llegar a su destino. Por su parte, en el Enrutamiento Dinámico se generan automáticamente las rutas por las cuales viajan los paquetes. (Craig Partridge, 2006)

. Tabla Comparativa Enrutamiento Estático vs Dinámico

En la Tabla 1 se detalla las diferencias entre Enrutamiento Estático y Dinámico.

Tabla. 1. Comparación Enrutamiento Estático y Dinámico

ESTÁTICO	DINÁMICO
Genera carga administrativa y consume tiempo del administrador de red cuando son redes grandes. El administrador debe configurar manualmente las rutas en cada router de la red.	No genera mucha carga administrativa porque los routers aprenden dinámicamente a enrutarse de los demás routers de la red.
El router no comparte su tabla de enrutamiento con los routers vecinos (directamente conectados).	El router comparte su tabla de enrutamiento con los routers vecinos.
Los routers no tienen capacidad de reacción ante un fallo en la red.	Los routers tienen capacidad de reacción ante un fallo en la red.

2.2.3 Conmutador o Switch

El Switch es un dispositivo muy utilizado en las redes de datos, tiene la capacidad de aprender y almacenar las direcciones físicas (direcciones MAC) de los dispositivos, alcanzables a través de cada puerto del Switch. Si un equipo se conecta directamente al puerto del Switch; hace que éste almacene su dirección física. Esto permite que, a diferencia de los hubs (concentrador), la información que se dirige a un dispositivo dentro de la red, vaya desde el puerto de origen al de destino. Esto origina un aumento en el rendimiento de la red.

Los Switch pueden trabajar a nivel de capa 2 (Enlace de Datos) o capa 3 (Red) del modelo OSI. El Switch de capa 2 tiene como función principal dividir una red local en diferentes dominios de Colisión. El Switch capa 3, además de cumplir con las funciones de la capa 2, incorpora funciones de enrutamiento como es la determinación del camino basado en las direcciones IP de los dispositivos. (Networks, 1995)

2.2.4 Conmutación

. Conmutación por Paquetes

La conmutación de paquetes es un método de envío de datos en una red. Un paquete de información es fragmentado y cada fragmento busca el camino hasta llegar a su destino.

. Conmutación por Circuito

A diferencia de la conmutación de paquetes, en este tipo de conmutación se establece un canal dedicado para la transmisión de la información entre el origen y el destino. La comunicación por conmutación de circuitos comprende tres fases: el establecimiento del circuito, la transferencia de datos y la desconexión del circuito. (Baran, 1964)

2.2.5 Ancho de Banda

El Ancho de Banda, también conocido como Bandwidth, es la capacidad de información de datos que se puede enviar a través de una conexión de red en un período de tiempo dado. El ancho de banda se indica generalmente en bits por segundo (bps), kilobits por segundo (kbps) o megabits por segundo (Mbps). (Tanendaum, 2003)

La Tabla 2 muestra las tecnologías más conocidas con su respectivo ancho de Banda.

Tabla. 2. Tecnología y Ancho de Banda

TECNOLOGÍA	ANCHO DE BANDA
Modem / Dialup	56 Kbit/s
ADSL Lite	1.5 Mbit/s
T1/DS1	1.544 Mbit/s
E1/E-carrier	2.048 Mbit/s
Wireless 802.11g	54 Mbit/s
Fast Ethernet	100 Mbit/s
Gigabit Ethernet	1 Gbit/s

2.2.6 Latencia

La latencia está relacionada con el tiempo que le toma a un paquete viajar desde el nodo origen hasta el nodo destino.

Depende de tres factores:

1. Tiempo de propagación del bit por el enlace que depende del tiempo de propagación de la corriente (Coaxial, UTP) o luz (Fibra Óptica) por el enlace, además de cuánta distancia recorre el bit.

2. Máxima cantidad de datos que pueden ser transmitidos por la red sin que el paquete se segmente. Al tiempo de propagación del bit se le llama tiempo de transmisión.

3. Tiempos de espera para difundirse un paquete a través de un conmutador. A este tiempo se le denomina tiempo de cola. (Blake, 2003)

2.2.7 Throughput

El Throughput/ Rendimiento, en redes de comunicaciones, es la tasa promedio de éxito que tiene un paquete al ser enviado sobre un canal/enlace. Estos datos pueden ser entregados a través de un enlace físico o lógico.

El Rendimiento generalmente es medido en bit sobre segundo (bit/s), a veces en paquetes por segundo o en paquetes por un período de tiempo. (Rappaport, 2002)

2.2.8 Buffer

Un buffer es un espacio dentro de una memoria física que sirve para almacenar temporalmente datos mientras se mueve de un lugar a otro.

El Switch almacena los datos de entrada y salida en el buffer, esto permite que los datos no se pierdan cuando existe tráfico elevado dentro de la red y puede influenciar en la latencia de los paquetes. Cuando el buffer de un puerto se llena, el dato se pierde y se lo retransmite. (Rappaport, 2002)

2.2.9 Interfaz de Red

La Interfaz de Red es un periférico que sirve para conectar una computadora a un dispositivo de Red. Tiene la capacidad de enviar y recibir los datos a través de un medio físico.

Dentro de la Interfaz de Red existe la circuitería necesaria para comunicarse, utilizando la capa física o de enlace estándar tal como Ethernet, Wi-Fi o fibra óptica. En el micro código de la Interfaz de Red se encuentra el stack de protocolos que permite la comunicación a gran escala, como es el Internet con el protocolo IP. (Posey, 2006)

2.2.10 Enlace

Es el canal o medio por el cual los datos viajan a través de la red. Este puede ser un medio guiado (cables de Red UTP, fibra óptica, coaxial) o un no guiado (conexión inalámbricas).

2.2.11 Modelo OSI

El Modelo Open System Interconnection (OSI) divide en 7 capas (Figura 5.) al proceso que transmite los datos; cada capa cumple con una función específica.

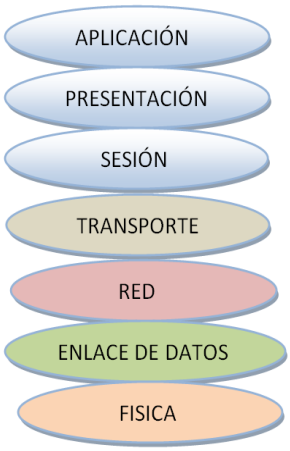


Figura. 5. Modelo OSI (Dally & Towles, 2004)

. Capa Aplicación

En esta capa se encuentran las aplicaciones/software de los usuarios finales y controla las funciones de éstas.

. Capa Presentación

Transforma el formato de los datos originados por un host origen/destino, a un formato común, para proporcionar una interfaz estándar para la capa de aplicación.

. Capa Sesión

Se encarga de establecer la conexión o sesión entre el host de origen y el host de destino.

. Capa Transporte

Permite la comunicación confiable de extremo a extremo. Los protocolos que se encuentran en la capa de transporte son TCP y UDP.

. Capa Red

En ella encontramos el direccionamiento lógico de los host. El protocolo que gobierna esta capa es el IP en su versión 4 (IPv4) y 6 (IPv6). Los dispositivos de capa 3 se encargan del enrutamiento de los paquetes.

. Capa Enlace de Datos

En esta capa están las direcciones físicas (MAC) de los dispositivos o host que se encuentran dentro de la Red.

. Capa Física

Hace referencia a las características eléctricas y físicas de los dispositivos. Los datos son enviados en 1s y 0s (bits).

2.2.12 Protocolo Data Unit (PDU)

La forma que adopta una sección de datos en cualquiera de las capas se denomina PDU (Unidad de Datos de Protocolo). Durante la encapsulación, cada capa encapsula las PDU que recibe de la capa superior, de acuerdo al protocolo que utiliza. (Kozierok, 2011).

En cada proceso, el PDU toma un nombre distinto como muestra la figura 6

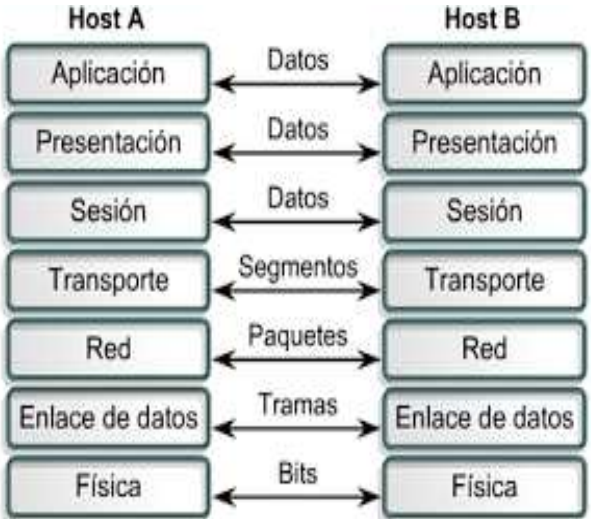


Figura. 6. PDU Modelo OSI

(Kozierok, 2011)

CAPÍTULO 3

NETWORK-ON-CHIPS (NoC)

En la actualidad, las redes de datos intrachip constituyen una parte fundamental del desarrollo tecnológico para contrarrestar las limitantes de comunicación que presentan los chips que utilizan las arquitecturas basadas en buses de datos. Por ejemplo, el diseño toma mucho tiempo en ser desarrollado y resultaría imposible la escalabilidad. Una solución basada en las redes de datos de los sistemas computacionales ayudará a solventar los problemas que se tienen en la actualidad; se obtendrían algunos beneficios tales como la reutilización de los componentes, arquitecturas, aplicaciones y se reduciría notablemente el tiempo de implementación y desarrollo.

Las NoC (redes intrachip) son un nuevo modelo de arquitectura, un conjunto de elementos de conmutación e interfaces de red interconectados entre sí para garantizar la comunicación a los distintos componentes dentro de un chip, proporcionando mecanismos de interconexión física y lógica para la transferencia de información entre los elementos de procesamiento con la finalidad de permitir la escalabilidad de los recursos (elementos de procesamiento) y reducir los tiempos de implementación. (Kumar S. , Jantsch, Soininen, & Forsell, 2002).

A continuación, en la Tabla 3, se muestran las ventajas y desventajas que presentan los buses de datos y las NoCs.

Tabla. 3. Ventajas(V) y desventajas(X) entre los buses de datos y las NoC

Buses		NoC	
Cada elemento agregado causa una capacitancia parásita que afecta al rendimiento eléctrico.	X	La conexión de los elementos es punto a punto para todos los tamaños de red, por lo tanto el rendimiento local no se degrada	V
Los retardos causados en el bus de datos causan bloqueos	X	Las decisiones de ruteo son distribuidas	V
El ancho de banda es limitado y compartido con todas las unidades	X	El ancho de banda aumenta de acuerdo al tamaño de la red	V
Las pruebas del bus son largas y problemáticas	X	El puesto localmente dedicado BIST (Built In Self Test) realiza las pruebas rápidamente y completas	V
La latencia del bus es mínima	V	La congestión interna de la red puede causar latencias prolongadas	X
Los conceptos son simples y fáciles de entender	V	Los diseñadores necesitan actualizarse a los nuevos conceptos (Keidar & Cidon, 2010)	X

3.1 LA ARQUITECTURA DE UNA NoC

La arquitectura de una NoC es la encargada de permitir la comunicación entre los elementos de procesamiento, por lo tanto, existen dos grandes objetivos que deben cumplirse:

- Posibilitar que se cree el hardware de los elementos de procesamiento de forma independiente y luego, al ser añadidos a la red, que estos formen parte de la NoC.
- Dar flexibilidad a la escalabilidad y configuración de la red para diferentes tipos de necesidades, a medida que la complejidad aumente.

A continuación se dará a conocer cómo está compuesta una arquitectura NoC, la cual es muy similar a una red de datos computacional:

- Interfaz de Red
- Conmutador
- Elemento de Procesamiento
- Topología
- Modelo OSI
- Enlace

3.1.1 La Red NoC

Para explicar de forma sencilla los elementos que conforman una red NoC se tomará como ejemplo una topología 2D Mesh (Figura 7), puesto que los elementos de procesamiento y los conmutadores son independientes del tamaño de la red y los algoritmos de enrutamiento son de fácil comprensión. (Tsai, Lan, Hu, & Chen, 2012)

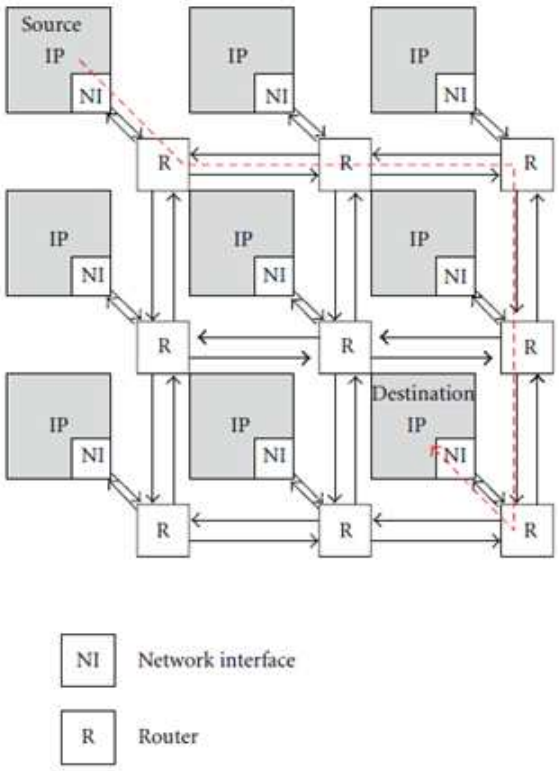


Figura. 7. Arquitectura NoC en una topología Mesh

(Tsai, Lan, Hu, & Chen, 2012)

Una NoC está compuesta básicamente de los elementos de procesamiento, interfaz de red, conmutadores y las interconexiones entre los elementos (enlaces), para así realizar las comunicaciones respectivas; todos estos elementos forman una malla.

Cada conmutador está conectado a 2 ó más conmutadores a través de enlaces tanto de entrada como de salida. Un enlace consiste en una conexión unidireccional punto a punto entre conmutadores o entre un conmutador y un elemento de procesamiento.

. Interfaz de Red (NI)

El módulo de la interfaz de red (NI) transforma los paquetes de datos generados por los elementos de procesamiento en fixed-length flow-control digits (flits). Los flits son asociados para formar un paquete de datos que consta de una cabecera (head flit), una cola (tail flit) y varios flits en medio, conocidos como el cuerpo (body flits). Este arreglo de flits es direccionado desde el origen hasta su destino saltando entre diferentes conmutadores. (Wiklund, 2005)

. Conmutador

El conmutador es el que se encarga de direccionar los flits a su destino correspondiente; para esto, cada conmutador posee cinco puertos de entrada y cinco de salida correspondientes a las direcciones norte, sur, este, oeste y un puerto para la conexión hacia la interfaz de red. Cada puerto es conectado al puerto del conmutador vecino mediante un cable físico (enlace). La funcionalidad del conmutador es direccionar el flit que ingresa por un puerto de entrada al puerto de salida apropiado hasta que llegue a su destino final. Para realizar esta función, el conmutador posee un buffer por cada puerto de entrada, un multiplexor de 5x5 para direccionar el tráfico al puerto de salida deseado y, finalmente, el control lógico para garantizar el correcto despacho del flit. (Figura 8). El flit de cabecera específica a dónde debe ser dirigido el paquete.

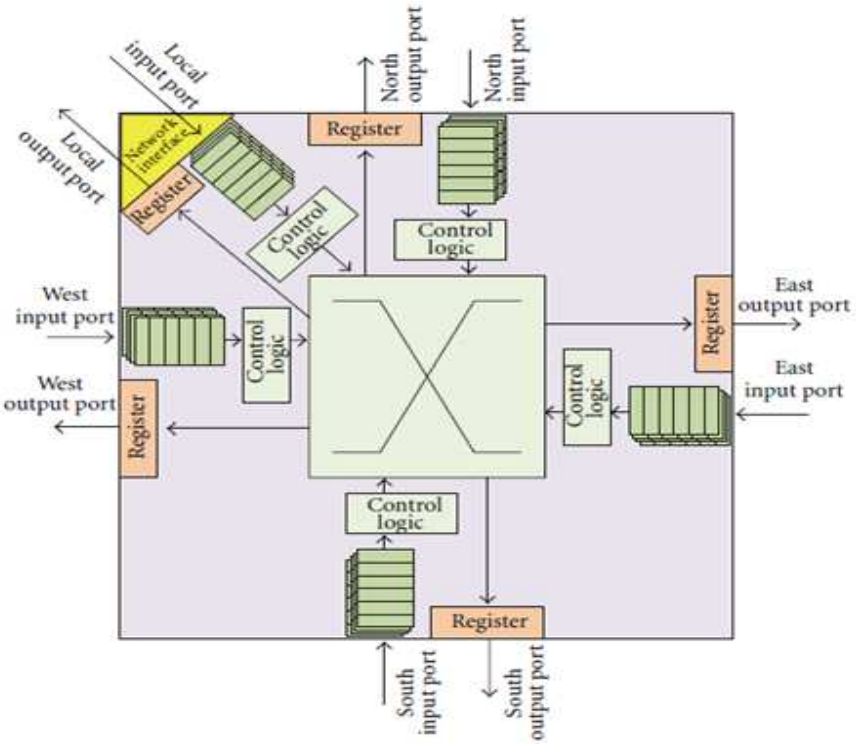


Figura. 8. Arquitectura de un conmutador

(Tsai, Lan, Hu, & Chen, 2012)

Luego de que se examina la cabecera, el control lógico del conmutador mediante el algoritmo de enrutamiento determina qué dirección de salida debe tomar el resto del paquete (cuerpo, cola). (Tsai, Lan, Hu, & Chen, 2012)

. Elementos de procesamiento

Las NoC permiten la conexión de elementos de procesamiento arbitrarios. Un ejemplo típico es la conexión de procesadores embebidos, DSP (procesadores digitales de señales), almacenamiento, memorias locales y con bloques de propiedad intelectual (IP) reconfigurables.

Debido a que los elementos de procesamiento utilizan la misma señal de reloj para sincronizarse, un recurso puede ser la combinación de los diferentes tipos existentes. La comunicación interna del recurso es sincrónica en la (Figura 9) NI = Interfaz de Red, P=Procesador, D= DSP, c=Caché, M= Memoria y RE= Bloque de propiedad intelectual.

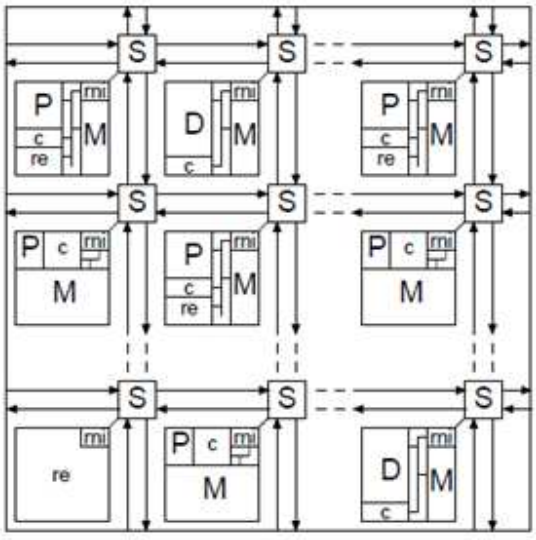


Figura. 9. Tipos de Elementos de procesamiento dentro de una NoC (Kumar S. , Jantsch, Soininen, & Forsell, 2002)

3.2 TOPOLOGÍA

La topología de una red es una configuración geométrica a nivel lógico, la cual permite la conexión de los diferentes elementos de la red. Existen muchas topologías diferentes, desde las más sencillas como una conexión punto a punto, hasta las más complejas como las jerárquicas.

3.2.1 Rendimiento Teórico de las Topologías

Un pequeño resumen del rendimiento teórico de las topologías más comunes puede ser visto en la Figura. 10.

La topología en anillo es muy sencilla de implementar pero tiene las mismas limitaciones de la arquitectura de buses de datos.

En cambio, en las topologías más poderosas como Fat Tree, su diseño es mucho más complejo a la hora de la interconexión de los enlaces.

	Ring	2D Mesh	2D torus	Binary tree	Fat tree
No. of nodes	N	$\sqrt{N}$	$\sqrt{N}$	$2N-1$	$2^{N/4-1}$
Bisection BW	2	$\sqrt{N}$	$2\sqrt{N}$	1	N
Links	Bidir	Bidir	Bidir	Bidir	Bidir
Complexity					
Wiring	Low	Low	Low	Low	High
Wire length	Short	Short	Medium	Medium	Long
Routing	Low	Medium	Medium	Low	High

Figura. 10. Rendimiento teórico de diferentes topologías de redes de N nodos (Wiklund, 2005)

En la Figura 10. No. of nodes hace referencia al número de nodos que son necesarios para realizar la comunicación total de la red. Bisection BW indica el peor ancho de banda que la topología puede presentar. Links muestra el sentido en el que se genera la comunicación.

En cuanto a la complejidad de cada una de las topologías se analiza el número de enlaces (Wiring), la longitud de cada enlace (Wire length) y el ruteo (Routing) para determinar qué topologías son las más adecuadas de acuerdo a su aplicación.

. Mesh

La característica de una red mesh (Figura 11) es la interconexión de sus nodos, creando una matriz en forma de malla con una dimensión de NxN. Los nodos exteriores solamente tienen dos nodos conectados a ellos.

. Torus

La característica de una red torus (Figura 11) es similar a una red mesh, con la diferencia de que sus nodos extremos están conectados a los nodos opuestos.

3.2.2. Propiedades de 2d Mesh y Torus

Tanto la 2d Mesh como la Torus (Figura 11) son las topologías más utilizadas en las NoC. Las principales ventajas de estas son: el buen desempeño que presentan, la facilidad del enrutamiento y su sencilla implementación en el chip. La Torus presenta un mejor desempeño cuando trabaja con tráfico randómico; esto se debe a la uniformidad de la conexión que esta topología presenta. La distancia lógica entre routers se reduce, siempre y cuando la distancia física aumente. (Wiklund, 2005).

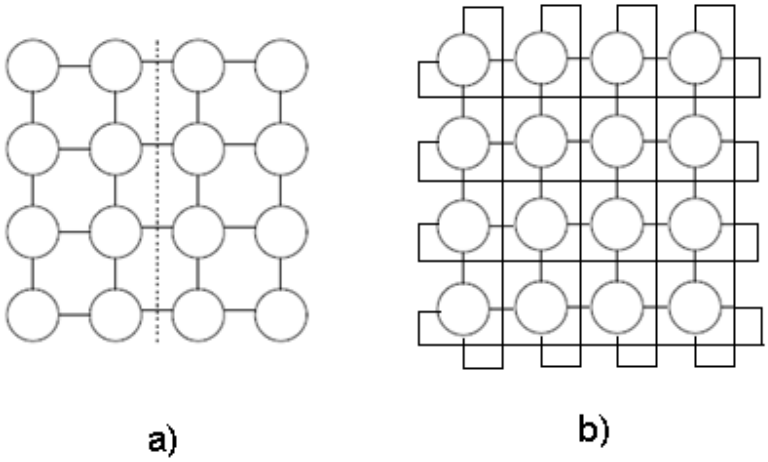


Figura. 11. Topología a) Mesh y b) Torus

3.3 MODELO OSI

Las 7 capas del modelo OSI (Open System Interconnect) fueron desarrolladas para un propósito general de las redes. Cabe recalcar que las 7 capas del modelo OSI no son estándares para la creación de una red, pero sí nos dan las pautas necesarias para lograr comprender el comportamiento de la misma; un ejemplo serían las NoC que, a diferencia de una red de propósito general (Internet) donde no se conocen los números de nodos existentes y está en constante crecimiento, se conocen todos los parámetros de la arquitectura como el número de nodos existentes, la topología y sus algoritmos de enrutamiento (Wiklund, 2005).

Con este conocimiento adicional de la infraestructura de la red en el chip, se puede omitir un grupo de capas del modelo OSI (Tabla 4), al reducir el número de capas del modelo OSI. De igual manera se reduce la latencia en la red.

Las capas del modelo OSI que se utilizan en una NoC son las siguientes:

Tabla. 4. Modelo OSI cubierto en las NoC

#	Capa	NoC
7	Aplicación	Unido
6	Presentación	
5	Sesión	
4	Transporte	Transporte
3	Red	Red
2	Datos	Unido
1	Física	

3.3.1 Capa de Aplicación

Esta capa es la que se encarga de la comunicación entre los elementos de procesamiento. En ella se establece la conexión entre los elementos de procesamiento que van a comunicarse entre sí.

Como los elementos de procesamiento pueden ser de diferentes tipos, los mensajes que estos envían también son distintos. Esta capa es la encargada de realizar la conversión de los diferentes tipos de formatos que utilizan los elementos de procesamiento para que así se puedan comunicar entre ellos (Thid, A Network on chip Simulator, 2002).

3.3.2 Capa de Transporte

Para prevenir el encolamiento del buffer e impedir la congestión del tráfico, algunos esquemas de manejo deben ser aplicados para poder guiar el transporte de los paquetes en la NoC. La capa de transporte se encarga de los problemas del control de flujo y de la congestión de los paquetes. Los parámetros claves de una NoC, como la latencia y el troughtput, son críticamente afectados por la congestión causada en la contención de los elementos de procesamiento. La contención de los elementos de procesamiento hace referencia a la disputa que se tiene entre los elementos de procesamiento para el despacho de los datos. Por lo tanto, la resolución eficaz y rápida de las disputas entre los elementos de procesamiento es la clave para evitar congestión en la red. Los algoritmos de enrutamiento son utilizados para mejorar la eficiencia de la utilización de los elementos de procesamiento de red disponibles con el fin de llegar a un mejor rendimiento en cuanto a la comunicación. (Tsai, Lan, Hu, & Chen, 2012)

3.3.3 Capa de Red

La topología de la red es una característica importante en esta capa. La capa de Red determina el cómo están conectados los elementos de procesamiento; es decir, la conexión entre los enlaces y nodos de la red interconectada. Las topologías irregulares pueden ser creadas al hacer una mezcla entre las diferentes arquitecturas de comunicación en forma jerárquica, híbrida o asimétrica, lo que nos ofrece redundancia en la conectividad y personalización de la red, a costo de la complejidad y el área utilizada. La optimización de la topología es importante, ya que esta afecta directamente a la conectividad entre los conmutadores y por ende la distancia entre los elementos de procesamiento aumenta.

Las NoC son homogéneas puesto que los elementos como los conmutadores, enlaces y las interfaces de red son los mismos. El diseño de las NoC debe ser lo suficientemente flexible para cubrir con la interconexión de los elementos de procesamiento sin que se aumente la complejidad del diseño de la misma. La mayoría de diseños de NoC utilizan una topología mesh o tours debido a los beneficios de rendimiento que presentan y al alto nivel de escalabilidad para un sistema de dos dimensiones (Kumar, Jantsch, & Soininen, 2000).

La capa de red también tiene que realizar el direccionamiento de los datos entre los conmutadores y elementos de procesamiento. Primero, el algoritmo encargado de empaquetar y desempaquetar trabaja con la descomposición del mensaje en paquetes en el nodo fuente y la composición en los nodos de destino. Segundo, la transmisión del paquete es ejecutada, tomando en cuenta el algoritmo de enrutamiento utilizado en la topología (Dally & Towles, 2004).

Los algoritmos de enrutamiento determinan qué camino debe tomar el paquete desde su nodo de origen hasta su destino. Las principales funciones de los conmutadores son: determinar qué dirección debe tomar el paquete y resolver los problemas de direccionamiento cuando el mismo camino es solicitado por diferentes nodos.

Los conmutadores están compuestos de un switch interno y de un control lógico para determinar qué camino del switch tomar (Guerrier & Greiner, 2000).

3.3.4 Capa Física

La característica principal de la capa física está centrada en los conductores y receptores de las señales de transmisión así como al desarrollo de nuevas tecnologías para la transmisión de las mismas. En las NoC, la transmisión de las señales es importante puesto a que, al trabajar con voltajes pequeños, éstas disminuyen el ruido que causa la lectura errónea de los datos en los enlaces.

El ruido eléctrico, la interferencia electromagnética, y la radiación inducida pueden producir errores en la sincronización y en la transmisión de los datos. La principal característica de esta capa es manejar esquemas de control de errores y la utilización de los enlaces físicos para lograr la fiabilidad de la comunicación.

En primer lugar, se debe desarrollar un modelo de falla. A continuación, un esquema de control de errores debe ser diseñado con características de baja potencia, gran ancho de banda y baja latencia.

En el diseño de las NoC, la transmisión por paquetes es una forma eficiente de combatir los errores en la transmisión de los datos (flits), ya que este puede ser recuperado con la retransmisión del mismo o aplicando algoritmos de control de errores. (Tsai, Lan, Hu, & Chen, 2012).

3.4 CONMUTACIÓN DE PAQUETES Y CIRCUITOS

A continuación, dos mecanismos de conmutación serán explicados: la conmutación de paquetes y la conmutación de circuitos.

En la conmutación de circuitos se establece un camino entre el origen y el destino antes de comenzar a transmitir los datos; una vez establecido el camino, se comienzan a transmitir los datos y cualquier otra comunicación que se desee realizar es denegada hasta que el camino sea liberado.

En la conmutación de paquetes, se transmiten los datos sin necesidad de establecer un camino de comunicación entre el origen y el destino; simplemente se determina el camino a tomar, a medida que transita el paquete de datos por los conmutadores.

Una red que utiliza la conmutación de circuitos es menos compleja, debido a que su función es la de realizar una conexión dedicada entre el origen y su destino.

Los puntos muertos (deadlock) no son ningún problema en la conmutación de circuitos, puesto que para transmitir los datos, se genera lógicamente un enlace dedicado entre el origen y el destino. En cambio, en la conmutación de paquetes, a medida que el paquete llega a su siguiente nodo, se busca un camino para llegar al destino; esto sin lugar a duda lleva a la existencia de puntos muertos, para lo cual una de las siguientes y posibles soluciones debe ser aplicada (Wiklund, 2005):

- La primera solución se vincula a las memorias internas (buffers) de los conmutadores; a medida que un paquete llega al conmutador, éste es almacenado en el buffer interno para luego ser despachado de acuerdo a la decisión tomada por el algoritmo de enrutamiento. Los tamaños de los buffers en las NoC son limitados, en tanto esto aumentaría la ineficiencia del conmutador y aumentaría la latencia en la red.
- La segunda solución es la creación de enlaces virtuales (wormhole routing) para evitar el encolamiento de los paquetes.
- La tercera solución es la restricción de caminos que también evitaría la generación de colas.

3.4.1 Latencia

En la conmutación de circuitos, una vez establecido el camino por donde se va a despachar el paquete, la latencia no se ve afectada puesto que el paquete, para llegar a su destino, va a dirigirse por el mismo camino. El problema que se presenta en la conmutación de circuitos sucede al momento de determinar el camino a tomar. Por ejemplo, en una red mesh de 4x4 se desea transmitir 28 paquetes entre los extremos diagonales opuestos, el número de salto que debe realizarse es de 7 hasta llegar a su destino, por lo tanto, si se realiza la transmisión entre extremos diagonales, la segunda conexión no se va a realizar hasta que el enlace de la primera conexión sea liberado.

Es así como vamos a tener una latencia máxima de 7x28 = 196 ciclos, tanto de la primera conexión como de la segunda, dando como resultado un retraso total de 392 ciclos para el despacho de las dos conexiones.

La conmutación de paquetes también sufre los efectos de la latencia, donde el retardo en el despacho de los paquetes puede llegar a ser de entre cientos a miles de ciclos; esto depende directamente del algoritmo de ruteo y el diseño del conmutador (Wiklund, 2005):

Tomando en cuenta el ejemplo anteriormente mencionado, el retraso total entre las dos conexiones va a ser siempre menor a 392 y mayor a 196. No se puede saber con exactitud ya que depende de los factores ya descritos.

3.4.2 Wormhole Routing

El modo básico de transportar los paquetes es el almacenamiento y envío, donde el paquete es recibido en su totalidad por el conmutador antes de que se realice su correspondiente envío. Este es el modo típico de operación de las redes computacionales, sin embargo, la técnica de almacenamiento y envió (Store and Forward)) resulta ineficiente cuando se habla de redes pequeñas y dedicadas como las NoC, ya que para realizar esto, el tamaño de los buffers tendrían que ser altos. La necesidad de almacenar completamente los paquetes dentro del conmutador puede complicar el diseño de conmutadores compactos, rápidos y de pequeño tamaño. Por lo tanto, la transmisión del paquete se realiza mediante la unidad más pequeña de información en la NoC que es el flit (flow-control digit).

El flit es la unidad de control de flujo de los paquetes, siendo los buffer de entrada y salida de un conmutador lo suficientemente grandes para almacenar algunos flits, permitiendo que el flit de cabecera continúe su camino por la red, sin necesidad de esperar que el cuerpo y la cola lleguen. Esto reduce significativamente la latencia en el

despacho de paquetes. Este método ayuda a aumentar el rendimiento general de la red. (Wiklund, 2005).

En la figura 12 se muestra el diagrama de la funcionalidad del wormhole routing.

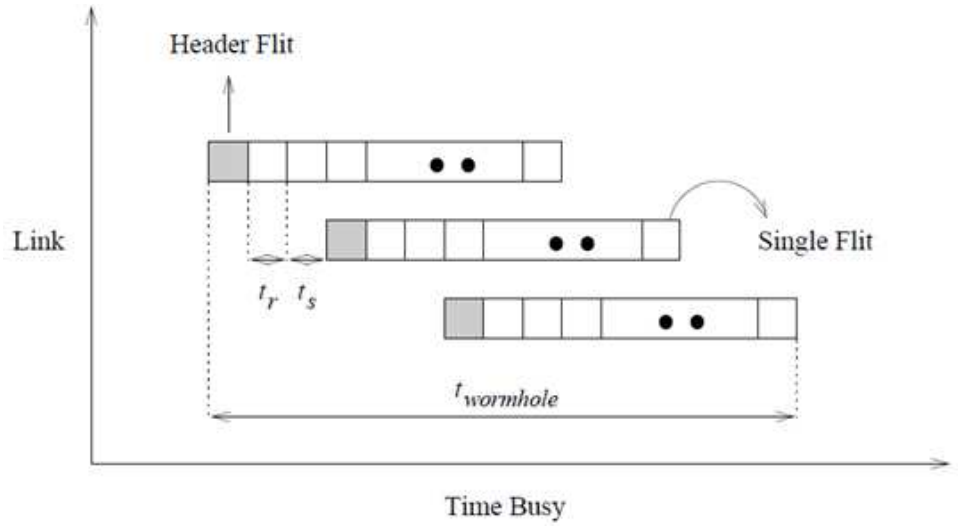


Figura. 12. Conmutación de un paquete mediante wormhole

3.4.3 Deadlock

El problema de Deadlock (camino muerto) se da cuando un elemento de la red no puede transmitir datos debido a que no existen los recursos necesarios para hacerlo.

Un ejemplo claro se puede visualizar en la conmutación de circuitos. Cuando (figura 13) el medio de transmisión está siendo utilizado por la comunicación establecida entre

los nodos B y D, los nodos A y C no van a poder comunicarse entre ellos; a esta situación se la conoce como deadlock.

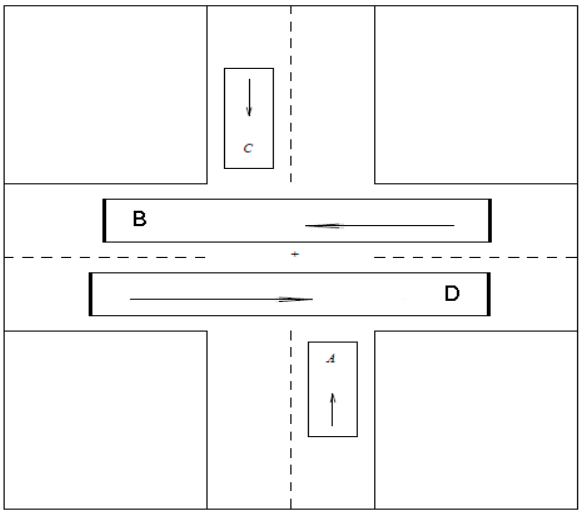


Figura. 13. Tráfico estancado por deadlock.

Otra situación en la cual se puede generar un deadlock (figura 14) es el caso en el que el buffer de B se encuentre lleno y A tenga que transmitir el paquete a B, debido a que el buffer de B no puede recibir más paquetes, la transmisión de A entra en un deadlock.

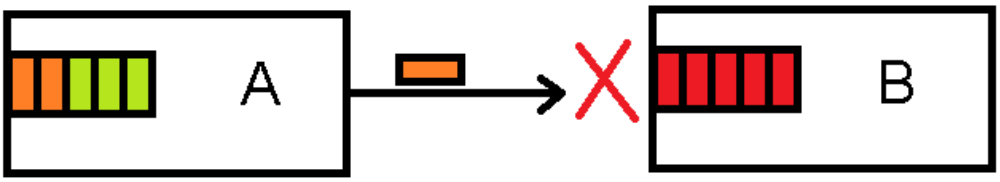


Figura. 14. Deadlock causado por tamaño de Buffer

3.5 ALGORITMO DE RUTEO

En una NoC basada en una topología en forma de malla, sea esta Mesh, Torus o sus diferentes variaciones, es sencillo lograr determinar el camino más corto para el enrutamiento, empleando algoritmos de dimensión variable tales como el XY (Keidar & Cidon, 2010).

En estas topologías es fácil calcular la distancia entre el origen y el destino como suma de las diferencias de posiciones en todas las dimensiones. El algoritmo de dimensión variable envía los paquetes cruzando los conmutadores en un orden estrictamente descendente o ascendente, reduciendo la diferencia del destino hacia el origen en una unidad antes de encaminar el paquete por el siguiente conmutador.

Para redes n-dimensionales, los algoritmos de dimensión variable ayudan a la prevención de bloqueos (deadlocks). Estos algoritmos son muy conocidos y han recibido varios nombres como encaminamiento XY, oddeven, westfirst, entre otros.

A continuación se analizará el algoritmo de enrutamiento XY, puesto que los algoritmos de dimensión variable se subdividieron a partir de éste. (Pardo Carpio, 2002)

Este algoritmo se describe en la figura 15 donde Internal es el enlace de conexión al recurso local. Estos algoritmos asumen que la cabecera del paquete lleva la dirección absoluta del destino; la primera sentencia (offset) calcula la distancia del nodo actual al nodo destino.

Algorithm: XY Routing for 2-D Meshes
Inputs: Coordinates of current node $(Xcurrent, Ycurrent)$
and destination node $(Xdest, Ydest)$
Output: Selected output $Channel$
Procedure:
$Xoffset := Xdest - Xcurrent;$
$Yoffset := Ydest - Ycurrent;$
if $Xoffset < 0$ **then**
$Channel := X-;$
endif
if $Xoffset > 0$ **then**
$Channel := X+;$
endif
if $Xoffset = 0$ **and** $Yoffset < 0$ **then**
$Channel := Y-;$
endif
if $Xoffset = 0$ **and** $Yoffset > 0$ **then**
$Channel := Y+;$
endif
if $Xoffset = 0$ **and** $Yoffset = 0$ **then**
$Channel := Internal;$
endif

Figura. 15. Algoritmo de ruteo para Topología 2D Mesh

Se tienen como datos el nodo destino de forma matricial (Xdest, Ydest) y el nodo origen (Xcurrent, Ycurrent). Se calcula la diferencia entre los nodos destino y el origen (Xoffset, Yoffset). Si la diferencia (Xoffset, Yoffset) es menor a cero, se aumenta en uno la dimensión (Xcurrent, Ycurrent); si la diferencia (Xoffset, Yoffset) es mayor a cero, se decrece en uno la dimensión (Xcurrent, Ycurrent); cuando la diferencia (Xoffset, Yoffset) es igual a cero, el paquete es despachado por el canal interno.

CAPÍTULO 4

SIMULADORES DE NoC

Los sistemas embebidos están formados por una arquitectura relativamente complicada, la cual requiere de numerosos elementos de procesamiento; por lo tanto, demanda un diseño óptimo en las NoC para asegurar un correcto manejo del tránsito interno de los datos a fin de que la comunicación entre los diferentes elementos de procesamiento pueda presentar un mejor rendimiento.

Para facilitar el desarrollo de los sistemas embebidos que contengan una red NoC han sido planteadas varias herramientas para evaluar el desempeño de las arquitecturas NoC antes de su implementación. Algunas de estas propuestas han sido presentadas por distintos equipos de investigación científica.

Las herramientas que existen para evaluar las NoC dependen del propósito para el cual fueron creadas. Podemos distinguir dos grandes clases de herramientas que son: sintetizadores o compiladores y simuladores. Los sintetizadores se refieren a los aspectos físicos con que se generan las arquitecturas (espacio, energía disipada, material). Los sintetizadores recientes son mucho más poderosos y exactos y de ellos existen versiones comerciales como FlexNoc de Arteries, INOC y Works CHAIN de Silistix.

En los simuladores se consideran dos criterios al momento de evaluar una arquitectura NoC: un estimado de la energía consumida y el rendimiento computacional (troughput, latencia). Cuando se desea diseñar una arquitectura NoC, contar con una evaluación de estas características es de suma importancia. (Achballah & Saoud, 2011).

Este capítulo se centrará solamente en los simuladores, puesto que el objetivo del proyecto de grado se vincula a la implementación de uno de ellos. Por lo tanto, no es necesario profundizar respecto a los sintetizadores.

4.1 SIMULADORES

Cuando un sistema es simulado, éste se representa en un grupo de módulos que están interconectados entre sí. La forma en que cada uno de estos módulos es ejecutado e interactúa se conoce como modelo de ejecución.

Las indagaciones científicas realizadas con respecto a los simuladores de NoC conducen a clasificarlos en dos grandes categorías o niveles de investigación: Redes y Conexión.

A continuación se describirá la orientación que cada una de éstas tiene. (Thid, A Network on Chip Simulator, 2002).

4.1.1 Nivel de Red

Las investigaciones a nivel de red son las más solicitadas por la comunidad científica, esto se debe a que las NoC se encuentran en fase de desarrollo. Los temas más discutidos en este nivel son los siguientes:

a) Topología: regular, no regular o mixta;
b) Protocolos: conmutadores;
c) Control de datos: algoritmos de ruteo, mecanismos antibloqueo, enlaces virtuales, buffers;
d) Calidad de Servicio (QoS): throughtput, latencia.

4.1.2 Nivel de Conexión

Las investigaciones realizadas en el nivel de interconexión es una consecuencia directa al nivel de red. Puesto que las interconexiones son usadas para unir las interfaces de red hacia los conmutadores y también la conexión entre ellos. Una conexión no optimizada afecta al rendimiento de la NoC, por lo tanto, la interconexión entre los nodos también tiene que ser estudiada. Los temas que se encuentran a este nivel son:

a) Sincronización,
b) Paralelo y Serial,
c) Confiabilidad,
d) Enlaces.

4.2 SIMULADORES DE NoC

A continuación, se mostrará una lista de los simuladores de NoC; cabe recalcar que esta lista no muestra todas las herramientas existentes pero sí las más comunes.

4.2.1 NS-2

El simulador NS (Network Simulator) es una herramienta disponible en múltiples plataformas desarrollada por DARPA, los creadores del internet, que ofrece soporte para la simulación de todo tipo de redes tanto cableadas como inalámbricas. Se trata de uno de los simuladores de redes más utilizado entre la comunidad docente e investigadora del área de redes de computadores.

NS-2 fue desarrollado originalmente para simular redes computacionales. Sin embargo, como las NoC comparten muchas de las características con las redes computacionales, NS-2 fue ampliamente utilizado por muchos investigadores acerca de las NoC.

Varios estudios se han realizado utilizando la herramienta de simulación NS-2, lo que lo convierte en un simulador muy fiable a la hora de comparar el rendimiento entre dos diferentes arquitecturas. Cabe recalcar que NS-2 es un simulador de código abierto desarrollado en C++ por lo que los investigadores pueden compartir su información (Simulator T. N., 2013).

4.2.2 Noxim

Esta herramienta fue desarrollada por el equipo computacional de la Universidad de Catania. Está desarrollada en SystemC y permite al usuario evaluar una arquitectura 2D Mesh variando ciertos parámetros. Noxim permite evaluar a la NoC en términos de troughput, latencia y poder consumido. (NOXIM, http://noxim.sourceforge.net/, 2013).

4.2.3 SunFloor

SunFloor es una herramienta de apoyo para el diseño NoC. Se puede utilizar en las primeras fases de diseño para sintetizar la más adecuada topología con estos parámetros como entrada (modelo, energía y espacio, objetivos de diseño). A partir de estos datos, SunFloor genera especificaciones listas para ser traducidas a una arquitectura integral, por lo general en lenguaje SystemC. SunFloor 3D es una extensión de la versión posterior. La principal característica es la generación de especificaciones para arquitecturas en 3D. Ambas versiones fueron desarrolladas por el equipo del Prof. Giovanni De Micheli. (SunFloor3D, 2012).

4.2.4 Orión 2.0 H

Esta herramienta fue desarrollada por el equipo de la Universidad de Princeton en el año 2003. Es un simulador dedicado exclusivamente a la estimación del poder consumido y el espacio de las arquitecturas NoC. Una de sus mejoras, comparado con otros simuladores, es que existe el soporte para probar nuevos semiconductores a través de los modelos de transistores y su capacitancia de acuerdo a cómo evolucione la industria. (Et al, 2002).

4.2.5 BookSim

Fue desarrollada entre los años 2002-2010 en la Universidad de Stanford. La versión actual, BookSim 2.0, es compatible con una amplia variedad de topologías tales como Mesh, Torus y Fat tree; ofrece diversos algoritmos de enrutamiento e incluye numerosas opciones para personalizar la micro arquitectura de la red del enrutador. (Simulator B. I., 2013).

4.2.6 Worm_sim Simulator

Worm_sim proporciona una solución eficiente para permitir al usuario especificar condiciones de tráfico arbitrario para la NoC. El usuario tiene el control sobre la velocidad de transmisión de cada nodo IP, el tamaño del paquete, su distribución, etc. Permite al usuario adjuntar un archivo de rastreo para cada nodo IP individual, para que el sistema bajo simulación imite el estado del tráfico exacto de aplicaciones reales. Fue desarrollado en el año 2005 por CMU. (ENGINEERING, 2013).

4.2.7 VNOC

Se ha integrado el modelo de carga (energía) de Orión. Asimismo cuenta con una interfaz gráfica de usuario útil para la depuración y la visualización de la congestión de los enrutadores. Se la desarrolló a partir del año 2009. (tools).

4.2.8 Nostrum

NoC Nostrum Simulator Enviroment (NNSE) es parte del proyecto de Nostrum y contiene un simulador basado en SystemC. Fue desarrollada por KTH – Royal Institute of Technology entre los años 2002-2006; en ésta se utiliza una interfaz gráfica de usuario para seleccionar tamaño, topología, enrutamiento y patrones de tráfico. En base a estos parámetros de configuración, los resultados de la simulación se pueden mostrar en una variedad de gráficos. (Environment, 2009).

4.2.9 Nirgam

La Universidad de Southampton la desarrolló en el año 2007. NIRGAM está basado en SystemC y presta un apoyo sustancial al desarrollo de las NoCs ya que se puede experimentar con el diseño de NoCs, en términos de algoritmos de enrutamiento y nuevas propuestas de topologías. (NIRGAM, 2012).

4.2.10 GPNoCsim

En este proyecto se presenta un modelo de propósito general para NoC. Está escrito en código abierto, basado en un entorno de simulación de red que se ha desarrollado completamente en GpNoCsim, mediante la simulación de varias arquitecturas conocidas. Además, se proporcionan las directrices para simular las nuevas arquitecturas haciendo modificaciones sencillas. Como un simulador de propósito general para las arquitecturas NoC, gpNoCsim ofrece a los diseñadores la flexibilidad de la configuración del tráfico, la topología y algoritmos de ruteo. (Simulator for Network-On-Chip, 2006).

4.2.11 Tabla de Simuladores

Existen también otros simuladores de NoC en el mercado. La siguiente figura (Figura 16) resume las herramientas que se han encontrado:

Tabla. 5. Simuladores NoC, encontrados

#	Nombre	Año	Grupo
1	NS-2	1995	DARPA
2	Noxim	2010	Universidad de Catania
3	Darsim	2009	MIT
4	SunFloor - 3D	2006-09	EPFL
5	Orion	2003-09	Universidad de Princeton
6	INSEE	2005	Universidad Basca
7	Atlas	2005	Universidad Federal de Brazil
8	Nocic	2004	Universidad de Massachussetts
9	Pestannna	2004	Laboratorios Phillips
10	Pirate	2004	Escuela Politecnica de Milan
11	Sunmap	2004	Universodad de Stanford
12	NoCgen	2004	-
13	FlexNoc	-	ARTERIS
14	Inoc	-	-
15	Chain	-	Silistix
16	BookSim	2002-10	Universidad de Stanford
17	Worm_sim Simulator	2005	CMU
18	NoC Simulator	2007	Universidad de Las Palmas de Gran Canaria
19	VNOC	2009	-
20	Sicosys	2008	Universidad de Cantabria
21	Nostrum	2002-06	KTH – Royal Institute of Technology
22	Nirgam	2007	Universidad de Southampton
23	NoCSim	2002-06	TAMU
24	gpNoCsim	2006	Bangladesh University of Engineering and Technology

4.3 PARÁMETROS A EVALUAR

Lo que se desea evaluar en una NoC es un paso importante para lograr clasificar a las arquitecturas. Existen 3 criterios generales tomados en cuenta por la comunidad de investigadores: área, poder consumido y rendimiento computacional (troughtput y latencia). Existen también otros criterios como la pérdida de paquetes o la longitud de los cables; pero no son criterios discutidos a la hora de proponer una NoC.

En el estudio de los simuladores sobre las NoC se puede encontrar que, en general, todas las herramientas se centran en el nivel de red, lo cual hace referencia a dos de los tres criterios anteriormente mencionados. (Achballah & Saoud, 2011).

La siguiente tabla enlista los simuladores de NoC que se pudieron recolectar mostrando las características que cada una de éstas facilita.

Tabla. 6. Resumen de los Parámetros entrada/salida de Simuladores NoC

#	Simulador	TR	TB	DP	AR	PIR	ES	DT	CA	CP	T	L	Disponibilidad
1	NS-2	+	+	+	+	+	+	+	+	+	+	+	+
2	Noxim	+	+	+	+	+	+	+	-	+	+	+	+
3	Darsim	+	+	+	+	+	+	+	-	-	+	+	-
4	SunFloor -3D	+	-	-	-	-	-	-	-	+	+	+	-
5	Orion	-	+	-	-	-	-	-	+	+	-	-	-
6	Atlas	+	-	+	+	+	-	-	-	-	+	+	+
7	Pirate	+	+	-	-	-	-	-	+	+	-	-	-
8	Sunmap	+	+	-	-	-	-	-	-	+	+	+	-
9	uSpider	+	+	-	-	-	-	-	-	-	+	+	-
10	NoCgen	-	+	-	-	-	-	-	-	-	+	+	-
11	FlexNoc	+	+	+	+	+	+	+	+	+	+	+	commercial
12	iNoc	+	+	+	+	+	+	+	+	+	+	+	commercial
13	Chai Tool	+	+	+	+	+	+	+	+	+	+	+	commercial

TR= Tamaño de Red	**ES= Estrategia de Selección**
TB= Tamaño del Buffer	DT= Distribucion de Trafico
DP= Distribucion de Paquetes	CA= Consumo de Area
AR= Algoritmo de Ruteo	CP= Consumo de Poder
PIR= Packet Injection Ratio	T= Throughput
	L= Latency

4.3.1 Selección de Simulador Base

Para la implementación del simulador, primero se tiene que seleccionar un simulador base puesto que el iniciar este proyecto desde cero, implicaría realizar una investigación científica más especializada la cual requeriría de un grupo de investigación del tipo que se ejecutan para obtener un título de postgrado y no de pregrado como es este caso. Asimismo, se debería desarrollar un estudio sobre cómo crear un simulador, incluyendo todos los procesos que esto conlleva.

Con la finalidad de escoger el simulador base adecuado se procedió a evaluar diferentes simuladores para obtener resultados propios que, junto a la información mostrada en la Figura 17, permitió seleccionar el simulador base. A continuación se explica el procedimiento general para la evaluación de los simuladores:

- Lectura del Manual de Usuario

- Ejecución de Simulador

- Obtención de Resultados

- Revisión de estructura y código del Simulador.

.Lectura del Manual de Usuario

Primero, se procede a descargar la documentación que faciliten los desarrolladores en los portales web. (Figura 16)

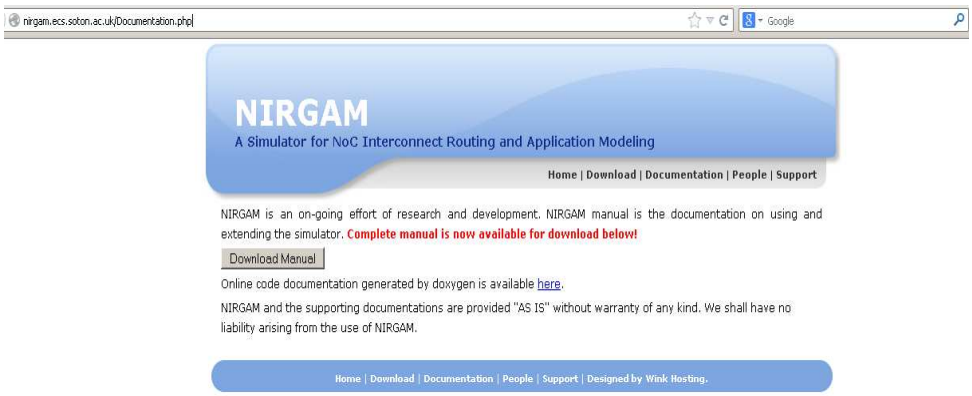


Figura. 16. Lectura del Manual de Usuario

Una vez descargado el manual de usuario, el enfoque estará en las siguientes secciones: instalación, descarga y ejecución.

Se sigue el procedimiento descrito en la instalación, el cual conlleva a la descarga de los componentes necesarios para la instalación del simulador y su compilación. (Figura 17)

Running Simulations

This chapter provides a detailed documentation on configuring and running simulations.

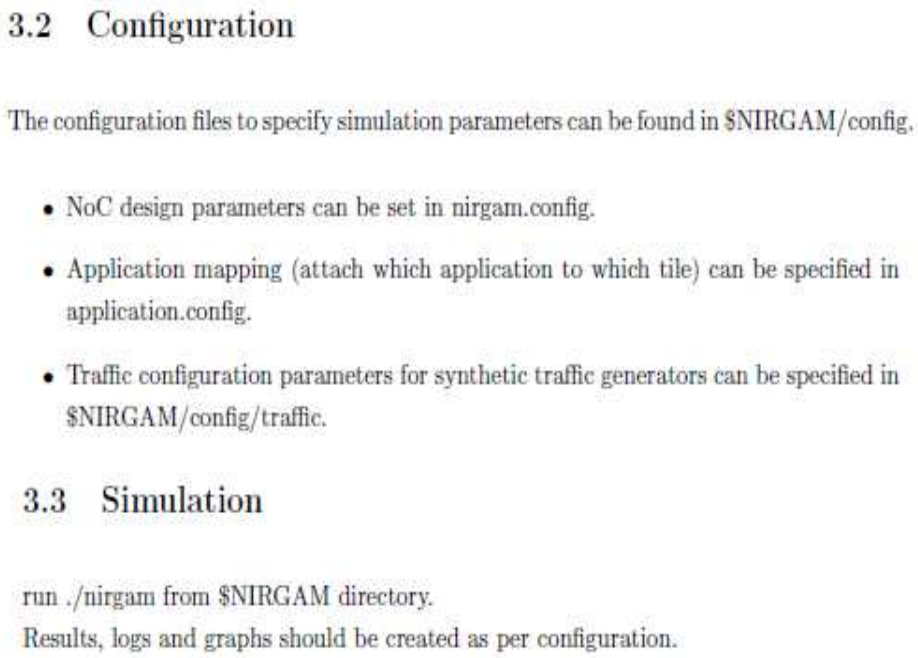
3.1 Installation

1. Download and install systemC from www.systemc.org.
2. Download the simulator archive nirgam.tgz and extract to a location of your choice.
   You should see a folder 'nirgam' having the directory structure as Figure 2.3.
3. cd into the nirgam directory.
   Hereafter we will refer to the path to nirgam directory as $NIRGAM.
4. Edit $NIRGAM/Makefile.defs to set variable SYSTEMC equal to your systemC installation path.
5. run make.
   This should create an executable 'nirgam' in $NIRGAM.

Figura. 17. Running Simulations

.Ejecución del Simulador

Una vez instalado, se ejecuta al simulador con los parámetros de default mediante el procedimiento descrito en el manual del usuario. (Figura 18)



3.2 Configuration

The configuration files to specify simulation parameters can be found in $NIRGAM/config.

- NoC design parameters can be set in nirgam.config.
- Application mapping (attach which application to which tile) can be specified in application.config.
- Traffic configuration parameters for synthetic traffic generators can be specified in $NIRGAM/config/traffic.

3.3 Simulation

run ./nirgam from $NIRGAM directory.
Results, logs and graphs should be created as per configuration.

Figura. 18. Ejecución del Simulador

En ciertos casos, los simuladores no se ejecutan como se espera debido a la falta de soporte y continuidad que éstos tienen, por lo que se debe hallar la solución en foros.

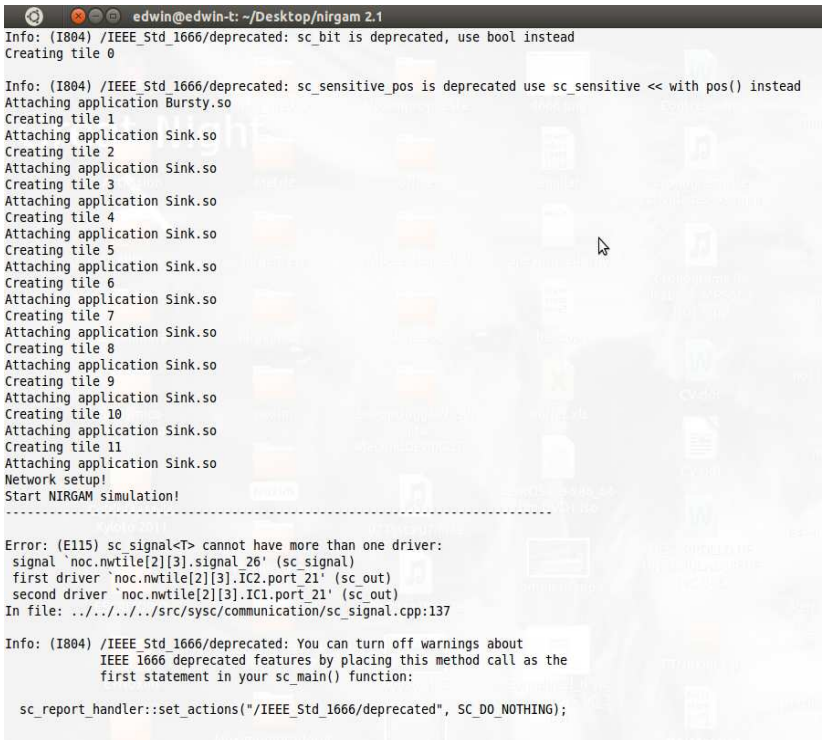
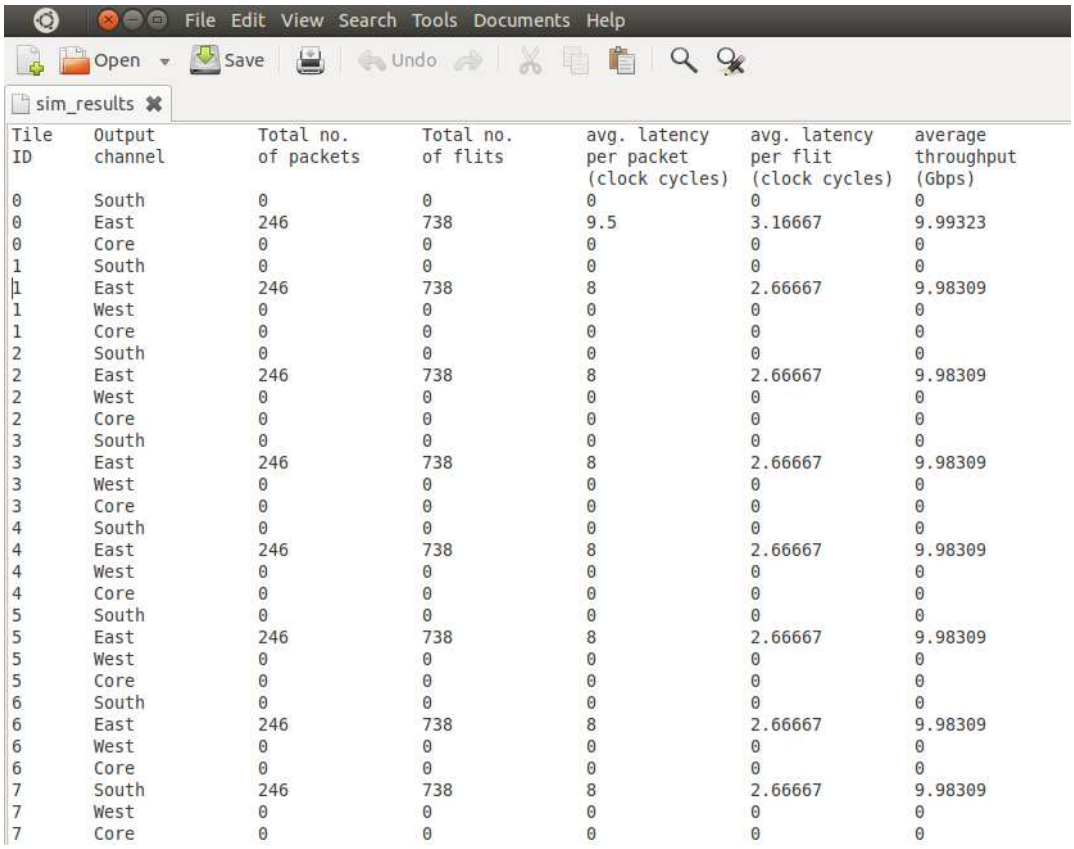


Figura. 19. Falla en los Simuladores

Figura. 20. Falla en los Simuladores

.Obtención de Resultados

Una vez ejecutado el simulador, se observan los resultados que éste entrega y se los interpreta de acuerdo a lo indicado por el manual de usuario. (Figura 21)



Tile ID	Output channel	Total no. of packets	Total no. of flits	avg. latency per packet (clock cycles)	avg. latency per flit (clock cycles)	average throughput (Gbps)
0	South	0	0	0	0	0
0	East	246	738	9.5	3.16667	9.99323
0	Core	0	0	0	0	0
1	South	0	0	0	0	0
1	East	246	738	8	2.66667	9.98309
1	West	0	0	0	0	0
1	Core	0	0	0	0	0
2	South	0	0	0	0	0
2	East	246	738	8	2.66667	9.98309
2	West	0	0	0	0	0
2	Core	0	0	0	0	0
3	South	0	0	0	0	0
3	East	246	738	8	2.66667	9.98309
3	West	0	0	0	0	0
3	Core	0	0	0	0	0
4	South	0	0	0	0	0
4	East	246	738	8	2.66667	9.98309
4	West	0	0	0	0	0
4	Core	0	0	0	0	0
5	South	0	0	0	0	0
5	East	246	738	8	2.66667	9.98309
5	West	0	0	0	0	0
5	Core	0	0	0	0	0
6	South	0	0	0	0	0
6	East	246	738	8	2.66667	9.98309
6	West	0	0	0	0	0
6	Core	0	0	0	0	0
7	South	246	738	8	2.66667	9.98309
7	West	0	0	0	0	0
7	Core	0	0	0	0	0

Figura. 21. Obtención de Resultados

.Revisión de Estructura y Código de Programación

Una vez instalado, ejecutado y obtenido los resultados, se observa la estructura que presenta el simulador, es decir el cómo los desarrolladores modelan a las NoC.(Figura 22)

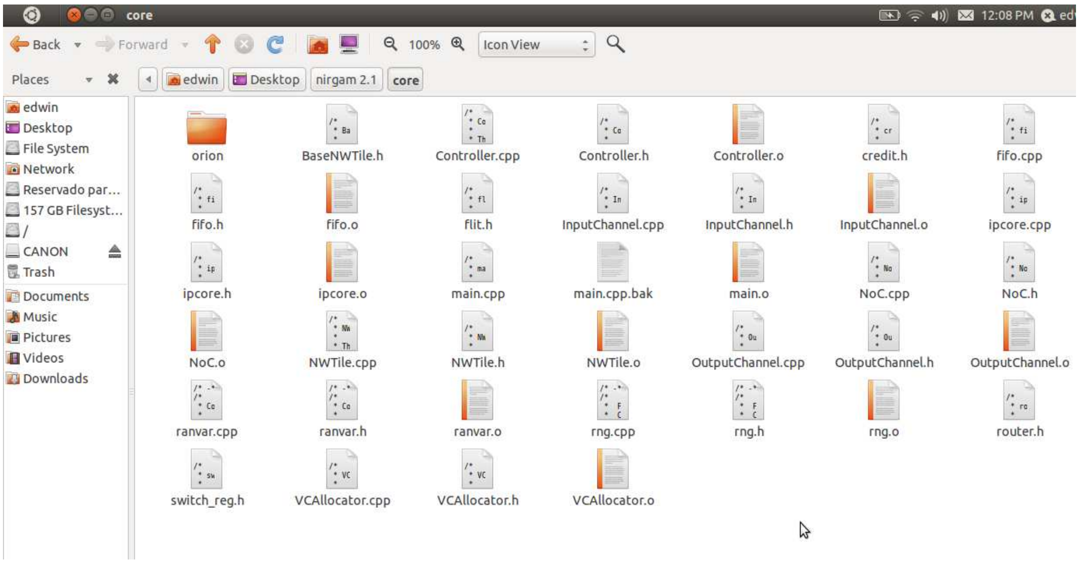


Figura. 22. Revisión de Estructura y Código de Programación

Finalmente, se observa a breves rasgos el código para comprender un poco más a los desarrolladores.

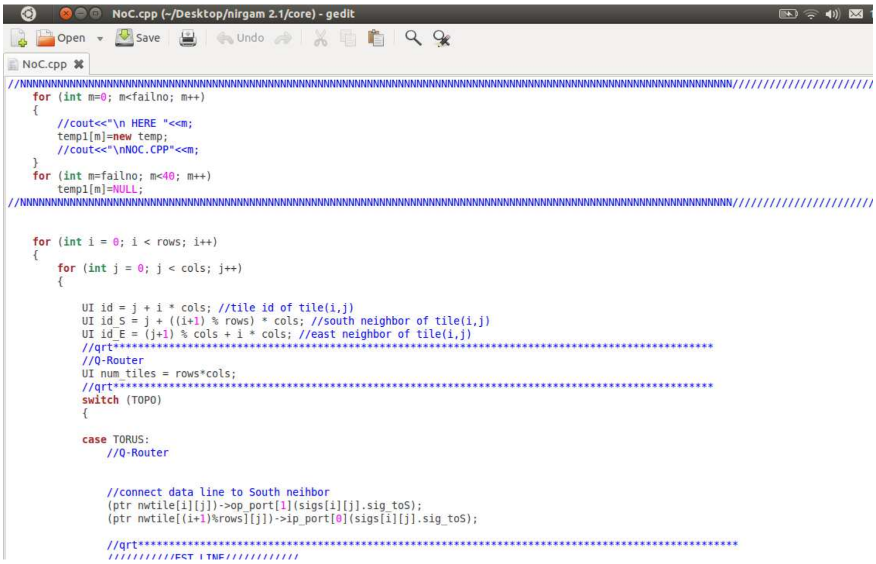


Figura. 23. El código para comprender los Desarrolladores

Una vez evaluados diferentes simuladores y recolectada la información de otros, se procedió a realizar la selección del simulador base. En el Anexo 1 se lista los simuladores evaluados junto con sus características.

Luego de haber analizado las ventajas y desventajas que presentan los simuladores existentes en el mercado, al igual que los parámetros que estos permiten controlar, los resultados que entregan (Figura 17) y el código de programación que utilizan, se ha optado por seleccionar a Noxim como simulador base para el propósito del proyecto, debido a que es un simulador completo en cuanto a su estructura de modelamiento a una arquitectura NoC.

Noxim está conformado por módulos que corresponden a cada uno de los elementos mencionados en el capítulo 3; de igual manera, permite controlar varios parámetros de entrada y entrega 2 de los 3 criterios a tomar en cuenta a la hora de analizar una arquitectura NoC. En el siguiente capítulo se dará a conocer una descripción más detallada de cuáles son estos parámetros.

Está escrito en código abierto y puede ser descargado bajo los términos de la licencia GPL (Licencia Pública General) que es utilizada en el mundo del software libre y garantiza a los usuarios finales (personas, organizaciones, compañías) la libertad de estudiar, compartir (copiar) y modificar el software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

El lenguaje de programación de Noxim es SystemC, definido como un lenguaje de descripción de hardware como lo son VHDL y Verilog. Tanto el lenguaje de programación VHDL y Verilog son utilizados en el desarrollo de los SoC, por lo tanto Noxim, al utilizar System C como base, arrojará resultados semejantes a los reales.

Entre la comunidad de investigadores de arquitecturas NoC, la herramienta más utilizada como referente de simulación es Noxim (Figura 26).

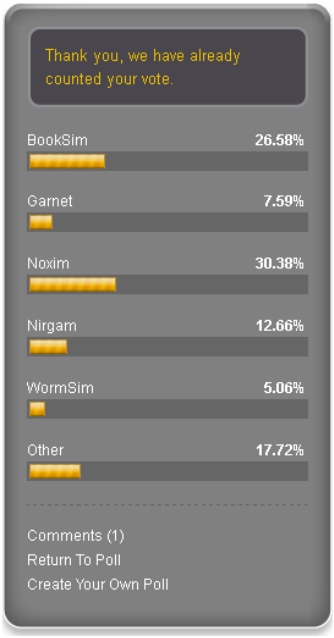


Figura. 24. Los 5 Simuladores de NoC más populares

(Network-On-Chip, 2012)

CAPÍTULO 5

SIMULADOR BASE NOXIM

Noxim es un simulador de NoC desarrollado en la Universidad de Catania (Italia). Es un simulador cuyo lenguaje de programación es SystemC, lenguaje descriptor de sistema basado en C++ que puede ser descargado de SourceForge bajo los términos de la licencia GPL.

Noxim se apoya en SystemC para modelar una red de interconexión por módulos de la forma más real posible. SystemC es un lenguaje de descripción de hardware similar a VHDL y Verilog; su principal característica radica en el modelado de sistemas a nivel de comportamiento. Los objetos descritos bajo este lenguaje son capaces de comunicarse durante una simulación de tiempo real utilizando señales de cualquier tipo. Un sistema en SystemC está formado por un conjunto de módulos que describen cierta funcionalidad y se comunican con otros módulos a través de canales o eventos.

Noxim tiene una interfaz de comandos de línea para definir varios parámetros de la NoC; en particular, el usuario puede modificar el tamaño de la red, del buffer, del paquete a ser distribuido, el número de paquetes, el algoritmo de ruteo, la velocidad de los paquetes a ser inyectados al sistema y la distribución del tráfico.

El simulador permite evaluar a la NoC en términos de troughput, latencia y energía consumida. Estos resultados son entregados al usuario en términos de promedio y unitarios.

Siendo más precisos, el usuario puede obtener las diferentes evaluaciones de las características de la NoC incluyendo el número total de paquetes/flits recibidos, el promedio global del troughput y latencia, el retraso máximo global, la energía total consumida. Adicionalmente, se puede obtener el retraso, troughput y energía consumida por cada comunicación realizada (NOXIM, www.Noxim.org, 2013).

5.1 PARÁMETROS DE ENTRADA Y SALIDA

Parámetros de Entrada

- Dimensión X de la matriz.
- Dimensión Y de la matriz.
- Tamaño de buffer (en flits).
- Tamaño mínimo y máximo de cada paquete (en flits).
- Algoritmo de ruteo.
- Selección de estrategia de bufferizado.
- Tipo de distribución de PIR (Package Inyection Rate).
- Tipo de distribución de tráfico.
- Nodos específicos en los cuales habrá más tráfico (hot-spot).
- Tiempo después de haber iniciado la simulación, empieza la recolección de datos.
- Ingresar un número para crear aleatoriedad en la distribución de tráfico.
- Tiempo de simulación (en ciclos)

Parámetros de Salida

- El número total de paquetes recibidos
- El promedio global del troughput y latencia
- El retraso máximo global
- La energía total consumida

5.2 INSTALACIÓN

Se procederá a explicar la instalación del simulador Noxim que puede ser ejecutado solo en la plataforma Linux.

1. Descargar el simulador Noxim de la dirección: www.Noxim.org
2. Crear una carpeta de nombre "Noxim" en una ubicación de direccionamiento fácil y descomprimir allí los archivos y carpetas.
3. Descargar la librería SystemC de la dirección de internet: www.systemc.org/downloads/standards/ (Se requiere registrarse de forma gratuita).
4. Ubicar la carpeta descomprimida en la misma ubicación de la carpeta anterior.
5. Instalar el compilador build-essential mediante el comando:

   sudo apt-get install build-essential
6. Abrir una pantalla de Terminal.
7. Direccionarse a la carpeta "systemc-2.2.0"
8. Escribir las siguientes instrucciones:
   - *mkdir objdir*
   - *cd objdir*
   - *export CXX=g++*

- *../configure*
- *make*
- *make install*
- *cd ..*
- *rm –rf objdir*

Si da un mensaje de error al usar el comando “make” se lo arreglará de la siguiente manera:

- En la carpeta “systemc-2.2.0” se ingresará al directorio ../src/sysc/utils/
- Se abrirá el archivo “sc_utils_ids.cpp” y en las primeras líneas se añadirá:

  #include <cstdlib>

  #include <cstring>

Se vuelve a realizar los pasos anteriores empezando con el paso del comando “make”.

9. Se dirige a la carpeta “Noxim” e ingresa al directorio “bin”.
10. En este directorio se edita el archivo Makefile.defs y se reemplaza la parte derecha del “=” en la línea 8, por la dirección de instalación de la carpeta systemc-2.2.0
11. En la pantalla del terminal se direcciona a la carpeta “bin” descrita anteriormente y se utiliza el comando “*make”*.
12. Finalmente, se escribe “*./Noxim*” y el programa simulará los datos que tiene por default.

5.3 MANUAL DE USUARIO

Para ver las opciones del simulador Noxim, se debe abrir el terminal y ubicarse en el directorio “bin” descrito en la instalación. Una vez allí, se utilizará el comando:

./Noxim –help

Las opciones que se muestran en el “-help” son descritas a continuación:

- ***./Noxim –verbose N***

Aquí se selecciona el nivel de datos a presentarse en la salida que generará el simulador Noxim, donde N puede ser: 1=low, 2=medium, 3=high, default off.

Por defecto, la opción está en default off que permitirá obtener sólo las siguientes características principales: número total de paquetes recibidos, número total de flits recibidos, retraso promedio global, throughput promedio global, throughput, latencia máxima y energía total consumida.

En el nivel low, se adicionará a los datos anteriores los parámetros de configuración y se podrá observar el trabajo hecho por cada elemento de la NoC.

En el nivel medium no existe diferencia (por ahora) con el nivel anterior.

En el nivel high se adicionará a todo lo descrito en los dos primeros niveles (low y medium) la información detallada acerca de los flit por toda actividad realizada en cada elemento NoC.

- ***./Noxim –trace FILENAME***

  Esta opción permite generar un archivo "FILENAME.vcd". Mediante el programa "gtkwave" se puede visualizar las formas de onda de cada señal del SystemC utilizadas por el simulador.

- ***./Noxim –dimx N -dimy N***

  Esta opción permite establecer el tamaño de la red así como el largo (dimy) y ancho (dimx) de la matriz que representa la malla (mesh) de la NoC. Por defecto es de 4 por 4.

- ***./Noxim –buffer N***

  Esta opción permite definir el tamaño del buffer de cada canal del router. El tamaño se expresa en flits. Por defecto su tamaño es de 4 flits.

- ***./Noxim –size Nmin Nmax***

  Establece el tamaño mínimo y máximo de cada paquete; está expresado en flits. Los valores por defecto son min=2, máx=10.

- ***./Noxim –routing TYPE***

  Esta opción permite especificar uno de los algoritmos de ruteo descritos a continuación:

xy	Algoritmo de ruteo XY
westfirst	Algoritmo de ruteo West-First
northlast	Algoritmo de ruteo North-Last
negativefirst	Algoritmo de ruteo Negative-First
oddeven	Algoritmo de ruteo Odd-Even
dyad T	Algoritmo de ruteo DyAD con threshold T
fullyadaptive	Algoritmo de ruteo Fully-Adaptative

- ***./Noxim –warmup N***

  Con esta opción se puede establecer un tiempo de inicio para la recolección de estadísticas. N se expresa en ciclos.

- ***./Noxim –seed N***

  Esta opción permite establecer la simiente del generador de números aleatorios usados por el simulador. Por defecto, utiliza la función estándar time().

- ***./Noxim –volumen N***

  Esta opción permite detener la simulación cuando se llega al número máximo de ciclos o cuando N flits son entregados.

- ***./Noxim –sim N***

  Esta opción permite especificar el número de ciclos de reloj que deben ser simulados. El valor por default es 10000 (diez mil) ciclos.

5.4 CÓDIGO BASE

Las librerías que se compilaron en el momento de la instalación de Noxim se encuentran en el directorio src de la carpeta Noxim y contienen los archivos .cpp que incluyen las funciones y los archivos .h de las constantes.

Si se realiza cualquier modificación en el código, se deberán seguir los siguientes pasos:

1.- Guardar el archivo modificado.

2.- Abrir una ventana "Terminal" y direccionarse a la carpeta bin dentro del simulador Noxim.

3.- Escribir "*make clean*" y dar enter.

4.- Escribir "*make*" y dar enter.

5.- ¡Listo! El programa está actualizado.

Se procederá a explicar de forma general que contienen las funciones creadas en los archivos .cpp los cuales son los siguientes:

NoximCmdLineParser:

Se encarga de imprimir mensajes en el terminal y de modificar los parámetros por defecto de la simulación. Los mensajes a imprimirse se llaman mediante las funciones descritas a continuación.

- showHelp(): Imprime las opciones para modificar los parámetros de la simulación.

- showConfig(): Imprime los parámetros con los cuales está configurada la simulación.
- checkInputParameters(): Imprime mensajes de error debido a que se ingresó de manera quivocada un parámetro como números negativos, letras o rangos inválidos.
- parseCmdLine(): se encarga de modificar los parámetros de default por los parámetros que ingresamos mediante la línea de comandos en el terminal.

NoximGlobalRoutingTable:

Se generan las tablas de enrutamiento en cada nodo y hace un mapeo de sus vecinos para cada canal; contiene las funciones:

- direcction2ILinkId(node_id, dir): convierte una dirección en un LinkId (dirección de un link entre 2 nodos).
- oLinkId2Direction(LinkId): convierte un LinkId en una dirección.
- getNodeRoutingTable(node_id): obtiene la tabla de ruteo de un nodo específico.
- Load(file): carga una tabla de ruteo definida por el usuario.

NoximLocalRoutingTable:

Aquí se configura la tabla de enrutamiento de cada nodo y se definen los canales admisibles entre un nodo y sus links para llegar a cada destino; además, se guarda en las tablas de ruteo.

NoximReservationTable:

Esta tabla define los puertos que están disponibles para el envío de paquetes, los puertos que no son válidos por no estar conectados a otro nodo y los puertos que están temporalmente inhabilitados debido a la gran cantidad de tráfico que pasa por ese canal.

Las funciones que podemos llamar son:

- clear(): limpia la tabla de reservación.
- isAvalible(port_out): indica si un puerto está disponible.
- getOutputPort(port_in): indica qué puerto de salida está conectado con un puerto de entrada.
- reserve(port_in, port_out): conecta un puerto de entrada con uno de salida.
- reléase(port_out): libera un puerto determinado.
- invalidate(port_out): invalida un puerto para que no pueda se reservado o liberado.

NoximBuffer:

El buffer está modelado como un contenedor de paquetes de almacenamiento tipo FIFO (First In First Out). Las funciones que podemos llamar son:

- SetMaxBufferSize(int): establece el tamaño máximo de flits que puede contener.
- SetMaxBufferSize(): indica el tamaño máximo del buffer.
- IsFull(): devuelve un booleano true si el buffer está lleno.
- IsEmpty(): devuelve un booleano true si el buffer está vacío.

- Push(): Ingresa un flit en el buffer.
- Drop(): Sub función del push. Cuando el buffer está lleno, detiene todo proceso de recepción.
- Pop(): saca un flit del buffer.
- Front(): Sub función del pop. Cuando el buffer está vacío, detiene todo proceso de transmisión.
- Size(): indica el tamaño actual del buffer.
- getCurrentFreeSlots(): indica el número de slots libres en el buffer.

NoximRouter:

Los routers poseen buffers encargados de administrar los flits que llegan a un nodo y guiarlos al siguiente nodo hasta su destino final mediante algoritmos de ruteo. Realizan un ruteado de dimensión variable. Para aceptar un nuevo flit se deben cumplir 2 condiciones:

1. Que exista un requerimiento de flit entrante.
2. Que haya un slot libre en la dirección de entrada del flit.

Las funciones que se pueden llamar son:

- rxProcess(): protocolo de recepción, mediación wormhole y señal de ack de recibo.
- txProcess(): protocolo de transmisión, reservación, preparación de datos de ruteo, contador de flits ruteados y reenvío de flits.
- getCurrentNoPData(): informa el estado de los buffer de canales vecinos.
- bufferMonitor(): monitoreo del estado de los buffer en el nodo actual.

- Configure(): configuración del router con los números de ciclo a simular, Id, tabla de ruteo y tamaño de los buffers.
- getRoutedFlits(): número de flits ruteados.
- getFlitsCount(): número de flits en el router.
- getPower(): energía consumida.
- reflexDirection(direction): dirección reflejada.
- getNeighbordId(direction): obtención del ID de un router adyacente.
- InCongestion: devuelve un booleano true si existe congestión en cualquier canal del nodo actual.

NoximProcessingElement:

Aquí es donde se generan los paquetes para propósitos de simulación con números aleatorios que determinarán de qué nodo a qué nodo se van a transmitir los paquetes, por lo que también se generan las distintas distribuciones de tráfico basadas en lo que se haya seleccionado en el terminal.

Aquí se encuentra el protocolo de comunicación ABP (Alternating Bit Protocol) que funciona mediante ack (acuse de recibo). Se encarga de recibir y transmitir los paquetes de modo que el router tiene como única función poner cada paquete en el canal adecuado.

Las funciones que se pueden llamar son:

- randInt(): número aleatorio.
- rxProcess(): proceso de recepción de flits mediante el protocolo ABP.
- txProcess(): proceso de transmisión de flits mediante el protocolo ABP.

- nextFlit(): toma el siguiente flit del paquete actual.
- canShoot(): booleano que señala true cuando un flit puede ser enviado.

NoximTile:

En cada nodo se encuentra un tile, que es la unión de un Processing Element y un router como observa en la Figura 27. En el archivo Tile.h se encuentra la declaración de cada señal que contiene; además, crea un objeto ProcessingElement que genera la información (payload) en forma de paquetes para ser transmitidos a otros nodos y crea un objeto router encargado de dirigir los paquetes a través de la malla. Este archivo sólo se encarga de conectar las señales entre un ProcessingElement y un router (a los buffers se los toma en cuenta como parte del router).

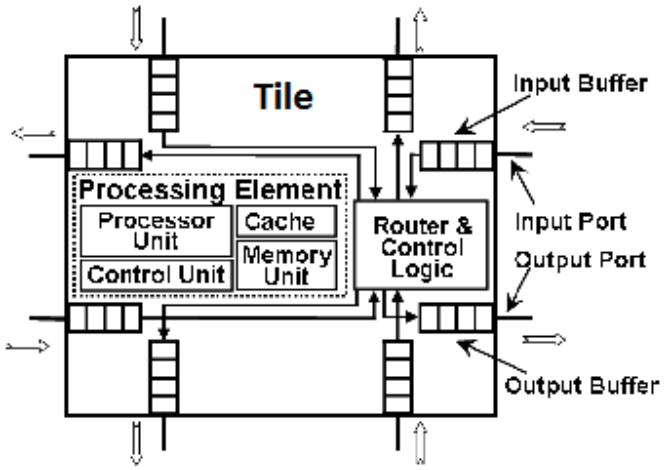


Figura. 25. Descripción de un tile

NoximNoC:

Crea una malla con un tile en cada nodo; además conecta todas las señales entre nodo y nodo como se muestra en la Figura 28, dando un número a cada nodo para realizar las

comunicaciones. Las señales de reloj y de reset de cada tile están unidas a un clock y reset global, los cuales se establecen en el archivo NoximMain.cpp.

Las tablas de ruteo también son indexadas en esta librería. Es invalidado el canal de los tiles que no poseen vecinos en ciertas direcciones por ejemplo los localizados en los bordes y que no tienen vecinos; por ejemplo el nodo 0 no tiene vecino al norte, ni al oeste de su posición.

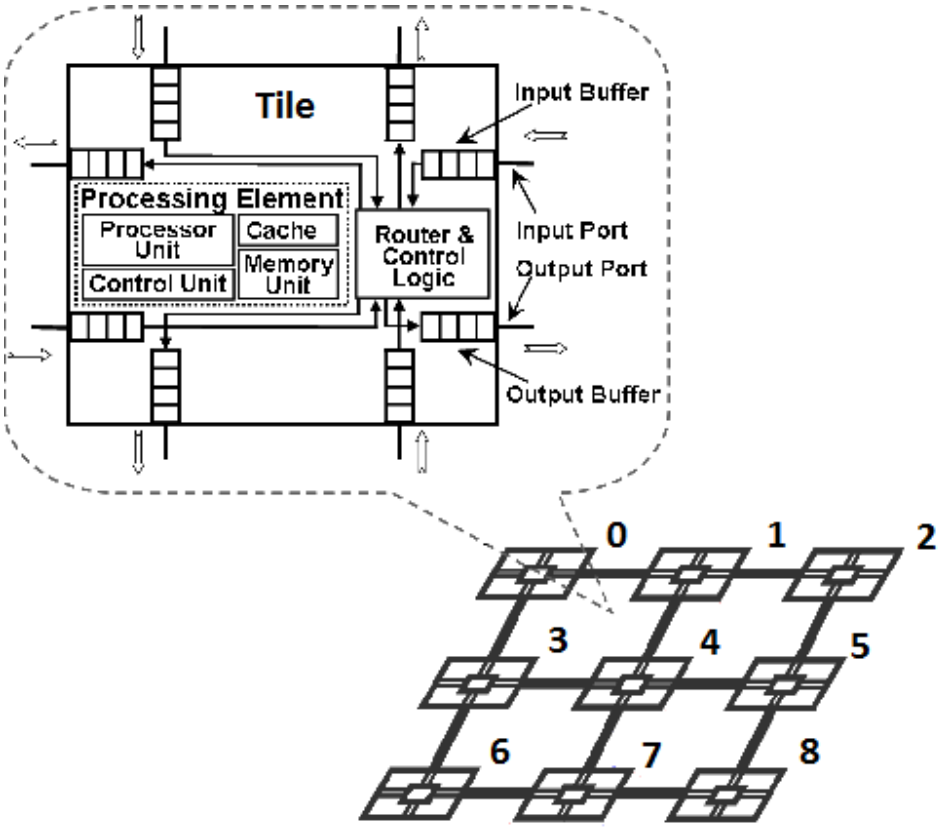


Figura. 26. Malla de tiles

NoximPower:

Para calcular la energía que se disipa al realizar comunicaciones en una NoC se modeló cada elemento y se obtuvieron los siguientes resultados:

- La energía promedio disipada por un flit en un salto se estimó en 0.151nJ, 0.178nJ, 0.182nJ and 0.189nJ para XY, Odd-Even, DyAD y NoP-OE, respectivamente.

Se asume que el tamaño del tile es de 2mm x 2mm y que los tiles fueron distribuidos de una manera regular en el floorplan. La capacitancia de los cables se estableció en 0.50fF por micron, así que se considera que, por un promedio del 25% de actividad del router, la cantidad de energía consumida por un flit al realizar un salto de interconexión será de 0.384nJ.

Las funciones que se pueden llamar son:

- getPower(): obtiene la energía consumida por todos los elementos.
- getPwrRouting(): obtiene la energía consumida al rutear un flit.
- getPwrSelection(): obtiene la energía consumida por el buffer.
- getPwrForward(): obtiene la energía consumida al reenviar flits.
- getPwrStandBy(): obtiene la energía consumida mientras el router está en espera de un flit.
- getPwrIncoming(): obtiene la energía consumida al recibir un flit.

NoximStats:

Se encarga de recolectar los datos, realizar cálculos estadísticos de la simulación e imprimir en el terminal todos los datos obtenidos de cada nodo. Las funciones que se pueden llamar son:

- configure(node_id,warmUpTime): configura el tiempo en el cual se empezará a recolectar datos de la simulación.
- receivedFlit(arrivalTime, Flit): escribe en un registro el historial de flits recibidos.
- getAverageDelay(): señala el delay promedio de todos los nodos.
- getAverageDelay(src_id): indica el delay promedio de un nodo.
- getMaxDelay(): muestra el delay máximo de todos los nodos.
- getMaxDelay(src_id): indica el delay máximo de un nodo.
- getAverageThroughput(): expone el throughput promedio de todos los nodos.
- getAverageThrougput(src_id): muestra el throughput promedio de un nodo.
- getRecibedPackets(): indica el número de paquetes recibidos.
- getRecibedFlits(): señala el número de flits recibidos.
- getTotalCommunications(): indica el número de comunicaciones que se realizaron durante la simulación.
- getCommunicationEnergy(src_id,dst_id): energía consumida al comunicarse dos nodos dados.
- searchCommHistory(src_id): búsqueda de las estadísticas de un nodo específico.
- showStats(); despliega en el terminal las estadísticas obtenidas por cada nodo.

NoximGlobalStats:

Se encarga de recolectar las estadísticas globales a lo largo de la simulación. Las funciones que utiliza las hereda de NoximStats y guarda los resultados en matrices para su posterior impresión por consola. Las estadísticas globales que se imprimen son:

- ✓ Total de paquetes recibidos.
- ✓ Total de flits recibidos.
- ✓ Delay promedio global.
- ✓ Throughput promedio global.
- ✓ Throughput.
- ✓ Delay máximo.
- ✓ Energía total.

Noxim Main:

Es el primer módulo a ser llamado al utilizar el ejecutable Noxim. Las variables se inicializan en una variable global llamada NoximGlobalParams que se pueden modificar a través del terminal.

- Se llama a la librería SystemC.
- Imprime en pantalla los títulos del simulador.
- Ya que se utiliza la librería SystemC 2.2 hay algunas funciones obsoletas que se emplean y, al ejecutar Noxim, nos da un warning, por lo que se procedió a desactivar estas advertencias.
- Se imprime los parámetros establecidos para la simulación.
- Para las señales de reloj se crea un sc_module llamado “clock” con un período de un nanosegundo.
- Se crea una señal de reset de tipo booleana.

- Se crea una malla tipo NoximNoC llamado “NoC” de dimX por dimY. (Ver NoximNoC.cpp).
- Se asigna a NoC la señal de reloj y reset creada anteriormente.
- Se crean todas las señales de rastreo que se utilizarán en la simulación las mismas que, posteriormente, serán guardadas en un archivo vcd.

El valor mínimo de la variable free_slots y free_slots_neighbor es 0 y el valor máximo es igual al tamaño de buffer.

Como se puede ver en el simulador, no se crean buses de datos sino que, mediante métodos dentro de las clases NoximFlit y NoximNop, se extrae la información requerida como nodos origen y destino.

Entre sus principales funcionalidades están:

- Generar tráfico de forma aleatoria.
- Inicia la simulación dando como parámetro el tiempo que durará la misma.
- Muestra los resultados por consola.
- Si faltaron ciclos y hay un error al mostrar los resultados, se informa al usuario mediante un mensaje en la consola.

Evaluación del Desempeño del Simulador Base (Noxim)

Se procedió a evaluar el funcionamiento del simulador base modificando los parámetros de entrada que alteran el desempeño de la NoC.

La única topología que presenta el simulador base es la Mesh. La red puede ser de dimensión NxM, siendo N y M valores entre 2 a 9. El tamaño del buffer puede tomar cualquier número natural, el tamaño del paquete en flits puede tomar como mínimo el valor de 3, que está representado por la cabecera, el cuerpo y la cola y, como máximo, cualquier número natural.

Se tienen varios algoritmos de enrutamiento, los cuales son derivaciones del algoritmo de enrutamiento XY como por ejemplo: westfirst, northlast, negativefirst, entre otros.

El tráfico puede ser generado de forma randómica, transpuesta, shuffle, entre otros. Esto se refiere a la selección realizada entre origen y destino del despacho del paquete.

Finalmente, se puede seleccionar desde qué ciclo de reloj se comienza a recolectar los datos, cuántos datos van a ser recolectados y cuántos ciclos de reloj se van a ejecutar.

La medida utilizada por el simulador como tiempo está descrita en ciclos que, dependiendo de la velocidad del reloj, se lo podría cuantificar en segundos. El simulador entrega los resultados en ciclo puesto que son una unidad independiente de la velocidad del reloj.

A continuación se muestra una tabla de resumen de los parámetros ingresados al simulador para evaluar el desempeño del mismo. (Tabla 7)

Tabla. 7. Parámetros ingresados al simulador para evaluar el desempeño

Topología	Dimensión	Buffer	Tamaño Paquete	Tráfico	Algoritmo de Ruteo	Calentamiento	Simulaciones
Mesh	4x4 default	4 default	10 default	Randómico default	xy	1000	100000
Mesh	4x4 default	4 default	10 default	randómico default	westfirst	1000	100000
Mesh	4x4 default	4 default	10 default	randómico default	northlast	1000	100000
Mesh	4x4 default	4 default	10 default	randómico default	negativefirst	1000	100000
Mesh	4x4 default	4 default	10 default	randómico default	oddeven	1000	100000
Mesh	4x4 default	4 default	10 default	randómico default	dyad T	1000	100000
Mesh	4x4 default	4 default	10 default	randómico default	fullyadaptive	1000	100000

Lo primero que se evaluó es la diferencia en términos de latencia que presenta el uso de diferentes tipos de algoritmo de enrutamiento, para así determinar cuál es el que presenta mejor desempeño. Los demás parámetros de entrada se dejaron con sus valores por defecto. (Tabla. 8.)

Tabla. 8. Diferencia en términos de latencia

Algoritmo de Enrutamiento	Latencia Promedio (ciclo)	Troughtput (flits/ciclo)	Energía Consumida (J)
xy	9,96736	0,0597886	0,000739829
fullyadaptive	10,0034	0,0601212	0,000680777
oddeven	10,0304	0,0602578	0,000759469
westfirst	10,0488	0,0592297	0,000735064
northlast	10,1449	0,0600997	0,000746871
negativefirst	10,2086	0,0602534	0,000749439
dyad T	10,3977	0,0599632	0,000760331

Una vez realizadas las simulaciones correspondientes, se procedió a seleccionar 3 algoritmos de enrutamiento, los cuales presentan un mejor desempeño en cuanto a términos de latencia. XY es el algoritmo que presenta un mejor funcionamiento, seguido por el algoritmo fullyadaptive y oddeven.

Seleccionados los 3 algoritmos, se procedió a variar la dimensión de la matriz para observar el comportamiento del simulador. También, como se sabe, el tamaño del buffer puede producir encolamiento lo cual vendría a generar latencia; por lo tanto, se seleccionó un tamaño de buffer arbitrario para la evaluación.

La siguiente tabla muestra los parámetros de entrada ingresados al simulador para evaluar el desempeño de las diferentes dimensiones que presenta (Tabla 9)

Tabla. 9. Parámetros de entrada ingresados al simulador

Topología	Dimensión	Buffer	Tamaño Paquete	Tráfico	Algoritmo de Ruteo
Mesh	3x3 - 6x6	100	10 default	randómico default	Xy
Mesh	3x3 - 6x6	100	10 default	randómico default	fullyadaptive
Mesh	3x3 - 6x6	100	10 default	randómico default	oddeven

A continuación se muestra el promedio de los resultados obtenidos de las 3 dimensiones evaluadas: (Tabla 10)

Tabla. 10. Promedio de los resultados

Algoritmo	Latencia Promedio (ciclo)	Troughtput (flits/ciclo)	Energía Consumida (J)	Dimensión
Xy	9,98218	0,0603193	0,00149121	3 a 6
Fullyadaptive	10,0094	0,0599127	0,00135839	3 a 6
Oddeven	10,0715	0,0599022	0,00150777	3 a 6

Se observa claramente que el algoritmo de enrutamiento XY es el que mejor desempeño presenta en cuanto a términos de latencia.

También se observa que Noxim es un simulador muy versátil a la hora de modificar los parámetros de entrada, la configuración de los parámetros de entrada es sencilla. Noxim, entrega los resultados en forma de promedio, facilita la comprensión, comparación e interpretación de los mismos.

La estructura de Noxim está dividida en diferentes módulos. Cada módulo corresponde a un elemento de la NoC, por ejemplo si se desea cambiar la estructura del Buffer simplemente se dirige al módulo Buffer.cpp y se modifica lo que se desea, de igual manera sus desarrolladores de Noxim están dispuestos a explicar cualquier inquietud que se les pregunte a sus correos personales.

CAPÍTULO 6

SIMULADOR EtNoC BASADO EN NOXIM

Una vez analizado la estructura del código del simulador base Noxim, se realizaron las modificaciones y las pruebas correspondientes para garantizar el correcto funcionamiento del mismo y así generar un simulador nuevo basado en su totalidad en Noxim .

En este capítulo se hablará de las modificaciones generales que se realizaron al simulador base Noxim para poder generar un nuevo simulador.

6.1 MODIFICACIONES

Para cumplir con el objetivo principal del proyecto de tesis. “Implementar un simulador NOC que permita evaluar el desempeño de arquitecturas NoC existentes y de una nueva propuesta”, se tuvo que realizar ciertas modificaciones en el simulador base.

En resumen el simulador base trabaja únicamente con una topología 2D-Mesh. Los parámetros de entrada que se pueden modificar son los siguientes: tamaño de la matriz, tamaño del buffer del conmutador, tamaño del paquete expresado en flits, algoritmo de enrutamiento (xy y sus derivaciones), delimitar un número máximo de simulaciones a realizarse, definir el número máximo de ciclos de reloj a ejecutarse, establecer un número generador de datos aleatorios y la forma de distribución del tráfico.

Para lograr crear dos arquitecturas nuevas de prueba en el simulador base Noxim, dos cambios específicos se realizaron.

Primero cambiar la Topología de la red y el segundo cambiar el algoritmo de enrutamiento para que se adapte a la nueva topología.

6.1.1 Topología de la Red

El primer cambio que se tuvo que realizar, fue la modificación de la topología de la red utilizada por el simulador base; se realizó el cambio de una topología Mesh a una topología Torus, y finalmente de una topología Torus a la nueva topología propuesta la cual la llamaremos TorusS.

Tanto la topología Mesh como Torus se encuentran explicadas en el capítulo 3.

A continuación se explicara cómo está compuesta la topología TorusS.

La topología TorusS (Figura 30) interconecta todos los conmutadores entre sí, es similar a la topología Torus con la diferencia que los conmutadores externos se interconectan diagonalmente a su conmutador opuesto el método de interconexión realizado se encuentra explicado en la siguiente sección.

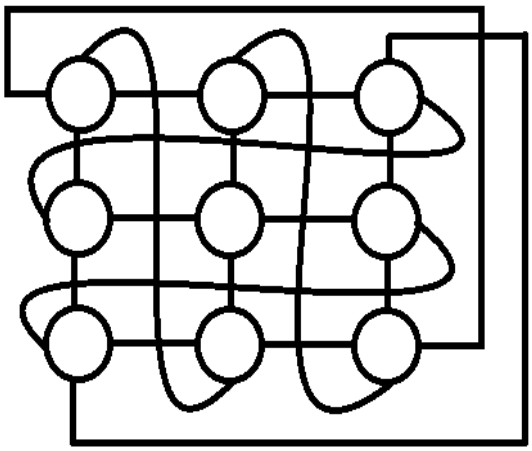


Figura. 27. Topología TorusS

Para realizar el cambio en el simulador base, se tuvo que modificar el módulo correspondiente a la topología el cual se encuentra determinado en el archivo fuente NoximNoC.cpp, se efectuaron las modificaciones necesarias para producir las interconexiones tanto para la topología Torus y TorusS.

En resumen, Noxim para la conexión entre conmutadores utiliza las 4 direcciones determinadas en el capítulo 3, norte, sur, este y oeste.

Por ejemplo en la Figura 31 si A desea comunicarse con B, al norte de A se lo debe conectar al sur de B, esto se lo debe realizar siempre bidireccionalmente, al sur de B se lo debe conectar con el norte de A.

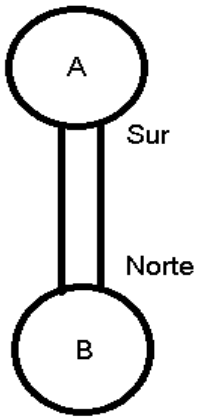


Figura. 28. Conexión entre dos conmutadores

. TorusS

En el siguiente ejemplo (Figura 32) se procederá a explicar cómo se realizó la interconexión de la topología TorusS de dimensiones 2x2, puesto que la misma interconexión se aplica en una red de NxN. Se omitirá la explicación de la interconexión de la topología Torus puesto que se la realiza de similar manera.

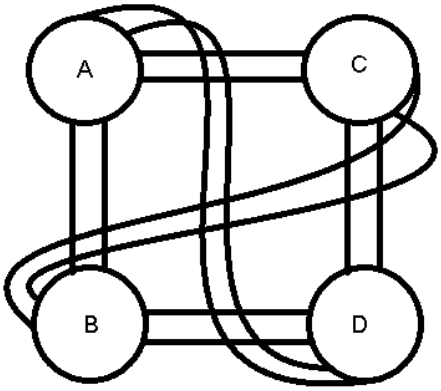


Figura. 29. Conexiones Topología TorusS

Para la conexión interna entre conmutadores se mantuvo la establecida en una topología Mesh. Para la conexión externa se determinó que el norte de A, se conecte al sur de D, el este de B, se conecte al oeste de C, el norte de C se conecte al sur de B y finalmente el este del A, se conecte al oeste de D.

6.1.2 Algoritmo de Enrutamiento

Una vez realizadas las modificaciones en cuanto a la topología se tuvo que adaptar el algoritmo de enrutamiento para que así este determine cuál es el mejor camino a tomar.

Las modificaciones se las realizaron en el módulo NoximRouter.cpp el cual posee los algoritmos de enrutamiento.

Debido a que las NoC utilizan algoritmos de enrutamiento de dimensión variable (ver capítulo 3), la mejor ruta es calculada con la diferencia entre el destino y el origen tanto verticalmente como horizontalmente.

. Algoritmo de Enrutamiento Torus

La arquitectura Torus (Figura 33) puede ser probada con dos algoritmos de ruteo, el algoritmo de ruteo de borde, el cual especifica que si su destino se encuentra en el lado opuesto del origen el paquete será enviado por los bordes (Figura 34) y el algoritmo de ruteo completo, el cual primero calcula la mejor ruta entre el destino y el origen para luego proceder al despacho del paquete (Figura 30)

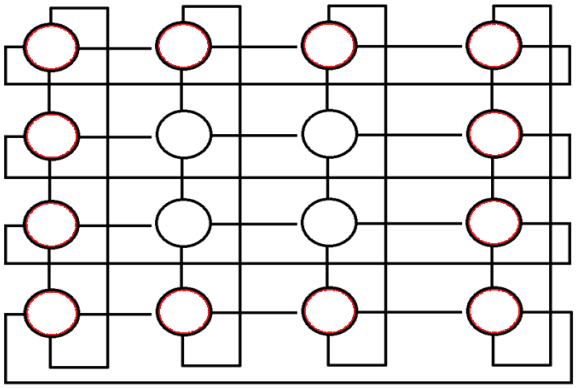


Figura. 30. Red Torus

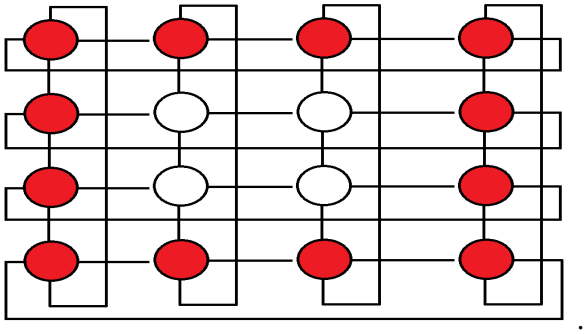


Figura. 31. Topología Torus con algoritmo de ruteo borde

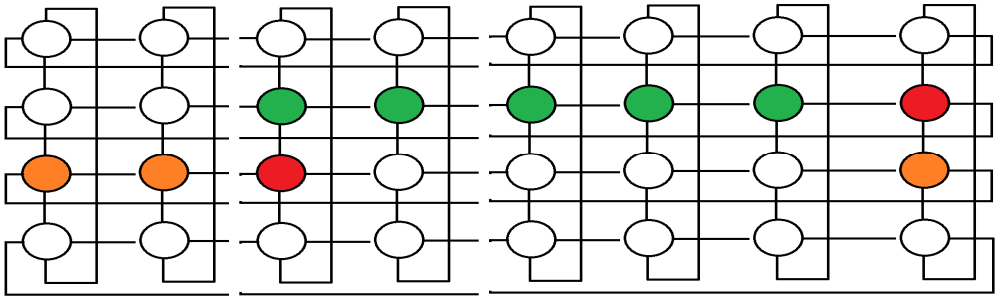


Figura. 32. Topología Torus con algoritmo de ruteo completo

En la topología TorusS solamente se tiene un algoritmo de ruteo completo, el cual determina la mejor ruta del origen al destino, para luego despachar los paquetes

En el siguiente ejemplo (Figura 33) se procederá a explicar cómo se determina la mejor ruta en la topología TorusS de dimensiones 2x2, puesto que la misma determinación de la mejor ruta se aplica en una red de NxN. Se omitirá la explicación del algoritmo de enrutamiento de la topología Torus puesto a que se lo realiza de similar manera.

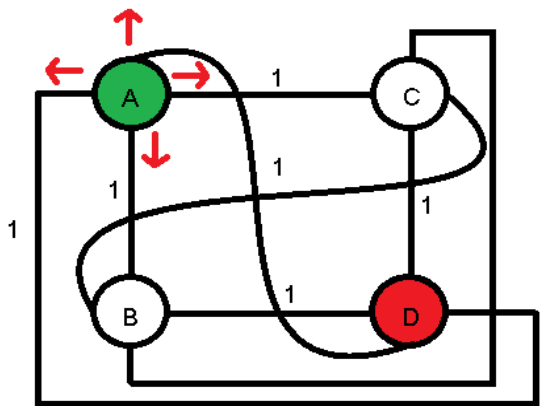


Figura. 33. Algoritmo de Ruteo TorusS

Si un paquete de A, desea llegar a D; se calculan todas las posibles rutas que este puede tomar en sus cuatro direcciones (norte, sur, este y oeste), para luego determinar que ruta es la menor, para posteriormente tomarse.

De A hasta D tanto por el norte y este, se calcula que si da un salto por esas direcciones faltaría para llegar al destino cero saltos. En cambio por las direcciones del oeste y el sur se calcula que si da un salto faltaría para el destino 1 salto. Por lo tanto se selecciona despachar el paquete por el norte o este. El código empleado se encuentra en al Anexo 2.

6.1.3 Interfaz Gráfica

Adicionalmente como Noxim se ejecuta en línea de comandos (CLI), se creó una interfaz gráfica. En la cual se seleccionan los parámetros de entrada deseados (Figura 37) y se adicionó que la distribución del tráfico no sea solamente randómico, sino que el usuario pueda determinar tanto el origen como destino del paquete (Figura 38).

Todos los resultados entregados por Noxim son visualizados en el terminal, también se adicionó que los resultados entregados por Noxim sean exportados a un archivo tipo Excel (Figura 34).

La Figura 34 es la primera pantalla que se muestra en el simulador. El primer paso es ingresar la dimensión de la matriz, una vez ingresada la dimensión de la matriz, si se desea seleccionar la distribución del tráfico (desde que origen a que destino), para esto simplemente se debe seleccionar el cuadro origen y el cuadro destino en la matriz desplegada, luego seleccionar Generar Tráfico.

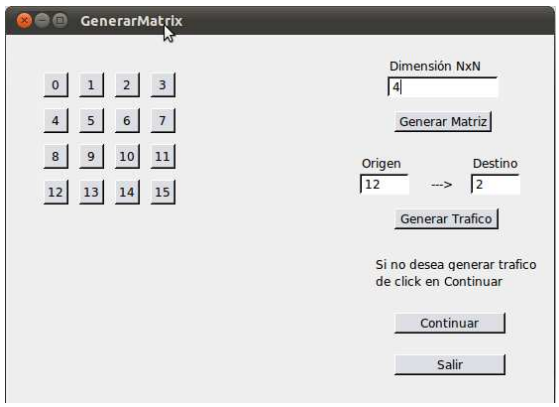


Figura. 34. Interfaz Gráfica EtNoC

Una vez aplastado el botón continuar, si se generó tráfico, el programa emite un archivo de texto que, posteriormente, es leído internamente por el simulador, mismo que dirige los paquetes desde el origen seleccionado hasta el destino.

En el siguiente formulario se puede modificar los parámetros de entrada como el buffer, numero de flits, el tamaño de cada paquete.

Para evaluar las diferentes tipos de arquitecturas uno simplemente tiene que aplastar el botón correspondiente a la arquitectura que se desea simular.

Los resultados son desplegados en un cuadro de texto, los valores de los resultados que nos entrega son exportados a varios archivos de texto de acuerdo a la arquitectura, cada simulación realizada es anexada a este archivo, el cual posteriormente puede ser desplegado en LibreOffice (Excel)

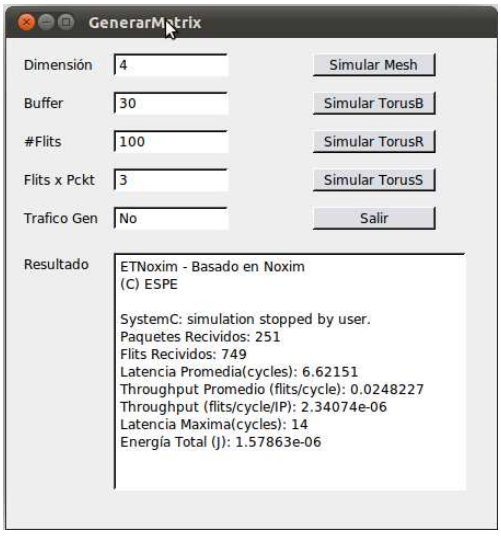
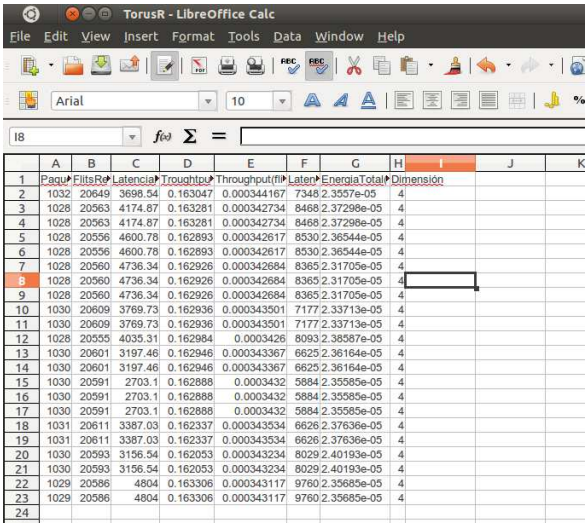


Figura. 35. Parámetros de entrada y despliegue de Resultados

Los resultados son desplegados en un cuadro de texto, los valores de los resultados que nos entrega son exportados a varios archivos de texto de acuerdo a la arquitectura, cada simulación realizada es anexada a este archivo, el cual posteriormente puede ser desplegado en LibreOffice (Figura 39). El manual de usuario para la instalación y ejecución del simulador se encuentran descritos en el Anexo 3



	A	B	C	D	E	F	G	H	I	J	K
1	Pagu	FlitsRe	Latencia	Troughtpu	Throughput(fil	Laten	EnergiaTotal	Dimensión			
2	1032	20649	3698.54	0.163047	0.000344167	7348	2.3557e-05	4			
3	1028	20563	4174.87	0.163281	0.000342734	8468	2.37298e-05	4			
4	1028	20563	4174.87	0.163281	0.000342734	8468	2.37298e-05	4			
5	1028	20556	4600.78	0.162893	0.000342617	8530	2.36544e-05	4			
6	1028	20556	4600.78	0.162893	0.000342617	8530	2.36544e-05	4			
7	1028	20560	4736.34	0.162926	0.000342684	8365	2.31705e-05	4			
8	1028	20560	4736.34	0.162926	0.000342684	8365	2.31705e-05	4			
9	1028	20560	4736.34	0.162926	0.000342684	8365	2.31705e-05	4			
10	1030	20609	3769.73	0.162936	0.000343501	7177	2.33713e-05	4			
11	1030	20609	3769.73	0.162936	0.000343501	7177	2.33713e-05	4			
12	1028	20555	4035.31	0.162984	0.0003426	8093	2.38587e-05	4			
13	1030	20601	3197.46	0.162946	0.000343367	6625	2.36164e-05	4			
14	1030	20601	3197.46	0.162946	0.000343367	6625	2.36164e-05	4			
15	1030	20591	2703.1	0.162888	0.0003432	5884	2.35585e-05	4			
16	1030	20591	2703.1	0.162888	0.0003432	5884	2.35585e-05	4			
17	1030	20591	2703.1	0.162888	0.0003432	5884	2.35585e-05	4			
18	1031	20611	3387.03	0.162337	0.000343534	6626	2.37636e-05	4			
19	1031	20611	3387.03	0.162337	0.000343534	6626	2.37636e-05	4			
20	1030	20593	3156.54	0.162053	0.000343234	8029	2.40193e-05	4			
21	1030	20593	3156.54	0.162053	0.000343234	8029	2.40193e-05	4			
22	1029	20586	4804	0.163306	0.000343117	9760	2.35685e-05	4			
23	1029	20586	4804	0.163306	0.000343117	9760	2.35685e-05	4			
24											

Figura. 36. Resultados entregados por EtNoC

CAPÍTULO 7

ANÁLISIS DE RESULTADOS

La finalidad del proyecto de tesis es comparar el rendimiento de dos arquitecturas NoC existentes (Mesh,Torus) y de la nueva propuesta (TorusS) en cuanto a latencia, troughtput y energía consumida, para lo cual es necesario plantear el mejor escenario de simulación, a fin de que los resultados obtenidos puedan ser comparables.

7.1 OBTENCIÓN DE PARÁMETROS DEL ESCENARIO DE SIMULACIÓN

Lo primero que debe considerarse es el número de muestras (paquetes despachados) que se ejecutarán en la simulación; hay que tomar en cuenta que el simulador entrega como resultado el promedio de todos los paquetes despachados.

Para calcular el número de combinaciones sin repetir de cada matriz se incluye el número de elementos (m) y el número de elementos a combinar (n). El simulador base permite realizar matrices como mínimo de 2x2 y como máximo de 9x9; además, los elementos a combinar son dos: el nodo de destino y el nodo de origen.

$$C_m^n = \frac{m!}{n!\,(m-n)!}$$

Con los datos de los números de elementos y número de elementos a combinar, se muestra en la siguiente Tabla 8 el número de muestras que serán necesarias, de acuerdo al tamaño de la matriz, para obtener todas las combinaciones posibles sin repetirlas.

Tabla. 11. Número de Combinaciones Sin Repetir

Dimensión de la Matriz	Elementos a Combinar	Número de Elementos	Combinaciones sin repetir
2	2	4	6
3	2	9	36
4	2	16	120
5	2	25	300
6	2	36	630
7	2	49	1176
8	2	64	2016
9	2	81	3240

De acuerdo a la ley de probabilidad de Laplace que enuncia "La probabilidad de un suceso elemental es igual al cociente entre el número de casos favorables a ese suceso y el número de casos posibles" se tendría como resultado (Tabla 9) la probabilidad que existe de obtener una de las combinaciones sin repetir en cada matriz.

$$P(A) = \frac{número\ de\ casos\ favorables\ a\ A}{número\ de\ casos\ posibles}$$

Tabla. 12. Probabilidad que existe en cada una de las matrices

Combinaciones sin repetir	Probabilidad (%)
6	16,66666667
36	2,777777778
120	0,833333333
300	0,333333333
630	0,158730159
1176	0,085034014
2016	0,049603175
3240	0,030864198

El número de paquetes a ser despachados depende directamente del tamaño de la matriz. Por ejemplo, en una matriz de 2x2 solamente son necesarias 6 combinaciones sin repetir para obtener el promedio de todos los paquetes a despacharse; en cambio, en una matriz de 9x9 son necesarias 3240 combinaciones sin repetir para obtener todos los paquetes a despacharse.

Por lo tanto, no es lo mismo despachar 100 paquetes y promediarlos en una matriz de 2x2 que en una de 9x9, ya que 100 paquetes en una matriz de 9x9 equivalen al 3% de las posibles combinaciones sin repetir ninguna existente, lo cual llevaría a generar un error en el cálculo del promedio de los datos.

Existe la posibilidad de ingresar manualmente todas las combinaciones sin repetir pero, a partir de una matriz con dimensión 3x3, se convertiría en una actividad tediosa. Por este motivo, es de suma importancia calcular el número de paquetes randómicos a ser despachados.

El número de muestras a realizar en una simulación randómica debe contemplar la posibilidad de que existan dos nodos iguales (AA,BB…) y el orden a tomarse de los nodos (AB o BA). Así, el número de muestras se lo debe calcular con el número de combinaciones con repetición y tomando en cuenta el orden (Tabla 10).

$$C_m^n = \frac{n!}{(m-n)!}$$

Tabla. 13. Número de Combinaciones repetidas y con orden

Dimensión De la Matriz	Elementos a Combinar	Número de Elementos	Numero de Muestras
2	2	4	16
3	2	9	81
4	2	16	256
5	2	25	625
6	2	36	1296
7	2	49	2401
8	2	64	4096
9	2	81	6561

Se procederán a realizar varias simulaciones (Figura 40), donde el valor de los paquetes despachados será entre 10 hasta 17000 en una matriz de 9x9 para demostración de la teoría anteriormente explicada.

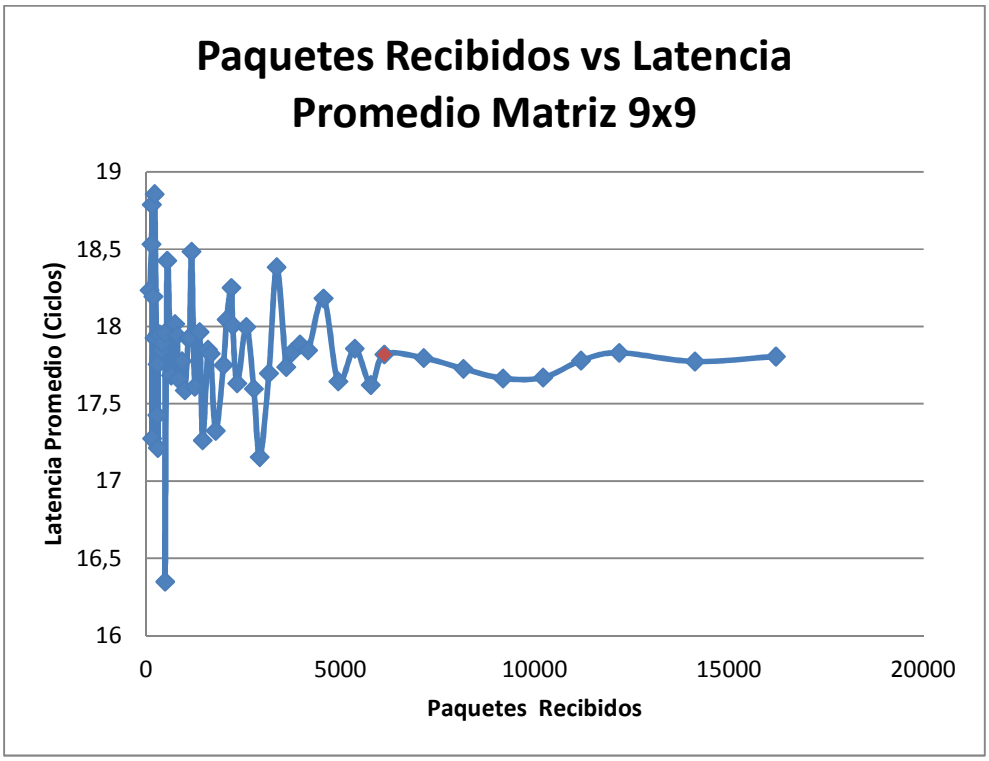


Figura. 37. Latencia Promedio de N paquetes despachados

Como se observa, en una matriz de 9x9 se alcanza estabilidad aproximadamente a partir de los 6128 (punto rojo) paquetes despachados; esto se debe a que la probabilidad de obtener todas las combinaciones sin repetir aumenta. En ese sentido, para la obtención de datos de las tres diferentes arquitecturas y para cualquier dimensión de la matriz, se optará por realizar el despacho como mínimo de 6000 paquetes en cada simulación para que el margen de error sea menor.

-volume 30000 –size 5 5

Comando para garantizar el despacho de aproximadamente 6000 paquetes

Se determinó si el margen de error que existe entre el resultado randómico y el resultado teórico esperado es aceptable; posteriormente se realizó la comparación con matrices de 2x2 y 3x3 debido a que su valor teórico puede ser calculado con el número de combinaciones sin repetir para lo que se debe comprender cómo el simulador calcula la latencia (Figura 41).

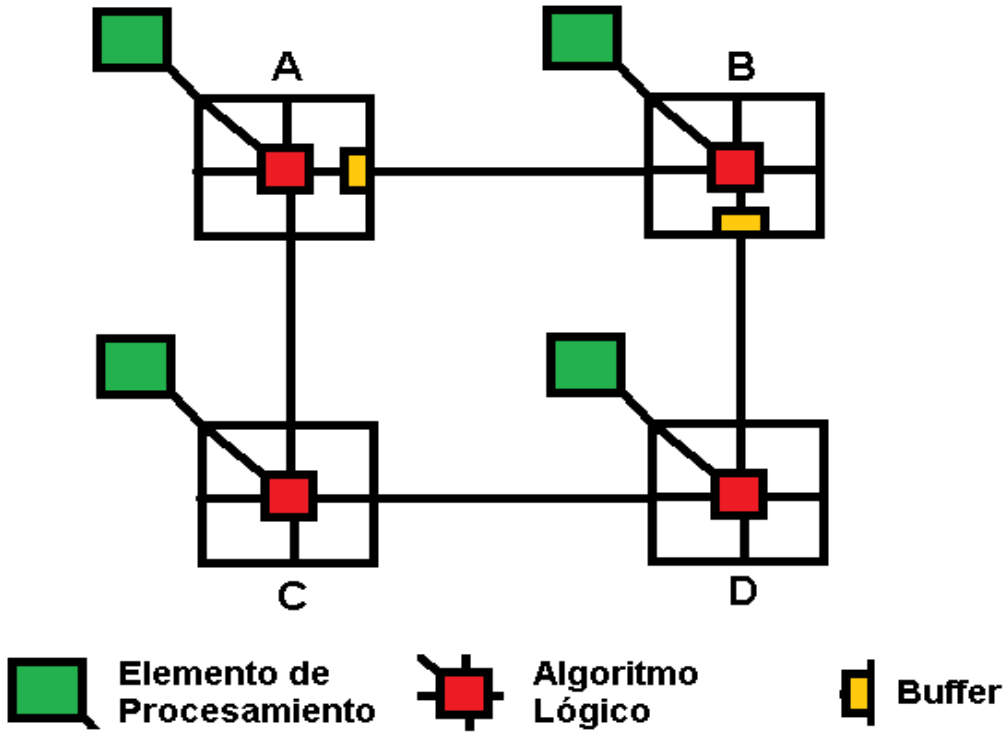


Figura. 38. Estructura de la NoC para el despacho de paquetes

Tomar en cuenta que la latencia es el número de saltos (ciclos) que un flit tarda en llegar desde su origen hasta el destino. Se demostrará cómo se calcula el número de saltos desde A hasta B (Tabla 11) y desde A hasta D (Tabla 12).

Tabla. 14. Número de saltos desde origen(O) destino(D) desde nodo A-B

	A			B		Saltos
PE	AL	Buffer	PE	AL	Buffer	Ciclos
O1	D1					1
	O2	D2				1
		O3		D3		1
			D4	O4		1

Tabla. 15. Número de saltos desde origen(O) destino(D), desde nodo A-D

	A			B			D		Saltos
PE	AL	Buffer	PE	AL	Buffer	PE	AL	Buffer	Ciclos
O1	D1								1
	O2	D2							1
		O3		D3					1
				O4	D4				1
					O5		D5		1
						D6	O6		1

Desde el elemento de procesamiento (PE) hasta el Algoritmo Lógico (AL) del Router A se realiza un salto; luego, desde el AL hasta el buffer o dirección a despacharse el paquete del Router A otro salto; desde el buffer del Router A hasta el AL del Router B otro; y finalmente, desde el AL hasta el PE del Router B un último salto, por lo tanto, para que un paquete llegue desde A hasta B se demorará 4 saltos. De igual forma, los paquetes son despachados desde A hasta D dando como resultado 6 saltos.

Para obtener el valor teórico se debe realizar el promedio del número de saltos de todas las combinaciones sin repetir (Tabla 13).

Tabla. 16. Promedio del número de saltos de las combinaciones sin repetir

Combinaciones Sin Repetir	Saltos
AB	4
AC	4
BD	4
CD	4
BC	6
AD	6
Promedio Latencia	**4,66666667**

Para una matriz de 2x2 el promedio de la latencia a esperarse es de 4,6666 y, para una de 3x3, el promedio es de 6.

Se calculó el margen de error con respecto al valor teórico esperado (línea azul) luego de promediar varias simulaciones tanto para una matriz de 2x2 (Figura 42) y 3x3 (Figura 43).

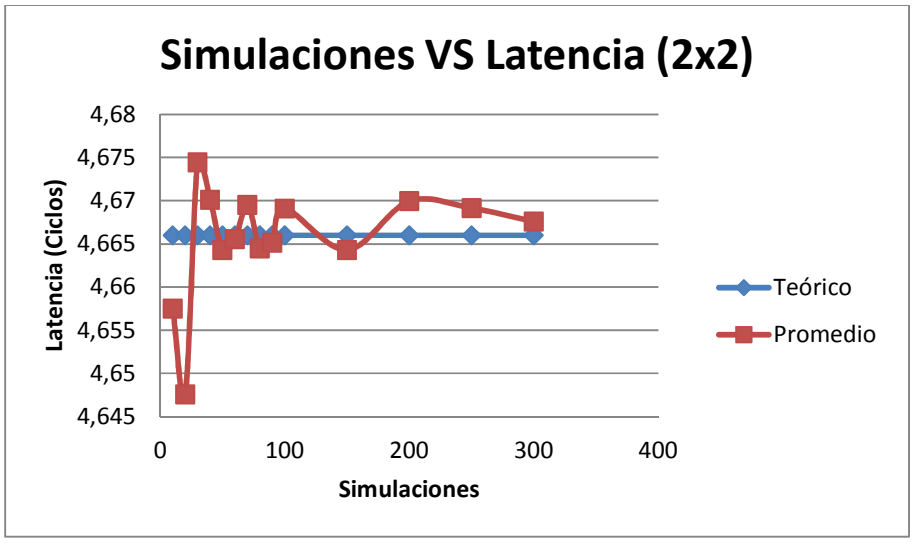


Figura. 39. Resultados randómicos vs resultado teórico en una matriz 2x2

Para una matriz de 2x2 el punto más alejado al valor teórico luego de promediar 40 simulaciones como mínimo fue de 4,66952, el cual representa un margen de error del 0,0754%

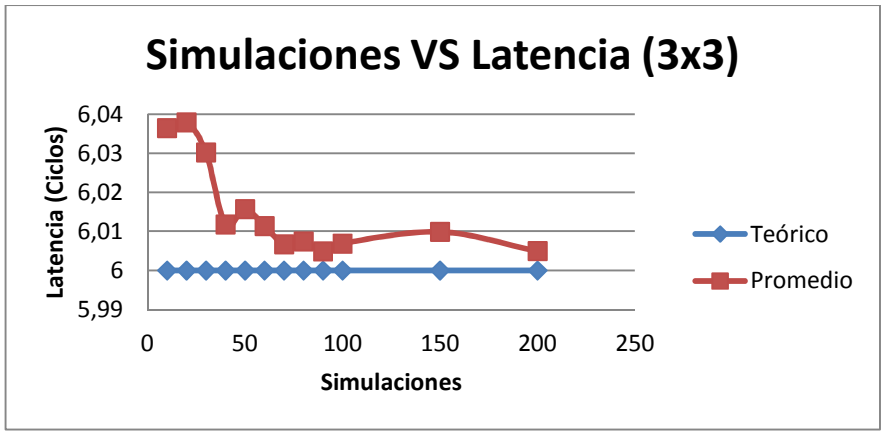


Figura. 40. Resultados randómicos vs resultado teórico en una matriz 3x3

Para una matriz de 3x3 el mayor margen de error que existe, en comparación al valor teórico, es del 0,25%.

Podemos decir que son márgenes de error bajos en comparación al valor teórico, por lo que el número calculado para paquetes a despacharse anteriormente es el adecuado y el número de simulaciones como mínimo a promediarse será de 40.

El siguiente análisis que se ejecutó fue determinar si la simulación a realizarse tenía que ser randómica o si ésta tenía que ser controlada (Figura 44) para poder garantizar la comparación adecuada y correcta entre las diferentes arquitecturas de NoC desarrolladas.

Para la simulación randómica se determinó el tamaño de la matriz, el número de flits a recibirse y el tamaño del paquete en flits.

./noxim -dimx 3 -dimy 3 -volume 30000 -sim 200000 -size 5 5

Para la simulación controlada se definió el tamaño de la matriz, el número de flits a recibirse, el tamaño del paquete en flits y la semilla (seed), que es la encargada de generar de qué origen a qué destino se enviará el paquete. Esto quiere decir que en toda simulación que realicemos con esta semilla, las combinaciones a generarse entre origen y destino serán las mismas.

./noxim -dimx 3 -dimy 3 -volume 30000 -sim 200000 -size 5 5 -seed 20

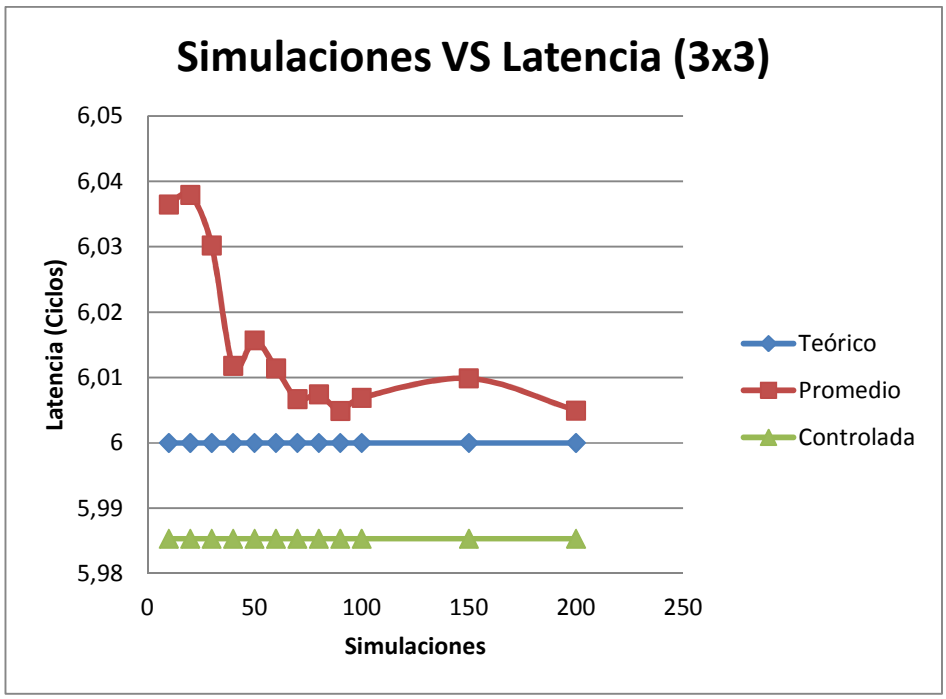


Figura. 41. Resultados randómicos vs resultado teórico vs controlado en una matriz 3x3

La simulación randómica genera resultados aleatorios que, en el momento de realizar una comparación entre las arquitecturas NoC desarrolladas, podrían producir un margen de error menor al 1% y la comparación realizada no sería idéntica.

En cambio, la simulación controlada nos entrega un resultado estable que nos garantizará una comparación pareja entre las arquitecturas desarrolladas; esto se debe a que cuando nosotros colocamos el comando –seed garantizamos que los paquetes vayan de los mismos orígenes a los mismos destinos siempre, en todas las arquitecturas.

7.2 BUFFER

Como conocemos, el tamaño del buffer puede generar encolamiento de los paquetes y producirá latencia al sistema. Para que esto no afecte a los resultados obtenidos de las diferentes arquitecturas, se determinó el tamaño del buffer adecuado para que el encolamiento sea descartado. Se realizó una simulación randómica variando el tamaño del buffer con pasos incrementales de dos (Figura 45).

./noxim -dimx 9 -dimy 9 -volume 25000 -sim 200000 -size 5 5 -buffer N –seed 10

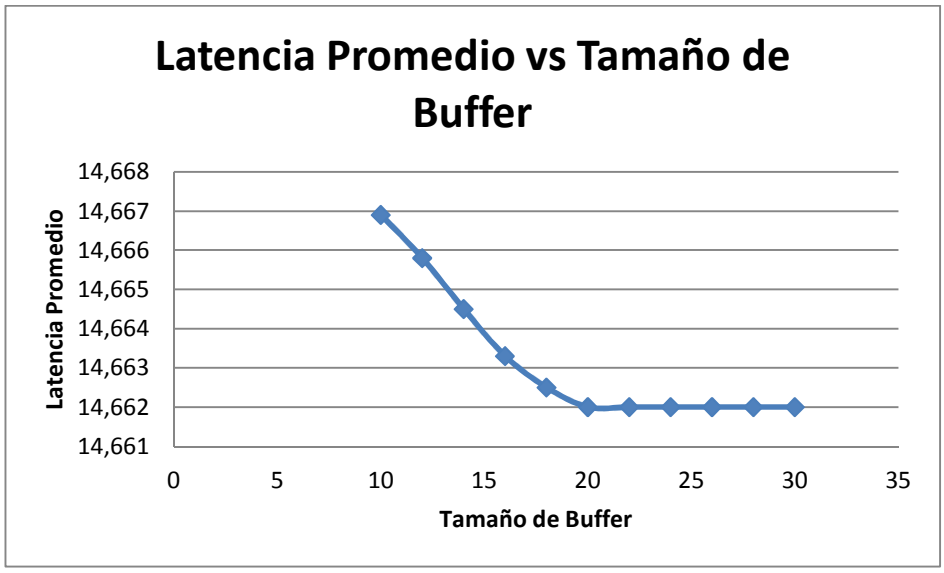


Figura. 42. Latencia promedio al incrementar el tamaño del buffer

Como era de esperarse, en la Figura 45, a medida que el tamaño del buffer incrementa, el retraso en el sistema llega a ser el mismo debido a que no existe el encolamiento que se genera cuando el buffer se llena. Se ve que a partir de un tamaño de buffer mayor a 20 el resultado es el mismo, por lo tanto, para poder comparar las arquitecturas, se optará por utilizar un tamaño de buffer mayor a 20 flits.

7.2.1 Algoritmo de Ruteo

Puesto que los algoritmos de ruteo generan un tiempo de procesamiento en el sistema dependiendo la complejidad del mismo, el siguiente análisis determinará si el simulador base incluye este tiempo de procesamiento en los resultados que nos entrega para que la comparación a realizarse entre las arquitecturas desarrolladas sea lo más justa posible.

El simulador Noxim permite utilizar una gran cantidad de algoritmos de ruteo como el XY, oddeven, fullyadaptive, westfirst, entre otros; de ahí que para determinar si existe el retraso generado por el algoritmo de ruteo, se procedió a realizar una simulación

controlada determinando un tamaño del buffer óptimo para que no exista el retraso generado por éste.(Tabla 14)

./noxim -dimx 9 -dimy 9 -seed 10 -sim 25000 -routing xy -size 5 5 -buffer 20

Tabla. 17. Retraso Promedio

Ruteo	Paquetes	Flits	Retraso Promedio	Troughput Promedio
oddeven	5000	24999	24,9825	0,0212443
fullyadaptive	5000	25000	25,0435	0,0212773
xy	5000	24999	24,4585	0,0209352

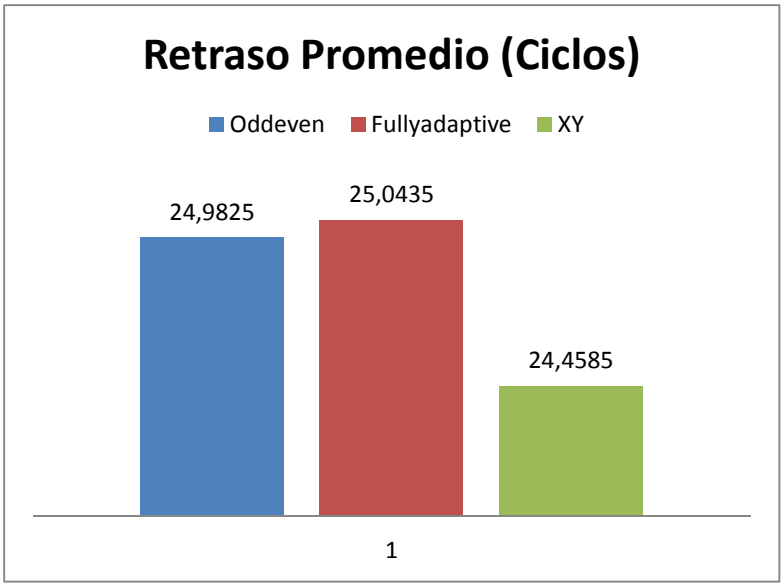


Figura. 43. Latencia promedio de diferentes algoritmos de ruteo

La complejidad del algoritmo utilizado se refleja en un aumento de tiempo, mismo que genera un retraso en el sistema. El algoritmo XY, al ser el más sencillo, genera un

retraso menor a la hora de despachar paquetes, por lo tanto, este algoritmo va a ser utilizado en la red Mesh para comparar con las otras arquitecturas.(Figura 46)

Una vez determinado el tipo de simulación, tamaño del buffer, número de paquetes a ser despachados, número de simulaciones a realizarse y algoritmo de enrutamiento para la red Mesh, se establecieron los parámetros para realizar la comparación entre las arquitecturas que se muestra en la siguiente Tabla 15:

Tabla. 18. Parámetros a Evaluar

Topología	Mesh	Torus	Torus	TorusS
Dimensión Matriz	2 a 9	2 a 9	2 a 9	2 a 9
Numero de Simulaciones	40	40	40	40
Número de Paquetes	+6000	+6000	+6000	+6000
Numero de Flits x Paquete	5	5	5	5
Tamaño del Buffer	+20	+20	+20	+20
Algoritmo de ruteo	XY	Borde	Completo	Completo
Simulación	Randómica			

7.3 ANÁLISIS DE RESULTADOS

7.3.1 Latencia

El primer parámetro a analizar entre las arquitecturas es la latencia (Figura 47).

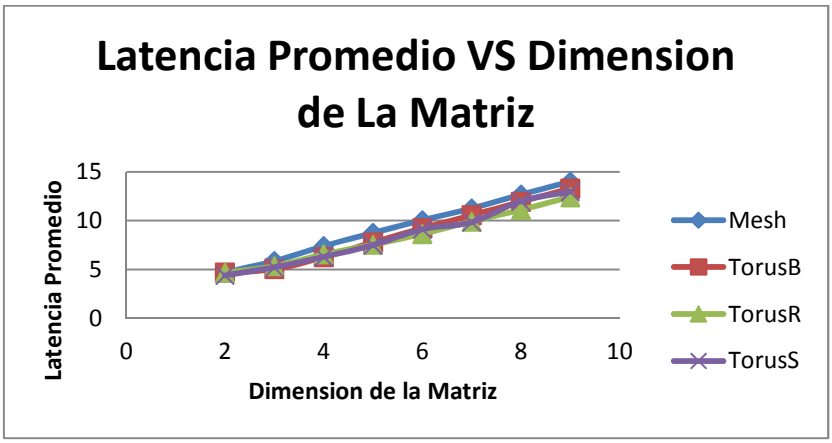


Figura. 44. Latencia promedio de las diferentes arquitecturas

Se observa que tanto las arquitecturas TorusB, TorusR y TorusS son superiores a la arquitectura Mesh en términos de latencia; esto se debe a la interconexión que disponen estas topologías al mejorar la velocidad en el despacho de paquetes, ya que no existe la limitante de la conexión entre los extremos como en una red Mesh. Debido a esto, los paquetes son despachados por los caminos más cercanos lo cual se refleja en términos de latencia.

Para dimensiones de 6x6 hasta 9x9 se puede determinar que la red TorusR es ligeramente superior a la arquitectura TorusS y superior a las demás arquitecturas. Sin embargo, se observan dos comportamientos interesantes que deben ser analizados a detalle. Primero, debe compararse y analizarse individualmente las arquitecturas desde la dimensión 2x2 hasta 5x5; segundo, se observa que a partir de la dimensión 6x6 la arquitectura TorusS no es estable, el motivo se lo explicará de mejor manera cuando analicemos la energía consumida de las arquitecturas:

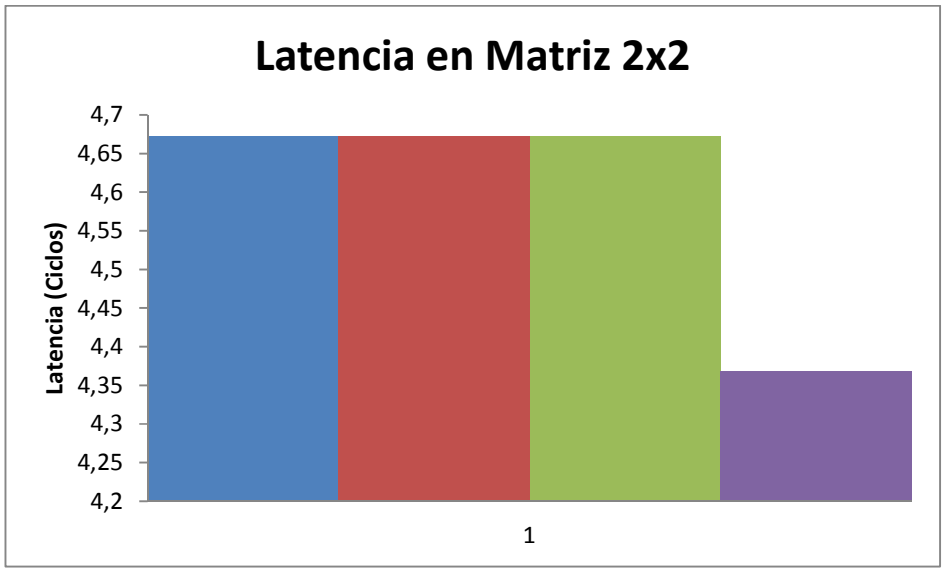


Figura. 45. Latencia promedio de las diferentes arquitecturas Matriz 2x2

Claramente, existe superioridad de la arquitectura TorusS con respecto a las demás arquitecturas; esto se debe a que en la arquitectura TorusS, con una matriz de dimensión 2x2, siempre existen 4 saltos entre origen y destino ya que todos los nodos están interconectados entre sí (Figura 48).

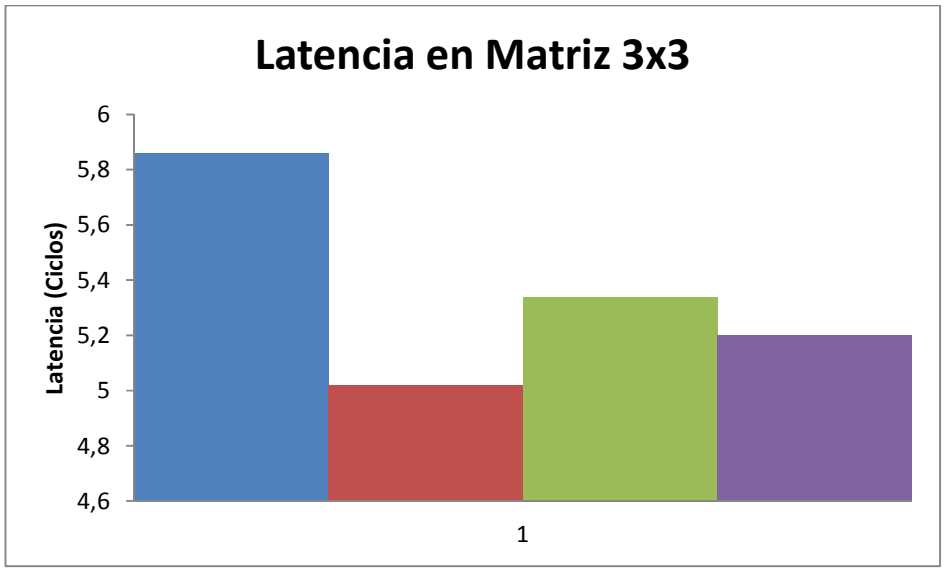


Figura. 46. Latencia promedio de las diferentes arquitecturas Matriz 3x3

En una red con dimensión 3x3 se observa que la topología Torus, con el algoritmo de ruteo de borde (TorusB), es superior a las demás arquitecturas. En esta dimensión, la mayoría de los paquetes serán dirigidos de los extremos y el algoritmo de ruteo utilizado en la arquitectura TorusB prioriza el despacho de los paquetes que se encuentran en sus extremos, a diferencia de la arquitectura TorusS y la arquitectura Torus con un algoritmo de ruteo completo (TorusR) que calculan la mejor ruta antes de despachar los paquetes (Figura 49).

Por lo tanto, a pesar que la topología de las dos arquitecturas es la misma (TorusB y TorusR), el algoritmo utilizado genera una ganancia en cuanto a la velocidad del despacho de los mismos.

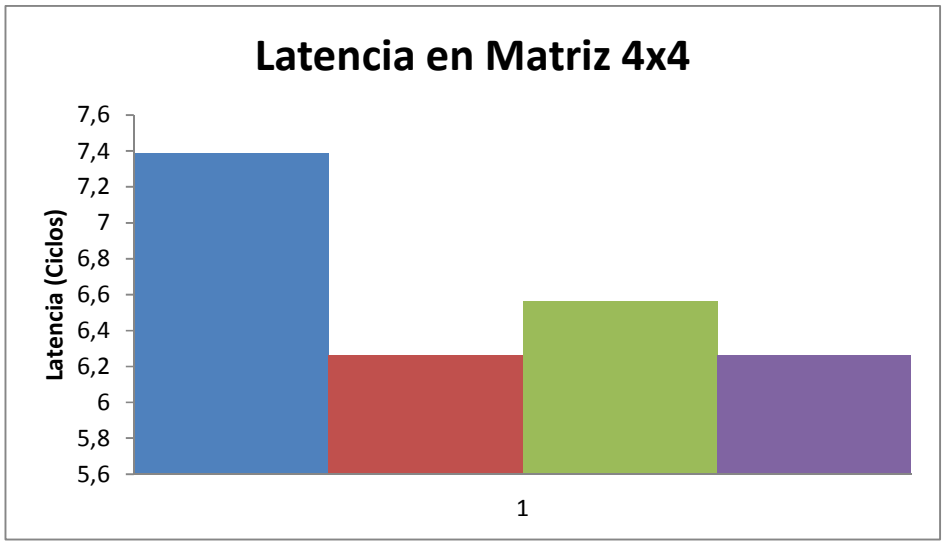


Figura. 47. Latencia promedio de las diferentes arquitecturas Matriz 4x4

Cuando se tiene una dimensión de 4x4, la mayoría de los paquetes son despachados desde los extremos, de ahí que se observa una ventaja de la arquitectura TorusB hacia las demás.

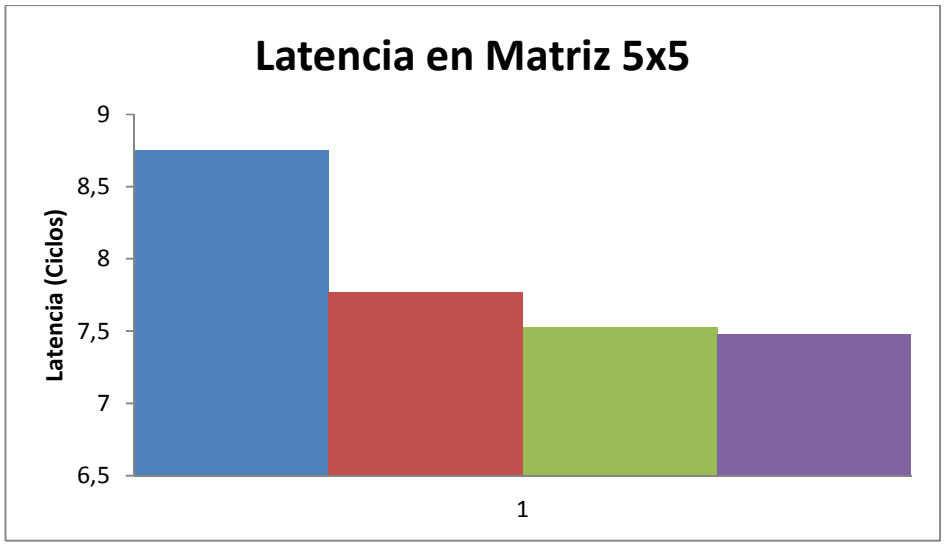


Figura. 48. Latencia promedio de las diferentes arquitecturas Matriz 5x5

Finalmente, a partir de una matriz de dimensiones 5x5, se observa que tanto una arquitectura TorusR y TorusS son superiores, puesto que éstas determinan el mejor camino para realizar el despacho de los paquetes, a diferencia de la TorusB (que solamente prioriza si el paquete destino se encuentra en un extremo) y de la Mesh (la cual no posee interconexión entre los extremos).

La dimensión de la matriz tiene un papel importante para determinar qué arquitectura es la mejor. Se determina que en una dimensión 2x2, la arquitectura TorusS es superior a las demás por la interconexión que presenta su topología. Cuando la dimensión sube a 3x3 y 4x4, la arquitectura que presenta una mejor respuesta es la TorusB; a pesar de que TorusR comparta la misma topología, no muestran un comportamiento idéntico ya que

quien determina la velocidad del despacho de los paquetes es el algoritmo de ruteo, por lo tanto, a una dimensión 3x3y 4x4 la TorusB es superior a las demás arquitecturas.

7.3.2 Troughtput

El siguiente análisis que se realiza en las arquitecturas es del troughtput. Para esto, en primera instancia, se debe saber que el troughtput es la velocidad con que los conmutadores procesan los datos internamente y eso se lo mide en datos por segundo, lo que en el simulador equivale a flits/ciclos (Figura 49).

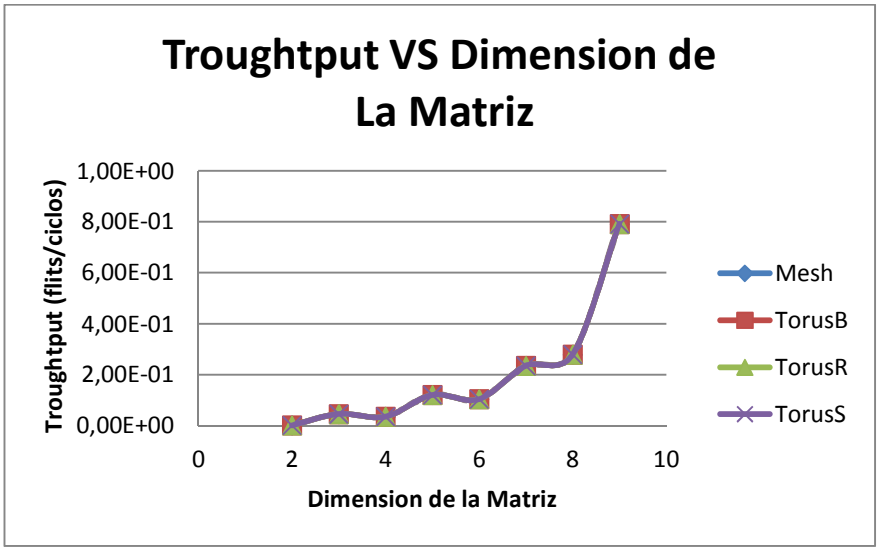


Figura. 49. Troughtput promedio de las diferentes arquitecturas

Debido a que todas las arquitecturas están desarrolladas en un mismo sistema, o en otras palabras todos los elementos que conforman a la NoC son los mismos, era de

esperarse una respuesta igual en todas las arquitecturas en términos de troughtput en tanto ningún elemento fue modificado. Se puede observar que a medida que crece la matriz, el troughtput incrementa. Esto se debe a que mientras el tamaño de la matriz sea mayor, mayor será el número de paquetes a despachar y, por ende, mayor el número de flits; el troughtput es calculado de acuerdo al número de flits que pasan por los elementos.

7.3.3 Energía Consumida

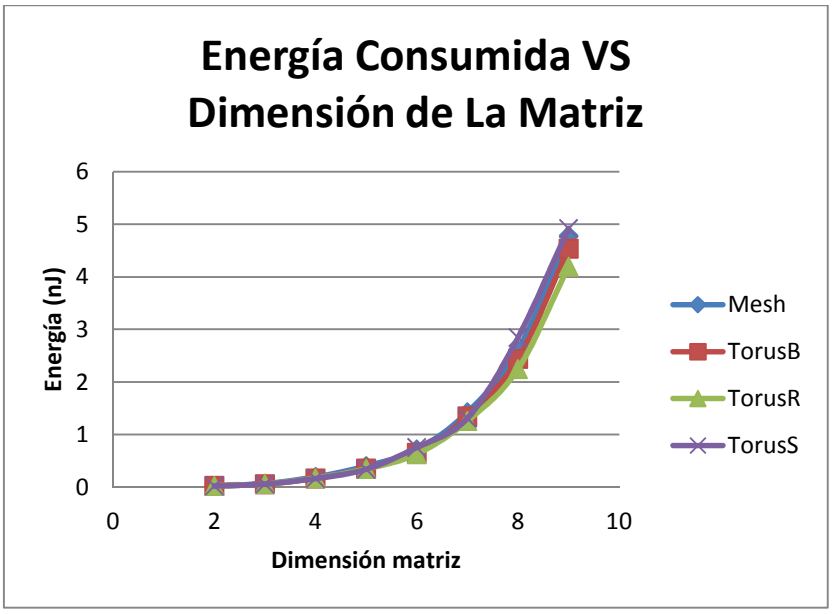


Figura. 50. Energía Consumida de las diferentes arquitecturas

Como podemos observar (Figura 53), a partir de una dimensión 6x6, la topología TorusS presenta un alto consumo de energía que se refleja en la variación observada en la Latencia.

Para explicar por qué sucede esto, primeramente se debe entender cómo el simulador calcula la energía consumida.

Cada salto que da un flit equivale a 0.384e-9 J, este dato fue calculado en una NoC de 2mmx2mm con una capacitancia de 0.50fF por micrón (NOXIM, www.Noxim.org, 2013) En el simulador, cada salto que un flit dé se suma al resultado de la energía total consumida.

Por lo tanto, el número de saltos de un paquete afecta a la energía total consumida.

El simulador entrega un dato interesante, motivo por el cual sucede este comportamiento a partir de una matriz de 6x6, que es la latencia máxima. Ésta representa el retardo más grande en el despacho de un paquete en el sistema; por ejemplo, en una red Mesh de 2x2, la latencia máxima que va desde A hasta D es de 6 saltos (ciclos).

La siguiente Tabla muestra la latencia máxima que se produjo durante la simulación de las tres arquitecturas para matrices de 2x2 hasta 9x9 (Tabla 16).

Tabla. 19. Latencia Máxima de las diferentes arquitecturas

	Latencia Máxima			
Dimensión	**Mesh**	**TorusB**	**TorusR**	**TorusS**
2	6	6	6	6
3	10	6	8	8
4	14	10	12	10
5	18	14	20	14
6	22	18	28	30
7	26	22	30	30
8	30	26	43	44
9	34	30	40	46

Definimos que a partir de una matriz de 6x6, en la arquitectura TorusR pero más notorio en la arquitectura TorusS, se tienen latencias máximas muy altas que se reflejan directamente en el consumo de energía.

Esto se debe a que el algoritmo de enrutamiento creado fue desarrollado a partir de una matriz de 4x4 y, como podemos observar, para dimensiones mayores como 6x6 tiene que ser pulido.

CAPÍTULO 8

CONCLUSIONES Y RECOMENDACIONES

8.1 Conclusiones

Del Proyecto de Tesis “Diseño e Implementación de un simulador de arquitecturas Network-On-Chip” se obtuvo las siguientes conclusiones:

La estructura de la NoC es más sencilla que las redes computacionales, ya que los elementos que conforman la arquitectura de la NoC no se encuentran sujetos a cambios posteriores a su implementación. Todos los elementos son ya conocidos y pre definidos en el diseño de la NoC, los cuales no van a estar sujetos a cambios dinámicos posteriores, como por ejemplo el número de elementos que la conforman o la interconexión de su topología.

Previo al desarrollo de una nueva propuesta de arquitectura NoC se debe conocer la estructura interna de los elementos que conforman ésta y su funcionalidad dentro de la red como la topología, el buffer, algoritmo de enrutamiento, enlaces; para así poder determinar en qué elementos se pueden realizar cambios que afecten al desempeño de la red.

La latencia depende directamente de la topología de la NoC debido a la interconexión existente entre los nodos que la conforman ya que, lógicamente, se puede reducir la distancia entre ellos. Por ejemplo, en la topología Torus y TorusS no existe la limitante de la interconexión de los nodos extremos opuestos como en la topología Mesh

La sencillez y optimización del algoritmo de enrutamiento reduce los tiempos de despacho de los paquetes, como se evidencia en la arquitectura TorusB en comparación con la arquitectura TorusR, ya que a dimensiones menores de 6x6 la arquitectura TorusB presenta una latencia menor a pesar de que las dos arquitecturas están conformadas por la misma topología con la variación del algoritmo de enrutamiento

El troughtput es la velocidad interna con que los elementos de red procesan la información. Si la estructura interna de los elementos de red es la misma en todas las arquitecturas, el troughtput es el mismo también en todas.

El desempeño en cuanto a términos de energía consumida de las arquitecturas TorusR, TorusB y TorusS es menor a la de la arquitectura Mesh. Se debe tomar en cuenta para futuros estudios que el simulador no considera la energía generada por la longitud del cableado, esto puede llegar a alterar los resultados obtenidos en cuanto a la energía consumida en las arquitecturas TorusR, TorusB y TorusS puesto a que el cableado viene a ser más largo.

Los simuladores de NoC se enfocan en entregar el desempeño computacional (latencia, troughtput) y la energía consumida de la arquitectura NoC. La mayoría de simuladores para formar la arquitectura NoC crean módulos separados de cada elemento que la componen, para luego unificarlos mediante señales. Esto ayuda a que los simuladores se asemejen a la programación utilizada en los sistemas VHDL y Verilog.

Cada programador tiene su manera de realizar su simulador, sin embargo, se mantiene una estructura similar en cuanto a la composición de las NoC. Por lo tanto, al saber cómo está estructurada la NoC, resultó sencillo entender y comprender dónde se debe modificar el código para así poder realizar nuevas arquitecturas.

Noxim es un simulador compuesto de varios módulos los cuales corresponden a cada elemento que compone la NoC, desde sus enlaces hasta la generación del tráfico. Lo cual ayuda a comprender aun más como se componen las NoC.

8.2 Recomendaciones

Del Proyecto de Tesis “Diseño e Implementación de un Simulador de Arquitecturas Network-On-Chip” se recomienda lo siguiente:

Las NoC abren un amplio campo para la investigación, por lo que es necesario realizar un estudio por separado de cada uno de los elementos que la componen (algoritmo de ruteo, buffer, topología), para así lograr optimizarlos al máximo y lograr en conjunto obtener un mejor desempeño.

El simulador no contempla la energía consumida, ocasionada por la longitud de los cables de las interconexiones entre los nodos externos; esto se deberá tomar en cuenta para futuros estudios.

Se debe realizar un estudio dedicado a los algoritmos de enrutamiento, para así poder garantizar un algoritmo óptimo que ayude a mejorar el desempeño de la NoC.

La mayoría de simuladores de NoC se encuentran desarrollados en el lenguaje de programación SystemC y g++ los cuales corren dentro del sistema operativo Linux, por lo tanto es recomendable saber utilizar este sistema operativo.

REFERENCIAS BIBLIOGRÁFICAS

Achballah, A. B., & Saoud, S. B. (2011). A Survey of Network-On-Chip Tools. *National Institute of Apllied Sciences and Technology, Dept. of Electrical Engineering*, Vol. No.2.

Baran, P. (1964). *IEEE Transactions on Communications Systems.* Santa Mónica, California: Communications Networks.

Bicsi, B. (2002). *Network Desing Basics for Cabling Professional.* McGraw-Hill: Hardcover.

Blake, M. B. (2003). *Coordinating Multiple Agents for Workflow-Oriented Process Orchestration.* Springer, Verlag: Information Systems and E-Business Management Journal.

Craig Partridge, S. B. (06 de 03 de 2006). *Data networking al BBN*. Recuperado el 15 de 07 de 2013, de Data networking al BBN: http://ieeexplore.ieee.org

Dally, W., & Towles, B. (2004). *Principles and Practices of Interconnection Networks.* USA: Morgan Kaumann, Waltham, Mass.

ENGINEERING, E. &. (15 de 01 de 2013). *http://www.ece.cmu.edu/~sld/wiki/doku.php?id=shared:worm_sim*. Recuperado el 18 de 07 de 2013, de http://www.ece.cmu.edu/~sld/wiki/doku.php?id=shared:worm_sim: http://www.ece.cmu.edu/~sld/wiki/doku.php?id=shared:worm_sim

Environment, N. T. (27 de 10 de 2009). *http://www.ict.kth.se/nostrum/NNSE/*. Recuperado el 13 de 07 de 2013, de http://www.ict.kth.se/nostrum/NNSE/: http://www.ict.kth.se/nostrum/NNSE/

Et al, W. (2002). *Orion: A Power-Performance Simulator for Interconnection Networks.* Istanbul, Turkey: In 35th Annual IEEE/ACM Internacional Symposium on Microarchitecture.

Guerrier, P., & Greiner, A. (2000). *A generic architecture for on chip packet-switched interconnections.* Paris: Design, Automation and Test in Europe Conference and Exhibition.

Keidar, I., & Cidon, I. (2010). *Zooming in on* . Springer-Verlag Berlin: Proceedings of the 16th international conference on Structural Information and Communication Complexity.

Kozierok, C. M. (20 de 09 de 2011). *Data Encapsulation, Protocol Data Units (PDUs) and Service Data Units (SDUs)*. Recuperado el 18 de 07 de 2013, de Data Encapsulation, Protocol Data Units (PDUs) and Service Data Units (SDUs): http://www.tcpipguide.com

Kumar, S., Jantsch, A., & Soininen, J. (2000). *Network-on-chip architecture and design methodology.* Proceeding of the Internacional Symposium on Very Large Scale Integration.

Kumar, S., Jantsch, A., Soininen, J.-P., & Forsell, M. (2002). *Network on Chip Architecture and Desing Methodology.* Pittsburgh, PA: VLSI. Proceedings. IEEE Computer Society Annual Symposium on.

Network-On-Chip. (29 de 09 de 2012). *Top 5 most popular NoC simulators*. Recuperado el 12 de 07 de 2013, de Top 5 most popular NoC simulators: http://networkonchip.wordpress.com/2012/09/29/top-5-most-popular-noc-simulators/

Networks, N. (1995). *Introduction to Internetworking.* Billerica, Massachusetts: BayNetworks. Inc.

NIRGAM. (04 de 08 de 2012). *http://nirgam.ecs.soton.ac.uk/*. Recuperado el 13 de 07 de 2013, de http://nirgam.ecs.soton.ac.uk/: http://nirgam.ecs.soton.ac.uk/

NOXIM. (10 de 07 de 2013). *http://noxim.sourceforge.net/*. Obtenido de http://noxim.sourceforge.net/: http://noxim.sourceforge.net/

NOXIM. (10 de 07 de 2013). *www.Noxim.org*. Obtenido de www.Noxim.org: www.Noxim.org

Pardo Carpio, F. (2002). *Arquitecturas Avanzadas.* Valencia.

Posey, B. M. (2006). *Networking Basics.* South Carolina: Networking Hardware.

Rappaport, T. S. (2002). *Wireless Communications, Principles and Practice.* New York: Prentice Hall PTR.

Simulator for Network-On-Chip. (23 de 04 de 2006). *General Purpose Simulator for Network-On-Chip*. Recuperado el 12 de 07 de 2013, de General Purpose Simulator for Network-On-Chip: http://www.buet.ac.bd/cse/research/group/noc/

Simulator, B. I. (12 de 02 de 2013). *http://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/BookSim*. Recuperado el 10 de 07 de 2013, de http://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/BookSim: http://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/BookSim

Simulator, T. N. (18 de 07 de 2013). *The Network Simulator -ns-2*. Obtenido de The Network Simulator -ns-2: http://www.isi.edu/nsnam/ns/

SunFloor3D. (08 de 11 de 2012). *SunFloor 3D: A Tool for Networks on Chip Topology Synthesis for 3D Systems on Chips*. Recuperado el 15 de 07 de 2013, de SunFloor 3D: A Tool for Networks on Chip Topology Synthesis for 3D Systems on Chips: http://infoscience.epfl.ch/record/150054

Tanendaum, A. S. (2003). *Computer Networks.* Amsterdam: Prentice Hall PTR .

Thid, R. (2002). *A Network on chip Simulator.* Stockholm: Master of Thesis in Electronic System Design.

Thid, R. (2002). *A Network on Chip Simulator.* Stockholm: Master of Science Thesis in Electronic System Design.

tools, S. (s.f.). *http://www.dejazzer.com/software.html*. Recuperado el 20 de 07 de 2013, de http://www.dejazzer.com/software.html: http://www.dejazzer.com/software.html

Tsai, W.-C., Lan, Y.-C., Hu, Y.-H., & Chen, S.-J. (2012). *Networks on chips: Structure anda Desing Methodologies.* Wisconsin-Madison: Electrical and Computer Engineering.

Wiklund, D. (2005). *Development and Performance Evaluation of Networks on Chip.* Linköping, Sweden: Department of Electrical Engineering.

FECHA DE ENTREGA

El presente proyecto fue entregado en el Departamento de Eléctrica y Electrónica, y reposa en los archivos desde:

Sangolquí, a _____

Elaborado por:

_____

Sr. Edwin Darío Tufiño Villacís

CC:1720539152

Autoridad:

_____

Ing. Ramiro Rios.

DIRECTOR DE LA CARRERA DE INGENIERÍA EN ELECTRÓNICA, REDES Y COMUNICACIÓN DE DATOS