INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

División de Tecnologías de Información y Electrónica

Diseño, Implementación y Evaluación de un Laboratorio Avanzado de Procesadores de Redes

Presentada como requisito parcial para obtener el grado de

Maestro en Ciencias con Especialidad en Ingeniería Electrónica (Sistemas Electrónicos).

Por

Octaviano Flores González

Monterrey, N.L., Mayo de 2007

© Octaviano Flores González , 2007

Diseño, Implementación y Evaluación de un Laboratorio Avanzado de Procesadores de Redes

por

Ing. Octaviano Flores González

Tesis

Presentada al Programa de Graduados de la

Escuela de Tecnologías de Información y Electrónica

como requisito parcial para obtener el grado académico de

Maestro en Ciencias

especialidad en

Ingeniería Electrónica con especialidad en Sistemas Electrónicos

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Mayo de 2007

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Escuela de Tecnologías de Información y Electrónica

Programa de Graduados

Los miembros del comité de tesis recomendamos que la presente tesis de Octaviano Flores González sea aceptada como requisito parcial para obtener el grado académico de **Maestro en Ciencias**, especialidad en:

Ingeniería Electrónica con especialidad en Sistemas Electrónicos

Comité de tesis:

Dr. Juan Arturo Nolazco Flores
Asesor de la tesis

Dr. Alfonso Avila Ortega
Sinodal

M.C. Santiago A. Rodríguez Pérez
Sinodal

Dr. Graciano Dieck Assad
Director del Programa de Graduados

Mayo de 2007

Dedicatoria

Dedico esta tesis a mi padres Roberto Octaviano y Aida Guadalupe, los cuales me dieron el principal recurso para llegar hasta esta etapa de mi vida, el amor. Además, me brindaron todo su apoyo, me animaron y siempre estaban convencidos que era capaz de culminar con éxito este nuevo capítulo en mi vida. A mis hermanos José Roberto y Juan Ramón y mi hermana Brenda, que me dieron la fuerza necesaria para poder continuar cada vez que me veía aflijido. Gracias por apoyarme y levantarme cada vez que me caía. Y al amor de mi vida, Nayeli, por toda la paciencia, comprensión y apoyo que me brindó durante este largo periodo.

Reconocimientos

A mi asesor Dr. Juan Arturo Nolazco Flores, por permitirme aportar un granito de arena al área de docencia e investigación, por su orientación y apoyo además de compartirme sus valiosos conocimientos.

Un especial agradecimiento a mi sinodal el Ing. Santiago Alberto Rodríguez Pérez, ya que no me brindó ayuda como un colega sino como un amigo en quien puedes confiar y pedir soporte para la culminación de esta tesis. Gracias por el apoyo que me dio en todo momento, por su disposición y experiencia. Por darme la oportunidad de dar clases a alumnos de carrera y poder compartir mis conocimientos, además de corregirme en todos mis errores, muchas gracias.

A mi sinodal el Dr. Alfonso Ávila Ortega, por haber sido un excelente maestro durante mi estancia en el TEC, por su ayuda y acertados comentarios.

A Dinora Nayeli Garza Santos, por haber salvado mi vida en toda ocasión, por estar disponible en todo momento y por nunca decirme que no. Además, por mostrarme cuál era mi límite y demostrarme que era posible llegar más lejos de lo que yo creía.

A Margarita Olaya Cabrera, por todo el tiempo que me dedicó a pesar de que ella también andaba bajo presión de la tesis, agradezco demasiado el apoyo y soporte que siempre me dio.

A Adrián Terán de la compañía Grasshoper, por todo el tiempo y apoyo que me dio para la implantación del sistema operativo de la tarjeta.

A la compañía de Windriver e Intel. A Windriver por dar soporte técnico a instituciones educativas y a Intel por haber donado esta poderosa herramienta tecnológica.

A mi prima Mayra Flores que vivió con nosotros este último semestre y siempre me animó.

A mis amigos del departamento Igmar, Isai, Ulises, Enrique, Roberto, Jorge, Sara,

Vidhya, Brenda y Adriana, por todos esos momentos de alegría y tranquilidad que me brindaron.

A mis amigos de toda la vida: Alex, Nacho, Paco, Jaime, Cesar, Diego, Enrique, Héctor, Fermín por los ánimos y por siempre estar al pendiente de mí.

Y a mis vecinas de cubículo Lorena, Tania y Adriana, por que me motivaron cada vez que podían.

Muchas gracias a todos

Octaviano Flores González

Instituto Tecnológico y de Estudios Superiores de Monterrey
Mayo 2007

Diseño, Implementación y Evaluación de un Laboratorio Avanzado de Procesadores de Redes

Octaviano Flores González , M.C.
Instituto Tecnológico y de Estudios Superiores de Monterrey, 2007

Asesor de la tesis: Dr. Juan Arturo Nolazco Flores

Este trabajo muestra el Diseño, Implementación y Evaluación de un nuevo Laboratorio para el procesamiento de redes utilizando la tarjeta ENP2611 de Radisys. Se diseñó una topología de laboratorio, se utilizó el sistema operativo de VxWorks y se desarrollaron prácticas. Estas prácticas se dividieron en dos segmentos, en el primer segmento se utilizó una computadora convencional x86 y el nivel de las prácticas fue básico teniendo como función ser catalizadores del aprendizaje para las prácticas del segundo segmento, las cuales utilizan el procesador de red IXP2400. Estas prácticas están enfocadas en incrementar el aprendizaje y habilidades de los alumnos permitiéndoles adquirir los conocimientos necesarios para diseñar, implementar, depurar y evaluar una aplicación de red. Se aplicaron evaluaciones objetivas y subjetivas a un grupo de 28 alumnos para corroborar y llegar a conclusiones sólidas del impacto del laboratorio en su aprendizaje. Se aplicó la evaluación objetiva al principio y otra al final del laboratorio, donde se llegaron a las conclusiones donde el 64 % del grupo opina que las prácticas tienen muy buen diseño ya que sirven para aprender a desarrollar, además de que el 96 % del grupo aseguró que el laboratorio tiene un nivel intermedio-superior tanto para la enseñanza didáctica como la investigación. Esto concluye en que se tiene un excelente diseño de laboratorio ya que aceleró el aprendizaje a tal grado en que los alumnos se sintieron capaces de desarrolar código para el procesador de red.

Índice general

Reconocimientos vi

Resumen viii

Índice de figuras xi

Capítulo 1. Introducción 1
- 1.1. Definición del Problema . . . 2
- 1.2. Objetivo . . . 2
- 1.3. Justificación . . . 3
- 1.4. Contribución . . . 3
- 1.5. Organización . . . 3

Capítulo 2. La Relación Teoría-Práctica 5

Capítulo 3. Marco Teórico (ENP-2611 y Procesador IXP2400) 10
- 3.1. Introducción . . . 10
- 3.2. Tarjeta ENP-2611 . . . 11
- 3.3. La arquitectura del procesador IXP2400 . . . 13
- 3.4. Procesador Integrado Intel Core XScale . . . 13
- 3.5. Ejemplo de Otros Procesadores de Redes . . . 20
  - 3.5.1. Procesador Funcional Agere . . . 20
  - 3.5.2. CISCO . . . 25
  - 3.5.3. Motorola . . . 30
- 3.6. Tendencias de los Procesadores de Redes . . . 34

Capítulo 4. Caso Estudio: Implementación del Laboratorio de Procesadores de Redes (IXP2400) 36
- 4.1. Historia de trabajo y desarrollo de prácticas con computadoras x86 . . . 36
- 4.2. Topología del Laboratorio de Procesadores . . . 38
- 4.3. Objetivos del Laboratorio . . . 39
- 4.4. Estructura del Laboratorio . . . 40

4.5. Prácticas del Laboratorio 40
4.5.1. Ambiente de desarrollo Integrado IDE: Tornado 41
4.5.2. El ambiente de desarrollo IDE: Developer Workbench 43
4.5.3. Simulación de Código 43
4.5.4. Implementación de Código 45

Capítulo 5. Experimentos y Resultados 46
5.1. Evaluación Objetiva 46
5.2. Evaluación Subjetiva 49
5.2.1. Preguntas enfocadas al catalizador de prácticas 50
5.2.2. Análisis de conocimientos que los alumnos sienten que adquirieron 53
5.2.3. Análisis de los aspectos a mejorar en el Laboratorio 55
5.2.4. Aspectos positivos y negativos del laboratorio 56
5.2.5. Diseño de las prácticas 58
5.2.6. Nivel de las Prácticas del Laboratorio 59
5.2.7. Capacidades que le ayudaron a los alumnos a realizar las prácticas 60
5.2.8. Comentarios finales de los alumnos 61
5.3. Analisis de Resultados 63

Capítulo 6. Conclusiones 66
6.1. Conclusiones 66
6.2. Trabajo Futuro 67

Apéndice A. Evaluación Objetiva 68

Apéndice B. Evaluación Subjetiva 72

Apéndice C. Prácticas Haciendo uso de una Computadora Convencional 77

Apéndice D. Prácticas Haciendo uso del Procesador de Red IXP2400 93

Bibliografía 161

Índice de figuras

2.1. Ciclo del Conocimiento 5
2.2. Datos del Conocimiento 6
2.3. Conceptos del Aprendizaje 7
2.4. Teoria-Aprendizaje 8

3.1. Tarjeta ENP-2611 de Radisys 12
3.2. La unidad Funcional del IXP2400 [10] 14
3.3. Diagrama de bloques del Microengine IXP2XXX [10] 16
3.4. Segmentación del procesador funcional Aguere 21
3.5. Hardware de Interfases I/O [1] 23
3.6. Arquitectura Interna del APP550 [1] 24
3.7. Performance Routing Engine PRE [4] 26
3.8. Performance Routing Engine 2 PRE2 [4] 27
3.9. Arquitectura PXF [4] 28
3.10. Arreglo de 4 filas donde cada columna tiene una función diferente del procesador PXF [4] 29
3.11. Diagrama de Bloques del Procesador de Red C-5 31
3.12. Grupo funcional de los Procesadores de Redes [11] 35

4.1. Topología del Laboratorio de NP 38
4.2. Software de Comunicación: NP IDE [5] 39
4.3. Sistema Host - Sistema Tarjeta [5] 42

5.1. Resultado de los exámenes de diagnóstico 47
5.2. Respuestas correctas-incorrectas de la 1ª Evaluación 48
5.3. (a) Antes del Laboratorio y (b) Después del Laboratorio 48
5.4. (a) Antes del Laboratorio y (b) Después del Laboratorio 49
5.5. Mejoramiento en la Evaluación 49
5.6. Hash vs IP-Five-Tuple 50
5.7. Bridge vs Recepción, Contador, Transmisión 50
5.8. Bridge, Clasificador y NAT vs Testing 51
5.9. Multithread de Bridge vs Semáforo de Tornado 51

5.10. Clasificador vs IP-Five-Tuple 52
5.11. NAT vs Prácticas del Nuevo Laboratorio 52
5.12. Prácticas usando NIC convencional vs Prácticas del Laboratorio IXP2400 53
5.13. Retroalimentación Subjetiva de los conocimientos adquiridos en el Procesador de Red IXP2400. 54
5.14. Retroalimentación Subjetiva de los conocimientos adquiridos en el procesador de Red IXP2400 55
5.15. Aspectos a mejorar en el Laboratorio 56
5.16. Aspectos positivos del laboratorio 57
5.17. Aspectos negativos del laboratorio 58
5.18. (a) Diseño de las Practicas y (b) Justificación 58
5.19. (a) Nivel del laboratorio (b) Razones 59
5.20. Esfuerzo cognitivo de las prácticas del NIC convencional 60
5.21. Esfuerzo cognitivo que genera las prácticas del NP 62
5.22. Comentarios finales de los alumnos 62

C.1. (a) Hub y (b) Bridge 80
C.2. Topología del Bridge 81
C.3. Topología 85
C.4. El concepto de una Topología NAPT 87
C.5. Topología del NAPT del Laboratorio 90

D.1. New Project 95
D.2. Create Project 96
D.3. Create Module 97
D.4. Toolchain and confirmation 98
D.5. Workspace 98
D.6. Adding Code 99
D.7. Building the project 100
D.8. Download the project 101
D.9. Download the project 101
D.10.Name of the target server 102
D.11.Shell Window 103
D.12.Memory Usage 104
D.13.WindView Colleciton Configuration 105
D.14.A wrong Graph 106
D.15.A good Graph 107
D.16.The Simulation 108
D.17.Target Server in Development Environmnet 108
D.18.Target Server in Development Environmnet 110

D.19.Partitioning of Packet Receive and Packet Processing 113
D.20.Hardware-Supported Multithreading 115
D.21.Packet Dataflow View 116
D.22.Create a Project 117
D.23.Create List File 118
D.24.Assign List File to Root File 119
D.25.Assign List File to Microengine 119
D.26.Run Simulation 120
D.27.Run Simulation 122
D.28.Assembler include directories 123
D.29.Simulator Code View for Microengine Assembly Code 124
D.30.Data Watch Window 126
D.31.Reassembling 64 byte Mpacket Contiguosly into Buffer 128
D.32.Format of the RX_THREAD_FREELIST_0 and RSW 129
D.33.Transferring Data from DRAM to a TBUF Element 131
D.34.The Format of TCW and Validating the TBUF Element 132
D.35.The MSF Device Simulator's Devices and Network Connections Dialog 133
D.36.The SImulator Data Stream Dialog 135
D.37.The Simulator Traffic Assignment 136
D.38.The implementation of Scratchpad Rings. 138
D.39.Using the SRAM Queue Array as Cache 141
D.40.The Format of a Queue-array Element for an SRAM Ring 142
D.41.The Format of an Entry on an SRAM Queue 143
D.42.The Contents of a Queue-array Element for an SRAM Queue. 143
D.43.Flowcharts for the Enqueue and Dequeue Operations in the SRAM Controller 144
D.44.The Sample Application Data Flow 147
D.45.Unaligned IP Header in an Ethernet Packet 150
D.46.Unaligned Access Instructions 152
D.47.A Simple Hash table 153
D.48.Modified Hash Table 157
D.49.Probability of Dropping a Packet 158
D.50.Communication between Target Server and Target Agent 159
D.51.Network Processor Lab Topology 160

Capítulo 1

Introducción

Hoy en día, las universidades están más preocupadas por llevar las clases a un nivel en donde, no sólo la teoría es el medio de aprendizaje, sino que la práctica se vuelve un recurso importante para que los alumnos adquieran mayores habilidades y conocimientos. Esto se ha podido lograr a través de la incorporación de nuevas tecnologías que transforman la manera tradicional de enseñar y aprender, ofreciendo a los estudiantes una manera dinámica y motivante de desarrollar nuevas habilidades.

Algunos cursos de redes están enfocados primordialmente en enseñar a través de prácticas basadas en los principios de ruteadores, switches y hubs, entre otros. La mayoría de los laboratorios están diseñados para instruir en un nivel de usuario y administrador, lo cual no brinda a los alumnos la oportunidad de adquirir la experiencia técnica para diseñar, desarrollar e implementar una aplicación de redes basada en las necesidades de la industria.

Desde el año 2000, muchas generaciones de procesadores de redes se han creado para resolver el problema del rendimiento de procesamiento de paquetes que los ruteadores están empezando a sufrir en la actualidad [2]. Compañías de telecomunicaciones han notado el potencial de estas tecnologías y han desarrollado software que ayuda a monitorear DNS, detectar intrusos, monitorear el tráfico de los ruteadores, etc.

Uno de los procesadores de redes más actuales en el mercado es el IXP2400 de Intel. Este permite que el procesamiento de un paquete pueda ser dividido en múltiples tareas, las cuales se pueden ejecutar y comunicar unas con otras de una manera más rápida.

Esta tesis pretende compartir la experiencia adquirida al implementar un nuevo laboratorio utilizando el procesador de redes IXP2400. En México no existe ningún laboratorio que se imparta con esta tecnología. El primer acercamiento con algo similar es el desarrollo de un laboratorio en Pittsburgh, pero se implementó una versión anterior (IXP1200). También se intentó implementar en Campus Guadalajara en el 2005, pero

debido a la complejidad de esta tecnología, se dejó de impartir.

En esta tesis se demostrará que es posible acelerar el aprendizaje de conceptos complejos partiendo de prácticas sencillas para posteriormente, a través del uso de la tecnología, migrar ese conocimiento a un nivel más alto.

Las prácticas básicas utilizando una computadora convencional son: tablas hash, implementación de un puente, clasificador de paquetes y implementación de un NAPT. Posteriormente se utiliza la tarjeta ENP-2611 de Radisys, con la que se realizan las prácticas avanzadas: IDE Tornado, IDE Developer Workbench, Recepción-Procesado-Transmisión, Procesado en un simple Thread y Probando un Programa Real.

El reto de esta tesis, es diseñar e implementar un laboratorio que pueda ofrecer un aprendizaje práctico utilizando un procesador de redes de Intel (IXP2400) y el Sistema Operativo de Windriver. Las prácticas se crearon de tal manera que le permiten a los estudiantes crear las habilidades necesarias para conectar, desarrollar, eliminar errores y probar programas complejos.

1.1. Definición del Problema

Los cursos de redes que actualmente se imparten por el Departamento de Ciencias Computacionales están enfocados a presentar los conceptos fundamentales requeridos para entender el funcionamiento de los protocolos de redes de datos, para operar equipo de red, así como analizar matemáticamente o por simulación redes de datos. Además el curso de procesadores de redes presenta los conceptos básicos: arquitectura y funcionamiento de red. Por esta razón el laboratorio de Procesadores de Redes va más alla y complementa este conocimiento con la habilidad de desarrollar, implementar y evaluar nuevas aplicaciones en el área de redes como ruteadores, switches, detectores de intrusos, etc.

1.2. Objetivo

Diseñar e implementar un laboratorio utilizando el procesador de red IXP2400, ofreciéndoles a los estudiantes conocimientos prácticos que les permitirán adquirir la experiencia necesaria para conectar, desarrollar, depurar y probar programas complejos.

Principalmente se cubrirán los siguientes aspectos:

- La implementación del sistema operativo VxWorks en la tarjeta ENP-2611.

- La implementación de una nueva topología de red a través de la cual se desarrollarán las prácticas.
- El diseño y desarrollo de prácticas enfocadas a acelerar el proceso de aprendizaje.

1.3. Justificación

Con el crecimiento de equipos complejos basados en la nueva tecnología de procesadores de redes ha sido necesaria la creación de cursos que requieren de aplicaciones prácticas para poder tener un acercamiento real de lo que se está haciendo en la industria. De esta forma se van a crear laboratorios que ayudarán a desarrollar las habilidades y capacidades que se necesitan para poder formar profesionistas capaces de resolver las necesidades que la tecnología actual demanda.

1.4. Contribución

Esta tesis brindará las bases para crear un nuevo laboratorio de procesadores de redes, el cual estará disponible para el curso de Procesadores de Redes de la carrera de Ingeniero en Tecnologías Electrónicas ITE y también como materia tópico para cualquier carrera de sistemas.

1.5. Organización

Esta tesis está organizada de la siguiente manera:

En capítulo 2 se presenta la relación del la teoría con la práctica donde van de la mano para crear un espiral en movimiento que crea conocimiento.

En el capítulo 3 se presenta la descripción de hardware de la tarjeta de Radisys, la cual fue donada por Intel al Tecnológico de Monterrey. También se presenta la descripción del hardware del procesador de red IXP2400, el cual es la base del funcionamiento de la tarjeta y por consiguiente del laboratorio.

En el capítulo 4 se presenta lo que se había desarrollado de prácticas sencillas previo al procesador de red, además de las nuevas prácticas haciendo uso de la nueva tecnología. También se presentan las gráficas resultantes de la evaluación de diagnóstico y la evaluación subjetiva que se les aplicó a los alumnos.

En el capítulo 5 se presentan los resultados, donde se hace un análisis a fondo de todas las gráficas arrojadas por las evaluaciones aplicadas.

Finalmente en el capítulo 6, se dan a conocer las conclusiones de la implementación de este nuevo laboratorio y el trabajo a futuro de este mismo.

Capítulo 2

La Relación Teoría-Práctica

El conocimiento es un conjunto de datos, hechos, verdades o información que es almacenada a través de la experiencia o del aprendizaje. La palabra conocimiento se refiere a la acción y el efecto de conocer las cualidades y relaciones de las cosas, lo cual nos lleva a un nivel de comprensión de lo que está a nuestro alrededor.

El conocimiento pasa por un ciclo, y la utilización del nuevo conocimiento es el inicio de la creación de más conocimiento. Esto se puede representar a través de la siguiente Figura (2.1).

Existen dos tipos de conocimientos:

- Conocimiento Explícito: es aquel que puede ser fácilmente comunicado y compartido, ya que es expresado por palabras, oraciones, números, fórmulas, etc.
- Conocimiento Tácito: es un conocimiento difícil de comunicar y transferir a los demás, ya que involucra creencias, experiencias, creatividad, sabiduría y perspectivas.

El conocimiento se divide en 4 tipos de saber:

- Qué: son los hechos y realidades. Este tipo de conocimiento se adquiere de manera explícita.

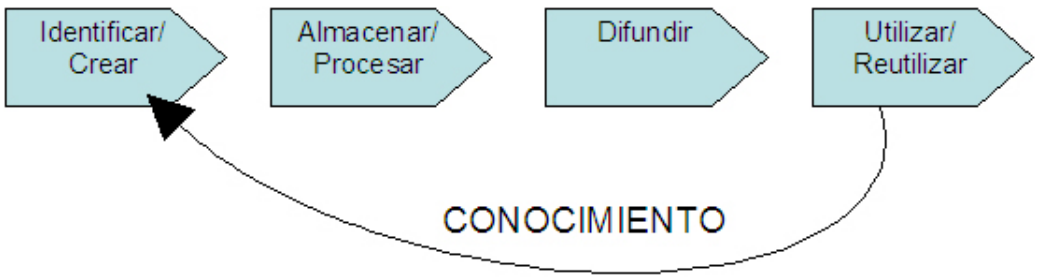


Figura 2.1: Ciclo del Conocimiento

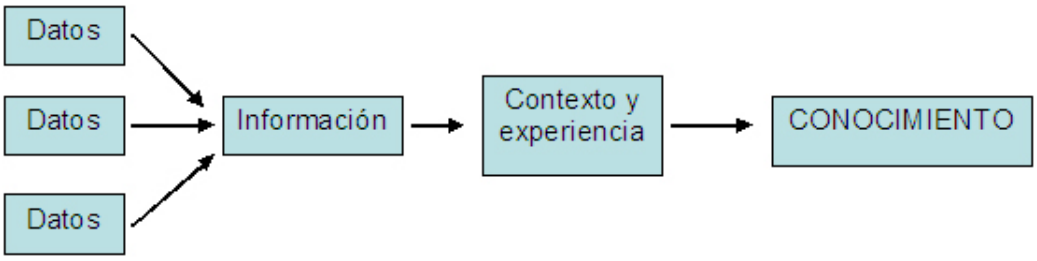


Figura 2.2: Datos del Conocimiento

- Por qué: Para entender el por qué se requiere del apoyo de un laboratorio o universidad, y lo facilita el adquirir los conocimientos de manera práctica. Este conocimiento se adquiere de manera explícita.
- Cómo: son las habilidades o capacidades para hacer algo. Este conocimiento se adquiere de manera tácita.
- Quién: se refiere a los expertos que saben qué y cómo hacer qué. Este conocimiento se adquiere de manera tácita.

El conocimiento se forma a través de datos que son asociados a un objeto o estructurados para convertirse en información. La información relacionada a un contexto o experiencia se convierte en conocimiento. Lo siguiente se puede demostrar a través de la Figura (2.2).

Un concepto importante que se relaciona directamente con el conocimiento es el aprendizaje. El aprendizaje es el proceso de adquirir conocimientos, habilidades, actitudes, valores, experiencia y enseñanza, entre otros. Para lograr un aprendizaje es necesario que los conocimientos se asimilen, se analicen y se juzguen.

Para aplicar estos conocimientos y lograr llegar a un aprendizaje se utiliza la teoría y la práctica. La teoría representa aquella información que se obtiene de diversas fuentes para almacenarla como conocimientos. Se identifica con el pensar, entender, comprender la información.

La práctica es lograr llevar aquella teoría a un nivel de aplicación más orientado a la realidad. Es mostrar con hechos aquello que se aprendió, que los alumnos demuestran sus conocimientos de manera lógica, aplicada e integrada en un contexto más interactivo y dinámico.

La práctica nos ayuda a llevar todo conocimiento a la aplicación. Esto significa, darle una verdadera utilidad al conocimiento, lo cual proporciona oportunidades de

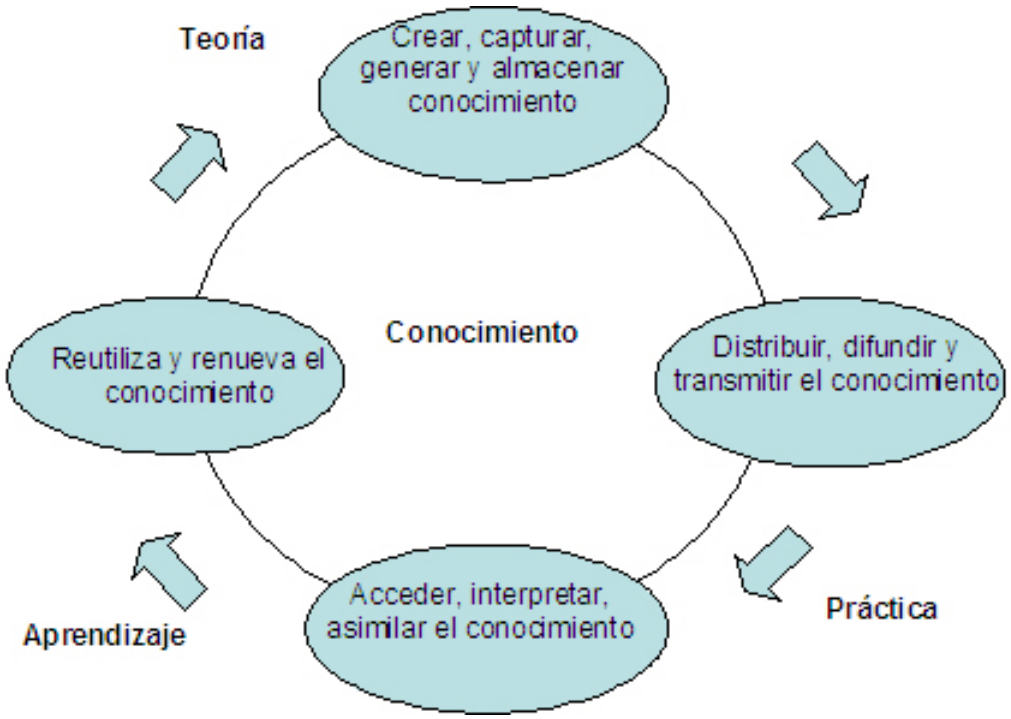


Figura 2.3: Conceptos del Aprendizaje

desarrollo y de usar el pensamiento crítico.

Por lo tanto, vemos que la teoría sin práctica o viceversa, suena incongruente. Se requieren de ambos para obtener los conocimientos y llevarlos a la práctica. Con la teoría hay muchos conocimientos e información almacenada, pero con la práctica hay una evolución de los mismos, se llevan a la acción y por tanto, se adquieren nuevos aprendizajes o se modifican y mejoran con los que ya se contaba.

Como se ha visto, todos los conceptos de aprendizaje, conocimientos, teoría y práctica están relacionados directa o indirectamente. Este ciclo, se puede ver a través de la Figura (2.3), en donde se pueden ver de manera integrada.

Actualmente existen muchos cursos que se imparten de manera teórica. Esto les da a los alumnos toda la información necesaria acerca de la materia. Sin embargo, no les brinda la experiencia de llevar esos conocimientos a un nivel de implementación donde le permita al alumno realizar pruebas de desempeño que acerquen lo más posible a lo que la industria y el sector demandan. Actualmente varias materias de ingeniería en sistemas, van de la mano de un laboratorio. Este, pretende mostrar el lado práctico de la materia, así el alumno logrará desarrollar mayores habilidades de razonamiento y adquirir conocimientos técnicos especializados en su área. Pero lo importante no es

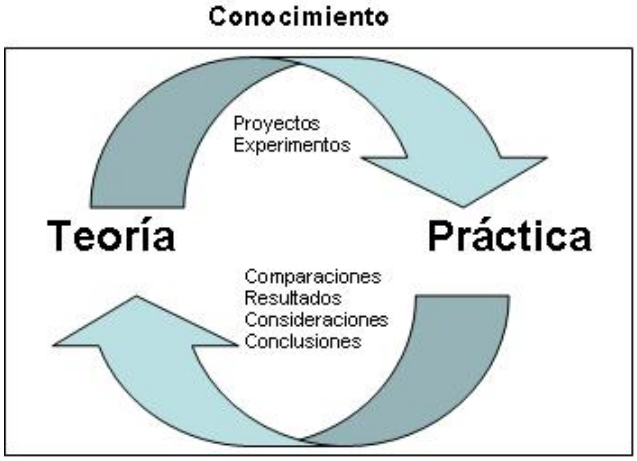


Figura 2.4: Teoria-Aprendizaje

tener una clase teórica y otra práctica, sino encontrar la relación entre ambas, ya que el mayor aprendizaje se da cuando ambas van de la mano.

La teoría y la práctica cuando se imparten en conjunto se dice que se crean las bases del conocimiento [8]. La práctica nace a través de las bases teóricas para poder experimentar y realizar proyectos. La práctica realiza comparaciones, consideraciones y se obtienen resultados con el fin de establecer nuevas teorías que después se comprueban a través de la práctica y así sucesivamente. Es por esto que se dice que ambas se encuentran en un espiral en movimiento que crea conocimiento, como se muestra en la figura (2.4).

Esto ayudará a crear un laboratorio en donde se desarrolle la enseñanza y el aprendizaje, además de darle las herramientas necesarias para desarrollar experiencia relacionada con el ámbito laboral y experiencia curricular. Sin embargo, es importante recordar que para lograr este objetivo, se deben considerar ciertas variables como lo son: infraestructura, equipo tecnológico, y la capacitación del instructor.

Es por ello que es importante tomar en cuenta lo siguiente:

- Ilustrar a través de ejemplos enfocados a las situaciones reales.
- Usar adecuadamente el equipo y el material.
- Ayudar al alumno a resolver las dudas o inquietudes que se le presenten durante el desarrollo de la práctica.
- Monitorear que el alumno esté realizando las actividades necesarias para completar la práctica.

- Que el material que se está impartiendo en el laboratorio vaya de la mano con aquel que se les da durante clase.

Los cursos en donde se involucra el hardware, tienden a involucrar conocimientos que son un puente entre diversas materias. En esta tesis se involucran conocimientos de otros cursos como: Microprocesadores, Lenguaje de Programación, Redes, Lenguaje Ensamblador y Estructura de Datos, entre otras. Lo importante al realizar un laboratorio en donde se involucran tales materias con contenido complejo, es el diseñar de manera intuitiva la práctica además de tener el equipo tecnológico adecuado. Cuando estas dos van de la mano, se logra que el alumno adquiera los conocimientos técnicos esperados, ya que recibe la teoría por parte del instructor, aplican ejemplos que simulan la vida real en la tecnología brindando el entendimiento total de su funcionamiento y al mismo tiempo se refuerzan los conocimientos previos y se adquieren nuevos.

Es importante que se dé esa correlación Teórica-Práctica en el laboratorio, es decir que haya correspondencia efectiva entre los conocimientos adquiridos en clase con la comprobación práctica y experimental y lograr tratar al alumno como un profesionista en entrenamiento.

La práctica despierta en los estudiantes la idea de experimentar, de ver cómo funcionan las cosas. Al realizar las prácticas, los alumnos se enfrentarán con variables que afectan el desarrollo de la práctica. Esto permitirá no sólo que apliquen conocimientos técnicos, sino que desarrollen habilidades de auto-estudio, razonamiento profundo y toma de decisiones.

El objetivo del enfoque teórico-práctico del laboratorio es que el alumno adquiera los conocimientos, los solidifique de manera práctica, trabaje en conjunto con otros alumnos y profesores además de la confianza para solucionar problemas profesionales y personales [12].

Capítulo 3

Marco Teórico (ENP-2611 y Procesador IXP2400)

3.1. Introducción

Los procesadores de redes son un concepto totalmente nuevo en el área de telecomunicaciones. Estos semiconductores proveen una gran versatilidad y escalabilidad en muchas aplicaciones de redes. Haciendo una mejora en el desempeño de los procesadores de redes. La tarjeta que se va a implementar en el nuevo laboratorio es el *Embedded Network Processor* ENP-2611 la cual fue fabricada por Radisys. Esta tarjeta lleva consigo un procesador de red fabricado por Intel: el IXP2400. Por lo que este capítulo se encarga de hacer un gran énfasis en el funcionamiento de este procesador de red el cual es la base de este nuevo laboratorio.

El procesador de red IXP2400 de Intel habilita un rápido desarrollo de los servicios de redes inteligentes previendo una programación flexible, código re-usable y alto desempeño en el procesamiento de paquetes. El procesador IXP2400 soporta una gran variedad de aplicaciones WAN y LAN que requieren un amplio rango de velocidades como OC-3 a OC-48. Un alto desempeño y escalabilidad es logrado a través de una arquitectura innovadora: *Microengine* el cual incluye una arquitectura cache con una distribución multithread que habilita características de pipeline en software [14]. La característica innovadora del Microengine es la comunicación *inter-thread* para un eficiente procesamiento de líneas de alta velocidad y hardware de propósito general que soporta algoritmos de red avanzados. El Microengine juega un rol en el software creado por *Intel Exchange Architecture* (IXA) donde su función principal es de almacenar y reenvía. Dando una gran flexibilidad y un enriquecido procesamiento de red, haciendo posible la convergencia en ambientes de comunicación robustos.

3.2. Tarjeta ENP-2611

El Embedded Network Processor 2611 (Figura 3.1), es una tarjeta compleja con 32/64 - bit PCI 2.2 y con 3 puertos modulares. El ENP-2611 usa el potencial del IXP2400 para proveer más de 5.4 giga-operaciones por segundo. Haciéndolo candidato perfecto para aplicaciones a una gran variedad de aplicaciones de desempeño alto, como: switches multiservicios, routeadores, aparatos de acceso de banda ancha, infrastructura inalambrica, seguridad con detector de intrusos y sistemas de prevención (IDS/IPS), voz sobre IP (VoIP), Red Virtual Publica (VPN), y otros productos relacionados.

El ENP-2611 provee la ventaja de ser totalmente programable a diferencia de cualquier Circuito Integrado de Aplicación Específica (ASIC), el cual ya viene hecho con ciertos protocolos o características de funcionalidad las cuales no pueden ser cambiadas. Por ejemplo: El en caso de que se descubriera una vulnerabilidad en la seguridad de las redes el ENP-2611 puede ser reprogramado para reestablecer protección en contra de esta nueva vulnerabilidad. Además de poder correr velocidades de Gigabit Ethernet (GbE) haciendo desempeños con gran flexibilidad y bajos costos comprados con los ASICs.

A continuación se presentan las principales características del ENP-2611:

- Estándar PCI 2.2
- Procesador de Red IXP2400 Intel DE 600 MHz [16].
- Tres puertos ópticos de Gigabit Ethernet para tráfico de texto plano.
- Tres *Small form Factor Pluggable* (SFP) 1000Base - Sx o 1000Base -Tx Transceiver.
- Un puerto 10/100 Base - Tx para control de trafico plano o servicios de debuggeo.
- Un máximo de direcciones de memoria de 2 Gbytes de DDR ECC SDRAM
- Un máximo de direcciones de memoria de 64 Mbytes de QDR SRAM
- 16 Mbytes de StarataFlash Memory
- 32/64 - bit PCI bus width
- Capacidad de actualizaciones de software remoto.
- Soporte de Intel: IXA SDK
- Soporte de microcódigo y herramientas de desarrollo XScale de Intel

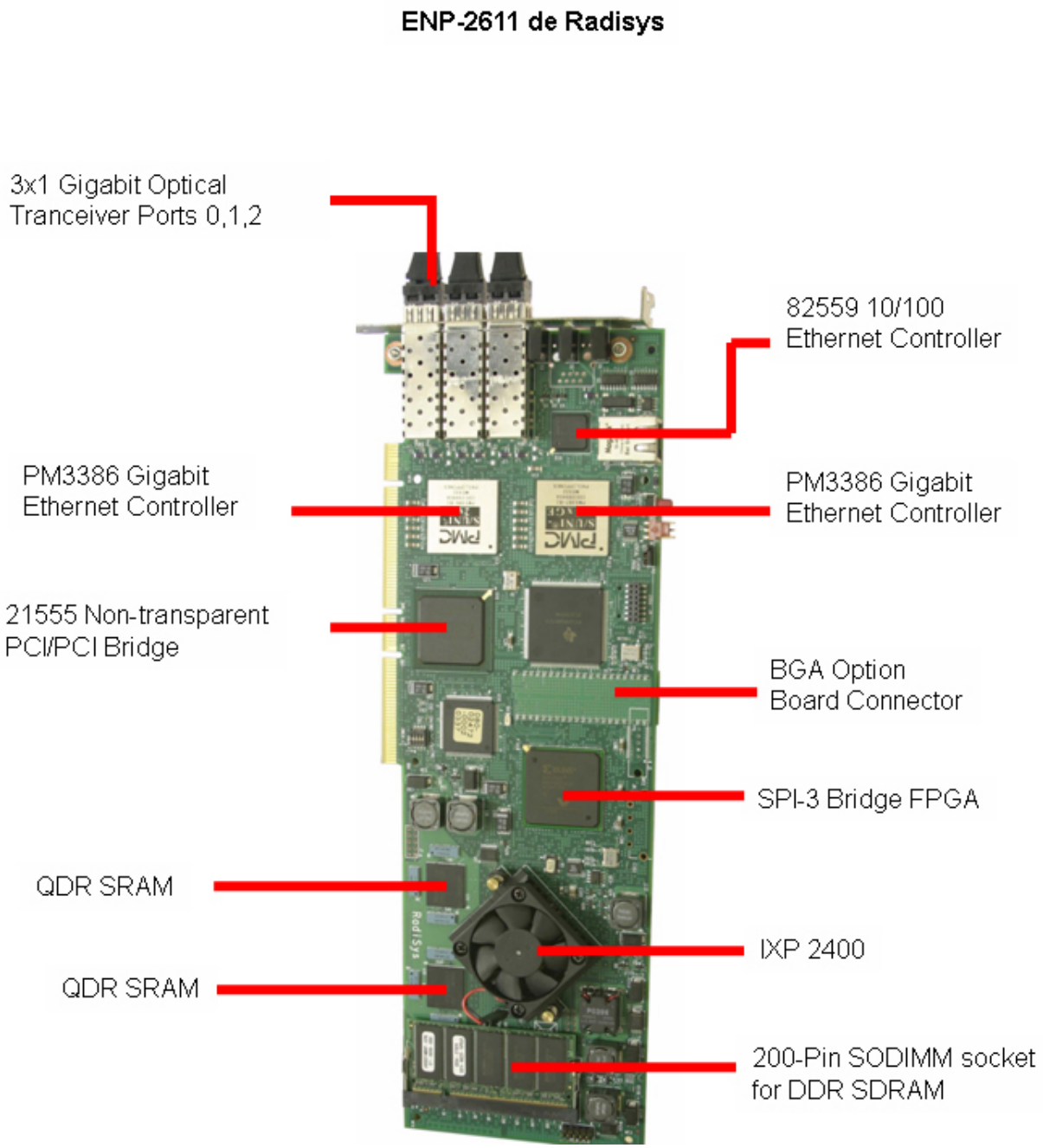


Figura 3.1: Tarjeta ENP-2611 de Radisys

- Soporta MontaVista Linux Professional Edition 3.1 y WindRiver VxWorks Version 5.4

3.3. La arquitectura del procesador IXP2400

En esta sección se hará una breve mención de las interfases externas del procesador de red IXP2400, como se muestran en la Figura 3.2:

- **Dos interfaces unidireccionales de 32 bits (Rx y Tx) programables para ser SPI-3, UTOPIA 1/2/3 o CSIX - L1.**
  Soporta los estándares industriales de célula y paquete, las cuales serían interfases al medio haciendo entregas de 4Gbps de desempeño. Simplifica el diseño e interfase a ASICs fabricados de empresas como: CISCO, Huawei, Motorola, etc. Cada ruta es configurable para 4x8 bits, 2x16 bits, 1x32 bits o combinaciones de 8 y 16 bits.

- **Una interfase estándar DDR DRAM**
  Memoria de soporte de 64 bits de bus de datos, donde el procesador de red almacena el paquete y lo reenvía.

- **Dos interfases estándar QDR-SRAM**
  Memoria de 32 bits para la búsqueda de tablas y lista de accesos, o coprocesadores (como CAM/TCAM)

- **PCI 2.2 64 bits /66 MHz I/O Interfase**
  Soporta estándares industriales de conexión al host.

A continuación se darán explicaciones más detalladas de los misceláneos internos más importantes del procesador IXP2400.

3.4. Procesador Integrado Intel Core XScale

Intel Core XScale es un procesador de 32 bits de propósito general (ARM Versión 5), usado para inicializar y manejar el procesador de redes [3]. También es usado para el manejador de excepciones, conexiones lentas de procesamiento y otras tareas de control. A continuación se menciona una lista extensa de todas las características del procesador Intel XScale para lograr su alto procesamiento:

Multiplicador/Acumulador (MAC)
La unidad MAC soporta multiplicaciones/acumuladores en dos ciclos y puede mantener una razón de operación MAC por cada ciclo de reloj. Muchas mejoras fueron realizadas al MAC para soportar algoritmos de codificación de audio, el cual incluye 40 bits de

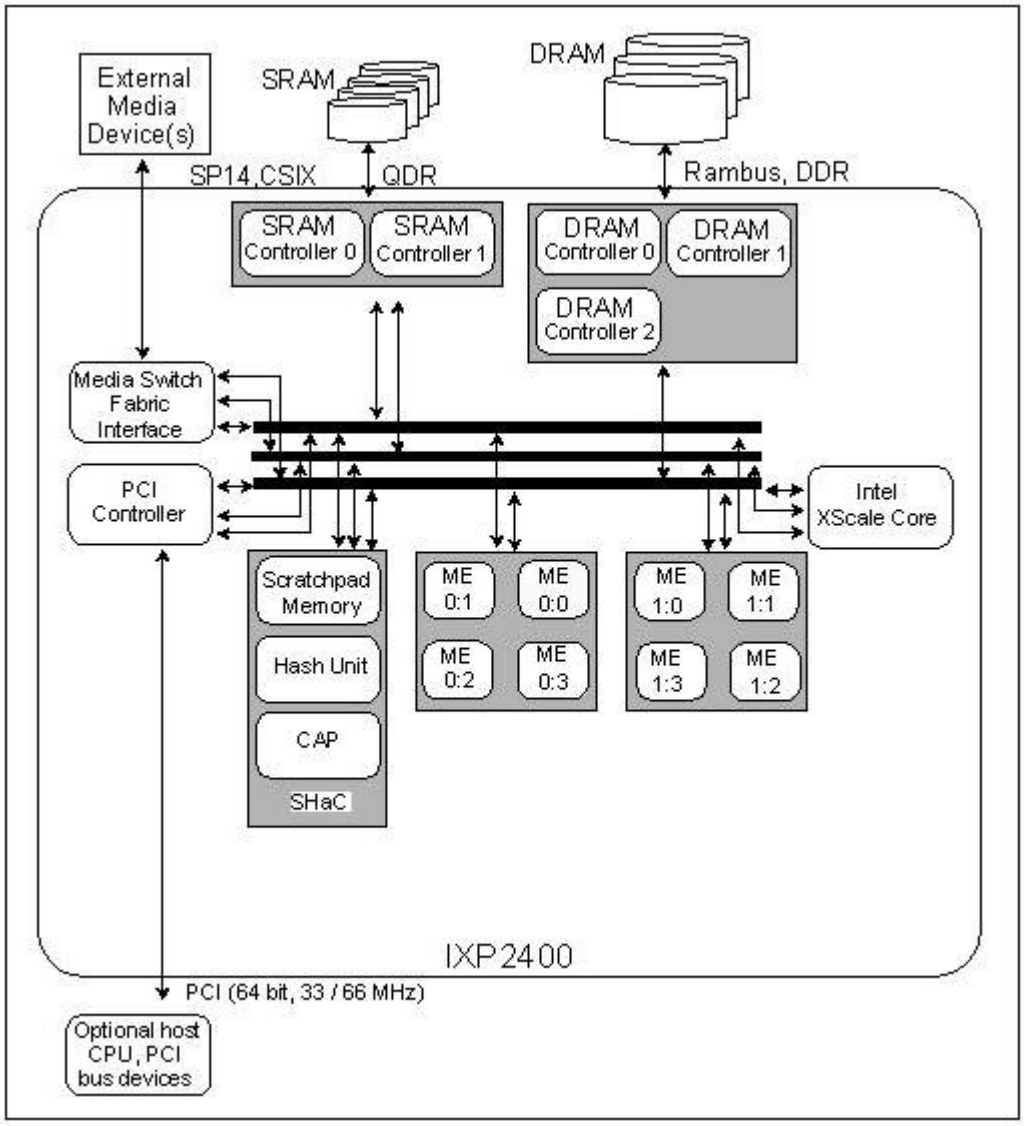


Figura 3.2: La unidad Funcional del IXP2400 [10]

acumulador y soporta valores para paquetes de 16 bits.

Administrador de Memoria
La micro arquitectura Intel XScale implementa la unidad de Administración de Memoria (MMU) especificada en el manual de referencia de la arquitectura ARM. El MMU provee protección en el acceso virtual y físico de las traducciones de memoria.

Instrucciones Cache
La arquitectura del Intel XScale implementa 32 Kbytes, 32 maneras de asociar instrucciones cache con una línea de 32 bytes. Todas las peticiones que fallan a la instrucción cache generan peticiones de lectura a memoria externa de 32KB.

Data Cache
La micro arquitectura Intel XScale implementa 32 Kbytes. Cada cache tiene una línea de tamaño de 32 bytes, además de soportar escritura hacia delante y hacia atrás en el cache.

Ocho *Microengines* integrados programables con 4k de almacenamiento de instrucciones de programa

El Microengine (MEs) hace la mayoría del procesamiento del paquete en el IXP2400. Hay 8 Microengines, conectados en 2 clusters de 4 Microengines como se muestra en la Figura 3.3. Los Microengines tienen acceso a todos los recursos compartidos (SRAM, DRAM, MSF, etc) así como también conexiones privadas a los Microengines adjuntos.

Control de almacenamiento
El control de almacenamiento es una RAM, el cual mantiene el programa que el Microengine ejecuta. Este mantiene 4096 instrucciones, cada una es de 40 bits de ancho. Este es inicializado por el Intel Core XScale. El control de almacenamiento es protegido por paridad para evitar errores.

Contexto
Hay 8 contextos de hardware (threads) disponibles en el Microengine. Para permitir un cambio eficiente de contexto, cada contexto tiene su propia serie de registros; Contador de Programa y registros locales específicamente para contexto. Teniendo una copia por contexto elimina la necesidad de mover información específica del contexto a memoria compartida y registros del Microengine. Rápidos cambios de contexto permiten a un thread estar haciendo operaciones computacionales mientras otros threads esperan para completar transferencias de IO (Típicamente accesos de memoria externa) o por una

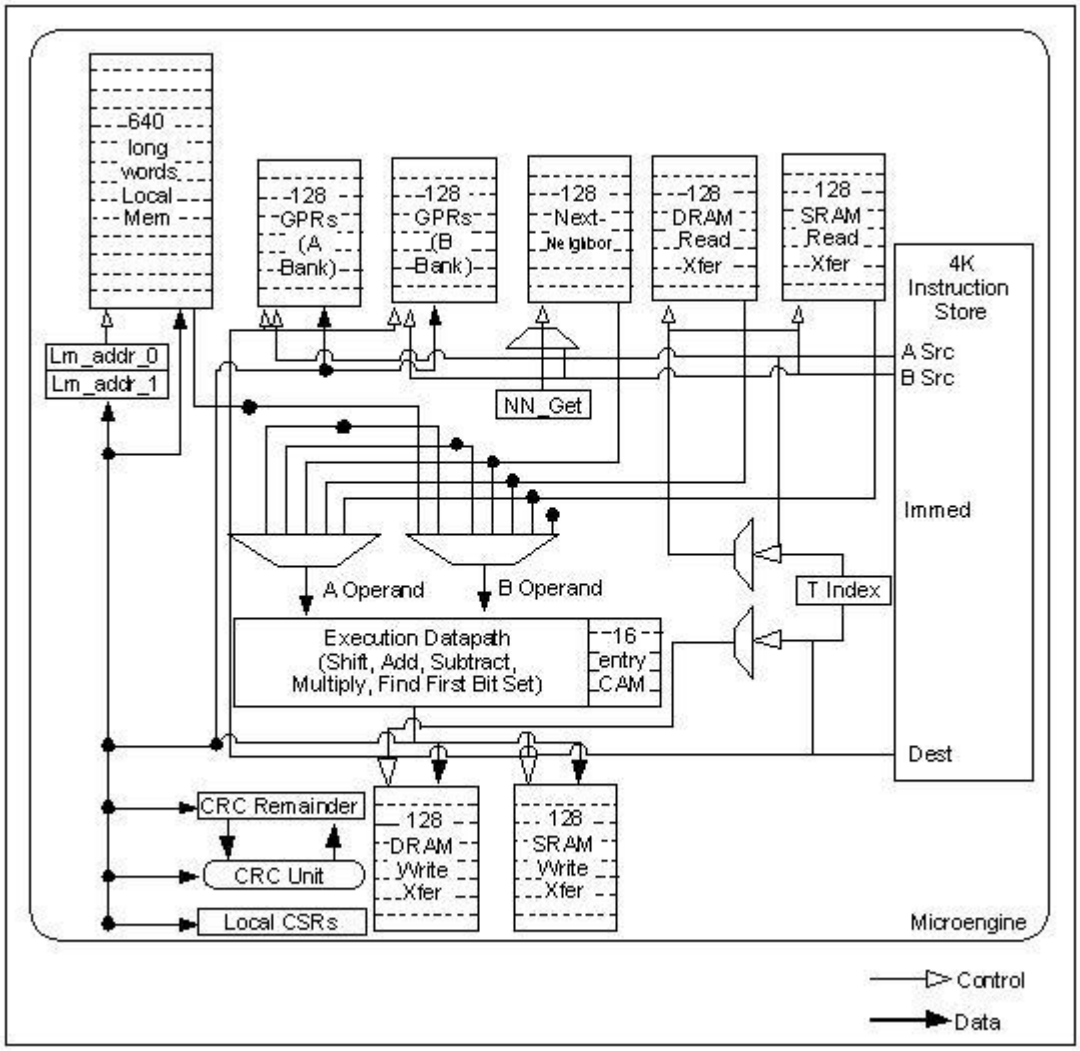


Figura 3.3: Diagrama de bloques del Microengine IXP2XXX [10]

señal de otro Contexto o unidad de hardware.

Registros de Datos Cada microengine contiene cuatro tipos registros de datos (32 bits):

1. 256 Registros de Propósito General
2. 512 Registros de Transferencia
3. 128 Registros del Próximo Vecino
4. 640 32- bits palabras de Memoria Local

GPRs son usados para propósito de programación general. Estos son de lectura y escritura exclusivamente debajo del control de programa. Cuando los GPRs son usados como fuente en una instrucción, provee de operandos al *exclution data path*. Cuando es usado como destino en una instrucción, estos son escritos con el resultado del *exclution data path*. Los GPRs seleccionados son codificados en la instrucción.

Los registros de transferencia (Abreviado como Registros Xfer) son usados como transferencia de datos para los Microengines y localidades externas al Microengine (por ejemplo DRAM, SRAM, etc) Hay cuatro tipos de transferencia de registros: DRAM y SRAM, tanto de lectura como escritura cada uno. Típicamente, los accesos a unidades externas se realizan en respuesta a las instrucciones ejecutadas por los Microengines.

Los registros del *Next Neighbor*, cuando son usados como fuente de una instrucción, provee de operándo a la ejecución de datos. Los registros NN en un Microengine pueden ser escritos por un Microengine Adyacente o el mismo Microengine.

Memoria Local LM. La memoria local es un almacenamiento direccionable localizado en el Microengine. LM es tanto de lectura como de escritura exclusivamente debajo del control de programa. LM provee de operandos al *exclution data path* como una fuente, y recibe resultados como un destino. La localización del LM seleccionado es basada en uno de los registros de LM_ADDR el cual es escrito por la instrucción local_csr_wr.

Bloques de Hardware Especial en el ME. El ME también provee de los siguientes bloques especiales de hardware para poder asistir en varias tareas del procesamiento de paquete:

- Unidad CRC. Procesa 16 y 32 bits CRC. Esto acelera el desempeño de una aplicación ATM AAL5 SAR, además de poder realizar un hash de 16 bits.

- Generador de números pseudo aleatorios. Asiste en algoritmos de QoS para evitar congestionamiento, WRED, RED.
- Timer. Asiste en el soporte de métricas, políticas, rate shaping functionality es requerido en IP DiffSer y los servicios de administración de tráfico ATM
- Unidad de Multiplicación. Asiste en los bloques QoS como políticas, simplificación de congestionamiento, etc.

DDR DRAM

El controlador de memoria es responsable para el acceso a los bloques del chip DDR DRAM y provee un mecanismo para otras unidades funcionales. Los tamaños de DRAM soportados son: 64Mb, 128Mb, 256Mb, 512Mb y 1Gb; el canal DRAM puede ser usado por un sencillo o doble lado DIMM.

Un espacio de 2 GB está apartado para DRAM. El espacio de memoria está garantizado de ser contiguo desde la perspectiva del software. Si menos de 2GB de memoria esta presente.

Escritura y lectura a DRAM son generados por Microengines, XScale y el bus del PCI. Estos son conectados a controladores a través del Bus de comandos y buses push y pull. ECC (Código Corrector de Errores) es soportado, pero puede ser deshabilitado.

SRAM

El procesador de Red tiene dos controladores SRAM independientes, donde cada uno soporta *QDR pipeline synchronous static* RAM (SRAM) y coprocesadores que se adhieren a la señal de QDR. Se puede conectar al SRAM por los microengines, el Intel Core XScale y la unidad de PCI.

La memoria lógica tiene 4 bytes de ancho (32 bits); físicamente los pines de datos son dos bytes de ancho los cuales ocupan doble señal de reloj. La paridad de byte es soportado. Cada uno de los bytes tiene bit de paridad, el cual es asignado cuando el byte es escrito y es revisado cuando el byte es leído. También cuenta con la habilitación de bytes el cual selecciona qué bytes va a asignar cuando va a escribir menos de 32 bits.

Estructura de Comandos para las Colas

La habilidad de encolar y desencolar datos a una razón rápida es la clave para cumplir con el desempeño de la transferencia de paquetes. Este es un problema difícil, ya que así como envuelve referencias de memoria que deben cambiar muy rápido. El SRAM incluye un controlador de estructura de datos (Llamado el Q_array) y un controlador lógico para poder desempeñar eficientemente el encolado y desencolado. El Q_array tiene 64 entradas, donde cada una puede ser usada de la siguiente manera:

- Descriptor de Cola como Lista encadenada
- Cache de la lista encadenada recientemente usada
- Descriptor del Anillo

Media and Switch Fabric Interface

El *Media Switch Fabric* (MSF) es usado para conectar el IXP2400 a la interfase de capa física (PHY) y/o switch fabric. Vendría siendo la primera interfase que se topa el paquete antes de llegar a la memoria y ser procesador. El MSF contiene las siguientes características:

- Buses separador e independientes para la recepción y transmisión de datos (32 bits).
- Cada bus puede ser configurado de manera independiente.
- Una interfase de bus configurable: el bus puede funcionar como un bus de 32 bits o puede ser canalizado a dos buses independientes de 16 bits, o cuatro buses de 8 bits, o uno de 16 y dos de 8 bits.
- Cada canal puede ser configurado para operar cualquiera de los siguientes modos: UTOPIA, POS-PHY, SPI3 o CSIX. El bus puede opera de 25 a 125 MHz.

Scratchpad and Hash Unit

La unida de Scratchpad provee 16 KB de un chip de memoria SRAM que puede ser usado para operaciones de propósito general por el Intel Core XScale y el ME. El scratch también provee 16 anillos de hardware que pueden ser usados para comunicar entre el ME y el core.

PCI, Intel XScale Periféricos y Monitores de desempeño

El IXP2400 soporta un bus PCI de 64 bits. El PCI puede ser usado para conectarse a un host o un periférico de PCI. La frecuencia del reloj del bus de PCI es de 66 MHz.

La interfase de periféricos tecnológica de Intel XScale (XPI) es usada para conectar el IXP2400 al controlador de interrupciones, Timers, UART, Propósito General IO (GPIO) y la ROM FLASH.

La unidad de monitoreo de desempeño cuenta con contadores que pueden ser programados para contar los eventos en hardware de un seleccionado chip. Estos contadores son usados para analizar y sintonizar el desempeño del software corriendo en el chip.

3.5. Ejemplo de Otros Procesadores de Redes

Entre 1990 y cerca del 2000 los procesadores de redes se han convertido de una curiosidad interesante a una producción en masa. Una vez conocidos los beneficios de su producción, la demanda creció rápidamente. Para el 2004, cerca de 30 vendedores ofrecían a los procesadores de redes como productos.

Como era de esperarse cada producto comercial cubre un amplio rango de arquitecturas. En una mano los diseñadores se enfocaron a tener varios coprocesadores de propósito general y por otra a contener chips de propósito general.

En los sub-capítulos anteriores se ha hecho mención al procesador de red de Intel IXP2400 ya que fue con el que se trabajo y desarrollo esta tesis. A continuación se presentaran una serie de sub-capítulos donde se exponen las principales características de otros procesadores de redes.

3.5.1. Procesador Funcional Agere

El procesador funcional de Agere combina realmente la flexibilidad de un RISC con el desempeño de un ASIC. La arquitectura única de PayloadPlus usa tecnología patentada llamada Pattern Matching Optimization para lograr un desempeño mayor a 5x inclusive sobre un procesador RISC avanzado. Este desempeño alcanza el nivel de funcionalidad de un ASIC mientras permite la flexibilidad y programabilidad de un RISC. La arquitectura Payload Plus puede realizar estas ganancias en desempeño usando menos overhead, menos ciclos de reloj y más procesamiento de datos por ciclo

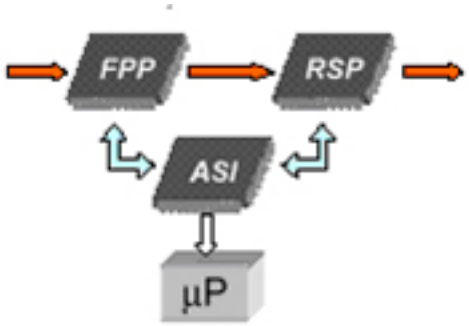


Figura 3.4: Segmentación del procesador funcional Aguere

de reloj que un RISC aumentado [1].

Comparado con los productos de la competencia basados en RISC aumentados, solo la arquitectura Payload Plus de Agere es capaz de soportar velocidades de ancho de banda de un OC-192 y en adelante mientras mantiene la flexibilidad y programabilidad de un RISC tradicional.

El procesados funcional de Aguere esta segmentado en dos componentes, el Fast Pattern Processor y el Routing Switch Processor (Figura 3.4). El FPP toma los datos de la interfaz física (PHY) y realiza un reconocimiento de protocolo, clasificación y reensamblaje. El FPP puede clasificar tráfico basado en información contenida en capa 2 a 7. Basado en procesamiento paralelo, el FPP puede proveer reensamblaje ATM y soportar búsqueda en tablas con millones de entradas y una gran variedad de longitudes.

Del FPP, el tráfico se mueve al RSP que maneja el encolamiento, la modificación de paquetes, el formato del tráfico y el QoS. El FPP y RSP hacen interface con el sistema Agere (ASI). El ASI en el componente de manejo del procesador y provee soporte para RMOM. El ASI también controla el movimiento de datos entre el FPP y el RSP para asegurar que se mueve a velocidades de cable.

Primer diseño de Agere:

Los procesadores Agere están dentro de los primeros procesadors de red, manejaban una velocidad de 2.5 Gbps y tenia el nombre de payload plus 2.5.

La arquitectura incluía 3 chips:

- Fast pattern procesor (FPP):

- Routing Switching Procesor(RSP)
- Agere System Interface(ASI)

Se utilizó la misma estructura básica que en la primera generación, pero todas las funciones fueron combinadas en un solo chip, lo que provocaba mejores tiempos de comunicación por el bus interno y estaba diseñado para trabajar a más velocidad (10 Gbps).

Arquitectura Externa:

Para ver la arquitectura de los procesadores agere nos vamos a enfocar en uno solo el APP550.

Entre las interfases externas del procesador se encuentran:

- Memorias: acceso para almacenar temporalmente los paquetes, filas, instrucciones y algunos otros parámetros, todos para el manejo de los paquetes y programación.
- Interfase con el coprocesador: Acceso a un coprocesador externo para ayudar al procesador en múltiples tareas.
- Interfases I/O: múltiples interfaces de entrada y salida de paquetes.
- Procesador de host (conectado por el pci bus) para conectarte con el cliente
- Scheduler: para conectarse con un Schedule externo.

En la Figura 3.5 se muestran todas las diferentes conexiones externas que se pueden encontrar en el APP550:

El "media interface hardware"es el encargado de iniciar el piceamuiento de paquetes que lleguen por el "ingress". Se encarga de dividir los paquetes en bloques de 64 bytes, envía un bloque por vez y envía información adicional como los bits que indiques cual es el primer y último bloque de un paquete, tamaño del bloque entre otros parámetros.

Memorias:

Como cualquier procesador de red el APP550 tiene una serie de diferentes memorias para las diferentes aplicaciones que se necesiten.

La tecnología utilizada en memorias es la siguiente:

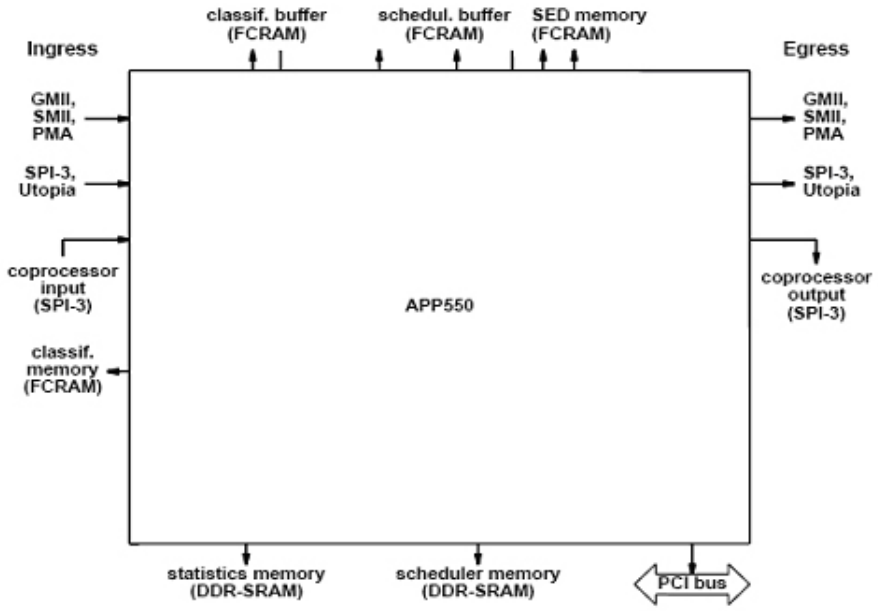


Figura 3.5: Hardware de Interfases I/O [1]

- Fast Cycle Random Access Memory (FCRAM): Bajo ciclo de reloj que permite un almacenamiento rápido de una secuencia de bytes. Esta memoria es usada principalmente para el almacenamiento de paquetes.
- Double Data Rate Static Random Access Memory (DDR-SRAM): Respuesta rápida para acceso de información de control y otras aplicaciones. Es usada para almacenar información de control y de tablas.

Arquitectura interna

En la Figura 3.6 se muestra la arquitectura interna del procesador:

Cada uno de los motores de operación son los siguientes:

- Input interface: recibe paquetes de coprocesador, fábrica y pci. Es la encargada de llevar los paquetes de ingress al clasificador. Es configurable para SPI-3, UTopia 2 o3, PLATO. Y es configurable hasta para 256 direcciones multiPhy (MPHY)
- Pattern Processing Engine (PPE): Es el chip que se encarga de la clasificación, utiliza una architetura no convencional (no utiliza fetch), es programable utilizando FPL, utiliza paralelismo y es invocado directamente cuando un paquete llega. Accesa diferentes memorias: Classifier PDU Buffer (CPDUB), Classifier Program Memory (CPM), Classifier Control Memory (CCM).

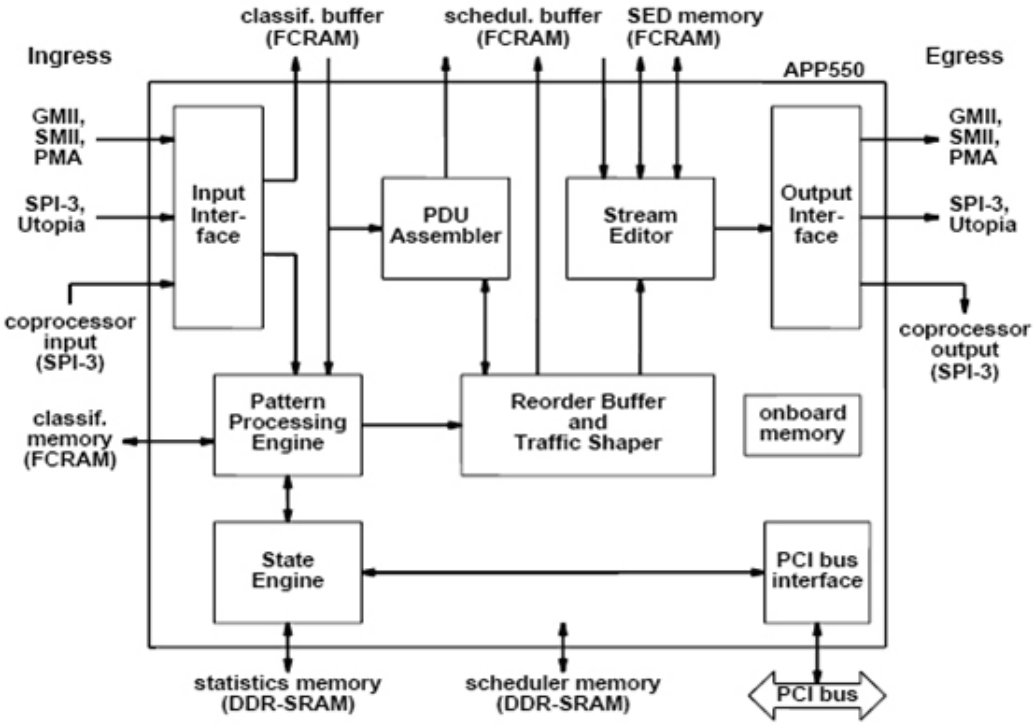


Figura 3.6: Arquitectura Interna del APP550 [1]

- PDU Assembler: Su principal tarea es la de combinar bloques para formar paquetes completos. Bloque de manejador de Tráfico: State engine: es invocado por el PEE y entre sus funciones esta la de colectar estadísticas, interfase con el cliente y configura y controla otras unidades. Esta unidad es programable con el lenguaje ASL.

- Reorder Buffer: es requerido para algunos protocolos del bloque de clasificación debido que utilizan paralelismo, además que beneficia algunos otros protocolos. Es invocado después de la clasificación y no es programable. Su función es proveer transferencia entre el clasificador y el manejador de tráfico.

- Manejador de buffers: Recibe paquetes del clasificador para decidir si el paquete se enlista o se desecha. Toma su decisión basado en uso de la memoria, problemas y resultados de la clasificación.

- Traffic Shaper: Es invocado por los paquetes que están por salir, se encarga de manejar las filas de salida. Es programable por el lenguaje ASL

- Stream editor(SED): maneja los cambios pertinentes para los paquetes de salida, como modificación y actualizaciones, como por ejemplo calcular el checksum o el cambio de un dirección MPHY.

Lenguaje de programación de Agere

Para permitir a los proveedores de hardware un rápido desarrollo y soluciones basadas en el procesador de Agere, Agere provee un antiguo ambiente de desarrollo llamado Functional Programing Language (FPL). Este lenguaje es un verdadero lenguaje de cuarta generación que ha sido diseñado para ser altamente intuitivo. Las instrucciones están codificadas como un protocolo.

El FPL de Agere provee a los desarrolladores con una enorme ventaja sobre el complicado código de RISC y permite a los desarrolladores acortar el tiempo de mercado.

El ambiente de programación de Agere también trae consigo un suite entero de testeo y simulación que permite a los desarrolladores probar totalmente aplicaciones antes de lanzarse. El IDE de Agere provee a los desarrolladores un completo SDK incluyendo un generador de tráfico y herramientas para simulación del mundo real. Más adelante hablaremos más sobre este lenguaje de programacion y sus características.

3.5.2. CISCO

En el 97, Scott y su equipo empezaron su camino rumbo a la que posteriormente se convirtió en la arquitectura Parallel Express Forwarding. Su idea era hacer pipelines paralelos. Y, en resumen, lo que querían era hacer un chip con 16 procesadores. El resultado de los esfuerzos del 97 culminó 18 meses después con la Toaster, el nombre no oficial que se le dio al conjunto de procesadores que se han estado utilizando en modelos de alto rendimiento de enrutadores de Cisco.

Mucha información de este procesador aún no se encuentra públicamente disponible, debido a que parte de la estrategia de Cisco incluye el no dar a conocer todos los detalles.

Arquitectura PXF

La arquitectura PXF logró acelerar ciertas tareas del procesamiento de paquetes que se necesitan para las necesidades actuales de la Internet. La idea detrás del PXF es balancear la velocidad de reenvío de paquetes con la flexibilidad necesaria para dar soporte a varias aplicaciones, facilitando la programación.

En una aplicación típica, como la es un enrutador, un dispositivo con esta arquitectura puede alcanzar a procesar millones de paquetes por segundo, respetando la idea

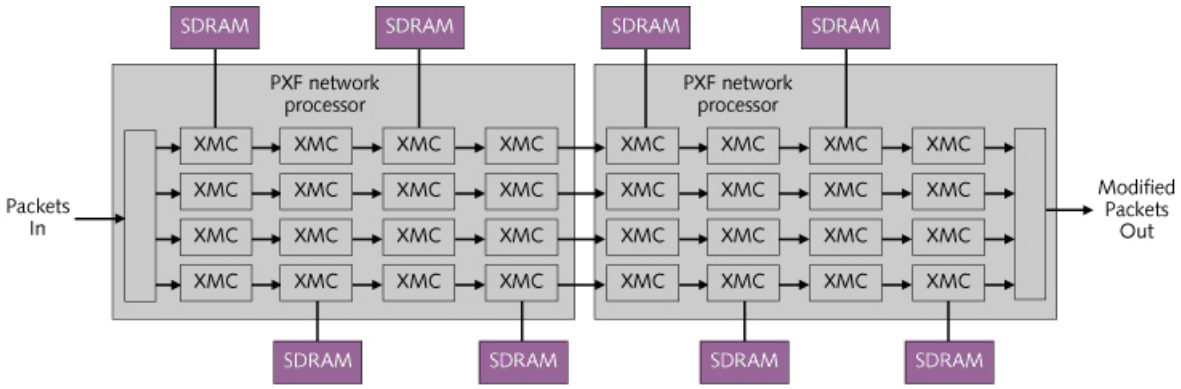


Figura 3.7: Performance Routing Engine PRE [4]

de una robusta flexibilidad.

El "engine"de reenvío del PXF permite procesar en paralelo con técnicas de pipeline. De hecho, parte del diseño de esta arquitectura adapta algoritmos del Cisco Express Forwarding, además de incluir diseño ASIC programable en varios elementos para lograr mayor ancho de banda, intentando mantener un nivel óptimo de programación [4].

Arquitectura

La panorama general de la arquitectura de las primeras dos versiones de Toaster se muestra en la Figura 3.7:

El PRE que se usa en modelos no tan recientes en enrutadores cisco ocupa dos procesadores PFX. Cada columna representa alguna función del procesamiento de paquetes. El primer procesador se para las funciones del "ingress", y el segundo para el "egress".

Se usan dos procesadores de red PXF en cada Performance Routing Engine (PRE), generando cuatro pipelines. Para el caso del Toaster 3, la arquitectura cambia significativamente (Figura 3.8).

Por otra parte, la versión de PXF usada es la Toaster 3, la cual, en vez de formar un arreglo de 4 x 4, es un arreglo de 8 x 2 por ASIC. Este nuevo diseño da una mayor capacidad en forma nativa.

Cada columna de CPUs tiene 16 CPUs y se suele manejar un espacio dedicado de memoria. Por ejemplo, para el enrutador 10000 Series PRE-2, cada columna cuenta con

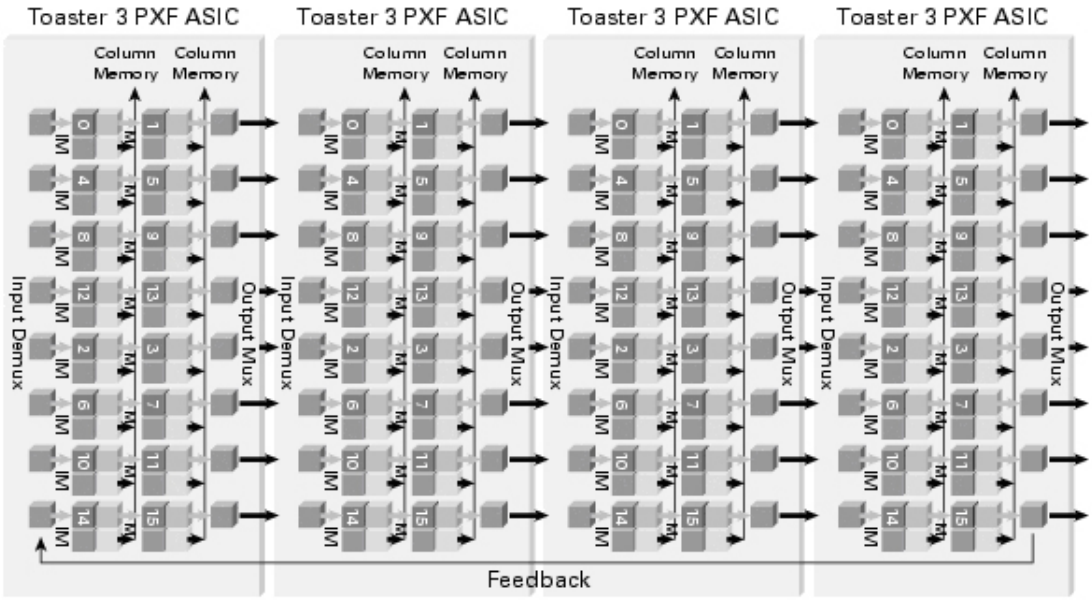


Figura 3.8: Performance Routing Engine 2 PRE2 [4]

128Mb de memoria ECC-protected FCRAM, donde las estructuras de datos y tablas de búsqueda necesarias para el procesamiento de paquetes y reenvío se almacenan. Cuando el procesador de enrutado envía una actualización de tabla de reenvío al más rápido FPGA, la memoria de la columna almacenará la información de reenvío para uso de esa columna específica del PXF.

Debido a que cada columna de CPUs en el PXF puede ser programada para proveer una serie específica de funciones, la columna de memoria para cada columna será única. De cualquier forma, debido a que los ocho CPUs en la misma columna darán el mismo procesamiento a los paquetes, pueden manejar una sola columna de memoria para que solo una copia del reenvío de información para cualquier función.

Algunas veces, después que un paquete ha pasado por las ocho columnas de CPUs, se requiere de procesamiento adicional para el paquete antes de que esté listo para transmitirse. En este caso, el paquete puede ser enviado de vuelta por las ocho columnas en el arreglo PXF. A este segundo paso se le conoce como "retroalimentación".

Algunas características pueden requerir retroalimentación para todos los paquetes, mientras que otras pueden requerir una retroalimentación basada en la combinación de funciones que necesitan ser aplicadas al paquete. La lista de funciones que pueden resultar en una retroalimentación puede, potencialmente, cambiar con diferentes versiones de microcódigo PXF y de las imágenes del software IOS de cisco.

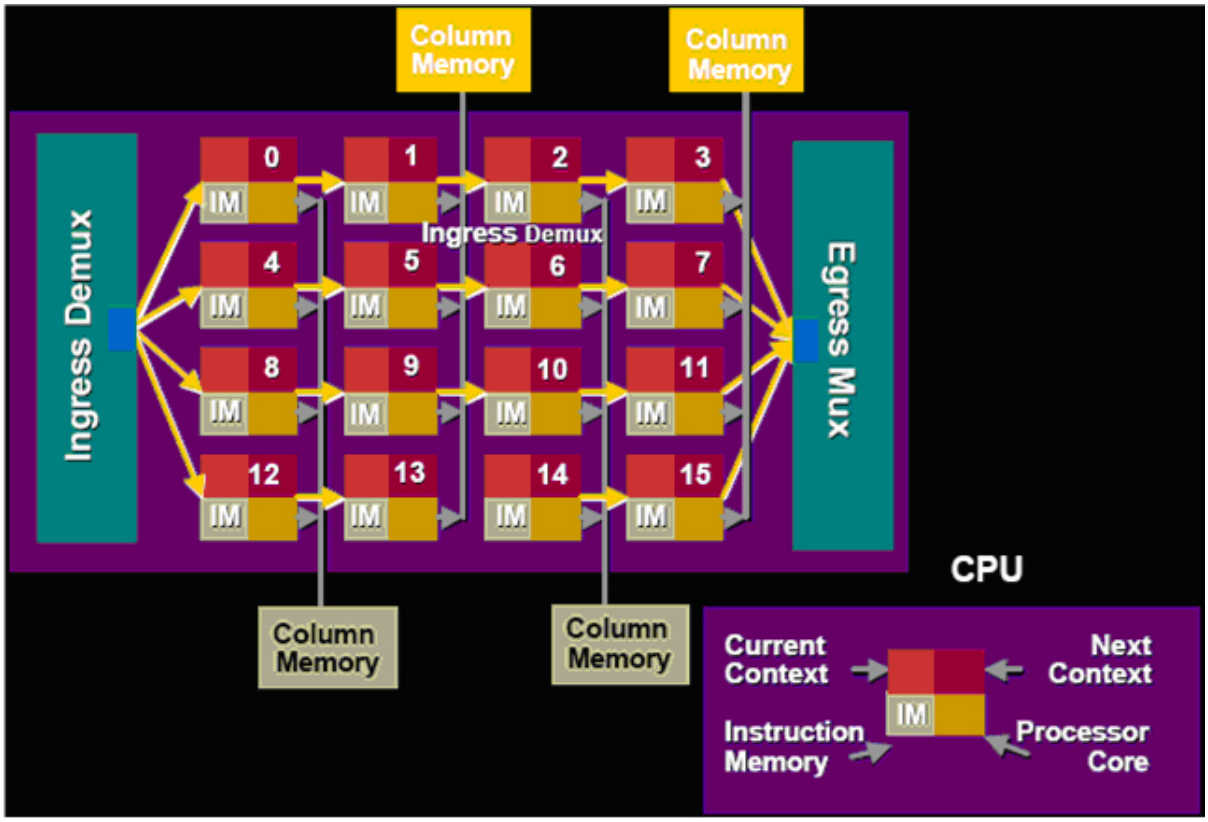


Figura 3.9: Arquitectura PXF [4]

Procesador PXF

Como se ha visto en clase de microprocesadores, parte fundamental del desempeño de los sistemas obedece a la arquitectura. Sin embargo, el procesador debe dar soporte para que esta arquitectura no tenga cuellos de botella.

Por otra parte, ya se ha revisado, en el apartado de introducción, cuáles son los cores de la arquitectura PXF. Lo siguiente es mirar aún más de cerca la parte que más representa la arquitectura.

En la Figura 3.9 Se muestra el complejo PXF. La arquitectura indicada es la de las versiones 1 y 2 del Toaster. Se aprecia la memoria compartida por columna y la forma en que fluyen los paquetes, desde el ingress hasta el egress.

En la sección de Arquitectura se explicó como es que una tarjeta externa, como la Cisco Cobalt 2, sirve de interfaz para depositar en el ingress los paquetes y tomarlos del egress. Se podría pensar que esta interfaz es análoga a la que realiza la SMF en la

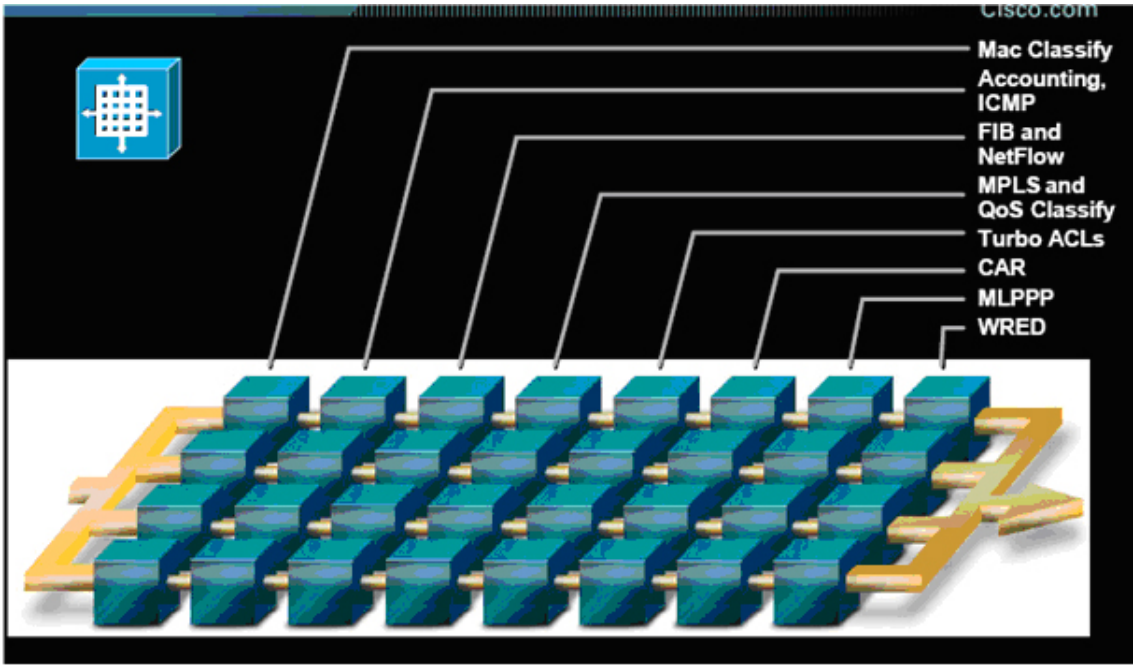


Figura 3.10: Arreglo de 4 filas donde cada columna tiene una función diferente del procesador PXF [4]

Intel IXP. Interesantemente, es el PXF quien controla a esta interfaz.

Por otra parte, cada core se compone de 4 elementos:

- Instruction Memory. Como se ha comentado, cada columna de cores puede ser programada, con ayuda del IOS de Cisco. La idea básica es que cada columna representa una "característica". Los paquetes van recorriendo columnas hasta terminar o, si es necesario, dar vueltas en la Red para recibir el procesamiento que requieren.
- Current Context. Dentro de lo que graba acá se encuentra la información requerida para el ensamblado del paquete original (recordar que al paquete se le separa del encabezado como parte del procesamiento).
- Next Context.
- Core del Procesador.. En cuenta a las funciones que se realizan en cada columna, como se puede corroborar en la lista de características soportadas por el procesador (Figura 3.10)

3.5.3. Motorola

La arquitectura del C-5e esta especialmente diseñada para aplicaciones de comunicación de red. El procesamiento de datos, tablas y funciones de manejo de filas esta integrado en esta arquitectura. Cuenta con chips de interfase de comunicación, chips de memoria (para payload, tablas) y muy poca necesidad de control por software lo cual lo convierte en un NP muy poderoso que por si mismo podría implementarse para manejar información altamente multiplexada, multipuertos, switches de múltiples protocolos, multiplexores y concentradores Sin embargo, si se necesita aún más poder de procesamiento de información o puertos se pueden interconectar varios C-5e por medio de un switching fabric.

A continuación se menciona las características de operación principales del NP de Motorola:

Poder de procesamiento

- Frecuencia de operación: 300 MHz
- 5 Gbps de bandwidth (non-blocking throughput)
- Más de 4500 MIPS (millones de operaciones por segundo) para manejo de servicios en el throughout del protocol stack.
- Más de 15 millones de paquetes por segundo por transmisión por cable.
- 17 RISC programables (Para cell/packet forwarding).
- 32 procesadores de datos seriales (para proceso de ráfagas de bits o "bit streams").
- Más de 133 millones de acceso a tablas por segundo.
- 3 buses internos con velocidad de 68 Gbps

Circuitos integrados

- 840 paquetes BGA (Ball Gris Array)
- 16 procesadores "Channel Processors"que incluyen:
- Embebido OC-3c, OC-12, OC-12c SONET framers
- Interfaz MAC programable
- Núcleos RISC

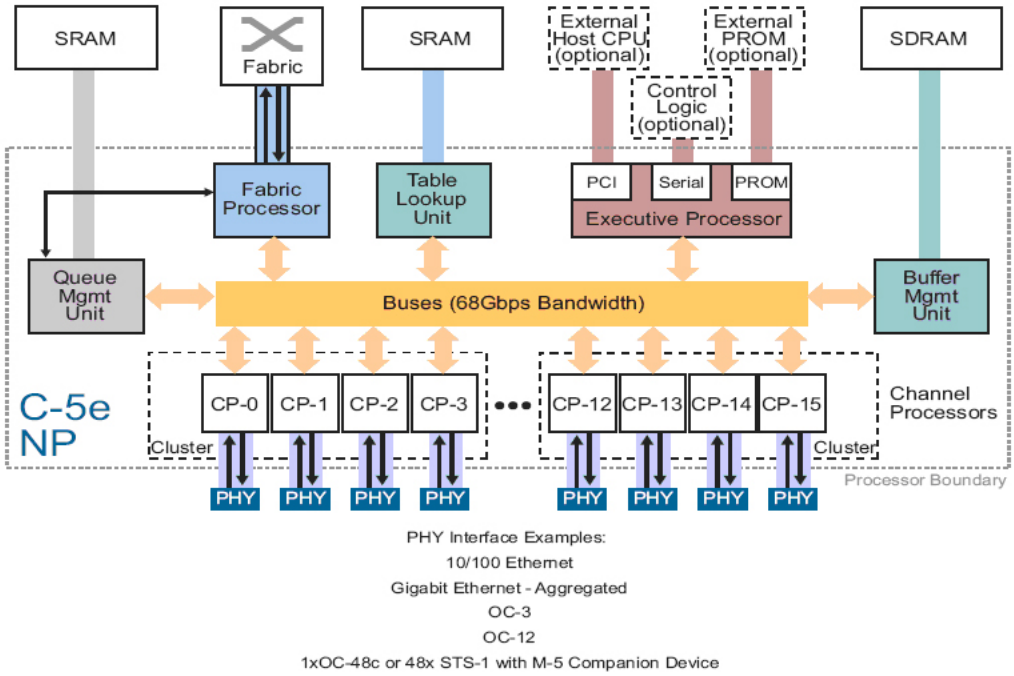


Figura 3.11: Diagrama de Bloques del Procesador de Red C-5

- Interfases Pin PHY programables
- Coprocesadores Embebidos para accesos a tablas (clasificación), manejo de buffers (payload control) y manejo de filas (implementación CoS/QoS)
- Fabric Processor y puertos dedicados
- Procesador Ejecutante RISC embebido (Embebed RISC Executive Processor).
- Interfase PCI bus integrada de 32 bits 33/66 MHZ

También dentro de sus características que el voltaje del Core es de 1.2 V y que consume 9.2 W típicamente. Su precio es de aproximadamente 350 dólares.

Este es un diagrama de Bloques del NP en el que se pueden observar las características de arquitectura arriba mencionadas Figura 3.11.

Modos de operación

La arquitectura del C-5e le permite trabajar en 3 modos de operación los cuales son: Modo dedicado (Single Channel Mode), en modo pipeline (Pipeline Channel) o en Modo Paralelo (Aggregate Channel Mode). Esto le permite al usuario ajustar el

funcionamiento del procesador de red de acuerdo a las necesidades que tenga, preemitiéndole moverse entre un mejor poder de procesamiento o una mejora en su ancho de banda.

- Modo dedicado (Single Channel Mode) Cada uno de los CP (Channel Processors) trabaja independientemente en modo full duplex.
- Modo Pipeline (Pipeline Channel Mode) Para lograr el escalamiento de los procesadores para una aplicación en particular, lo que se realiza es un link entre ellos para una misma ráfaga de bits; logrando así que el poder de procesamiento de cada uno de ellos se ejecute sobre cierta cantidad de información.
- Modo Paralelo (Aggregate Channel Mode) En este modo los 16 CP's del NP son agrupados en 4 grupos de 4 llamados "clusters", esto permite al NP compartir recursos como IMEM y DMEM y poder así lograr el paralelismo corriendo dichos clusters. Cada uno de ellos puede trabajar en una misma aplicación o trabajar en cosas totalmente independientes una de la otra.

Interfases soportadas

El C-5e puede trabajar con estándares de transmisión de datos seriales y paralelos manejando velocidades de DS1(1.54MBPS) a velocidades Ethernet de (1000 MBPS). Las siguientes son las interfases estándares soportadas:

- 10/100 Mb Ethernet (RMII)
- 1 Gb Ehternet (GMII y TBI)
- OC - 3C
- OC - 12
- OC - 48c
- OC - 48
- 100 Mbit Fibre Channel

Jerarquía de procesadores

Channel Processor:
El c-5e de Motorola tiene un Channel Processor programable y dedicado para cada una de las 16 interfases de procesamiento con las que cuenta la arquitectura para manejar paquetes de información. Cada uno de estos CP tiene un Serial Data Processor (SDP)

y un Channel Processor RISC Core (CPRC), los cuales de manera conjunta procesan información por medio de memorias de propósito general llamadas "Instruction Memory"(IMEM) y "Data Memory"(DMEM).

Executive Processor (XP):
El Executive processor sirve como un recurso centralizado de computo para el procesador de red C-5e y administra las interfaces del sistema. Una de las interfaces de sistema que maneja es la PCI bus, el cual se utiliza generalmente para la comunicación con un el host processor externo. Si se presenta un host processor puede proveer coordinación device-wide (por ejemplo, entre varios NPs C-5e), administración de red, señalización y podria posiblemente construir una tabla de ruteo para el dispositivo del cual el NP C-5e es parte. El XP puede realizar muchas de estas funciones por el mismo. El XP tiene acceso al bus global interno, ring bus y el payload bus.

Las funciones típicas del XP incluyen:

- Inicialización del chip y descarga de código.
- Mantenimiento de la tabla de ruteo y de switcheo (ya sea construyendo tablas o importando actualizaciones desde el host).
- Obtención de estadísticas desde el CP DMEM y el TLU.
- Detección y recuperación de fallas.
- Funciones de reenvío de trayectoria no crítica.

Fabric Processor (FP):
El Fabric Processor provee un puerto con amplio ancho de banda para la segmentación y reensamblado de PDUs a una velocidad mayor que OC-48. Se comporta como un puerto de interfase de red de alta velocidad (más de 125 MHz para dos rutas de 32 bits de datos) con funcionalidades avanzadas que permiten al NP C-5e interactuar con una solución de switcheo específico o con un mecanismo de switcheo. El FP puede ser configurado para formar 7 tipos diferentes de interfaces de mecanismos, entre las cuales: CSIX-L1, UTOPIA-1, -2, -3, PRIZMA, Power X (CSIX-LO), UTOPIA3 como un M-5. La flexibilidad de programación de FP permite soportar estándares y celdas de formato específicas de propietario.

El FP realiza mapeo y administración de flujo para y desde el mecanismo de swicheo. Puede recibir más de 159 flujos al mismo tiempo y soporta transmisiones con más de 128 prioridades, los flujos simultáneos pueden ser configurados como: una matriz de 32 puertos con cuatro niveles de prioridad o una matriz con 16 puertos con 8 niveles de prioridad. Estos flujos soportan:

- Topologías Unicast y Multicast.
- Control de flujo en niveles.
- Administración de congestión y control de flujo End-to-End.
- Segmentación y reensamblado de PDUs para y desde celdas de mecanismos uniformes configurables para el propósito de una mejor utilización del mecanismo y de Quality of Service (QoS).

3.6. Tendencias de los Procesadores de Redes

Debido a la gran versatilidad de los procesadores de redes se ha dedicado un subcapitulo para mencionar las tendencias de estos mismos. Se han categorizaron las aplicaciones de red que puede implementar un procesador de red en tres grupos funcionales: Administración de tráfico, grupo de calidad de servicio (TQG), y grupo del procesamiento de paquetes (PPG) [11]. En la Figura 3.12 se hace mención de las aplicaciones mas relevantes de cada grupo.

Administración de tráfico:

TGQ tiene una serie de aplicaciones relacionadas al enrutamiento, scheduling, colas, switching, señal y calidad de servicio. Estas aplicaciones contiene de ambas: procesamiento de control lineal y procesamiento de datos lineales. Las principales aplicaciones para esta categoría son: Wieght Fair Queuing WFQ y Random Early Detect RED, las cuales son soluciones al control de congestionamiento. En general, la congestión ocurre cuanto los paquetes llegan más rápido de lo que el ruteador o switch lo puede reenviar a otro enlace. Los dos algoritmos más representativos para el control de congestionamiento el algoritmo de scheduling y el algoritmo de administración de colas. El administrador de scheduling determina que paquete será el siguiente enviado y es usado primordialmente para administrar el ancho de banda a través de la cantidad de flujo.

Seguridad y Procesamiento del medio:

Así como el comercio industrial a crecido la seguridad, aplicaciones con cuentas como lo son el firewall, control de admisión, aplicaciones de encriptado, etc. se han vuelto bien conmún. Con el incremente en el ancho de banda, la demanda para la calidad del servicio en multimedia también ha incrementado. Manipulación de paquetes, traducciones del medio, H.323 y muchas otras aplicaciones de codificación son de im-

Grupo	Aplicaciones	Dato Lineal	Control Lineal
TQG (Administación de Tráfico y Grupo de Calidad de Servicio)	Routing	x	x
	Scheduling	x	x
	Content-based switching	x	x
	Weighted Fair Queuing	x	x
	Traffic Shaping	x	x
	Load balancing	x	x
	VLAN		x
	MPLS	x	x
	RSVP	x	x
	DiffServ	x	x
	IntServ	x	x
SMG (Grupo de Seguridad y Procesado del Medio)	Block Cipher Algorithm	x	
	Message digest algorithm	x	
	Firewall application	x	x
	IPSec	x	x
	Virtual Private Network	x	
	Publick encryption	x	
	H.323	x	
	Media transcoding	x	x
	Duplicate data supression	x	
PPG (Grupo Procesado de Paqute)	IP-Packet fragmentation	x	
	Packet encapsulation	x	
	Packet marking/editing	x	
	Packet Calssification	x	
	Checksum calculation	x	

Figura 3.12: Grupo funcional de los Procesadores de Redes [11]

portancia para el NP.

Grupo de Procesamiento de Paquetes:

Lo que incluye el procesamiento de paquetes es: la fragmentación de paquetes IP, enmarcado de paquetes, modificación y clasificación de paquetes. La mayoría de las aplicaciones son procesamiento de datos lineales.

Capítulo 4

Caso Estudio: Implementación del Laboratorio de Procesadores de Redes (IXP2400)

4.1. Historia de trabajo y desarrollo de prácticas con computadoras x86

La implementación de un Laboratorio para el curso de Procesadores de Redes es indispensable, ya que la mayoría de los conceptos teóricos no iban a ser aterrizados ni lograrían a alcanzar el grado de comprensión que se llegaría a tener con un laboratorio práctico. El laboratorio tiene como principal objetivo transmitir el funcionamiento de un Procesador de Red bajo ambientes robustos de la industria.

Para que la tarjeta pudiera arrancar y cargar el código de desarrollo requería del sistema operativo Hardhat que el libro: *Network System Design using Network Processors* recomienda. Para esto se contactó a MontaVista, empresa líder en proveer material de grado comercial en el desarrollo de plataformas Linux para aparatos inteligentes e infraestructuras de comunicación. El único inconveniente con MontaVista es que no brinda soporte a instituciones educativas, por lo que se decidió trabajar con el Kit de desarrollo, el cual no tiene habilitado todos sus módulos de operación. Esto provoca que la tarjeta solamente trabaje a un tercio de su capacidad.

Para poder usar este kit de desarrollo se tuvo que aprender a usar la herramienta de *cross-compiler* de Dan Kegel. Esta herramienta ayuda a construir programas para otras arquitecturas. En otras palabras el *cross-compiler* traduce nuestro programa para que sea entendido por otra arquitectura. Cabe mencionar que este concepto se les enseñó a los alumnos del semestre donde realizaron una práctica en la que se crooscompilaba una imagen de arranque para la arquitectura computacional del IXP2400 (Strong-Arm).

Los alumnos realizaron los siguientes pasos:

1. Bajar y descomprimir la herramienta de Kegel.

2. Editar archivo: *gcc-3.4.-glibc-2.3.5.dat*, para compilar una versión de Linux 2.6.16.

3. Compilar el *crosstool* para una arquitectura ARMEB.

4. Se añade el *path* de compilación para un ARMEB, en el archivo: .bash_profile

5. Descarga y descompresión del kernel: Linux 2.6.15.4.

6. Se aplica el parche al Linux 2.6.15.4 para añadir librerías de compilación de un ENP-2611.

7. Se añaden el comando: armeb-unknown-linux-gnu- al Makefile del Linux-2.6.15.4.

8. Descargar la imagen crosscompilada (zImage).

Después de crosscompilar la imagen y descargarla cuando se estaba corriendo el sistema operativo, se comprendió que se necesitaba la instalación completa del sistema operativo si se deseaba usar al 100 % de la capacidad del NP. Debido a que no se tenía el funcionamiento básico de los procesadores de redes para levantar una topología en la que los alumnos pudieran desarrollar o hacer pruebas de código, se desarrollaron 4 prácticas que se enfocan en enseñar los conceptos básicos del procesamiento de redes haciendo uso de una NIC convencional con una computadora x86.

Estas prácticas fueron desarrolladas en base a lo visto en el curso y trata principalmente los siguientes temas:

1. **Desarrollo de un HASH.** Los alumnos desarrollaron código en C en donde se aplican los conceptos básicos de una tabla HASH, la cual es utilizada por cualquier procesador de red para eficientizar búsquedas.

2. **Creación de un Puente.** Para la comprensión completa de un procesamiento de paquete, se requiere entender este proceso desde que llega a una interfase hasta donde sale de la misma. Por lo que se les encargó a los alumnos el desarrollo de un Bridge.

3. **Desarrollo de un Clasificador de paquetes.** Se les encargó a los alumnos el desarrollo de código en C, donde se clasificaban los paquetes y se pasaban a otras interfases, de acuerdo al protocolo que traía el paquete (TCP, UDP, etc.).

4. **Desarrollo de un Traductor de Direcciones de Red (NAT).** Esta práctica se enfocó en la habilidad de desarrollo y la comprensión total de un traductor de direcciones IP, que se usa en toda red de área local (LAN).

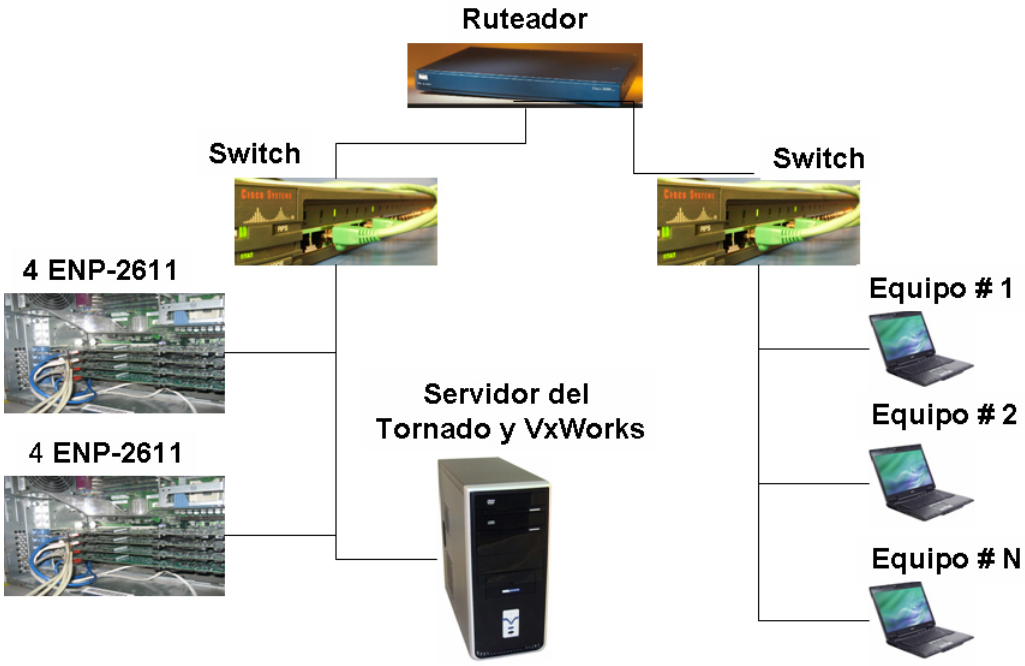


Figura 4.1: Topología del Laboratorio de NP

Estas prácticas tienen como finalidad dar una visión básica de lo que un desarrollador en procesadores de redes necesita saber, tanto de manera práctica como conceptual. La mejor manera en que los alumnos comprendan esta complejidad de desarrollo y se metan al código es encargando tareas en las que se enfrenten al reto de desarrollar funciones, clases, métodos, estructuras de datos, etc., donde implementen lo conocido e incrementen sus habilidades aplicando los conceptos de la materia.

4.2. Topología del Laboratorio de Procesadores

Debido al problema donde los procesadores de redes no corrían al 100 % se decidió contactar otra compañía que brindara soporte a Universidades como lo es WindRiver, empresa que desarrolló el Sistema Operativo de VxWorks y es líder a nivel mundial en *device software optimization* (DSO).

Haciendo uso del sistema operativo de WindRiver se buscó implementar una topología como la que se muestra en la Figura 4.1.

La topología tiene como principal finalidad establecer una conexión entre el equipo utilizado por los alumnos y los procesadores de redes. Con esta base los alumnos fueron capaces de cargar software en los NPs y de hacer pruebas de implementación. Esta

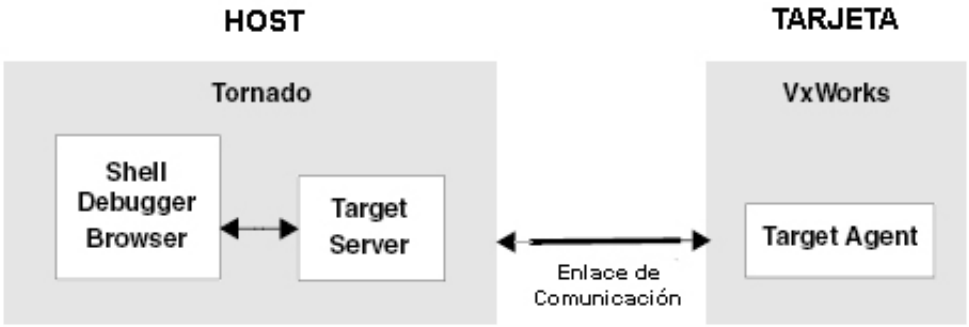


Figura 4.2: Software de Comunicación: NP IDE [5]

topología requería de cómputo como:

- Un servidor con ranuras PCI-X
- Dos servidores de licencia convencionales
- Un routeador y un switch

Al final de cuentas el conseguir todo este equipo hizo que se atrasara el desarrollo de este laboratorio, pero no imposibilitó su instalación y uso.

Cada NP tiene la capacidad de correr dos sistemas operativos: Hardhat y VxWorks donde ambos pueden descargar código a través del puerto RJ-45 usado el programa de desarrollo: Tornado. El VxWorks en específico implementa el *Target Agent* para usar la IP del NP y manejar la conexión entre la tarjeta y el software de desarrollo. Y la computadora que está corriendo el Tornado, usa el *Target Server* para usar la IP y puerto del Tornado, como se muestra en la Figura D.50.

Para cada NP, se tiene una IP asignada, además de que cada vez que reinician se descarga la imagen del sistema operativo VxWorks del servidor de Licencias.

4.3. Objetivos del Laboratorio

Se ha trabajado en la implementación del laboratorio de Procesadores de Redes en el departamento de Ciencias Computacionales por más de un año en donde las principales metas han sido:

- Que los estudiantes alcancen un alto grado de comprensión en la funcionalidad y la implementación de aplicaciones en el Procesador de Red. Además de que se reafirman los conceptos aprendidos en el curso de Procesadores de Redes.
- Los alumnos desarrollen habilidades para la comprensión de código específico para la programación del procesador de redes. Además del uso de simuladores sofisticados para correr ejemplos de aplicaciones básicas del NP.
- Los alumnos desarrollen el trabajo en equipo y administración de proyectos.

4.4. Estructura del Laboratorio

El laboratorio tiene tres componentes principales.

- La implementación de un programa real desarrollado por el equipo de trabajo IXA SDK de Intel.
- Una serie de prácticas, sesiones de laboratorio y tareas que en conjunto con las prácticas ya diseñadas, usando una NIC convencional, se acelere el proceso de aprendizaje de los alumnos en un corto tiempo.
- El último componente es el curso que se imparte donde se les asignan papers que hablan de la arquitectura que trae integrada la tarjeta, el procesador IXP 2400 y comparaciones de mercado en procesadores de redes. Este curso consiste de la lectura y comprensión del libro de Comer que habla de los procesadores de redes haciendo un principal énfasis en el procesador IXP2XXX de Intel.

Más adelante se hablará más en específico de algunos de estos componentes, dando explicación de lo que se hace y como funciona.

4.5. Prácticas del Laboratorio

Ya teniendo la topología del laboratorio instalada, se empezó a desarrollar el laboratorio al mismo tiempo se empezaba a implementar. El orden en que se planearon las prácticas fue de crecimiento en aprendizaje, donde las primeras que se enseñaban eran de nivel básico y poco a poco se iba incrementando la complejidad hasta llegar a la etapa de implementación y prueba. El laboratorio se enfoca básicamente en los siguientes aspectos:

- Uso de la Herramienta Tornado.
  Herramienta que se usa para crosscompilar y bajar código a la tarjeta. Además de que tiene un compilador y simulador convencional para poder hacer debug de programas a nivel de threads.

- Uso de la Herramienta Developer Workbench.
  Se encarga de compilar código *microengine assembler*, facilitar el desarrollo de código y simular el código compilado.

- Desarrollo de Código.
  Debido a la complejidad del código y el poco tiempo para enseñanza de este mismo, se han desarrollado prácticas de nivel básico para comprender escrituras a los distintos tipos de memoria (SRAM, DRAM, Scratchpad).

- Simulación de Código.
  Se pedía la explicación del código que el instructor le proporcionaba, donde los equipos se dedicaban a la deducción de código y simulación del mismo. También se pedía la explicación de la simulación a detalle en cada práctica y cada reporte.

- Implementación de Código.
  Se descargó código en las tarjetas, donde los alumnos tuvieron contacto físico con los procesadores de redes, llegando a la etapa de prueba de código y corrección de errores de la implementación.

Nuestro enfoque es acelerar el conocimiento de los alumnos a través de las prácticas ya vistas anteriormente usando una NIC convencional en una computadora x86 junto con estas nuevas prácticas las cuales van de un nivel básico a un nivel avanzado tomando conceptos ya vistos por las prácticas desarrolladas anteriormente.

Para la comprensión de lo desarrollado en este laboratorio se va a comentar más a fondo cada uno de los puntos vistos en las prácticas:

4.5.1. Ambiente de desarrollo Integrado IDE: Tornado

Tornado es un ambiente integrado para desarrollo de software-cruzado. Este provee una eficiente manera de desarrollar tiempo-real y aplicaciones embebidas. Tornado consiste de los siguientes elementos:

- VxWorks, sistema operativo de tiempo-real con alto desempeño.

- Herramientas de desarrollo (compiladores y programas asociados)

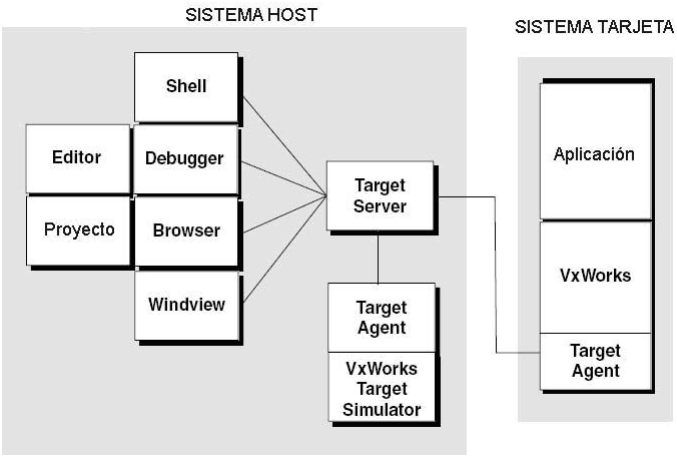


Figura 4.3: Sistema Host - Sistema Tarjeta [5]

- Ambiente de desarrollo Integrado (IDE) que facilita el manejo y construcción de proyectos, estableciendo y manejando la comunicación entre el host y la tarjeta. Corrido, depuración, y monitoreo de aplicaciones VxWorks.

Características principales del IDE:

- Un editor fuente - código integrado.
- La facilidad de un administrador de proyectos.
- Compiladores integrados de C, C++ y make.
- El *browser*, colección de datos, que ayuda a la visualización del sistema de la tarjeta.
- CrossWind, depuración mejorada con desplegado gráfico.
- WindSh, Comandos de Shell en lenguaje - C que controla la tarjeta.
- VxSim, versión integrada del simulador de tarjeta VxWorks

Tornado facilita la ejecución principalmente en el sistema host, con acceso compartido al mismo, enlace dinámico y tabla de símbolos para un sistema remoto de la tarjeta. La Figura 4.3 muestra la relación entre los principales componentes del host de Tornado y el sistema de la tarjeta. La comunicación entre las herramientas del host y el VxWorks es mediante el *target server* y el *target agent*.

4.5.2. El ambiente de desarrollo IDE: Developer Workbench

El Ambiente de Desarrollo Integrado IDE Developer Workbench facilita el desarrollo de lenguaje Microengine Assembler, desarrollado por Intel, aporta en los siguientes aspectos:

- Familiarización de Código. Debido a lo específico que es el código para el NP, el Developer Workbench facilita la identificación de funciones predefinidas en código C, además del código escrito en ensamblador.
- Identificación de código. El Developer Workbench hace distinción entre el código raíz y los códigos dependientes de este, siendo de gran ayuda por el inmenso código que se necesita para desarrollar cualquier aplicación en un procesador de red.
- Simulación de Tráfico. Esta herramienta de desarrollo tiene la capacidad de crear simulaciones de tráfico donde se le especifica el puerto de entrada, puerto de salida, tipo de paquetes a transmitir, repetición de paquetes, tamaño de paquete, etc.
- Estatus de Thread. Esta aportación del Developer Workbench ayuda a encontrar errores de sincronización de threads desplegando de forma gráfica y simultanea 64 threads que ejecutan en paralelo.
- Estatus de Memoria. Otra de las aportaciones que tiene esta herramienta de desarrollo es el desplegado de memorias Scratchpad, SRAM y DRAM, donde se le específica el rango de direcciones que desea observar.

Todas estas aportaciones que hace el Developer Workbench ayudan a la comprensión y familiarización de los conceptos vistos en clase. En las prácticas se hizo uso de esta herramienta en un basto tiempo, ya que es el compilador principal de lo que se programa en el NP. En la práctica #2 los alumnos desarrollaron un programa sencillo donde se incrementaba un espacio de memoria de forma indefinida. El objetivo era la familiarización con el código de desarrollo, la compilación y simulación en el Developer Workbench.

4.5.3. Simulación de Código

El manejar las especificaciones de simulación viene siendo demasiado útil, ya que se puede configurar de tal forma que se haga visible el error.

Durante la simulación se puede personalizar el Developer Workbench para que se despliegue de manera gráfica el estatus del procesador y de la aplicación del ME en progreso.

Entre los *Debug Views* más importantes están:

- Command Line View. Con el *command line view* se pueden establecer descargas de código, correr simulaciones, pudiendo ver el error detallado de forma interactiva.
- Thread Status View. Fue la primera interfase gráfica de usuario que se implementó en el Developer Workbench. En esta interfase se hace un desplegado de los threads que está corriendo cada microengine, teniendo la opción de encontrar en código la tarea específica que se está ejecutando en cada thread.
- Thread History. Se despliega lo que corre cada microengine en líneas horizontales. Cada línea representa un thread, por lo que el desplegado gráfico hace distinción de sus estados dependiendo del color: negro - thread en ejecución, amarillo - abortado, rojo - thread stalled, azul - microengine - idle, dotted - microengine deshabilitado.
- Memory Watch. En esta sección se hace un desplegado del contenido de las memorias de: SRAM, DRAM y scratchpad. El alumno tiene que especificar la dirección o rango de direcciones a desplegar.

El Developer Workbench cuenta con la capacidad de generar tráfico para poder simular programas en las que se procesa el paquete desde que está en el buffer de recepción hasta reenviarlo por el buffer de transmisión.

De las características del generador de tráfico cabe resaltar:

- Loggin. Se pueden capturar los datos del paquete en un archivo log a través de la interfase de tráfico o puerto.
- Stop control. Sirve principalmente para hacer la captura de una cierta cantidad de paquetes después de que empieza a simular, ya que se programa para decirle después de una cantidad de tiempo que se detenga.
- Data Stream. La creación de paquetes, donde se puede configurar el tipo de stream: POS IP, ATM AAL5, Ethernet IP, Ethernet TCP/IP, PPP TCP/IP, etc. Además de configurar los encabezados de los paquetes: PPP Header, IP Header, Data Payload, PPP Tailer, etc.

- Traffic Assignment. En esa sección se le asigna los *data stream* a un puerto en especifico, ya sea de entrada o salida.
- MSF Devices. En esta sección se específica el tipo de dispositivo que correrá el MSF: SPI3, SPI4, CSIX, etc, además de editar el puerto que estará usando el dispositivo creado.

Cabe mencionar el potencial de esta herramienta, ya que la mayoría de las prácticas hicieron uso de ésta. También se debe resaltar el simulador de tráfico, ya que sin este, no se hubiera podido realizar la práctica tres que es la de *Recibir, procesar y Transmitir*.

La práctica cuatro de *IP-Five-Tuple* también tuvo gran importancia el simulador de tráfico, ya que se podía analizar todo el paquete desde que entraba y salía además de guardar en un archivo log el paquete que estaba en el puerto de entrada y en otro archivo log el paquete que estaba en el puerto de salida.

4.5.4. Implementación de Código

La implementación de código fue uno de los principales objetivos de este laboratorio, ya que los alumnos tenían contacto físico con los procesadores. En esta quinta práctica los alumnos eran encargados de hacer el trabajo de un *"tester"*, donde cargaban código desarrollado por equipo de trabajo de Intel (IXA) en la tarjeta y se dedicaban a investigar los problemas que se desplegaban y trataban de solucionarlos.

Esta práctica englobaba todo lo visto en las prácticas anteriores, ya que utilizaba a su máximo potencial los dos IDEs. Donde el Developer Workbench servía de compilador y corrector de errores. El Tornado nos sirve como conexión por red al NP, cross-compilador de código e interfase de ejecución de código. Para que tuviera éxito en la práctica tenían que hacer uso de estas dos herramientas de trabajo.

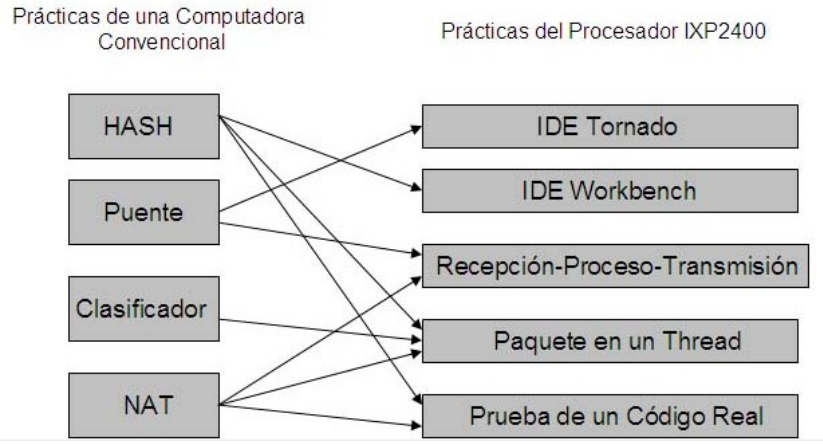
Capítulo 5

Experimentos y Resultados

Este laboratorio se implementó a mitad de semestre impartiéndolo como apoyo a la materia de Procesadores de Redes. Se tomó como muestra a un grupo de alumnos que cursaban la materia para realizar el estudio de esta tesis. El salón estaba conformado por 27 hombres y 1 mujer, los cuales tienen en promedio 22 años y están cursando el octavo y noveno semestre.

5.1. Evaluación Objetiva

Al principio de este laboratorio, se les aplicó un examen de 17 preguntas enfocadas a conceptos generales relacionados con Procesadores de Redes. Esto con el fin de poder evaluar los conocimientos de los alumnos, previo a cursar el nuevo laboratorio y lograr demostrar cómo los alumnos al final del semestre incrementaron sus conocimientos durante las clases teóricas y prácticas del laboratorio. Haciendo resaltar el diseño complejo del laboratorio donde se busca acelerar el proceso del aprendizaje impartiendo temas básicos en el laboratorio sencillo y haciendo uso de estos conceptos en las nuevas prácticas del laboratorio que usa el procesador de red, como se muestra en la Figura 5.1.



El examen tiene preguntas tanto de conceptos básicos del procesamiento de un paquete hasta preguntas del código de desarrollo de los procesadores de redes.

Cabe mencionar que cuatro de las diecisiete preguntas aplicadas fueron eliminadas de los resultados estadísticos debido a que fueron temas que no se alcanzaron a cubrir y que en la continuación del laboratorio se cubran.

A continuación se presentan los resultados del examen aplicado previo al laboratorio.

Como se puede apreciar en la Figura 5.1 se muestra como de 28 alumnos sólo aprobaron 10 alumnos y reprobaron el 58 % del salón, siendo el pase de 9 preguntas correctas. Se debe tomar en cuenta que este examen no les afectaba de forma académica por lo que hay que estar conscientes que muchos lo hicieron sin interés alguno. Además de esto se debe mencionar que se obtuvo un promedio de 8.1 respuestas correctas contra 8.8 respuestas incorrectas, donde la mayoría del grupo se encuentra reprobado.

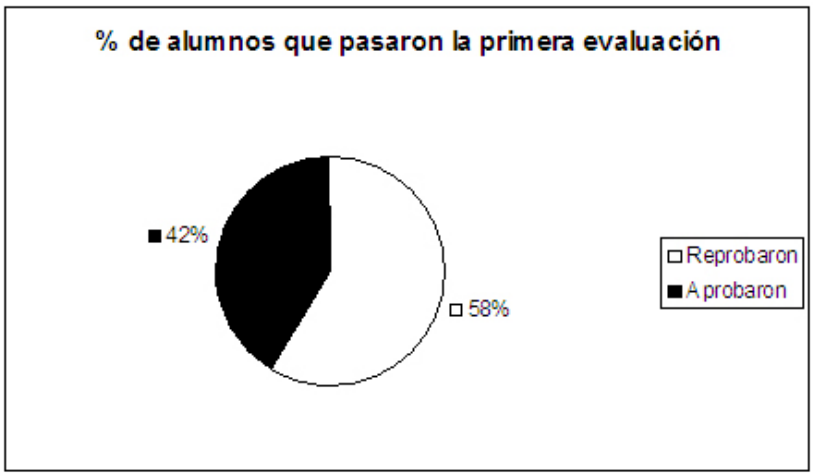


Figura 5.1: Resultado de los exámenes de diagnóstico

A continuación se muestra un comparativo de las respuestas correctas e incorrectas de cada una de las 17 preguntas diseñadas en el examen como se muestra en la Figura 5.2. Cabe resaltar que la única pregunta que todos los alumnos sacaron bien es una pregunta de concepto vista en el curso y que no se ha enseñado en el laboratorio de clase. Cabe mencionar el lado contrario donde la pregunta 7 que fue la más baja es uno de los conceptos principales que se enseñan en el nuevo laboratorio de procesadores de redes.

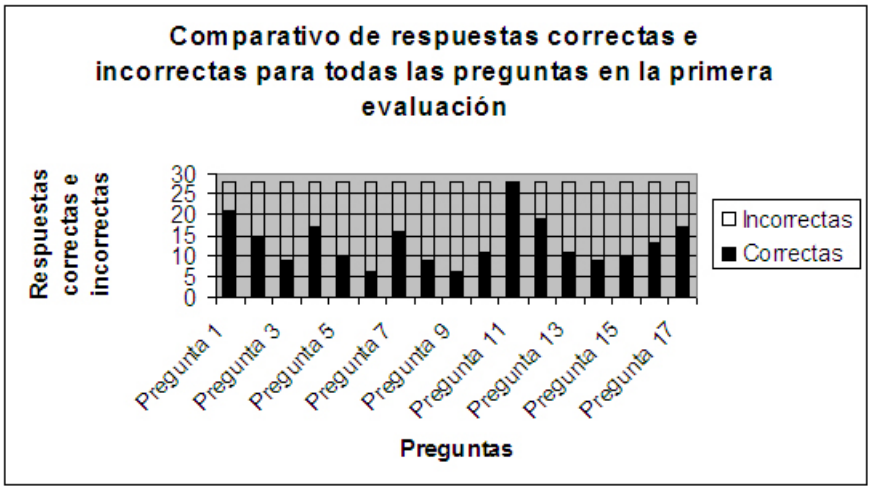


Figura 5.2: Respuestas correctas-incorrectas de la 1ª Evaluación

A continuación se presentan los resultados al término del laboratorio, donde se aplicó exactamente el mismo examen, con la misma muestra y con el mismo tiempo para hacer estudios de mejora de conocimientos.

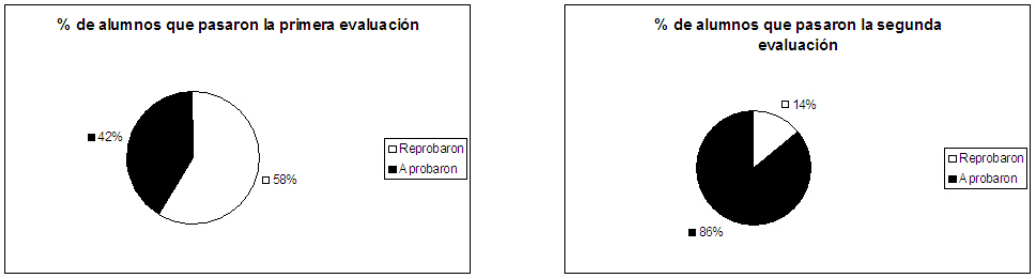


Figura 5.3: (a) Antes del Laboratorio y (b) Después del Laboratorio

Como se demuestra en la Figura C.1 inciso (a), sí hubo una mejoría significativa comparada con la Figura C.1 inciso (b), ya que para esta segunda evaluación donde se les aplicó el mismo examen con el mismo tiempo, hubo una mejora debido al promedio de 10.89 preguntas correctas por examen (Figura5.5), además de que el más alto obtuvo 15 preguntas correctas.

Como se puede apreciar en la Figura 5.4 inciso (b), que el promedio de aciertos de los alumnos en la segunda evaluación fue más alto por 2.87 preguntas en comparación de la Figura 5.4 inciso (a), siendo esto un reflejo de mejora excelente. También cabe resaltar que ahora no sólo hubo una sola pregunta que casi todos acertaran, sino que fueron cuatro aciertos que casi todo el grupo sacó correctas.

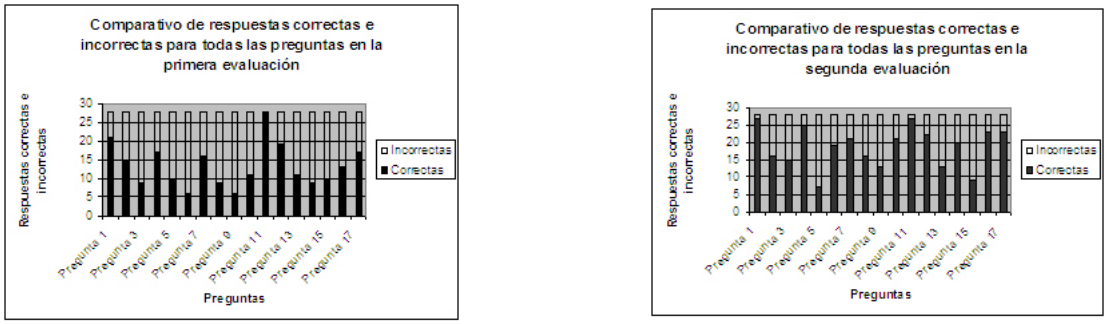


Figura 5.4: (a) Antes del Laboratorio y (b) Después del Laboratorio

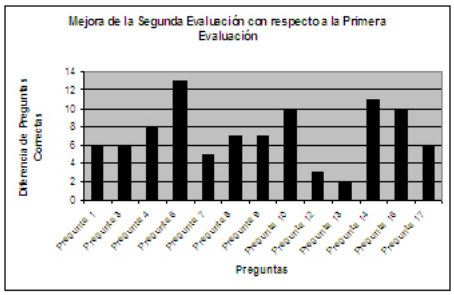


Figura 5.5: Mejoramiento en la Evaluación

Estas evaluaciones de diagnóstico que se les aplicaron a los estudiantes de la carrera de Ingeniería en Sistemas Electrónicos ISE. Se pudo llegar a la conclusión de que lo que se imparte en el laboratorio sirve para complementar el aprendizaje del curso. Además de que los alumnos se familiarizan y comprenden de una manera más tangible los componentes y herramientas de desarrollo de un procesador de red que en un curso teórico no se alcanza a cubrir.

Cabe mencionar que se quitarón las preguntas 2, 5, 11 y 15 dentro de las gráficas y estadísticas de resultados, ya que estos fueron temas que no se alcanzaron a ver debido al tiempo que se tenía disponible.

5.2. Evaluación Subjetiva

Al finalizar la última práctica de laboratorio se les pidió a los alumnos llenar una evaluación del laboratorio para poder tener retroalimentación de lo que ellos percibieron de lo visto en el nuevo laboratorio. Esta muestra consta de 23 hombres y una mujer, en promedio de 22 años y de octavo - noveno semestre. No todos alcanzaron a hacer las encuestas debido a que no fue posible localizarlos.

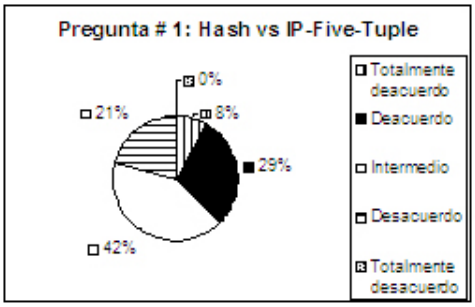


Figura 5.6: Hash vs IP-Five-Tuple

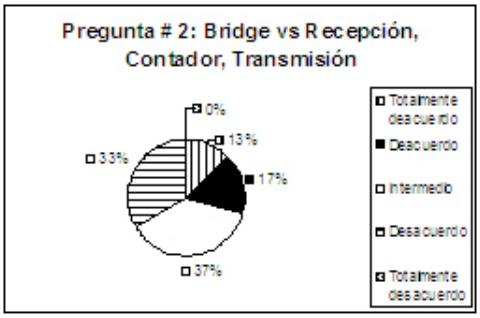


Figura 5.7: Bridge vs Recepción, Contador, Transmisión

5.2.1. Preguntas enfocadas al catalizador de prácticas

Las primeras siete preguntas se enfocaron en conocer qué tanto les ayudó a incrementar sus conocimientos. Haciendo un comparativo de las prácticas que se impartieron haciendo uso de una NIC convencional con las nuevas prácticas del laboratorio que usaron el Procesador de Rede IXP2400.

A continuación se muestran los resultados de cada una de estas preguntas en específico:

En la Figura 5.6, se muestra como el 79 % de los alumnos opinan que los conceptos aprendidos de Hash en la práctica 1 usando una NIC convencional les ayudan a resolver o comprender la práctica de *IP-Five-Tuple* y la práctica de *Testing*. Además de que sólo cinco personas no estuvieron de acuerdo con este aceleramiento en el aprendizaje.

En la Figura 5.7, se muestra como sólo un 33 %, que sería un tercio del salón no están de acuerdo en que los conceptos aprendidos de Bridge de la práctica # 2 sirvieron para resolver o comprender los conceptos de "Recepción, Contador y Transmisión" de la práctica # 3 del nuevo laboratorio. Siendo ocho alumnos los que no están de acuerdo en este aceleramiento en el aprendizaje de la nueva práctica.

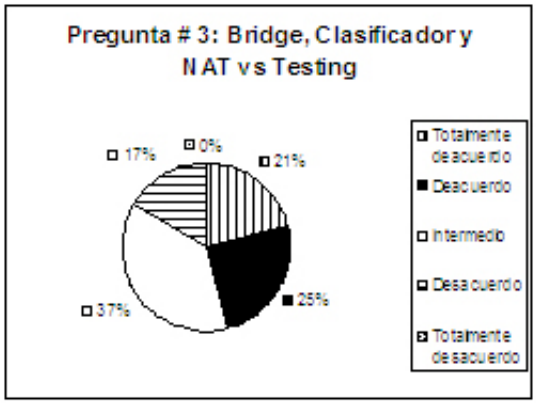


Figura 5.8: Bridge, Clasificador y NAT vs Testing

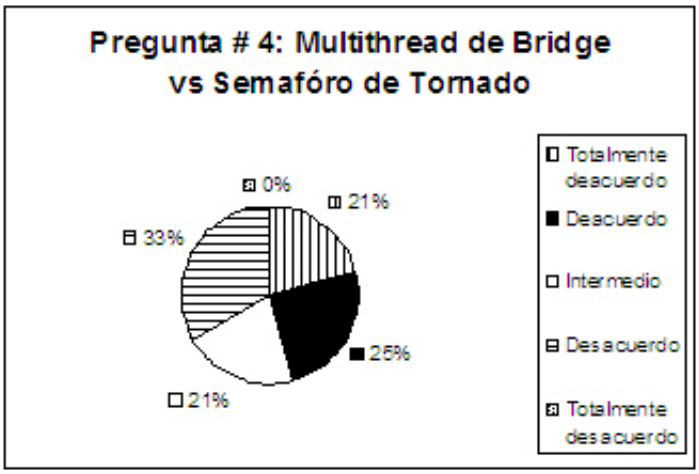


Figura 5.9: Multithread de Bridge vs Semáforo de Tornado

En la Figura 5.8, se muestra como el 46 % esta de acuerdo en que los conceptos aprendidos en el Bridge, Clasificador y NAT de la Práctica # 3 usando una NIC convencional sirvió de gran ayuda para resolver o comprender los conceptos vistos en la práctica de IP-Five-Tuple, que sería la cuarta práctica del nuevo laboratorio.

En la Figura 5.9, se muestra como más el 46 % del salón comenta que la práctica del NAT usando una NIC convencional, fue de gran aportación para resolver o comprender los conceptos vistos en alguna de las prácticas nuevas del laboratorio. Y esto tiene mucho sentido, ya que muchos de los conceptos vistos en esta práctica engloban varios de los temas visto en el nuevo laboratorio.

En la Figura 5.10, es la gráfica con mayor cantidad de personas en desacuerdo

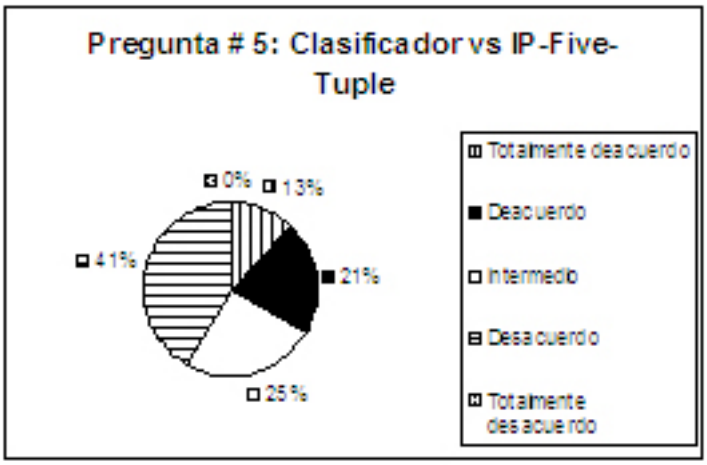


Figura 5.10: Clasificador vs IP-Five-Tuple

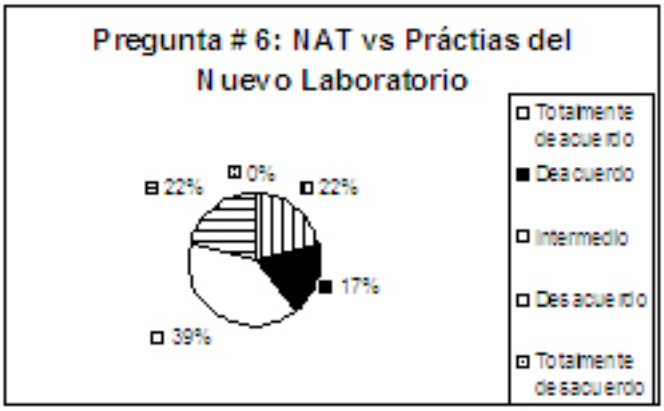


Figura 5.11: NAT vs Prácticas del Nuevo Laboratorio

con los conceptos aprendidos en una práctica como lo es el clasificador. Comentaron que sirvieron como aceleradores para resolver o comprender los conceptos vistos en la práctica de IP-Five-Tuple del nuevo laboratorio. En total fueron 10 alumnos los que no estuvieron de acuerdo.

En la Figura 5.11, se muestra como el 78 % de los alumnos opinan que los conceptos del NAT aplicados usando una NIC convencional sirvieron de ayuda para resolver o comprender algunas de las prácticas del nuevo laboratorio de IXP2400.

En la Figura 5.12, se muestra como el 77 % de los alumnos están de acuerdo en que las prácticas que se realizaron al principio de semestre, en las que se trabajó con una NIC convencional de una computadora x86. Aseguraron que fue de gran ayuda para comprender y realizar de manera eficiente las prácticas del nuevo laboratorio.

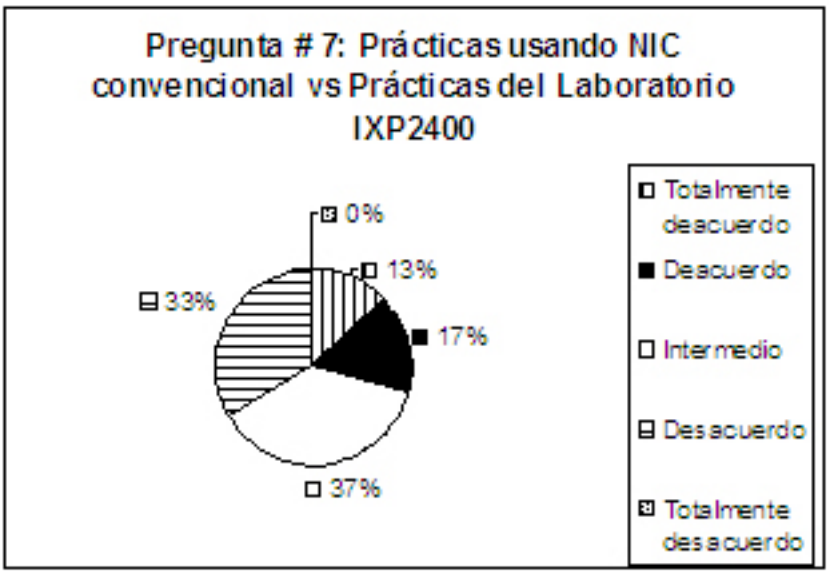


Figura 5.12: Prácticas usando NIC convencional vs Prácticas del Laboratorio IXP2400

5.2.2. Análisis de conocimientos que los alumnos sienten que adquirieron

Después de estas preguntas, los alumnos llenaron una tabla en donde iban calificando de Muy Alto a Muy bajo, dependiendo de los conocimientos que sienten que adquirieron y los que sienten que les faltaron:

En la Figura 5.13, se presentan los resultados de esta tabla que a continuación será analizada. Observando esta gráfica cabe mencionar que fueron muy pocas las califiaciones bajas y que la mayoría se centran en alto y medio.

Las calificaciones más altas fueron por parte de la documentación debido a que mucha de la documentación fue elaborada desde el semestre pasado y fue depurada de nuevo este semestre.

La calificación más baja estuvo enfocada a las dudas que se tuvieron en el NP, ya que los alumnos consideran que resolvieron la mayor parte de sus dudas. La otra calificación baja es acerca de la experiencia aprendida en el laboratorio de NP, aquí sólo demuestra que sigue faltando añadir más prácticas de implementación para que el alumno adquiriera la suficiente experiencia de campo.

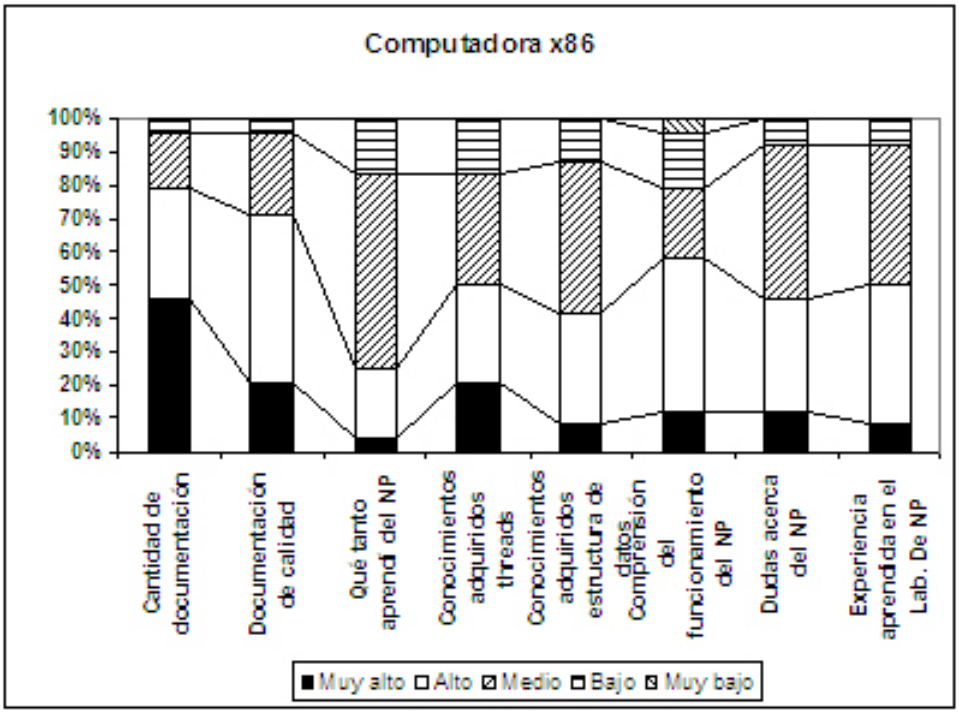


Figura 5.13: Retroalimentación Subjetiva de los conocimientos adquiridos en el Procesador de Red IXP2400.

En cuanto a los conocimientos adquiridos en el manejo de threads y estructura de datos, se aprecia que los alumnos consideran que han aprendido por lo menos a un nivel medio.

En la Figura 5.14 , se presentan los resultados adquiridos al aplicar la misma tabla pero analizando lo aprendido en las prácticas del laboratorio usando el procesador de red IXP2400.

Cabe resaltar que las calificaciones más altas son de documentación y eso se debió a la buena calidad de las prácticas donde se daban extensas explicaciones de lo que se iba hacer. También cabe resaltar que salió alta, tanto la comprensión del NP, como las dudas que se tienen acerca de este. Este resultado es normal, debido a que entre más conocimientos tengan de la tarjeta, más dudas les surgían.

La calificación baja de la documentación se entiende debido a que era mucha información y no se tenía en claro cuáles eran las de calidad. Es por ello que los equipos se perdían con tanta información y no lograban diferenciar la documentación importante.

En cuanto a los conocimientos adquiridos en el manejo de threads y estructura de

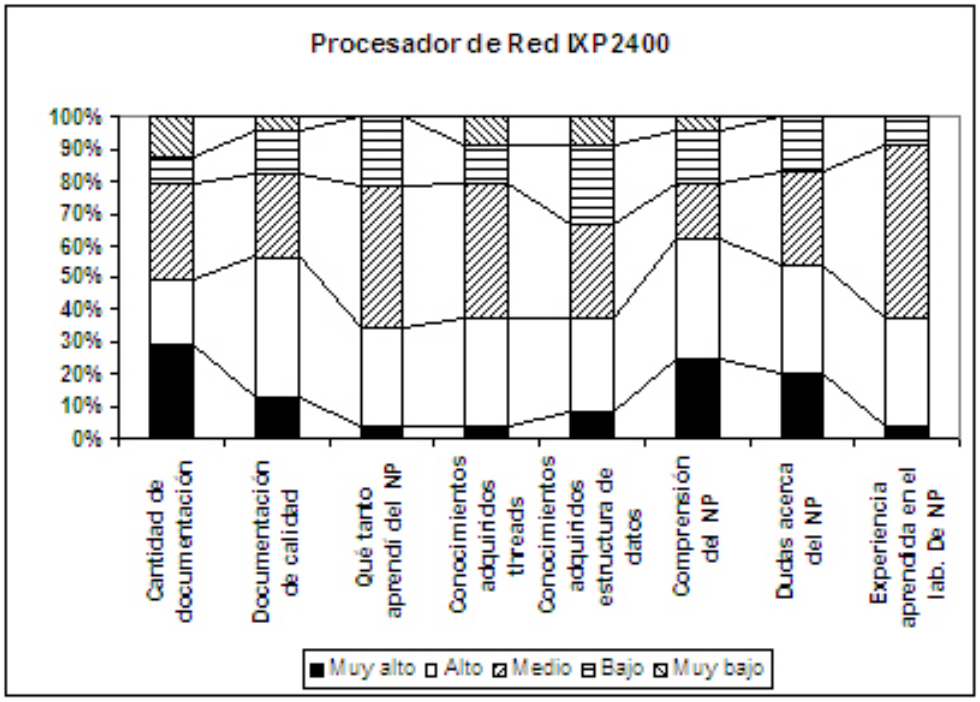


Figura 5.14: Retroalimentación Subjetiva de los conocimientos adquiridos en el procesador de Red IXP2400

datos se obtuvo una calificación baja, dado a que los conocimientos adquiridos fueron vistos principalmente en las prácticas anteriores, por lo que al momento de analizar estas nuevas prácticas se siente que no se adquieron conocimientos nuevos y a pesar de esto, hubo nuevos conocimientos que se debieron de aprender en las nuevas prácticas pero no todos lo alcanzaron a apreciar.

5.2.3. Análisis de los aspectos a mejorar en el Laboratorio

En esta sección se dejó una pregunta abierta donde los alumnos opinaban de los principales aspectos a mejorar en el laboratorio.

En esta gráfica 5.15 cabe resaltar que lo más que le faltó al nuevo laboratorio fue el tiempo para la realización de las prácticas, el cual se justifica debido a las escasas 6 semanas de clases que se tuvieron para implementar el laboratorio.

Otro aspecto importante es el saber que la opinión de los alumnos es buena en cuanto al nuevo laboratorio dando a entender que se debería de empezar con las nuevas prácticas desde un principio.

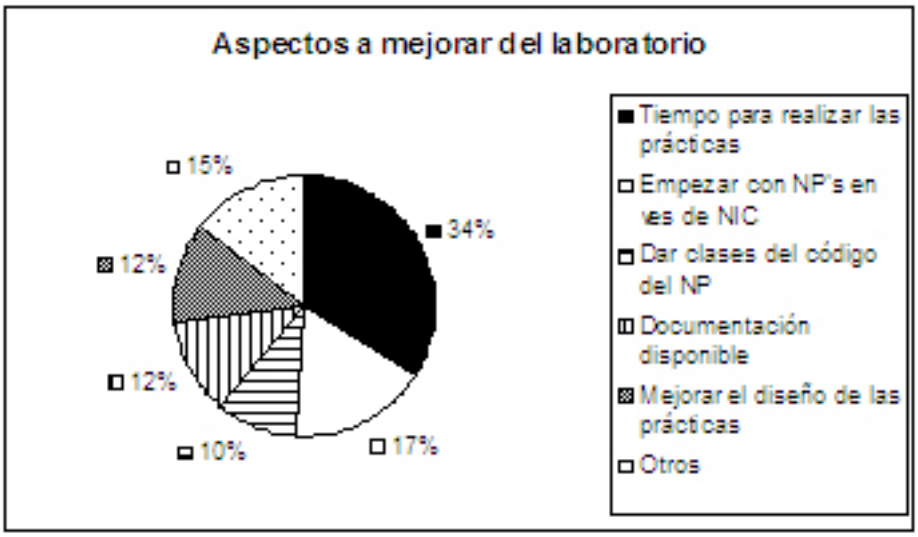


Figura 5.15: Aspectos a mejorar en el Laboratorio

Los aspectos a mejorar que salieron con el mismo porcentaje hablan de la documentación disponible dado a la gran cantidad existente; y de la mejora en el diseño de las prácticas debido a que es la primera vez que se da, además falta depurar errores de ortografía o imágenes en las prácticas.

Y por último son las prácticas de laboratorio que faltan por diseñar, pero realmente no pueden empezar a diseñar código, ya que la curva de aprendizaje del estudiante es muy grande y más con una tecnología que acaba de salir al mercado tan compleja que se recomienda empezar con simulaciones como se hizo en este laboratorio. Además de que sí se tuvo una práctica de desarrollo básica.

5.2.4. Aspectos positivos y negativos del laboratorio

En esta sección se estableció un cuadro comparativo donde en una columna venían los aspectos positivos y en la otra los aspectos negativos.

En la Figura 5.16, se muestran los resultados de la tabla de aspectos positivos que llenaron los alumnos. Donde el más alto es que los procesadores de redes les llamó la atención por completo, esto es muy interesante, ya que da a entender que la materia vale la pena por aprender a utilizar nuevas tecnologías.

Otro de los aspectos positivos que no se esperaban era el de HW y SW accesible ya que se tuvo problemas en la conexión del Procesador de Red hacia las computa-

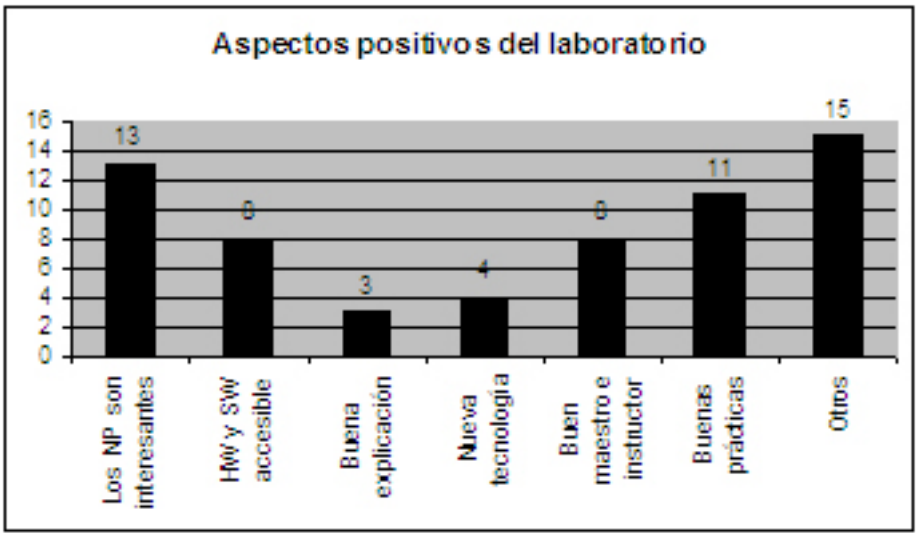


Figura 5.16: Aspectos positivos del laboratorio

doras de los alumnos, cuando apenas comenzaba el laboratorio, pero un día después de la práctica este error fue arreglado. Esto nos da a entender que los alumnos están conscientes de las posibles fallas técnicas que podrían tener.

Por último, cabe resaltar el comentario de las buenas prácticas, ya que como se podrá apreciar en la sección del apéndice se verá el diseño de las prácticas, las cuales están de muy buen nivel y tienen demasiada información para tener un laboratorio de cuatro horas muy ocupado.

En la Figura 5.17, se muestran los resultados de la tabla que se aplicó a los alumnos para que mencionaran los aspectos negativos que percibieron en el laboratorio.

Lo que más resalta y es muy cierto, es el poco tiempo que se tuvo para la elaboración de las prácticas, pero esto se entiende ya que el proyecto se levantó sobre la marcha de las clases y se tuvo que invertir mucho tiempo para comprenderlas, llevarlas acabo y entregar reportes y tareas.

La segunda que más se menciona es la falta de documentación, pero esta ya se explicó que era demasiada documentación y eran ciertos documentos en específico los que se deben de analizar y los estudiantes que no se alcanzaron a dar cuenta de esto cayeron en estos comentario.

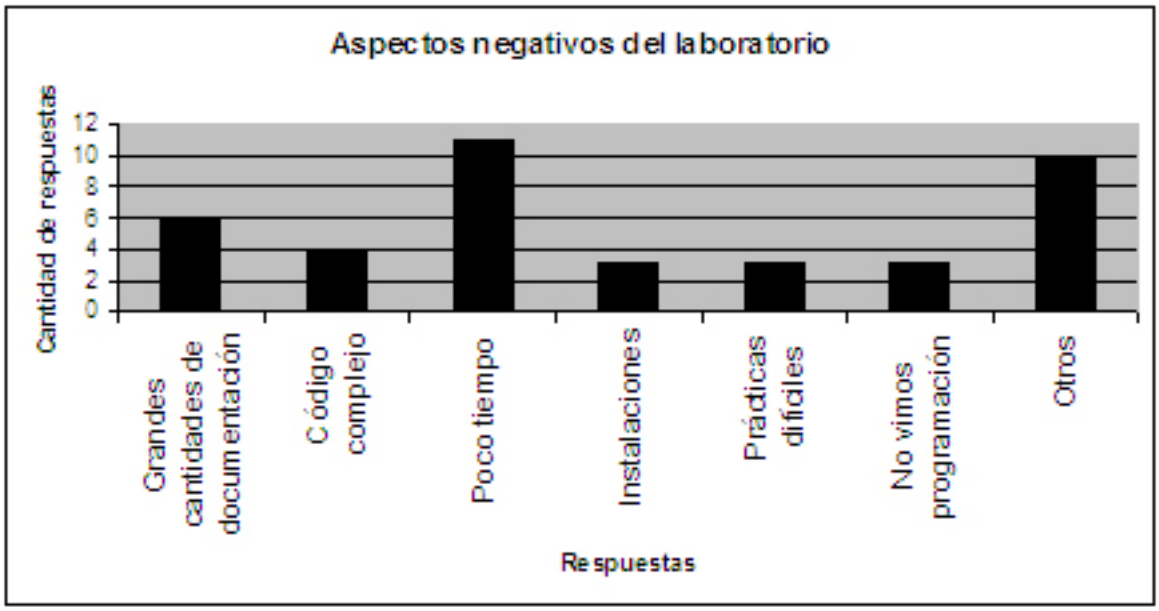
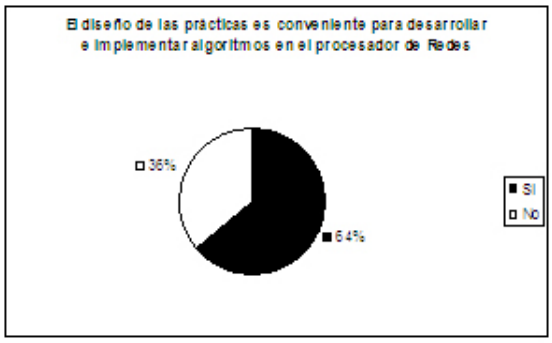


Figura 5.17: Aspectos negativos del laboratorio

5.2.5. Diseño de las prácticas

En esta parte de la evaluación se preguntó de forma directa si el diseño de las prácticas se considera conveniente para empezar a desarrollar o implementar algoritmos en el Procesador de redes.

A esta pregunta respondieron un 64 % de los alumnos afirmativamente y el 36 % negativamente (Figura 5.18, inciso (a)), siendo sorpresivo, ya que las prácticas están enfocadas principalmente a simulación y hacer la función de un "tester".



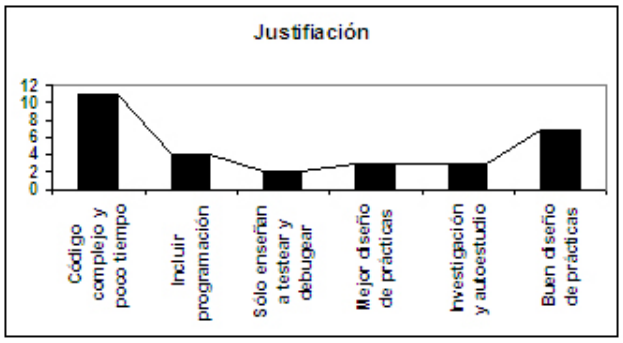


Figura 5.18: (a) Diseño de las Practicas y (b) Justificación

En la Figura 5.18 inciso (b), se muestran las principales respuestas que los alumnos escribieron para justificar el hecho de que la mayoría opinaba que las prácticas están

bien diseñadas para empezar a desarrollar en lo que serían los procesadores de redes. Cabe resaltar que la principalmente mencionada, es la del lenguaje complejo que tiene para programarse y el poco tiempo que se tuvo para aprenderlo.

Además de que por otro lado mencionan y hacen énfasis en que estaban de acuerdo con el diseño y que es una buena forma de enseñarlo.

Los demás son comentarios que hicieron una o dos personas, en las que preferirían incluir programación desde cero, no les gustó que se enseñara por medio de "testing", se pedía un mejor diseño de prácticas cuando muchos estaban a gusto con el diseño de prácticas, etc. Para estos comentarios se tiene que rediseñar completamente el laboratorio y cambiar el objetivo además de que no se considera que para una tecnología tan compleja se empiece a programar desde cero.

5.2.6. Nivel de las Prácticas del Laboratorio

En esta sección se hizo una pregunta directamente donde los alumnos responderían si las prácticas las consideraban de nivel: básico, intermedio o avanzado en comparación con otros laboratorios del Campus.

El 46 % de los alumnos contestaron que está en nivel avanzado, el 50 % contestaron que está en nivel intermedio y un 4 % contestó que está en nivel básico (Figura 5.19 inciso (a)). Todas estas respuestas nos dan a entender que los alumnos percibieron en general al laboratorio con un nivel alto, por lo que es muy bueno para cualquier laboratorio a nivel profesional. Para complementar su respuesta se les volvió a pedir una justificación.

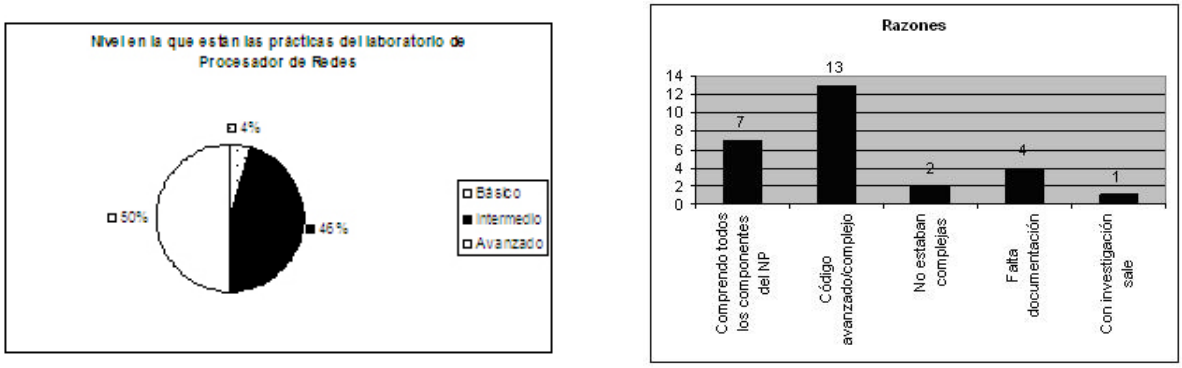


Figura 5.19: (a) Nivel del laboratorio (b) Razones

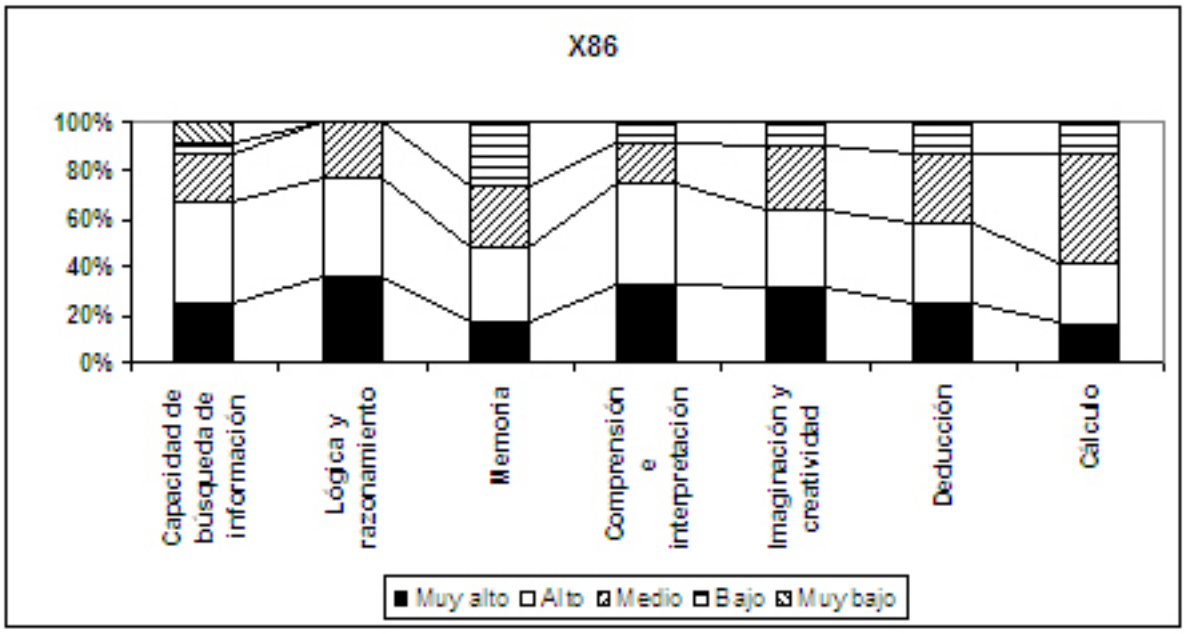


Figura 5.20: Esfuerzo cognitivo de las prácticas del NIC convencional

En la Figura 5.19 inciso (b), se muestran las diversas razones por las que los alumnos llegaron a deducir que el laboratorio tiene un nivel alto o intermedio principalmente.

La principal razón que la mayoría de los alumnos mencionó fue que el código era muy complejo para el tiempo que se tenía para enseñarlo. Esto ya se había comentado y se espera tener una mejor planeación de tiempo para el próximo semestre.

La segunda razón es que los alumnos alcanzan a comprender los principales componentes que utiliza el procesador de red, ya que son las que se tuvieron que utilizar para las prácticas.

5.2.7. Capacidades que le ayudaron a los alumnos a realizar las prácticas

En esta sección se les preguntó a los alumnos los aspectos que ellos consideran les ayudó a realizar las prácticas. Donde se les dió como opción, variables cognitivas: capacidad de busqueda, lógica y razonamiento, memoria, comprensión, imaginación, deducción y cálculo. Y calificaron estas opciones cognitivas de 0 a 5, donde 5 es muy alto y cero es muy bajo.

En la Figura 5.20. Se muestra el esfuerzo que tuvieron que realizar los alumnos en

las prácticas haciendo uso de un NIC convencional de una computadora x86, donde la lógica y el razonamiento fueron los principales motores para la resolución de las prácticas.

Cabe mencionar que la memoria fue la que los alumnos consideron haber utilizado menos, y eso se debe a que la mayoría de la información estaba documentada, sólo era necesario la búsqueda, comprensión y deducción.

Estas mismas opciones se les pidieron que calificaran los alumnos para las nuevas prácticas del laboratorio de redes.

En la Figura 5.21, se muestra como la deducción fue el esfuerzo deductivo que más utilizaron siendo la más alta de todas, después la comprensión e interpretación. Se obtuvieron estas respuestas, ya que la mayoría de las prácticas eran de análisis y simulación de código donde se les daba el código a los alumnos y ellos se encargaban de entender y simular el código, además de que el instructor se encargaba de corroborar que los alumnos realmente estuvieran entendiendo las prácticas haciendo preguntas directas de lo que se hace en el procesador de redes.

La que también era de esperarse que fueran muy bajas son la imaginación y creatividad, además del cálculo, ya que como se ha mencionado las prácticas están hechas para descifrar lo que ya se hizo, por lo que descartaba totalmente las posibilidades de creación o desarrollo, es más bien etapa de "tester".

5.2.8. Comentarios finales de los alumnos

En esta última parte se les pidió a los alumnos que hicieran cualquier comentario que sea importante para este laboratorio. Cabe mencionar que no todos respondieron esta pregunta en específico debido a que era la última pregunta y no era obligatoria contestarla.

En la Figura 5.22, se muestra como el 30 % de los alumnos opinan que faltó tiempo para la realización de las prácticas. Aquí se vuelve a tocar el tema del tiempo faltante para la realización de las prácticas el cual también ya fue mencionado, por lo que sólo cabe resaltar que sigue siendo un factor importante para el desempeño de las nuevas prácticas.

El otro 24 % menciona que las prácticas tienen un potencial importante y que sí debemos de darle continuidad al desarrollo de este laboratorio, ya que es un acer-

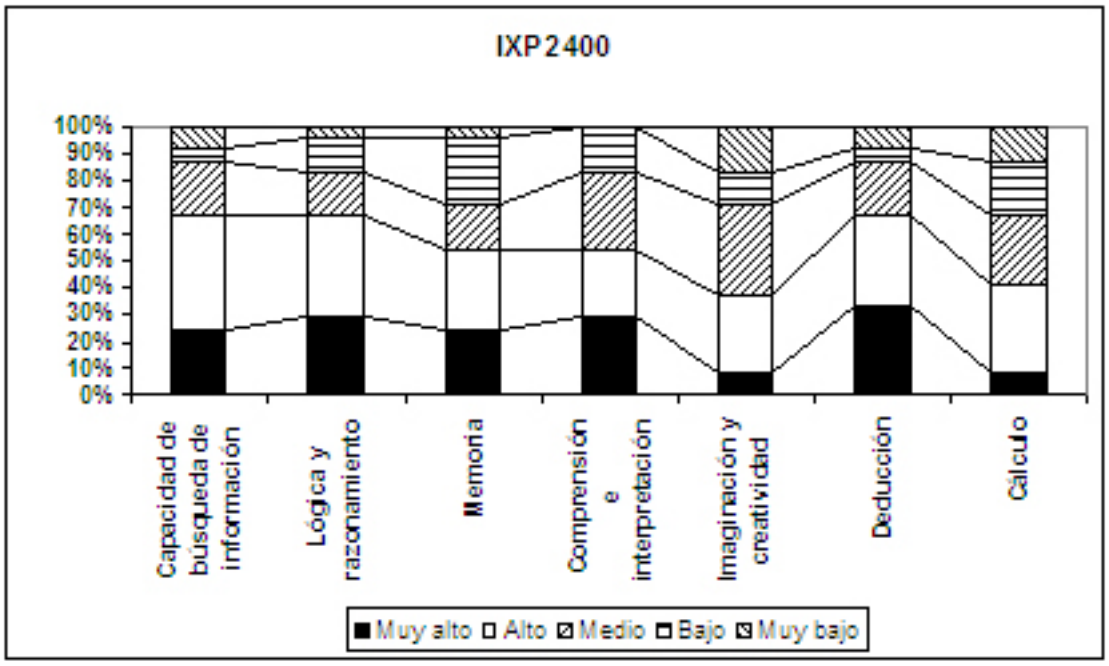


Figura 5.21: Esfuerzo cognitivo que genera las prácticas del NP

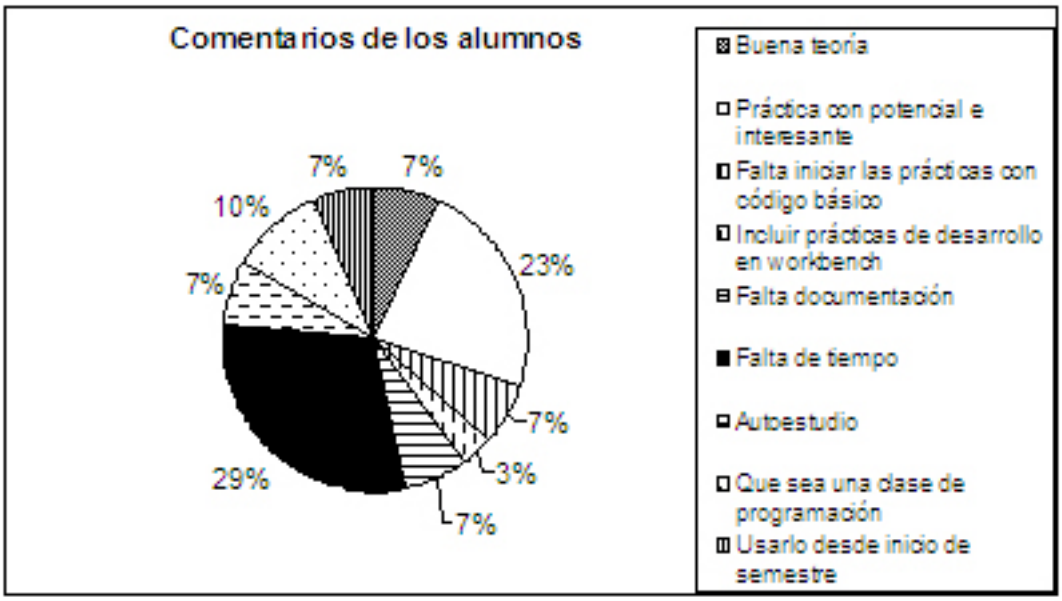


Figura 5.22: Comentarios finales de los alumnos

camiento muy bueno que se puede tener en el campo.

Y los comentarios con el menor porcetaje fue el 3 % del grupo haciendo mención que se deberían tener prácticas de desarrollo con el IDE Developer Workbench.

5.3. Analisis de Resultados

Se realizará un analísis de las gráficas arrojadas por la evaluación subjetiva y evaluación aplicada al principio y al final del semestre.

En la evaluación de diagnóstico que se les aplicó al principio y al final del semestre, sí se nota una mejoría de casi 3 preguntas correctas en promedio. Se debe tomar en cuenta que esta evaluación de diagnóstico se les aplicó sin aviso previo, además de que era sin afectar su calificación curricular. Por lo que sí se considera con más tranquilidad el tener un promedio de 10.89 del final contra 8.1 de respuesta correctas del principio, se está hablando de una excelente mejoría en el grupo.

A continuación se explica cómo se aceleró del conocimiento haciendo referencia a las prácticas usando una computadora convencional contra las prácticas del nuevo laboratorio:

- La práctica IDE de Tornado (Práctica # 1), del nuevo laboratorio de procesador de redes refleja el conocimiento adquirido al salir con mejor puntuaje en las preguntas 12 y 13, que tiene que ver con threads y semáforos. Estos datos también fueron corroborados de manera subjetiva al preguntarles a los alumnos si la práctica de Multi-thread de Bridge ayudó a resolver esta práctica en el problema del semáforo.
- La práctica Developer Workbench (Práctica #2), del nuevo laboratorio de procesadores de redes refleja los conocimientos impartidos al mejorar la nota el grupo de 11 a 21, siendo esto una gran diferencia de conocimientos grupales.
- La práctica Recepción, Conteo y Transmisión (Práctica # 3), del nuevo laboratorio de procesadores de redes refleja de los conocimientos adquiridos al haber mejora en las respuestas de la pregunta 7 y 8, que hablan de temas que tienen que ver con recepción, procesado y transmisión de un paquete. En la pregunta 7 se mejoró de 16 puntos a 21 y en la pregunta 8 se mejoró de 9 a 16 puntos. En esta práctica en particular se debió de haber sentido el aceleramiento en el

aprendizaje ya que se están usando conceptos de Bridge (Práctica # 2), donde se vio la partición del paquete, filtrado del protocolo Ethernet y reenvío de paquetes.

- La práctica de *Process in a single Thread* (Práctica # 4) del nuevo laboratorio de procesadores de redes refleja los conocimientos adquiridos al salir con mayor puntaje en las preguntas: 4, 6, 9, 14, 16 y 17. Todas estas preguntas engloban temas que se ven en Hash, Bridge y Clasificador que son las prácticas 1, 2, y 3 usando una NIC convencional. En estas prácticas se ven en particular los temas de: threads, coprocesadores y memoria jerárquica. Estos resultados fueron corroborados por la evaluación subjetiva al preguntar si los conceptos de Hash y Bridge ayudaron a resolver la práctica de IP-Five-Tuple.

- La práctica de *Testing a Real Program* (Práctica # 5) del nuevo laboratorio de procesadores de redes muestra los conocimientos adquiridos al mejorar el puntaje de las preguntas: 1 y 3 que se enfocan en temas de programación y algoritmos de redes. En esta práctica también se tuvo un aceleramiento en el aprendizaje, ya que se utilizaron conceptos vistos por la práctica de Bridge, Clasificador y NAT que sería la cuarta práctica usando una NIC convencional. Todos estos datos también fueron corroborados por las evaluaciones subjetivas que preguntaban si se había sentido algún aceleramiento en el aprendizaje en esta práctica en donde el 46 % de los alumnos estuvieron de acuerdo.

Al analizar los conceptos aprendidos por los alumnos se puede comparar como en las prácticas usando una computadora x86 la cantidad de documentación existente es de mejor calidad que la que se usó en los procesadores de redes, esto se debe a que las prácticas que ya estaban se han ido mejorando y la documentación esta cada vez más clara y concisa, en cambio con los procesadores de redes a pesar de ser la primera vez, la documentación en las prácticas está muy completa, sólo que faltó especificar a los alumnos en cuáles de todos los documentos basarse para estudiar.

Los conceptos aprendidos en el uso de threads y estructura de datos, es de esperase que se aprenda más la primera vez que se imparte que la segunda vez. Por último cabe resaltar que en las prácticas pasadas no tuvieron una comprensión alta del NP y mucho menos dudas con el NP. En cambio con las nuevas prácticas por lo mismo que aumentó su comprensión del funcionamiento del NP las dudas aumentaron.

Se hicieron varias preguntas con el mismo objetivo sólo que se preguntaban de diversas formas, esto con la finalidad de converger a una misma conclusión que nos refleje una opinión real de los aspectos que hacen sobresalir al laboratorio y los aspectos que se tienen que mejorar. Haciendo el análisis de los aspectos a mejorar, los aspectos

negativos y la opinión de los alumnos se llegó a la conclusión que se debe mejorar el tiempo para impartir las prácticas, debido a que casi todos mencionan que es muy poco tiempo para toda la carga que el laboratorio representa. También se hizo un análisis de los aspectos positivos, nivel de prácticas y opinión de los alumnos donde se concluyó que la opinión que todos tenían era que a pesar que la curva de aprendizaje de esta nueva herramienta era demasiado grande, el diseño de las nuevas prácticas las cuales atacaban los problemas con simulaciones tienen un gran potencial para aprendizaje [13].

Capítulo 6

Conclusiones

Esta tesis aportó en la creación de un laboratorio para la enseñanza de código de los procesadores de redes IXP2400 de Intel usando la tarjeta ENP-2611 de Radisys. Cabe mencionar que este laboratorio no existe en México a pesar de que en Guadalajara se intentó implementar, pero se dejó por la complejidad del mismo y que en Estados Unidos se ha implementado pero con la versión pasada del procesador de Red de Intel (IXP1200).

6.1. Conclusiones

La tesis llegó a un nivel en el que los alumnos desarrollaron los conocimientos básicos e intermedios para desarrollar algoritmos de redes en la tarjeta. Se logró tener un gran avance para la enseñanza de los alumnos en el área de redes donde se les abre la oportunidad de tener los conocimientos para poder diseñar y desarrollar. También se llegó al grado de vivir este espiral en movimiento que fomenta al conocimiento, donde los alumnos cuestionaban entre ellos, le preguntaban al instructor, estudiaban e investigaban por su cuenta.

Se lograron todos los objetivos planteados en esta tesis como lo son: La implementación del sistema operativo VxWorks en la tarjeta ENP-2611, ya que es el sistema con el que los alumnos descargan código a la tarjeta, la implementación de una nueva topología de red, a través de la cuál, se implementaron las nuevas prácticas del laboratorio; y por último el desarrollo de prácticas enfocadas a acelerar el conocimiento, ya que se obtuvo una retroalimentación completa con las evaluaciones de diagnóstico y subjetivas corroborando su desarrollo de habilidades y conocimientos con los procesadores de redes.

Un punto importante a mencionar es el tiempo en el que se pudo impartir estas prácticas ya que se realizaron en 6 semanas y no pudo haber sido posible de esta man-

era de no haber tenido oportunidad de por los menos 2 veces a la semana de tocar los temas que se estaban viendo en el laboratorio.

6.2. Trabajo Futuro

Este laboratorio tiene demasiado potencial para poder crecer en muchas áreas. Principalmente se debe de concluir el desarrollo de estas prácticas, ya que quedaron pendientes temas a cubrir como lo son:

- Ejecución de Threads en desorden
- Ejecución de Threads en orden
- Multi-Threaded Recepción y Transmisión Drivers

Se deben de reestructurar y asignar los tiempos completos de las prácticas para que los alumnos vean estos temas a fondo sin necesidad de estar tomando demasiado tiempo de las clases del curso.

Teniendo el laboratorio corriendo completamente, se pueden llegar a encargar proyectos finales donde los alumnos implementen código de un bridge, switch, o hasta un ruteador pudiendo de esta manera crecer en conocimientos.

También se pueden desarrollar proyectos de investigación, como:

- Mejorar el desempeño de una red implementando protocolos de redes en el procesador de red. Llegando a hacer modificaciones al gusto del desarrollador.
- Implementación de un detector de intrusos, haciendo pruebas y mejoras.
- Implementación de algoritmos de redes que se enfoquen a la calidad de servicio (QoS) [7].

Esta tecnología tiene un potencial impresionante y sería excelente que fuera explotado tanto por los alumnos como los profesores.

Apéndice A

Evaluación Objetiva

INSTITUTO TECONOLOGICO DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS MONTERREY

Nota: esta es una evaluación de diagnóstico por lo que se procura obtener tu respuesta honesta y sincera a las preguntas que se te hacen a continuación. No tendrá ningún efecto en tu calificación curricular.

1.- De las siguientes aportaciones del Procesador de Red, ¿Cuál es la más importante?
a) El bajo consumo de potencia.
b) El lenguaje de programación que utilizan.
c) Los coprocesadores que utilizan para realizar sus tareas.
d) El sistema operativo que cargan.

2.- ¿Cuál es el formato correcto de un paquete de ETHERNET, que corre bajo el protocolo de TCP/IP?, Nota: Lo primero que llega es lo que esta más a la izquierda.
a) ETHERNET – IP – UDP
b) IP - ETHERNET – TCP
c) TCP - ETHERNET - IP
d) TCP – IP - ETHERNET

3.- ¿Cuál es la razón del estudio de algoritmos como: bridge, switch, router, NAT, etc.?
a) Para poder entender a las redes.
b) Para poder desarrollar protocolos de red en el procesador de red.
c) Para dar un enfoque completo a la materia.
d) Para entender como funcionan los procesadores de red.

4.- ¿Cuál sería la razón de tener un orden jerárquico de memoria y procesadores en un procesador de red?
a) La clasificación de paquetes lo necesita.
b) Ayudan a realizar tareas específicas dando un mayor throughput.
c) Diseño económico de la tarjeta.
d) Los protocolos de redes así lo requieren.

5.- ¿Cuál sería una tarea diferente de un procesamiento de Entrada (Ingress) y de un procesamiento de Salida (Egress) en un procesador de Red?
a) Búsqueda de direcciones y reenvío de paquetes
b) Detección de errores
c) Modificación del encabezado
d) Procesado con seguridad agregada.

6.- ¿Cuál de los siguientes es considerado un coprocesador del procesador IXP2400?
a) CAM
b) SHaC
c) Scratch
d) MSF

7.- ¿En cuál de las siguientes memorias se guarda todo un paquete?
a) Scratch
b) Local
c) SRAM
d) DRAM

8.- ¿Cuál es la función del Media Switch Fabric en el Procesador de Red?
a) Hace el uso de colas para no perder los paquetes.
b) Es una interfaz de alta velocidad que usa el Procesador de Red.
c) Usa el SPI-4.2 para mandar y recibir paquetes.
d) Ninguna de las anteriores.

9.- ¿Cuál es la razón de usar anillos en los microengines?
a) Ayudan a procesar paquetes mientras los microengines hacen otras tareas.
b) Levantan un thread para conectar a los microengines.
c) Ayudan a procesar las altas tasas de transferencia de paquetes usando FIFO.
d) Ninguna de las anteriores.

10.- ¿Cuál es la diferencia de un Microblock y un Core Component?
a) Los dos hacen lo mismo pero son de distintas compañías.
b) Uno procesa tareas y el otro procesa funciones aritméticas.
c) Uno es código para los microengines y el otro es código para el procesador XsCale.
d) Todas las anteriores.

11.- De los siguientes programas, ¿cuál es el que se programaría en un NP.?
a) Bridge
b) Detector de Intrusos
c) Ruteador
d) Todos los anteriores

12.-Tipo de semáforo que maneja la librería de VxWorks
a) semBlib .-binary semaphores
b) semClib .- counting semaphores
c) semMlib .- mutual exclusion semaphores
d) Todas las anteriores

13.- En el siguiente código, ¿quién es la prioridad de la tarea? taskSpawn ("tCosmos", 2, 0, SIZE, (FUNCPTR) cosmos,0,0,0,0,0,0,0,0,0,0);
a) 2
b) 0
c) SIZE
d) FUNCPTR

Código A)

```
1 .begin
2       .if(ctx() == 0)
3               ctx_arb[voluntary]
4               start_simple_reverse#:
5               reverse_array_simple(old_array, new_array, 10)
6               end_simple_reverse#:
7               reverse_array_advanced(old_array, new_array_adv, 10)
8               end_advanced_reverse#:
9       .endif
10      ctx_arb[kill]
11 .end
```

14.- En el código A, la línea 2 se refiere:
a) Un cambio de contexto
b) Se realizara solamente cuando la variable ctx valga cero.
c) Solo se va a ejecutar en un thread
d) Ninguna de las anteriores

15.-¿Cuál de los siguientes serían razón para escoger una ejecución de threads en desorden?
a) Cuando un Microblock tiene loops que se necesitan sincronizar.
b) Cuando la aplicación requiere sincronización por-flujo
c) Si una aplicación tiene muchas ramificaciones en paralelo de gran tamaño.
d) Todas las anteriores.

16.-¿Qué es lo que se necesita para inicializar un Anillo Scratchpad?
a) El tamaño del Anillo
b) La dirección inicial del scratchpad donde los datos del Anillo van a estar.
c) El número de Anillo a usar.
d) Todas las anteriores.

Código B)

```
1  .begin
2        .reg addr
3        .sig ring_sig
4
5        immed32(addr, (RING_NUM<<2))
6        scratch[get, out_val, addr, 0, 1],
7              ctx_swap[ring_sig]
8
9        .if (out_val != 0)
10             immed32(addr, RING_COUNT_ADDR)
11             scratch[decr, --, addr, 0]
12       .endif
13 .end
```

17.-En el código B, menciona cuál es la linea que toma el número de anillo y uno o más registros de transferencia y trae el anillo con contenido de la memoria scratchpad.
a) 5
b) 6
c) 10
d) 11

MUCHAS GRACIAS POR TU APORTACIÓN!!!!
ESTE ANALISIS SERÁ UN ESTUDIO PARA UNA TESIS

Apéndice B

Evaluación Subjetiva

INSTITUTO TECONOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS MONTERREY
Departamento de Ciencias Computacionales
Laboratorio de Procesadores de Redes
Valoración inicial del ENP-2611

Contestar después de haber realizado la última práctica de NP.

- Datos demográficos:
- Sexo:
- Edad:
- Semestre:

- Consideras que los conceptos de Hash aprendidos en la Práctica 1, usando una NIC convencional, aceleró el proceso de aprendizaje de la práctica 4 (IP-Five-Tuple) usando el Network Processor IXP2400.

Totalmente de acuerdo **Totalmente desacuerdo**
o 1 o 2 o 3 o 4 o 5

- Consideras que los conceptos de Bridge y Clasificador de la Práctica 2 y 3, usando una NIC convencional, te ayudó a acelerar el proceso de comprensión de la recepción de paquetes, la partición de estos, el procesado y el reenvío de paquetes de la Práctica 3 (Rx_count_Tx), usando el Network Processor IXP2400.
-

Totalmente de acuerdo **Totalmente desacuerdo**
o 1 o 2 o 3 o 4 o 5

- Consideras que los conceptos vistos de Bridge, Nat y clasificador de las Prácticas 2,3 y 4 usando una NIC convencional, agilizo la comprensión de la práctica 5 (Testing) usando el Network Processor IXP2400.

Totalmente de acuerdo **Totalmente desacuerdo**
o 1 o 2 o 3 o 4 o 5

- Consideras que los conceptos de multithread de la Práctica 2 (Bridge), usando una NIC convencional, te ayudó a acelerar el proceso de comprensión de semáforos de la Práctica 1, usando el Network Processor IXP2400.
-

Totalmente de acuerdo **Totalmente desacuerdo**
o 1 o 2 o 3 o 4 o 5

- Consideras que las conceptos de Clasificación de paquetes de la Práctica 3, usando una NIC convencional, aceleró el aprendizaje de los conceptos de IP-Five-Tuple de la Práctica 4, usando el Network Processor IXP2400.

Totalmente de acuerdo **Totalmente desacuerdo**
o 1 o 2 o 3 o 4 o 5

- Consideras que los conceptos aprendidos de NAT en la práctica de Práctica 4, usando una NIC convencional, te ayudó a acelerar el aprendizaje de alguna de las prácticas usando el Network Processor IXP2400.
-

Totalmente de acuerdo **Totalmente desacuerdo**

o 1 o 2 o 3 o 4 o 5

- Consideras que alguna de las prácticas que se hicieron usando una NIC convencional te ayudaron o aportaron algún conocimiento para alguna de las prácticas del Network Processor IXP2400.

Totalmente de acuerdo **Totalmente desacuerdo**

o 1 o 2 o 3 o 4 o 5

Explica tu respuesta de por qué sí o por qué no.

Responde las siguientes preguntas enfocadas al uso de la tarjeta x86

	Muy alto	Alto	Medio	Bajo	Muy Bajo
1. Cantidad de documentación.					
2. La documentación, ¿era de calidad?					
3. ¿Qué tanto aprendí de la arquitectura del NP, usando una NIC convencional?					
4. Nivel de conocimientos adquiridos en uso de threads.					
5. Nivel de conocimientos adquiridos en estructura de datos.					
6. El tener contacto con la tarjeta directo, me motivó a comprender el funcionamiento del NP.					
7. Me quedaron dudas acerca del NP.					
8. Considero que el laboratorio me dio la experiencia necesaria para comprender el funcionamiento de un NP.					

Responde las siguientes preguntas enfocadas al uso de la tarjeta IXP2400

	Muy alto	Alto	Medio	Bajo	Muy Bajo
1. Cantidad de documentación.					
2. La documentación, ¿era de calidad?					
3. ¿Qué tanto aprendí de la arquitectura del NP usando una NIC convencional?					
4. Nivel de conocimientos adquiridos en uso de threads.					
5. Nivel de conocimientos adquiridos en estructura de datos.					
6. El tener contacto con la tarjeta directo, me motivó a comprender el funcionamiento del NP.					
7. Me quedaron dudas acerca del NP.					
8. Considero que el laboratorio me dio la experiencia necesaria para comprender el funcionamiento de un NP.					

- Indica aspectos que mejorarías del Laboratorio de Procesadores de Redes: prácticas, instructor, metodología, etc. ¿Por qué?

- Indica aspectos positivos y negativos a destacar del Laboratorio de Procesadores de Redes. (Prácticas, Instructor, Instalaciones, Prácticas interesantes, Prácticas aburridas, etc.)

POSITIVOS	NEGATIVOS

- El diseño las prácticas son convenientes para poder empezar a desarrollar o implementar algoritmos en el Procesador de Redes. ¿Por qué?

- ¿ En que nivel sientes que se encuentran las prácticas del Laboratorio de Procesadores de Redes en comparación con otros laboratorios?

  o Básico o Intermedio o Avanzado

  ¿Por qué?

- ¿Cuáles de estos aspectos (en escala de 0 a 5) te ha ayudado a realizar las prácticas del laboratorio de NP? (El cero es bajo y cinco es alto)

ESFUERZO COGNITIVO QUE FOMENTA LABORATORIO NP	PC-x86 0-5	IXP2400 0-5
Capacidad de búsqueda de información		
Lógica y razonamiento		
Memoria		
Comprensión e interpretación		
Imaginación y creatividad		
Deducción		
Cálculo		

Agradecería cualquier comentario que sea importante para considerarlo en este laboratorio:

Muchas gracias por tu sinceridad
Octaviano Flores González

Apéndice C

Prácticas Haciendo uso de una Computadora Convencional

Instituto Tecnológico y de Estudios Superiores de Monterrey
Práctica de Laboratorio # 1
Optimización en el procesamiento de datos: Tabla Hash

Marco teórico

Existen muchas formas de almacenar datos en una computadora: a la forma de almacenar estos datos en memoria comúnmente se les llama estructura de datos. Sin embargo, el compromiso que existe entre la necesidad en velocidad de acceso a los datos, velocidad de almacenamiento, ahorro de recursos y escalabilidad de la estructura, comúnmente nos hace decidir por una estructura en específico; la realidad es que no se puede tener todo a la vez. En redes, en especial cuando se trata del procesamiento de paquetes, lo que se desea es que el sistema pueda procesar a la mayor velocidad posible. En muchas ocasiones, nuestras aplicaciones requieren acceder a los datos utilizando como clave para el acceso parte de esos mismos datos. Esta necesidad de acceso asociativo es una situación muy común en el mundo real. Además de esto, es bien sabido que el acceso a memoria es algo que limita la velocidad de cómputo de la mayoría de las computadoras.

Ante esta situación, existe una estructura de datos que minimiza los accesos a memoria y de esta forma (si es bien diseñada) permite que la velocidad de acceso a los datos sea mínima, logrando el acometido que se tiene en redes con relación al procesamiento de paquetes. A esta estructura de datos se le conoce como tabla de dispersión de datos, o más comúnmente tabla hash, de la analogía con el significado estándar (en inglés) de dicha palabra en el mundo real: picar y mezclar.

Una tabla hash consiste básicamente en aplicar una función a una llave (algún dato que permita indexar los registros de la tabla, puede ser una cadena de caracteres o un número real), el valor que te regresa la función es un apuntador a una tabla, la cual contiene los registros de datos. La idea es que cuando se quiera hacer una búsqueda del dato, simplemente se aplique la función de hash y se pueda obtener directamente el apuntador al registro asociado, sin necesidad de hacer una búsqueda exhaustiva como es el caso de muchas otras estructuras de datos. Debe de ser claro, de esta forma, que el tiempo de acceso va a depender solamente de la complejidad de la función hash, por lo que este tiempo va a ser un valor constante, independiente (por lo menos idealmente) del número de registros de la tabla.

Sin embargo, las cosas no son tan sencillas, ya que en la práctica no se puede generar una función hash que genere un apuntador único para cada llave, por lo que cuando dos llaves diferentes generan un mismo apuntador, existe una colisión. Existen

muchos métodos para tratar las colisiones; sin embargo el que se sugiere en el libro y el que se sugiere aquí es el de doble hash con direccionamiento abierto. Doble hash se refiere a aplicar una nueva función hash cuando ha sucedido una colisión para poder crear un apuntador diferente; direccionamiento abierto se refiere a hacer solamente uso del arreglo de memoria reservado, y no hacer uso de listas o alguna otra estructura de datos para extender el arreglo.

Como desventaja de las funciones hash se tiene el hecho de que la necesidad de tener los datos dispersos en la tabla comúnmente tiene como consecuencia que se desperdicien recursos de memoria; por su puesto, esto es una desventaja mínima teniendo en cuenta las grandes ventajas que posee.

Objetivo

Aprender sobre un concepto que está enfocado a optimizar el procesamiento de datos y por consiguiente, el procesamiento de paquetes en redes. En específico, el objetivo es implementar una tabla de hash desde cero.

Descripción

Se pide un programa donde se implemente una tabla de hash en memoria dinámica. La tabla de hash debe de tener la capacidad de inserción y búsquedas de datos. Se sugiere el uso del algoritmo que se encuentra en el libro de texto (hash doble con direccionamiento abierto), pero cualquier otro algoritmo de hash que sirva para el mismo propósito estará bien. Se pide además que la tabla de hash tenga un propósito específico: sirva para procesar direcciones MAC; es decir, el programa deberá de tener la capacidad de insertar y buscar direcciones MAC en la tabla de hash.

Nota: Esta tabla de hash se usará en prácticas posteriores con algunas modificaciones, por lo que es muy importante la realización de la práctica y el total entendimiento de la misma.

Ing. Jorge A. Bustamante Horta (Ago-Dic 06)
Ing. Octaviano Flores González (Ago-Dic 06)
Ing. Roberto De León (Ene-May 07)

Instituto Tecnológico y de Estudios Superiores de Monterrey
Práctica de Laboratorio # 2
Implementación de un Bridge

Marco teórico:

Un bridge es un dispositivo de red que opera a nivel 2 del modelo de capas OSI. Esto quiere decir que un bridge NO es capaz de rutear paquetes, y es por esta misma razón que ni siquiera se entera de que protocolo de red está encapsulando el frame (un bridge solo ve frames). A diferencia de un hub, el cual crea un solo dominio de colisiones, el bridge es capaz de separar una red en dos dominios de colisiones diferentes; y esta es su principal ventaja.

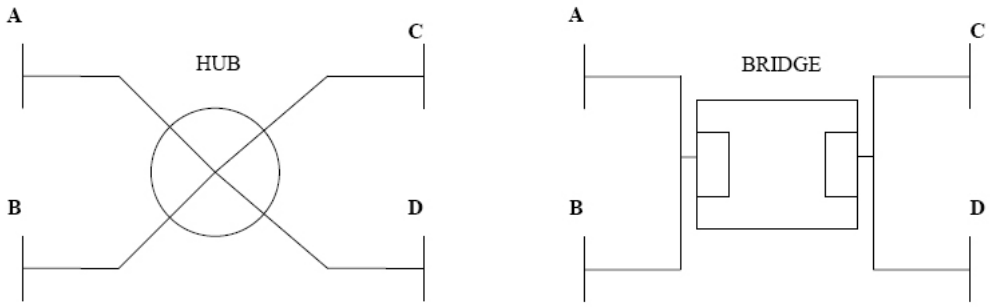


Figura C.1: (a) Hub y (b) Bridge

Un hub es simplemente una conexión compartida, un alambrado común. A,B,C y D están luchando por el mismo medio y es por está razón que las colisiones son mucho mas probables a ocurrir.

Un bridge separa a las computadoras C y D del dominio de colisiones que comparten A y B, de esta forma A solo lucha con B por el medio, mientras que C y D solo luchan por el medio entre ellos. Si C le manda un frame a D, tanto A como B deben de pasar inadvertidos de lo sucedido. Con todo esto debe de ser claro que el bridge ha separado la red en dos dominios de colisiones.

Un bridge más sofisticado podría además convertir de un protocolo de enlace de datos como lo es Ethernet a otro protocolo diferente (no aplica en está practica).

Objetivo:

El objetivo de esta práctica consiste en implementar en software un "Puente de Red"(Bridge) entre dos interfaces de red (NIC), dentro de una misma computadora

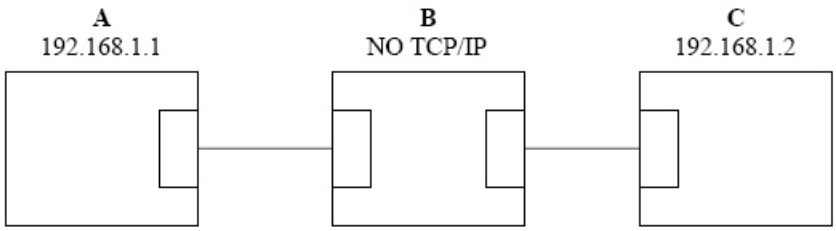


Figura C.2: Topología del Bridge

convencional.

Escenario:
Se tienen 3 computadoras: una de ellas tiene dos NIC instaladas y es la que servirá como Bridge; las otras dos sólo necesitan una NIC cada una y servirán solamente como terminales de prueba.

En la Figura C.2, las computadoras A y C son las terminales, mientras que la computadora B es la que tiene las dos interfaces y funcionará como Bridge. La computadora B no tiene implementado el protocolo TCP/IP en ninguna de sus interfaces. La computadora A y C tienen implementado el protocolo TCP/IP, y tienen configurada una IP estática 192.168.1.1/24 y 192.168.1.2/24, respectivamente. Se espera que la computadora A pueda hacer un ping exitoso a la computadora C.

Descripción:

La elección del lenguaje de programación a usar, así como las bibliotecas para interactuar con la NIC que se necesitan para realizar esta práctica está abierta a los alumnos; sin embargo, la explicación que se da en este apartado es referente a un lenguaje y a una biblioteca en particular: C++ y WinPCap. Se recomienda usar Visual C++ 6.0 para la realización de la práctica, aunque cualquier otro IDE estará bien. La biblioteca de desarrollo de WinPCap así como la documentación y ejemplos de uso de la misma se pueden encontrar en http://www.winpcap.org .

El API de WinPCap está desarrollado en C puro, por lo que en realidad no es necesario un archivo de C++ para realizar la práctica; sin embargo, quizás sea conveniente renombrar el archivo fuente .c como .cpp para poder hacer uso de estructuras de datos ya desarrolladas para C++.

En esta práctica, se espera que se implemente una tabla de hash como parte modular en el algoritmo del bridge: la tabla debe de tener la capacidad de guardar las

direcciones MAC destino que se van leyendo en los paquetes, y posteriormente poder buscar a que segmento de red pertenece una dirección dada.

Para la realización de la práctica también es necesario el uso de threads para poder estar escuchando por las dos interfaces a la vez; el uso de la técnica de polling(escuchar por una y luego por la otra consecutivamente) es insuficiente debido a que las funciones de recepción de paquetes son bloqueantes y una NIC no se podría dar el lujo de esperar a que la otra reciba un paquete para seguir operando.

Se recomienda además el uso de un Sniffer como Ethereal (http://www.ethereal.com) para poder debugear el funcionamiento del programa en la red.

El set de funciones definidas en el API de WinPCap que se pueden ocupar en esta práctica y que se recomiendan usar para evitar complicaciones de compilación, son las siguientes:

int pcap_findalldevs (pcap_if_t **alldevsp, char *errbuf)
Construct a list of network devices that can be opened with pcap_open_live().

pcap_t * pcap_open_live (const char *device, int snaplen, int promisc, int to_ms, char *ebuf)
Open a live capture from the network.

int pcap_loop (pcap_t *p, int cnt, pcap_handler callback, u_char *user)
Collect a group of packets.

typedef void(* pcap_handler)(u_char *user, const struct pcap_pkthdr *pkt_header, const u_char *pkt_data)
Prototype of the callback function that receives the packets.

int pcap_sendpacket (pcap_t *p, u_char *buf, int size)
Send a raw packet.

void pcap_freealldevs (pcap_if_t *alldevsp)
Free an interface list returned by pcap_findalldevs().

Nota: para mas información sobre esta funciones se puede referir a la documentación de WinPCap, bajo el apartado de Exported functions.

```
HANDLE CreateThread(
  LPSECURITY_ATTRIBUTES lpThreadAttributes,  // dirección de los atributos de seguridad del hilo
  DWORD dwStackSize,     // tamaño inicial de la pila del hilo en bytes
  LPTHREAD_START_ROUTINE lpStartAddress,   // dirección de la función del hilo
  LPVOID lpParameter,    // argumento para nuevo hilo
  DWORD dwCreationFlags,  // flags de creación
  LPDWORD lpThreadId     // dirección del identificador del hilo creado  );
```

Para el uso de threads se recomienda usar la función CreateThread que se encuentra en la biblioeca «Windows.h» que tiene en la Figura C

Nota: investigar como se debe de definir la función del thread.

El algoritmo que le da funcionalidad al bridge se encuentra en el libro de texto. A esta funcionalidad posiblemente sea necesario agregarle una validación para evitar un problema que surge: este problema consiste en que cuando una NIC recibe un paquete y lo manda por la otra NIC, esta ultima NIC al momento de mandar el paquete también lo recibe en su thread correspondiente y lo vuelve a enviar a la otra NIC; esto concluye en que los paquetes se queden ciclados produciendo un bajo desempeño de la red y posteriormente una saturación de la misma con sus correspondientes implicaciones (TIP: hay que validar también la dirección fuente para evitar este problema).

Ing. Jorge A. Bustamante Horta (Ago-Dic 06)
Ing. Octaviano Flores González (Ago-Dic 06)

Instituto Tecnológico y de Estudios Superiores de Monterrey
Práctica de Laboratorio # 3
"Clasificador de Paquetes"

Marco Teórico:

Cuando se refiere al procesamiento de paquetes, la velocidad de decisión con respecto a que flujo mandar el paquete es un aspecto muy importante a considerar; en otras palabras, la tarjeta de red debe de tener una capacidad de decisión adiestrada para poder "aventarle la bolita" al flujo correspondiente de la manera más rápida posible, y así poder seguir procesando los paquetes venideros (comúnmente los próximos en la fila). Para lograr una rápida decisión de flujo, existen varios métodos que sirven para este propósito, entre los más importantes se encuentran la demulticanalización y la clasificación de paquetes. La demulticanalización se refiere a ir checando encabezado por encabezado de cada protocolo (capa por capa), donde se ocupa validar que protocolo encapsula el protocolo anterior. La clasificación se refiere a identificar ciertos campos de todos los "supuestos"encabezados en un solo paso, para poder seleccionar a que flujo corresponde el paquete. Se puede observar de esta forma, que aunque la demulticanalización es más general y es un proceso más estructurado, la clasificación es un proceso que involucra mayor velocidad y por ende, permite que se procesen mayor número de paquetes consecutivos. Además de esto, se puede también optimizar la clasificación para validar primero los encabezados que son menos probables a ocurrir, de forma tal que se pretende descartar las opciones de clasificación lo más rápido posible. La clasificación de paquetes es algo muy usado en los dispositivos de red actuales, y muchos fabricantes ya están implementando en hardware este proceso para agilizar aún más el procesamiento de paquetes en dispositivos de red de alta velocidad.

Objetivo:

El objetivo de la práctica es implementar un clasificador de paquetes simulado (una vez que se clasifica el paquete, solo se despliega a que flujo pertenece el paquete y no se manda a ese flujo en realidad).

Descripción:

Se pide un programa que realice la clasificación de paquetes que están llegando por la red. El instructor mandará paquetes de diferentes tipos desde una computadora externa conectada directamente a la computadora donde está corriendo el programa, para observar si la clasificación es la correcta.

Figura C.3: Topología

De acuerdo a la Figura C.3 presentada se va a desarrollar un programa que va a tener la clasificación implementada en el Host 2.

Se desea que conforme vayan llegando los paquetes se vayan clasificando y se vaya desplegando en una tabla el número de paquetes y el número de bytes recibidos por el flujo en cuestión.

Se desea que el clasificador por lo menos clasifique los siguientes protocolos:

- Flujo 1: ICMP
- Flujo 2: Web
- Flujo 3: FTP
- Flujo 4: UDP

Ing. Jorge A. Bustamante Horta (Ago-Dic 06)
Ing. Octaviano Flores González (Ago-Dic 06)

Instituto Tecnológico y de Estudios Superiores de Monterrey
Práctica de Laboratorio # 4
"Implementación de un NAPT"

Marco teórico:

La red más grande del mundo, Internet, ha tenido un gran crecimiento en la última década. El protocolo TCP/IP, no fue diseñado en sus principios con el propósito de atender a una cantidad semejante de usuarios. El principal problema es que las direcciones lógicas de IPv4, en parte debido a la mala distribución de las mismas, se han ido agotando. Hace unos años, debido a este problema, se pensaba que sería el fin de IPv4 como tal, e incluso se empezó a desarrollar un nuevo estándar, el cual no se ha generalizado hasta la fecha: IPv6. El problema de la creciente necesidad de direcciones IP logró llamar la atención de ingenieros de todo el mundo, logrando consigo que se presentaran algunas propuestas de solución. Aunque IPv6 pretendía ser la solución más viable al problema, la realidad es que la infraestructura de TCP/IP se vendría abajo con un cambio tan radical, lo que representaría la perdida de una fuerte suma de dinero previamente invertido por la mayoría de las grandes empresas. Aunque el cambio de IPv4 a IPv6 parecía inminente, surgió una propuesta de solución que pareció ser la más apropiada en su momento, y la cual trajo consigo una repentina esperanza a la vida de la infraestructura de IPv4 previamente establecida: Network Address Port Translation (NAPT).

NAPT (también conocida como Port Address Translation, PAT) surgió como un derivado de la tecnología ya conocida como NAT (Network Address Translation). NAT, en su faceta más básica, consiste en la traducción de direcciones IP de una red a otra; es decir, permite la traducción de direcciones IP de una a una. NAPT, a diferencia del NAT clásico, no solo permite la traducción de direcciones IP de una a una, sino que también permite la traducción de direcciones IP de una a muchas. Para lograr este acometido, la implementación usa la asignación de puertos para relacionarla con una conexión en específico. De esta manera, NAPT puede traducir tantas conexiones como el número de puertos disponibles a asignar tenga el dispositivo en cuestión.

El dispositivo que implemente la NAPT tiene ciertas características bien definidas:

- Se puede visualizar como un dispositivo con posibilidad de recibir y mandar paquetes en dos segmentos de red diferentes (por lo que usualmente tiene 2 interfaces).
- Tiene un rango definido de puertos fuente a asignar en el segmento que tiene salida a la Internet (ó red default).

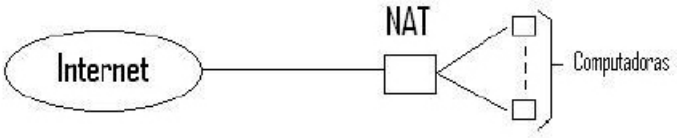


Figura C.4: El concepto de una Topología NAPT

- Usualmente se implementa en el borde de alguna red, para poder dar acceso a una red privada por medio del procesamiento de paquetes.
- Realiza cambios en los encabezados de los paquetes para cualquier paquete que pretenda pasar de un segmento a otro.

Ya que el dispositivo conecta dos segmentos diferentes, es conveniente hacer una distinción entre el tipo de direcciones IPs que se asignan para cada segmento. Como el uso de una NAPT es usualmente poder darle acceso a Internet a una red local, se distinguen dos tipos de red diferentes: la red pública, la cual contiene al segmento que está conectado a Internet; y la red privada, la cual contiene al segmento que conecta a los distintos hosts deseosos de acceso a Internet, conectados a la red local. Al IP que tiene cada uno de los hosts dentro de la red privada, se les llama comúnmente IP privado, mientras que al IP que contiene la interfase del dispositivo NAPT conectada al segmento que forma parte de la red privada, se le llama IP privado.
Cabe destacar que la idea general es que cualquier "host"dentro de la red pública (cualquier usuario de Internet) no sea capaz de darse cuenta de que existe una cantidad de hosts "colgados"de la dirección IP pública del dispositivo NAPT; simplemente visualizan a esa dirección IP como un dispositivo único; la existencia de la red privada es completamente transparente para la red pública (tan es así, que incluso el NAT en general ha sido implementado como simple medida de seguridad en una gran cantidad de casos).

Ejemplo:

Para explicar más a fondo el proceso que realiza el NAPT, se hace referencia a un ejemplo (Figura C.4). Supóngase que se tiene un NAPT que tiene un segmento que está conectado a Internet, con su respectiva IP pública asignada, quizá dinámicamente por algún servidor DHCP del ISP (Internet Service Provider). Por otro lado, la otra interfase de la NAPT está conectada a una red local 10.0.0.0/8. Supóngase que la dirección de IP pública que el ISP le asignó a la NAPT es la de 131.178.6.5. Supóngase que estáticamente se asignó la dirección IP privada de 10.0.0.1 a la segunda interfase del NAPT. Supóngase que a todos los hosts dentro de la red local se les asigna la puerta de enlace predeterminada de 10.0.0.1; es decir, se les asigna como puerta de enlace

predeterminada la interfase que corresponde a la NAPT. Supóngase también que existe una puerta de enlace predeterminada para la primera interfase de la NAPT con dirección IP de 131.168.6.1 dentro de la red pública. Teniendo todo esto en consideración, supóngase que un host con la dirección 10.0.0.101 dentro de la red local, intenta abrir una página en su navegador hacia www.google.com (64.233.103.107). En este momento el navegador crea un paquete de conexión TCP, con MAC destino correspondiente a la puerta de enlace predeterminada (la MAC de la NAPT), MAC fuente correspondiente a la del host, puerto destino 80, dirección de IP destino de 64.233.103.107, dirección fuente 10.0.0.101 y un puerto fuente 2000 (puerto asignado dinámicamente por el Sistema Operativo del host). En el momento en que el NAPT recibe el paquete TCP, identifica la conexión por medio de todos estos campos, guarda un registro de la misma en una tabla si es que no existía el registro con anterioridad, realiza ciertos cambios en los campos correspondientes del paquete en cuestión, actualiza los campos de checksum del encabezado IP y del encabezado TCP, y manda el paquete modificado por la otra interfase (la interfase conectada a la red pública).

Los encabezados que tiene que cambiar la NAPT antes de reenviar el paquete son: dirección IP y puerto fuentes, MAC destino y MAC fuente. Para el puerto fuente, la NAPT le asigna un puerto disponible dentro de un rango previamente establecido. De esta manera, para el ejemplo en particular, el paquete modificado sale de la NAPT hacia Internet con los siguientes campos: protocolo TCP (0x06), MAC destino la correspondiente al dispositivo con dirección IP 131.168.6.1, MAC fuente la correspondiente a la interfase de la NAPT por la que se manda este paquete, dirección IP destino 64.233.103.107, puerto destino 80, dirección IP fuente 131.178.6.5, puerto fuente 61000 (asignado por la NAPT).

Este paquete es transmitido por la primera interfase de la NAPT hacia el dispositivo correspondiente a la puerta de enlace predeterminada (131.178.6.1). Este último dispositivo es el encargado de "rutear"el paquete al destino (google). Cuando el paquete llega al servidor de google correspondiente, éste generará un paquete de acknowdlege y lo enviará de regreso al dispositivo NAPT. El paquete será "ruteado" a través de Internet y posteriormente llegará de regreso a la interfase de la NAPT conectada a la red pública. Este paquete de acknowdlege tendrá los siguientes datos al llegar al NAPT: protocolo TCP(0x06), MAC fuente correspondiente a la puerta de enlace predeterminada del segmento de la red pública, MAC destino correspondiente a la interfase de la NAPT conectada a la red pública, dirección IP destino 131.178.6.5, puerto destino 61000, dirección IP fuente 64.233.103.107, puerto fuente 80.

Al llegar el paquete al sistema NAPT por su interfase conectada a la red pública, el sistema validará los encabezados correspondientes para verificar si esa conexión ex-

iste como registro en la tabla. Como sí existe (ya que se creo cuando el host mandó el paquete en primera instancia por la otra interfase), el NAPT obtendrá del registro correspondiente los valores asociados de dirección IP local 10.0.0.101, y puerto local 2000, modificará los campos correspondientes, actualizará los campos de checksum del encabezado IP y del encabezo TCP y mandará el paquete modificado por la interfase conectada a la red privada.

De esta manera, al host 10.0.0.101 dentro de la red privada le llegará un paquete con las siguientes características: protocolo TCP, MAC destino correspondiente al host 10.0.0.101, MAC fuente correspondiente a la interfase de la NAPT conectada a la red privada, dirección IP destino 10.0.0.101, puerto destino 2000, dirección IP fuente 64.233.103.107, puerto fuente 80. De esta manera se completa el ciclo para este paquete y el host se prepara para mandar el siguiente paquete para sostener esta conexión que se mantiene con el servidor de google a través del sistema NAPT.

Tips de implementación:
La tabla a la que se hace referencia puede ser implementada por medio de una tabla de hash. Debido a que la tabla debe de poder ser accesada a través de dos llaves diferentes, es necesario tener dos tablas físicas para mantener registros lógicos; es decir, cuando llega un paquete por la interfase de la red pública se buscan diferentes campos para encontrar el registro, que cuando el paquete llega por la otra interfase. De esta manera es necesario mantener en memoria dos tablas hash que tengan relación entre si, para esto se puede hacer uso de apuntadores para tener referencia de un registro al otro y viceversa.
Una explicación más extensa de esta implementación se hace en el último capítulo del libro de texto.

Se recomienda definir funciones aparte para actualizar los encabezados de checksum de IP y TCP; se recomienda definir una función para modificar los demás campos de los encabezados. Se recomienda también mantener algún tipo de tabla o función para relacionar las IP con las direcciones MAC, y de esta manera emular el protocolo ARP. Todo esto es con el fin de poder desarrollar el algoritmo del sistema NAPT con mayor facilidad y claridad.

Aunque el protocolo UDP se maneja diferente al de TCP en cuanto a que no se necesita establecer una conexión, para efectos de la práctica se puede manejar UDP de la misma forma que TCP (de hecho los campos que necesitan ser modificados en un momento dado para el protocolo UDP casualmente caen dentro de los mismos apuntadores con respecto al encabezado TCP). Por todo esto, si el protocolo TCP funciona dentro del NAPT, no se debe de tener ninguna dificultad para hacer funcionar el pro-

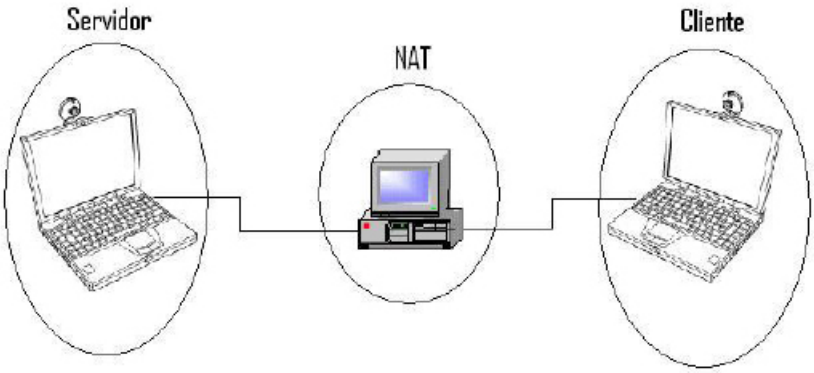


Figura C.5: Topología del NAPT del Laboratorio

tocolo UDP.

Para "debugear"el programa; es decir, para saber si los cambios a los encabezados se están haciendo correctamente, se puede hacer uso de un "sniffer". Se recomienda en específico el uso de "Ethereal", ya que posee la capacidad de ver todos los campos de los encabezados e inclusive permite ver si los "checksum"tanto de IP como de TCP están siendo calculados correctamente. Se recomienda correr el Ethereal en ambos segmentos de red al mismo tiempo al iniciar la conexión desde la computadora cliente, para poder seguir el paquete en todo momento y de esta manera poder ser capaces de identificar los errores que se pudieran encontrar en un momento dado. Se recomienda también aplicar un filtro para poder seguir solamente el paquete en cuestión y no obtener el despliegue de todos los paquetes que están siendo transferidos por la red. Para el ejemplo descrito en la práctica, se podría hacer uso de un filtro como: "ip.addr == 64.233.103.107", para filtrar solo los paquetes con dirección IP destino o fuente de www.google.com.

Objetivo:
Implementar en software un NAPT "Network Address Port Translationüsando una computadora convencional con dos NICs instaladas.

Escenario:
Se van a tener 3 computadoras de prueba. Una de ellas tiene dos NIC instaladas y es la que servirá como NAPT; las otras dos necesitan una NIC; la que emulara el Internet tendrá un servidor corriendo para poder dar servicio a la red que se quiera conectar, y la otra sólo necesitará del software de cliente correspondiente para hacer el "request"de servicio al servidor. La Figura C.5, lo ilustra.

Debido a que el stack de TCP/IP de Windows tiene su propio control de conexiones, será necesario deshabilitarlo para hacer funcionar el NAPT; no obstante, esto NO

quiere decir que dentro del programa del NAPT las direcciones lógicas correspondientes a las dos interfases no se vayan a tratar como si estas sí existieran. Por esta razón, no será necesario configurar ninguna dirección IP estática en ninguna de las dos interfases de red de la NAPT dentro de Windows; sin embargo, dentro del sistema NAPT deberá de tratarse a la interfase conectada al Servidor con la dirección IP 10.0.0.2 y a la interfase conectada al cliente con la dirección IP 192.168.1.1. Tanto la computadora "Servidorçomo la computadora "Cliente"estarán corriendo el protocolo TCP/IP y tendrán la dirección estática: 10.0.0.1 y 192.168.0.2, respectivamente. Se espera poder hacer "request.ª la dirección 10.0.0.1 desde la computadora cliente y tener acceso a los servicios proporcionados por el servidor.

Descripción:

Se desea implementar el escenario presentado anteriormente, por lo que se espera que el alumno lleve acabo la práctica haciendo uso de las herramientas vistas previamente en el laboratorio (Winpcap). Además de esto, cabe destacar que el NAPT a desarrollar tiene las siguientes características simplificadas:

- El sistema NAPT solo tiene dos conexiones.
- El sistema NAPT solamente maneja TCP y UDP.
- El sistema no va a manejar datagramas fragmentados.
- El sistema solo va a manejar comunicación inicializada de una computadora dentro de la red local.
- El protocolo ARP que necesita el NAPT para funcionar, será emulado por una tabla estática de direcciones MAC, que será introducida por el programador ya sea dentro del mismo programa antes de compilar o de manera dinámica por la interfaz de usuario.
- La tabla de registro de conexiones se creará de un tamaño considerable para evitar tener que borrar registros en un momento dado.
- No se hará uso de ningún mecanismo para la reutilización de puertos: se está asumiendo que nunca se acabarán los puertos disponibles para conexiones del NAPT.

Pre-Reporte:

1. Entregar el diagrama de estados de lo que se tiene que llevar a cabo para poder realizar la traducción de direcciones IP. La traducción de direcciones IP se refiere, desde que el paquete sale de la red privada y llega a un "host"dentro de la red pública y este mismo "hostregresa un "acknowlege", teniendo que salir de la red

pública y regresar a la red privada. Nota: pueden implementarse dos diagramas de estados distintos, uno para cada interfase.

2. Investigar la realización del cálculo de actualización de checksum con respecto a ciertos campos de los encabezados y reportar el algoritmo en pseudocódigo que se tendría que realizar para su implementación.

3. Investigar que es lo que se tendría que hacer diferente en el NAPT con respecto a los protocolos TCP y UDP, para que el sistema soportara también el protocolo de ICMP.

Ing. Jorge A. Bustamante Horta (Ago-Dic 06)
Ing. Octaviano Flores González (Ago-Dic 06)

Apéndice D

Prácticas Haciendo uso del Procesador de Red IXP2400

INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS MONTERREY

Practice #1: Tornado (Integrated Development Environment)

Pre-Practice:

Objective

Familiarize with the Integrated Development Environment debugging a simple code that calls tasks through semaphores.

Theoretical Framework:

Many courses in Computer Scince hace tools that help students in the learning process [15]. Tornado 2.2.1 provides a greater level of accessibility for users unfamiliar with Tornado, and increased productivity for experienced developers. Key features of this release are:

- The integrated version of the VxWorks target simulator (VxSim) introduced in Tornado 2.0
- The integrated version of theWindView logic analyzer for the target simulatorintroduced in Tornado 2.0
- an improved project facility
- a new debugger engine

The target simulator allows you to experience Tornado - as well as begin developing hardware-independent modules - without target hardware. The integrated version of WindView provides dynamic visualization of application behavior on the simulator. The new project facility provides automated graphical features for managing project files, configuring VxWorks, and building applications. The new debugger GUI is easier to use and provides more direct access to debug information[6].

Development:

This practice demonstrates the use of the project facility, the integrated versions of the target simulator and WindView, and the Tornado debugger. The tutorial is designed to provide new users with an accessible introduction to the key features of Tornado and their use in application development. It is also intended to provide experienced Tornado users with an introduction to major innovations in the development

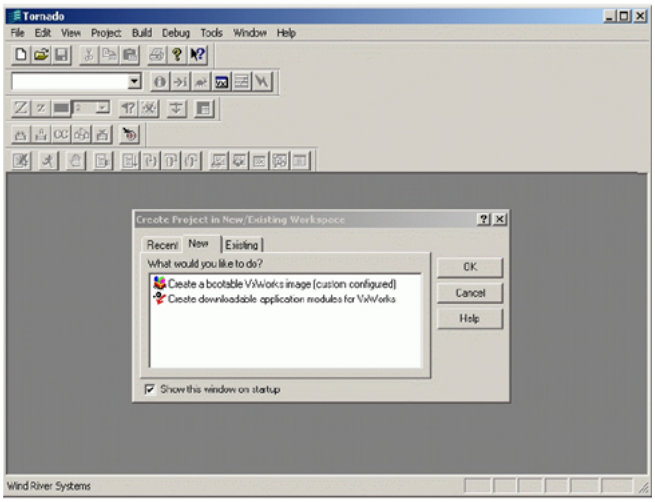


Figura D.1: New Project

environment.

The integrated version of WindView is designed solely for use with the VxWorks target simulator. WindView is available as an optional product for all supported target architectures.

Start Tornado

To start Tornado, select Start¿Programs¿Tornado2.2.1¿Tornado (with a default installation), or select the Tornado desktop shortcut (if you have created one). If Tornado is unable to connect to a Tornado registry, you will be prompted to start one. You can start a registry manually by selecting the Tornado Registry option from the Tornado program group menu.

The main Tornado window appears with the Create Project dialog box open by default (Figure D.1)

You can use the toolbars as floating palettes or dock them on the sides, top, or bottom of the main Tornado window.

Create a Project

If the Create Project in New/Existing Workspace dialog box is not displayed, click File¿New Project. Then select the option for a downloadable application (Figure D.2)

Click OK to continue.

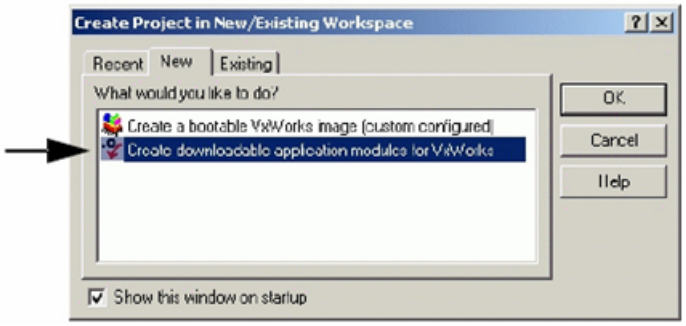


Figura D.2: Create Project

The Tornado application wizard appears. This wizard guides you through the steps of creating a new project. In the first step, the wizard prompts you for:

- A name for the project.
- The full path and name of the directory you want to use for the project files.
- An optional description of the project.
- The name and location of the workspace file, which contains information about the workspace, including which projects belong to it.

A project consists of the source code files, build settings, and binaries that are used to create an application.

A workspace is simply a grouping of one or more projects, which provides a useful means of working with related material, sharing code between projects, and associating related applications. Once a project and workspace have been created, a workspace window displays information about the projects that it contains.
You can accept the wizard's defaults for the project and workspace, but it is preferable to set them up outside of the Tornado installation tree. (If you do so, you won't have to untangle your work from the Tornado tree when you upgradeTornado.)

In the wizard dialog (displayed below):

- The project name is gizmo.
- The project directory is c:\projects\gizmo.
- The project description is lightning gizmo.
- The path and file name for the workspace are c:\projects\lightning.wsp:

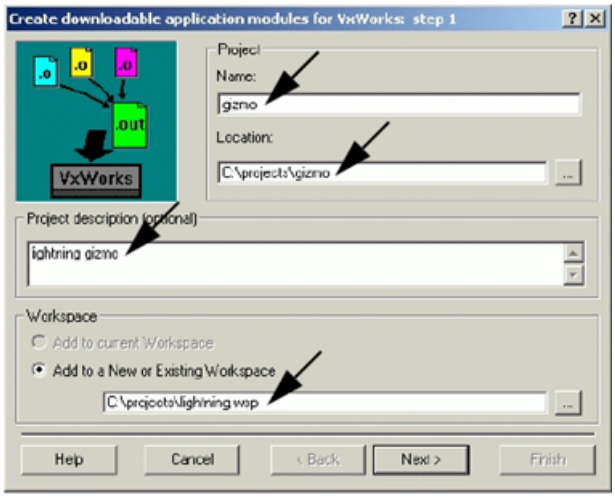


Figura D.3: Create Module

Click Next to continue (Figure D.3).

In the second wizard dialog, you identify the toolchain with which the downloadable application will be built.

A toolchain is a set of cross-development tools used to build applications for a specific target processor (preprocessor, compiler, assembler, and linker).
The default toolchain names for target simulators take the form SIMhostOsgnu (for example, SIMNTgnu).

NOTE: The directory path for the workspace must exist-or at least match the path (or an initial segment thereof) defined in the project Location field-for the wizard to complete project creation.

Select A toolchain and the option for the target simulator from the drop-down list, as illustrated in Figure D.4, click Next to continue and the final wizard dialog confirms your selections.
Click Finish to continue.

The Workspace window appears. The workspace window title includes the name of the workspace, and the window itself includes a folder for the project (Figure D.5)

NOTE: Context menus provide access to all commands that can be used with the objects displayed in, and the pages that make up, the Workspace window. Use the right mouse button to display an object's context menu.

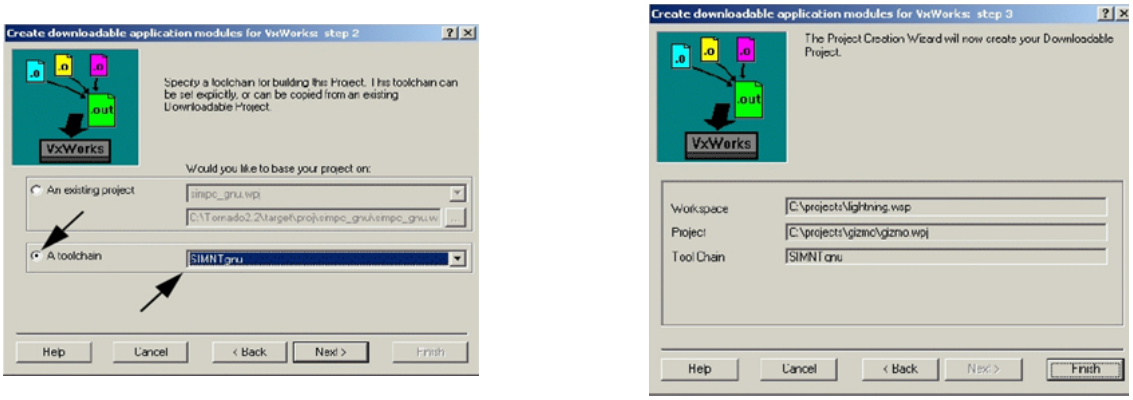


Figura D.4: Toolchain and confirmation

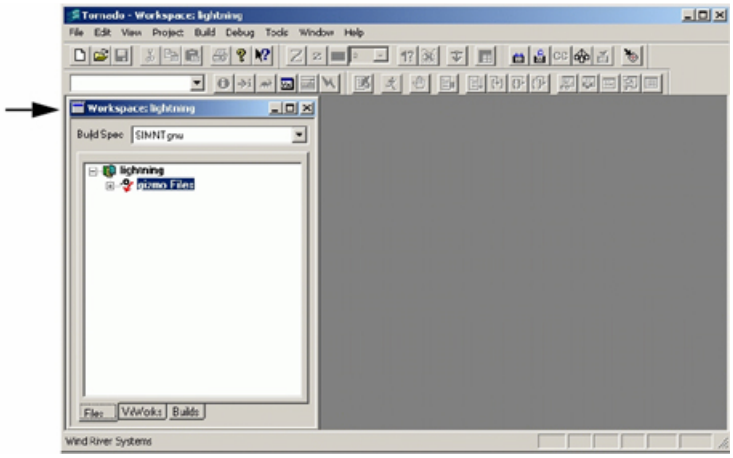


Figura D.5: Workspace

Add the Sample Source File to the Project

The sample program for this tutorial is cobble.c. It is a simple multi-tasking application that simulates a data collection system in which data is retrieved from an external source (for example, a device that generates interrupts as data comes in). The four cobble tasks are:

- tCosmos, which simulates an interrupt service routine (ISR) and generates
- new data.
- tSchlep, which collects the data.
- tCrunch, which performs calculations on the data to obtain a result value.
- tMonitor,which monitors the results and prints awarning message if they are outside of the safety range.

The cobble program also includes the progStart( ) and progStop( ) routines. Later in the tutorial, you will use these routines to start and stop the program.

Before you add the sample program to the project, copy it from the Tornado installation tree to another location, such as the project directory you have set up with the Tornado project wizard.

The sample file cobble.c is located in installDir \target\src\demo\start. In the examples provided thus far in the tutorial, c:\projects has been used as the base directory for our project work, and subsequent examples are based on cobble.c being copied to that directory.

After you have copied cobble.c to another location, add it to the project. Select Add Files from the context menu in the File view of the workspace (Figure D.6).

Then use the file browser that appears to select cobble.c.
Open the Object Modules folder in the Files view so you can see the names of the object files that are built from the project (Figure D.6).

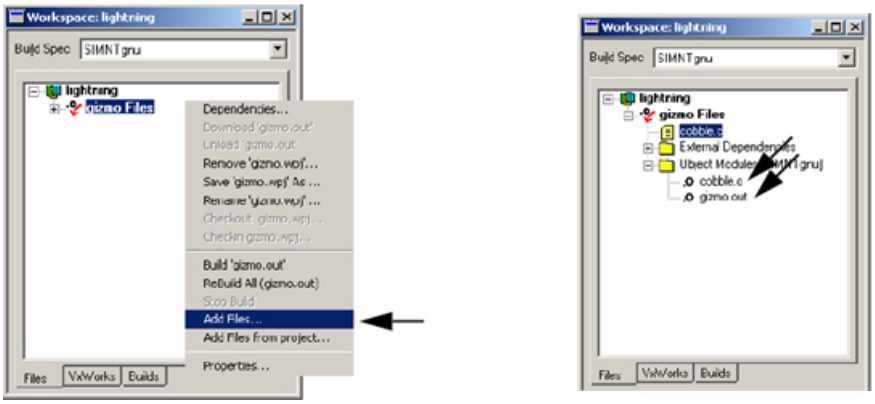


Figura D.6: Adding Code

The Tornado project facility automatically creates the file projectName.out as a single, partially linked module when a project is built. It comprises all of the individual object modules in a downloadable application project, and provides for downloading them all to the target simultaneously.

Build the Project

To review the default build settings defined when you created the project, select the Builds tab on the Workspace window, open the gizmo Builds folder, and double-

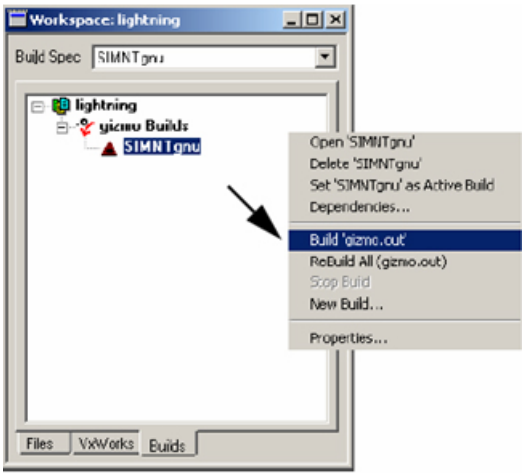


Figura D.7: Building the project

click on the default build name (for example, SIMNTgnu).

The properties sheet for the build appears, which you can use to review makefile rules and macros, as well as the compiler, assembler, and linker options used for the build. When you display the C/C++ compiler page, you will note that the Include debug info checkbox is selected by default. This selection ensures that optimization is turned off when the project is compiled with debug information.

Close the property sheet by clicking on the Cancel button.
Build the project by selecting Build 'gizmo.out' from the context menu (Figure D.7).

The option Build 'projectName.out' builds all project modules as a single, partially linked module that is optimal for downloading to a target.

Before Tornado builds the project, the Dependencies dialog box warns you that makefile dependencies have not been calculated for cobble.c.

Click OK to continue.

Tornado calculates makefile dependencies and proceeds with building the project. If any external dependencies are found, they are automatically included in the project and listed in the External Dependencies folder in the Files view of the workspace.
The build output window displays any errors and warnings. In this case, there are none.

Close the Build Output window with the window control.

Download the Project to the VxWorks Target Simulator

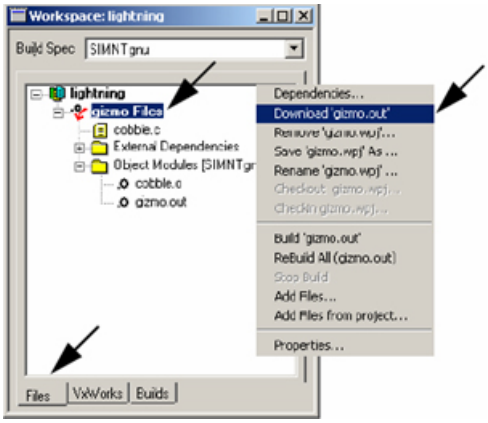


Figura D.8: Download the project

You can download your project from the Files view of the workspace, and start the integrated simulator, all as part of the same process.
First display the Files view of the workspace window.

Then select the project folder name with the right mouse button and select the Download 'gizmo.out' option from the context menu (Figure D.8).

Tornado prompts you to start a simulator,
Click Yes to continue.

The VxSim dialog box appears (Figure D.9).

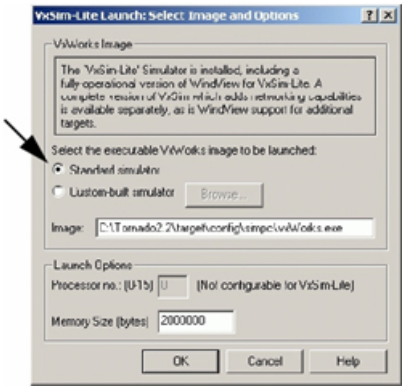


Figura D.9: Download the project

Leave Standard Simulator selected, and click OK to continue.
The target simulator window opens, and Tornado prompts you to start a target server.
Click OK in the VxSim-Lite Launch dialog to continue.

NOTE: Only one instance of the integrated VxWorks target simulator (VxSim-Lite) can be run at a time. To stop the simulator, close the VxWorks Simulator window.

A target server manages all communication between Tornado host tools, such as the debugger, and the target. The convention for target server names is target-Name@hostName.

In this example, the name of the target is vxsim (the default for the integrated simulator), and the name of the PC host is ala-tau.

The name of the target server is displayed in the Tornado launch toolbar (Figure D.26).

Figura D.10: Name of the target server

The toolbar includes buttons for starting Tornado tools such as the browser, shell, and debugger. The buttons are activated when the target server is started.

NOTE: Both the browser button and the shell button both have the letter i in them, which may be confusing. The i in the shell button follows the -> shell command prompt, where it is a shell command that displays target system information. The i in the browser button stands for information about the target.

Run the Application from the Tornado Shell

Before you run an application, it is often useful to configure and start the Tornado debugger, so that the debugger can respond automatically to any program exception.

The Tornado debugger (CrossWind) combines the best features of graphical and command-line debugging interfaces. The most common debugging activities, such as setting breakpoints and controlling program execution, are available through convenient point-and-click interfaces. Similarly, program listings and data-inspection windows provide an immediate visual context for the crucial portions of your application.

To configure the debugger, select Tools¿Options¿Debugger from the main Tornado window.When the Options dialog appears, select Always under Auto attach to tasks, so that the debugger will automatically attach to a task when an exception occurs.

Click OK to continue. For more information, see the Tornado User's Guide: Debugger.
To start the debugger, use the debugger button in the Tornado launch toolbar.

The back trace and variables windows open (you may close them), and the status line at the bottom of the main Tornado window announces that the debugger is running.

The Tornado shell (also known as WindSh) is a C-language command interpreter that allows you to invoke any routine from the shell command line that has been downloaded to the target. The shell also includes its own set of commands for managing tasks, accessing system information, debugging, and so on.

You can run the program from the shell command line. To start a shell, click the shell button.

When the shell window opens, run the program by entering the name of the main routine, progStart, at the command line (Figure D.11).

Figura D.11: Shell Window

.

Examine Target Memory Consumption The Tornado browser is a system-object viewer that is a graphical companion to the Tornado shell. The browser provides display facilities to monitor the state of the target system, including summaries of active tasks, memory allocation, and so on.
To start the browser, click the browser button in the Tornado launch toolbar.

When the browser appears, Select Memory Usage from the drop down list and click the periodic refresh button.

This will update the display every few seconds, and you will note that cobble.c has a voracious appetite for memory (Figure D.12) .

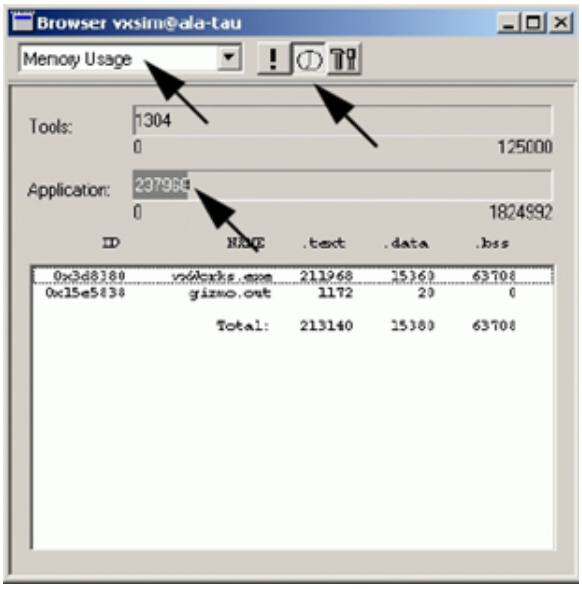


Figura D.12: Memory Usage

.

Stop the browser with the window control in the title bar.

Examine Task Activity

WindView is the Tornado logic analyzer for real-time applications. It is a dynamic visualization tool that provides information about context switches, and the events that lead to them, as well as information about instrumented objects such as semaphores, message queues, and watchdog timers.

Click the WindView button in the Tornado launch toolbar to display the WindView Control window.

The WindView Collection Configuration dialog box also appears (Figure D.13). Select Additional Instrumentation from the drop-down list and click OK.

.

To begin data collection, click the GO button in the WindView Control window.

NOTE: Except where otherwise specified, the default configuration of WindView is used for this exercise. If WindView settings have been changed during another session, the behavior may vary from the description below.

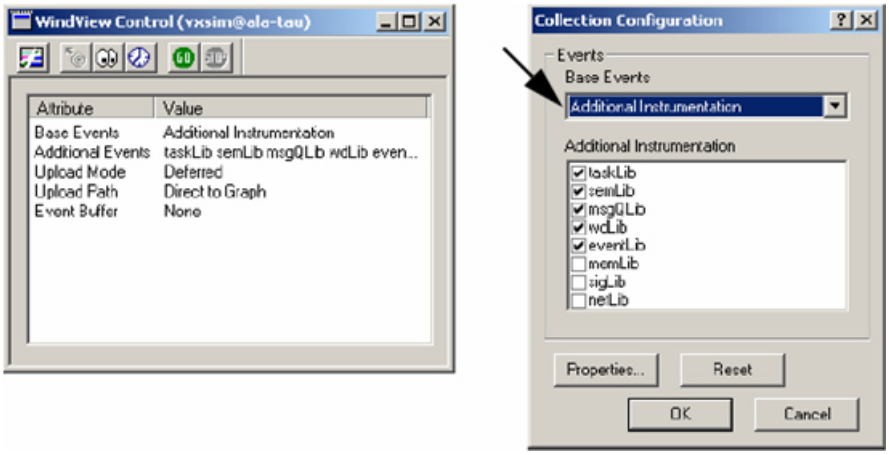


Figura D.13: WindView Colleciton Configuration

Wait a few seconds, and then click the update button in the WindView Control window to update the status of data collection.

When the Buffers value listed under Current Content reaches at least two, click the Stop button to end data collection.

You may notice a certain sluggishness in your machine's performance, which is due to the sample program's runtime misbehavior on the integrated simulator.
Before you upload the WindView data from the simulator target to the host, stop the sample program by entering progStop at the shell command line.

Then use the upload button to upload the data. A view graph is displayed while the data is uploaded.

When you maximize the view graph window, the data display should look something like the Figure D.14 (The color of the view graph display can be changed with Tools>WindView>Options.)

.

You can use the zoom buttons on the WindView toolbar to enlarge or decrease the size of the scope of the display, and scroll bar at the bottom of the view graph to display the rest of the data. (The zoom 100 % button displays all of the data at once.) The flag-shaped icons indicate semaphore gives and takes. The horizontal lines indicate task states (executing, pending, ready, and so on). For information about the meaning of the WindView icons, select Help>WindView Help>Legend.

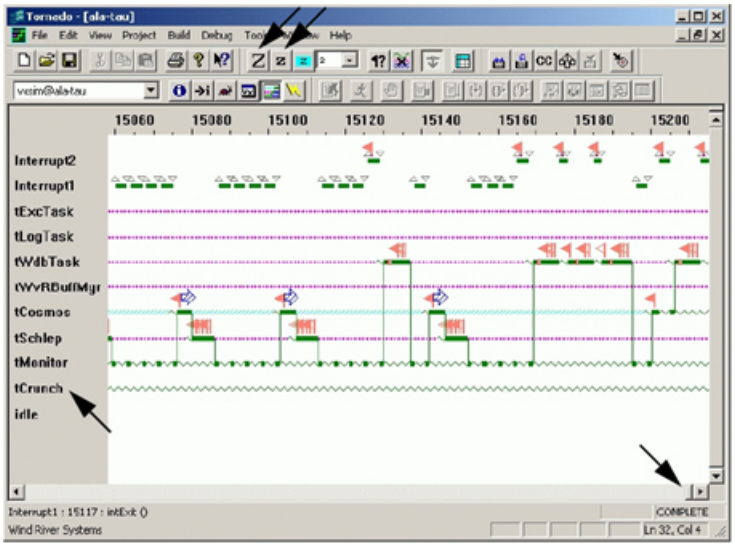


Figura D.14: A wrong Graph

Note that tCrunch, the task that processes data and removes nodes from a linked list, never runs.

Close the view graph and click No in response to the dialog that prompts you to save the data with: Save changes to hostName?

Close the WindView Control window as well.

Fix Task Priorities and Find the Next Bug

Fix the Last Bug and Take it for a Spin

Use the context menu in the Files view of the workspace to select the ReBuild All (gizmo.out) option. When the build is done, close the Build Output window.
Download the application to the target with the Download 'gizmo.out' option from the context menu.

To be sure that everything is working, start the debugger with the debugger button.

Start the program again from the Tornado shell with the command progStart.

Click on the Windview button to display the WindView Control window. Then click the GO button in the WindView Control window to begin data collection.

Wait a few seconds, and click the update button in the WindView Control window to update the status of data collection.

When the Buffers value listed under Current Content reaches at least two, click the Stop button to end data collection. Then use the upload button to upload the data. A view graph is displayed while the data is uploaded.

Maximize the view graph window and scroll through the WindView data. You should find that all tasks, including tCrunch, are running in an orderly manner, and the view graph data should look like the Figure D.15

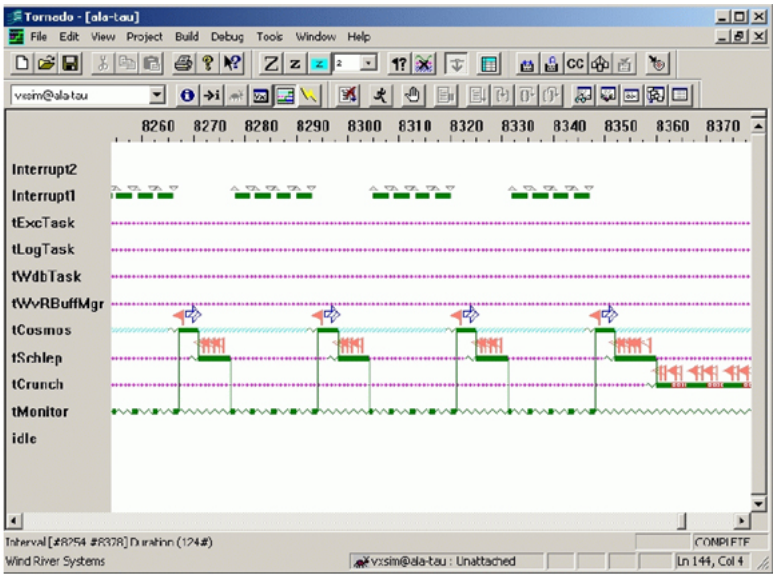


Figura D.15: A good Graph

.

You can check memory consumption once again with the browser by clicking the browser button in the Tornado launch toolbar.

When the browser appears, select Memory Usage from the drop-down list and click the browser's periodic refresh button. Every few seconds the display will be updated. Notice that memory consumption fluctuates modestly within a constant range.

Moreover, output in the simulator window indicates that the program is humming along nicely, providing you with information about when it's hot and when it's not (Figure D.16).

.

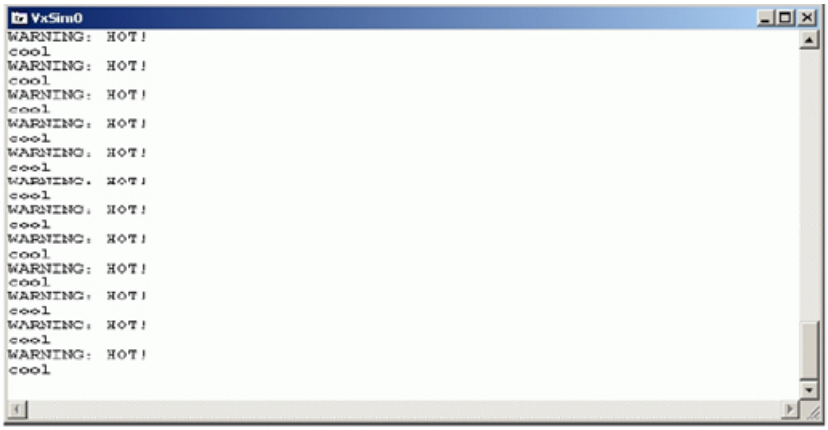


Figura D.16: The Simulation

Stop the program by entering progStop at the shell command line.

Practice:

Objective:

Learning basic command of VxWorks through the Target Server of Tornado.

Theoretical Framework:

A Tornado target server runs on the host and manages communications between Tornado host tools (such as the shell, debugger, and browser), and the target system itself (Figure D.50). A target server includes the host-resident target symbol table, and an object-module loader that inserts application modules into a running target system.

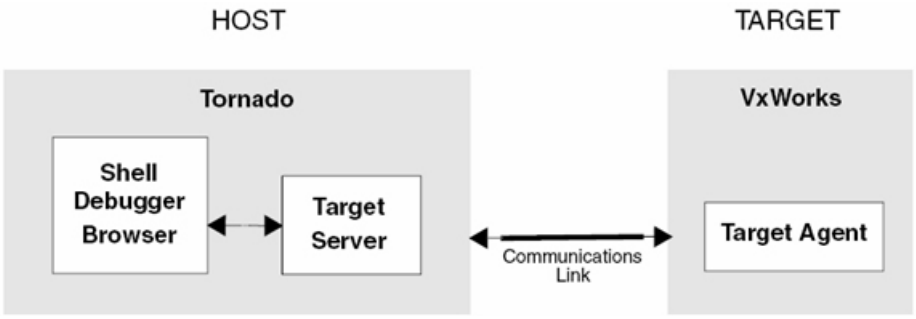


Figura D.17: Target Server in Development Environmnet

.

The target server communicates with the target system through the target agent, which runs on the target system either as a VxWorks task or externally from VxWorks. In order to communicate with the target agent, the target server uses a communication back end configured for the same communication protocol and transport layer as the target agent.

A target server must be configured for a target and started before host tools can interact with the target. It must be configured with a communications back end that supports the communication protocol and transport layer used by the target agent, or it will not work. When a target server is started, it identifies the target agent by the network name of a target board on which the target agent is running.

Development:

Using the Configure Target Server Dialog Box

The Configure Target Servers dialog box is a GUI tool for configuring and starting a target server. To display the box, select Tools>Target Server>Configure.

When you press the New button, Tornado displays the dialog box with default options.

Configure Target Server

.

Configure a new Target server clicking the New button.

Fill the fields with the follow properties:

- Description: Configuracion1
- Target Server Name: IP address of the host.
- Target Server properties: Back End
- Available Back End: wdbrpc (supports any IP connection)
- Timeout and Re-try: 3
- Target Name/IP Address: NP IP Address.

Then click to the button Launch.

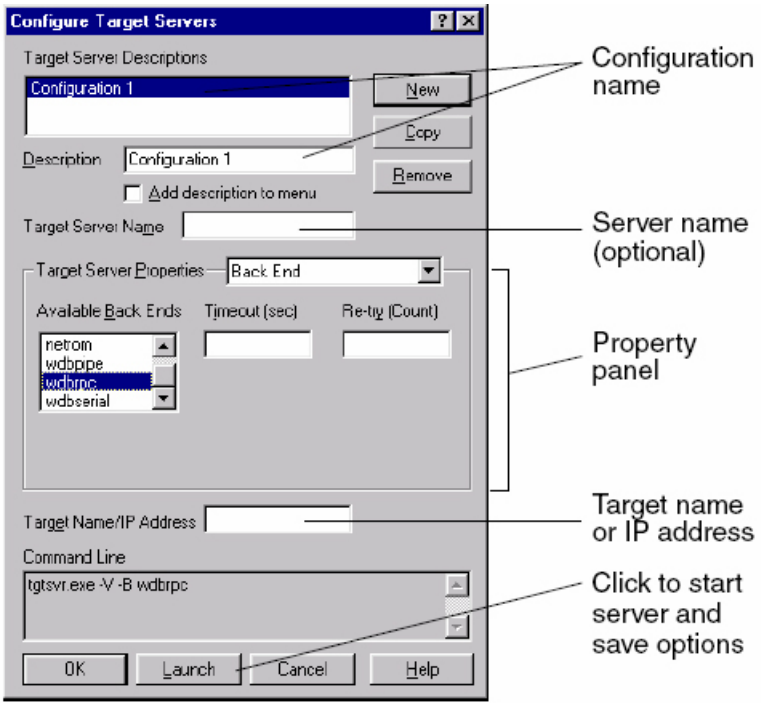


Figura D.18: Target Server in Development Environmnet

Note: For this point you must substitute the file of enp2611_be (C:\RadiSys\ENP_SDK_3.5\BoardSupport), with the one that your instructor gave you. Also, you must install the Software Develop Kit of IXA.

After having a successful connection you must proceed to answer the questionnaire that is attached to this practice.

This questionnaire must be printed and hand in to the instructor at the end of the practice. In addition you must show the pre-practice and all the commands that you test on the Vxworks.

Ing. Octaviano Flores Gzz. (Enero-Mayo, 07)

INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS MONTERREY

Practice #2: Developer Workbench

Pre-Practice:

Objective

Familiarize with the DevWorkbench debugging using a simple code that increments a register in each cycle.

Theoretical Framework:

Just imagine a command-line interface displaying the status of 128 threads running in parallel and thousands of packets being classified and queued for transmit. You could create a command-line watch that captures simulation state printing PC; printing a list of threads that are active; printing a packet data that is received, stored in DRAM, and transmitted; and printing a packet descriptor and queuing information. This situation was life with IXP2XXX network processor simulator before the Developer Workbench came along [9].

The problem is, simply, how do you make the tools do something and present to you the resulting information? That is what GUI does. It gives control over the functionality and information organized so you see what you need to know, when you need to know it. Even with a GUI, you will feel more comfortable doing some things from the command line, such as quick memory examination, or a go 100 (run the simulator 100 cycles) command. With the Developer Workbench, you can do all these operations from either the GUI or command line, plus you can assign shortcut keys for operations you do most frequently.

The way programs run on the IXP2XXX network processors differs from traditional processors. Popular tools suites give you a compiler, an assembler, and a debugger, each of which makes no assumptions about the application. However, with network processors, the objective is clear. The application is working on packets or cells, forwarding data in a network environment. With this in mind, tools can be made more application-friendly and debugging can be more packet-centric.

Parallel Processing

High-performance packet-forwarding applications must be partitioned in a way that provides maximum throughput from packet receive to packet transmit. You label the top-level partitions and list the sequence " receive to transmit " a something like this:

1. Packet Segment Receive
2. Reassembly
3. Lookup
4. Classification
5. Policing
6. Statistics Update
7. Queue for Transmit
8. Schedule Transmit
9. Packet Transmit

Tools: You need a tool to draw and connect these partitions, then attach key metrics to them, such as what tasks each performs, throughput, and memory bandwidths used, to capture the design at the top level. The tool must be flexible so that you can easily move the tasks from one partition to another as you balance the performance when assigning tasks to processing units. This tool is the IXP2XXX Product Line Architecture Tool.

Some tasks are simple, and can be performed in just a few cycles. For example, in IXP2XXX network processors, most of the work of receiving packet data from the network is done by the media switch fabric (MSF), so all Packet Segment Receive has to do is notify the MSF that it can take more data, get the status - that is, which receive buffer, how many bytes, and so on, when the data arrives, then verify the status. The next step, reassembly, involves much more work. Reassembly must get a buffer location in DRAM and copy the packet segment from RBUF to DRAM. Fortunately, the IXP2XXX network processor architecture has hardware supported linked lists, so getting buffer pointer is a single instruction. Also, since copying data from RBUF to DRAM is a single instruction, reassembly isn't too difficult. Packet data can come in on more than one MAC device port interface, so you must store the buffer location and other reassembly states while a series of data segments are being received for a

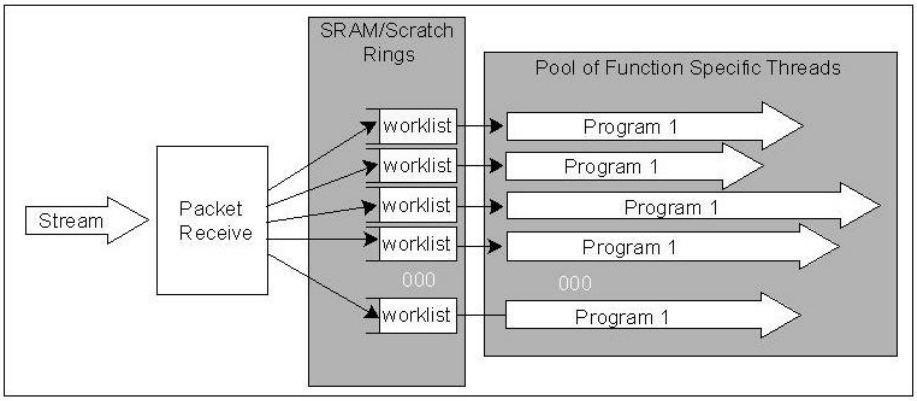


Figura D.19: Partitioning of Packet Receive and Packet Processing

packet. The buffer location must be updated by adding the size of each segment as it is received. There arent many ports, and this is Ethernet, which means successive segments of a packet come back to back on port. Microengine local memory can be used to hold this information. Packet reassembly state can be stored in a local memory array and indexed by port. If the receive port is a cell-based protocol, such as CSIX or ATM the reassembly state can be kept in SRAM, indexed by a protocol-specific connection identifier. Now the receive / reassembly partition receives, looks up, and updates port information and reassembles the packet in DRAM [9].

Tools: You need a tool that reports the number of packets received; whether there was an error, such as a missing start of packet indicator, with which you can capture the reassembled packet data and look at it later; and where you can watch the port information as it is updated in local memory.

You calculate that the packet segments arrive every 70 cycles. Ypu also know that the next stages need to get the packet header, hold onto it, and perform many operations. This segment is a good place for a partition. The first stage reassembles packets and dispatches them to the next arrival rate, and many parallel MEs can run the header processing stages. Fig D.19. Shows this arrangement.

Tools: You need a tool that can show the partitioning of code to MEs and execution of many threads tunning in parallel. You need to know when threads are idle and, when busy, what packets they are working on.

When packet processing is completed and the packet is modified, the packet is then queued for transmission, of forwarding. Scheduling, queuing, dequeuing, and transmitting are simpler operations that do not require a lot of packet data. Therefore, the pipe

can narrow again for these stages, witch each running at the transmit rate; for example, one enqueue/dequeue per 70 cycles, on schedule per 70 cycles, and one transmit per 70 cycles.

Tools: You need a tool that can watch queue status and show whether queues are backed up and causing stalls. You need to see packets being segmented and transmitted, and transmit rate. You need to report packets that are modified incorrectly, dropped, or transmitted to the wrong output interface.

Asynchronous Events

Packet data being worked on in a network device is always transient, moving through at a high rate. In fact, throughput for given functionality is the primary consideration. Traditional processor use a cache to accelerate their programs. In a cache, the likelihood of reusing data is very high for most single-user programs. A cache close to the execution data path of a processor usually speeds up the time it takes to access data. However, if the data is always changing, the cache could spend all of its time evicting old data and reading in new data from off-chip memory. Such is the case with network processor.

Except for the Intel Xscale core, the network processor does not use a cache, causing the microengines to be greatly reduced in size. Extra space is given to general-propose registers dedicated to each thread. A thread runs for a while, issues a memory I/O request, and swaps out. That thread is available to run again when the memory operation completes and sends a signal back to the microengine. Also, as it swaps out, another thread can immediately swap in and run, because its general purpose registers are live and dedicated. Fig D.20 shows eight threads executing on the same microengine.

Ideally, the eight threads of a microengine would tun at maximum efficiency by swapping out often enough to allow others to run an not so often as to be stalled waiting for I/O operations to complete.

Tools: You need a tool that can show threads as lines in time, executing and swapping out, issuing memory requests, and receiving event signals. The tool should mark places on the thread execution line that are significant code milestone, such as starting a packet processing stage. During the simulation, you should be able to scroll back in simulation history to see execution during any cycle.

Packets

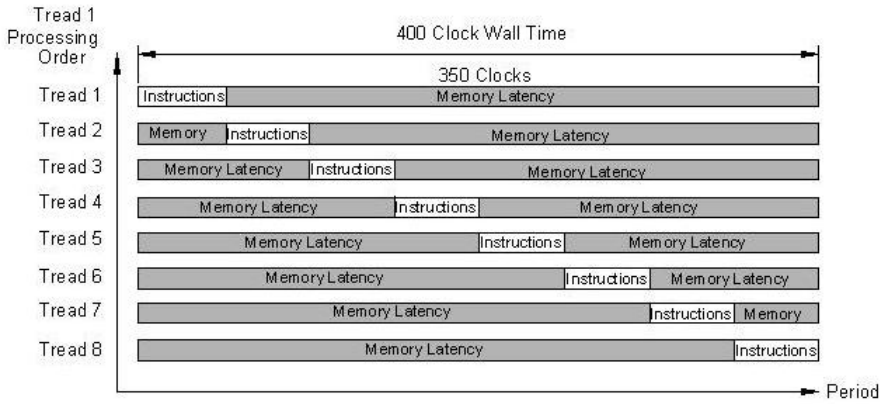


Figura D.20: Hardware-Supported Multithreading

Following parallel execution of threads and their memory references is not enough. The network processor application is all about packets. So why not show them? In may places, a packet can be checked as it goes through the processing stages. You need to observe packet data, packet derived information, and network connection state.

You need a tool that traces packets and displays all events related to selected packets, including code partitions as they execute packet-related data. From a high-level diagram of the packets actual execution, you should be able to zoom to code from any point in the diagram and trace quickly back to the cause of failure. Figure D.21 shows the concept of a scrolling history view that presents the events of selected packets.

Development:

Here you will see a fastest way to go from creating a project to running your code. You might even want to create several test projects to try out what you are learning.

You have installed the Intel IXA SDK and started the Developer Workbench. You would like to create a simple design, called "heartbeat" It makes a microengine periodically increment a global scratch memory counter, indicating that the design is still running [9].

The major steps to create and run this project are:

- Create the project
- Create, edit and insert the source file or files
- Create and assign the list file or files
- Run simulation

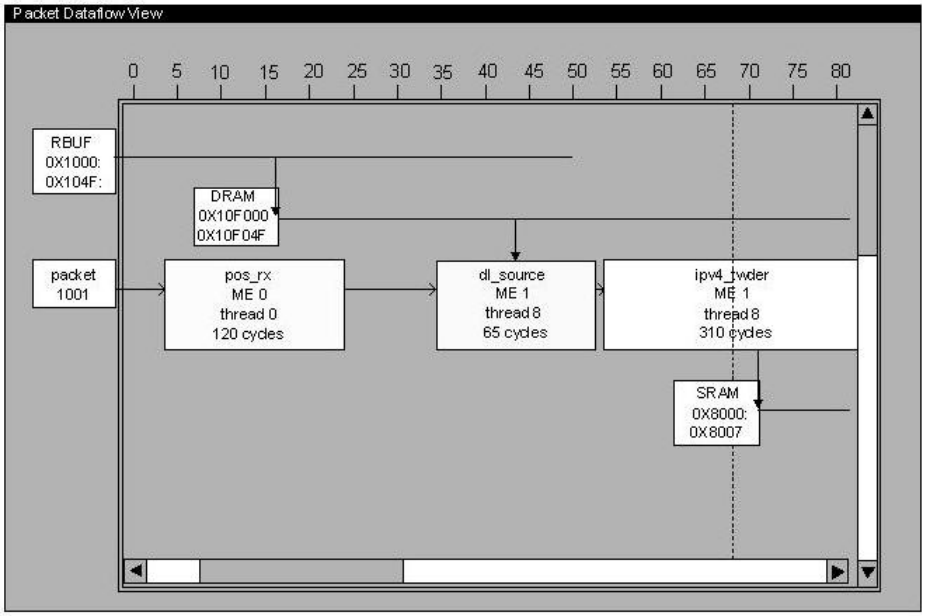


Figura D.21: Packet Dataflow View

Create the project:

From the File menu, select New Project [1]. The New Project dialog appears, as shows in the Figure D.22.

In this dialog, you must enter a project name [2] and select a chip version [3]. You may want to relocate your project to a specific parent directory. To relocate the project, modify the Location box. The actual directory name in the parent directory is the project name [4]. You can also name the chip if you are creating a multiprocessor project. Click OK and you are ready to put something in the project.

Create, Edit, and Insert the Source File:

Next, you must create a source file. This source file is a microcode file that runs a few instructions to initialize, then goes into the heartbeat loop. Microcode files have the .uc file suffix.

Here are the instructions that you must implement:

- #define used to define constants
- label#: - used for reference.

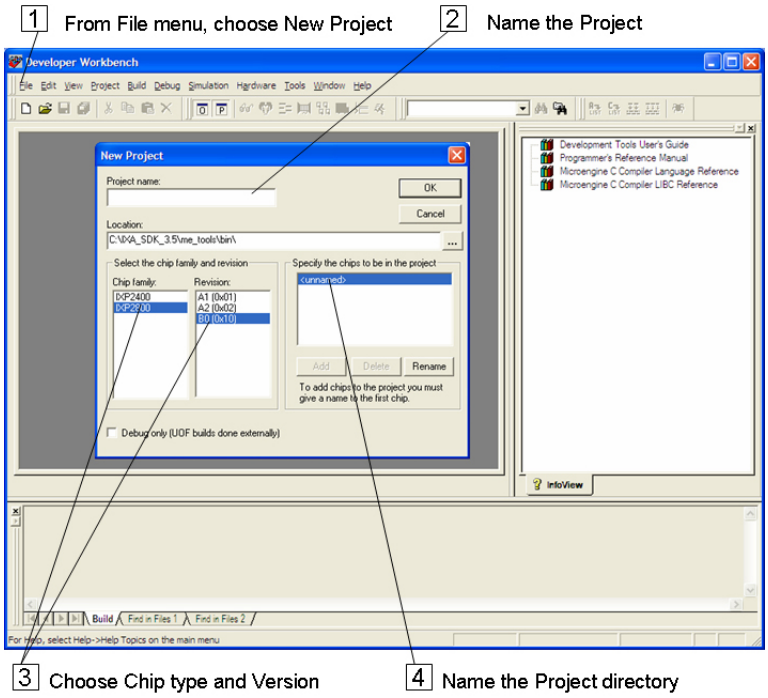


Figura D.22: Create a Project

- .reg - used to create registers.
- immed [*destreg, srcreg*] used to load values.
- scratch [*operation, optional_value, addr1, addr2*] - offer atomic operations.
- br[*label#*] - to jump a specific place

Let's give the code sequence a filename, say heartbeat.uc. To do this, choose Save As in the File menu and enter the file name in the dialog box.

Finally, the file must be added to the project. Choose Insert-> Assembler Source Files, from the Project Menu. A list of all source files in the project directory is shown; in this case, there is just one. Highlight heartbeat.uc and click Insert.

Create and Assign the List File:

The next step is to name and assign the list file. The list file is the output of the Assembler-the .uc goes in, and the .list comes out. In the Intel IXP2800 Network Processor, there are 16 microengines. Therefore, you can have up to 16 list files. Each list file has an associated root file, which is the main microcode that includes all others to be loaded from a given microengine.

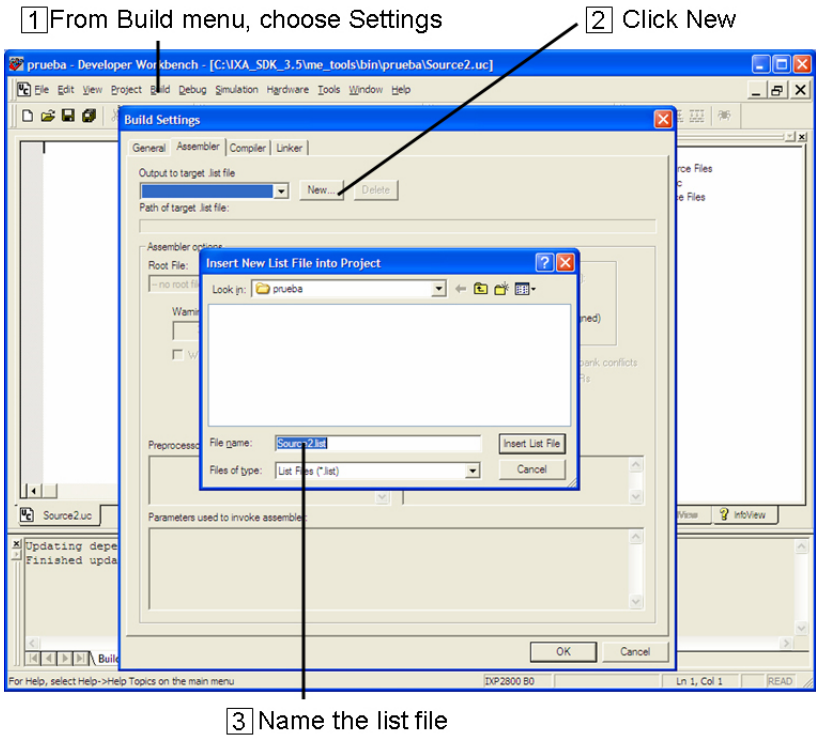


Figura D.23: Create List File

As shown in Figure D.23, choose Settings in the Building menu. Then select the Assembler tab of the Build Settings dialog. There are a number of optimization choices, but ignore those for now. Besides Output to Target List File, click New 2 to bring up the Insert New List File dialog box. In the file name box, type h. A list of files in your local directory is displayed. In fact, the filename heartbeat.uc appears in the File name box. Merely change the suffix to .list 3 and click OK.

The list file that you now have needs to be associated with two things: a root microcode file and microengine. As shown in Figure D.24 click the down arrow under Root File and choose heartbeat.uc

The Assembler settings are now completed. Now you can configure the Linker. While still in the Build Settings dialog, select the Linker tab, show in Figure D.25. The Linker takes as input the .list file, and produces a .uof file. The .uof file is the binary format that is loaded into the microengines.

For learning microcode or microengine C notation of for initial experimentation with various pieces of the application, you usually run just one microengine. In this test

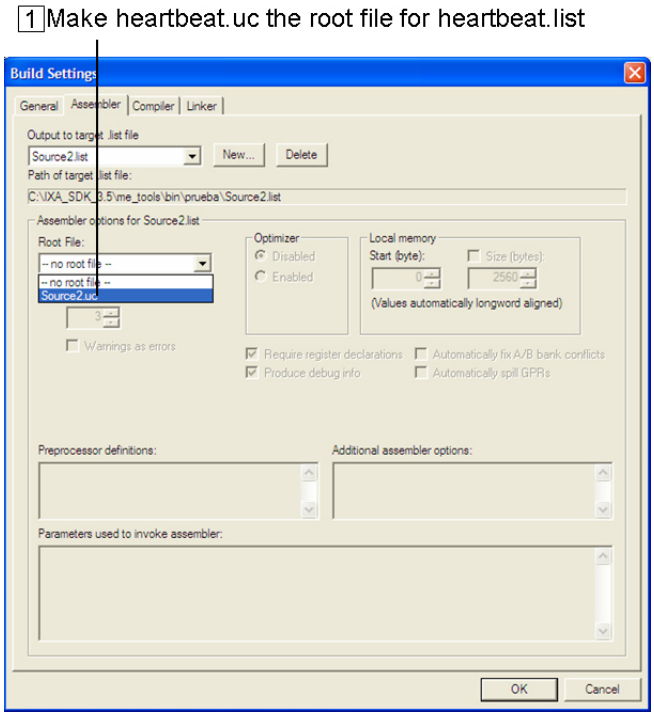


Figura D.24: Assign List File to Root File

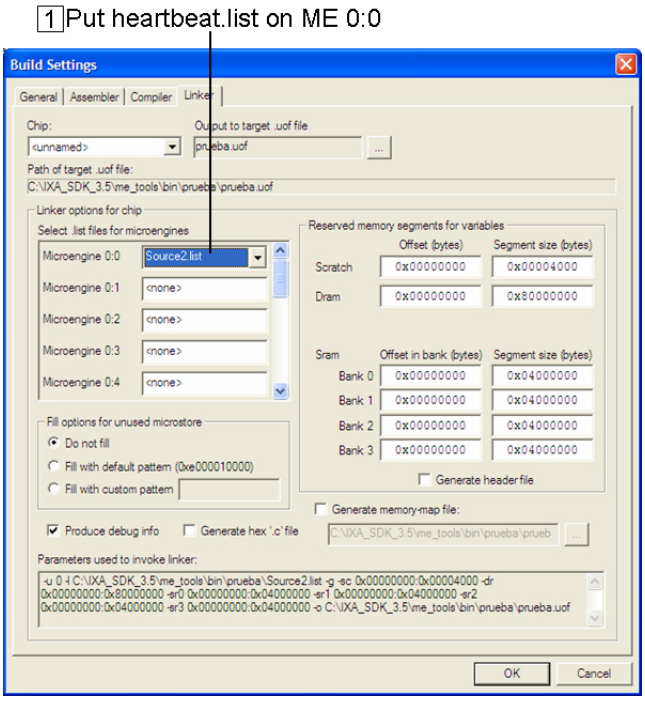


Figura D.25: Assign List File to Microengine

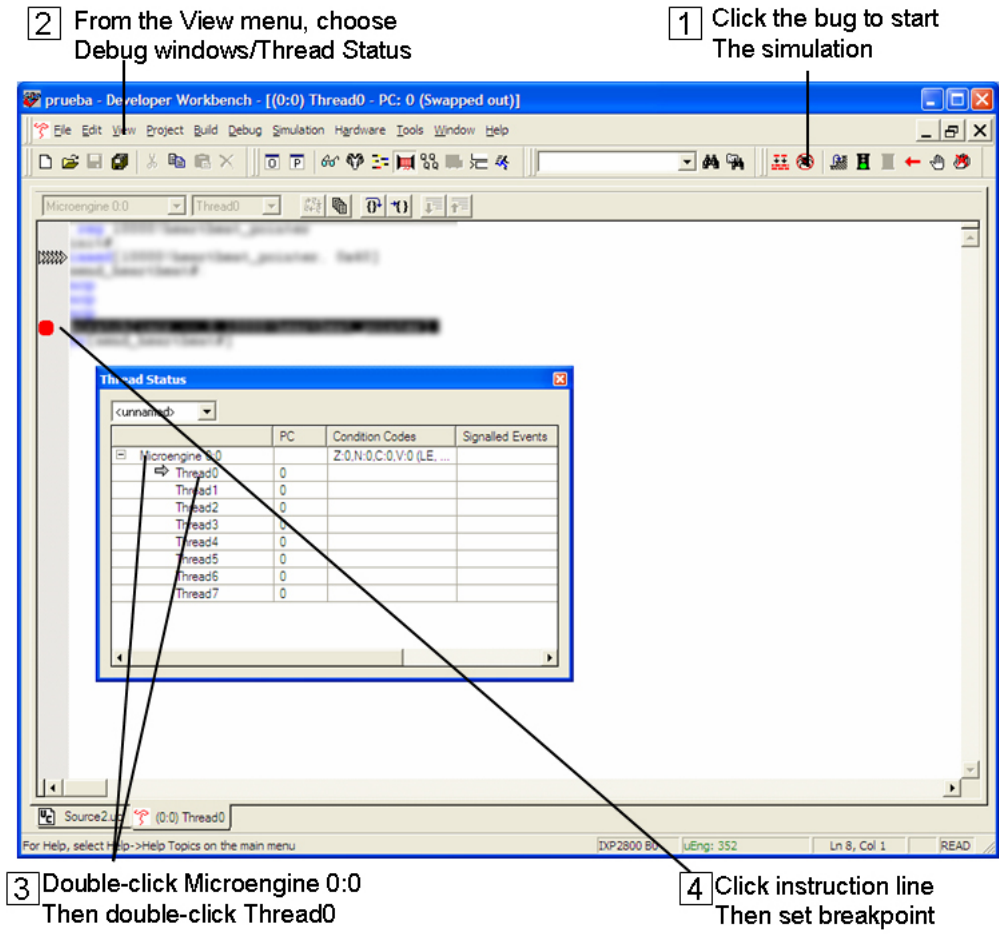


Figura D.26: Run Simulation

program, we don't want to many heartbeats!. Clock the down arrow at the box nest to Microengine 0:0, and select heartbeat.list [1]. Click OK, and you have completed your design.

Run Simulation:

You're probably thinking, How can I run simulation when I haven't even assembled it When you try to start up simulation, the Workbench detects whether any of the source files are newer than their related list files and forces a build for those microengines that need it. Figure D.26 shows the simulation view that is displayed when you click the Simulation Start "the bug"button [1]. Notice that Build menu is no longer present.

The simulation runs a few hundred cycles, performing various initializations in the processor's functional units. Following this, from the View menu, choose Debug Windows. A toggle menu appears allowing you to display or hide many different views of the simulation in progress. For example, selecting Thread status brings up a dialog box showing the status of all active threads [2]. This view initially presents the list of microengines that have list files; that is, code to run. Double click Microengine 0:0; the following appears for each thread [3]:

- Program Counter (PC) is the current instruction being executed
- Condition codes are flags for the las time an instruction set condition code:= zero, negative, carryout, overflow.
- Signaled event are events the thread is waiting on.
- Wakeup events identify those events that have completed.

The thread debug list view is opened by double-clicking on any given thread status line [3]. This view shows the instructions that have been loaded in the microengine, with PC to the left of each one. In the heartbeat case, six instructions appear in blue. In fact, all eight threads of microengine 0 are at PC 0, ready to execute these instructions.

A real program has many more instructions. The first thing to do is run the program to some significant milestone in the code and stop there. In this example, the program stops when the thread issues the increment to the heartbeat counter and checks to see if the address is correct.

You can follow this procedure 4 by first setting a breakpoint:

- Right-click on the scratch instruction, and select Insert/Remove Breakpoint. A red dot appears to the left of the instruction, indicating an unconditional " always stop "breakpoint.
- Second, click the Simulation Stop/Go button. Note that the button is green when the simulation is stopped. The simulation runs until the breakpoint occurs. Move your cursor over the operand 1000!_variable_name. The value of the register displays. Check to make sure is correct.
- Now lets remove the Breakpoint and tun for a few seconds. You can remove all breakpoints by clicking the No Breakpoint button.

As show in Figure D.27, the History screen displays the activity of threads and their memory references. In the heartbeat example, only one thread is running. Do you

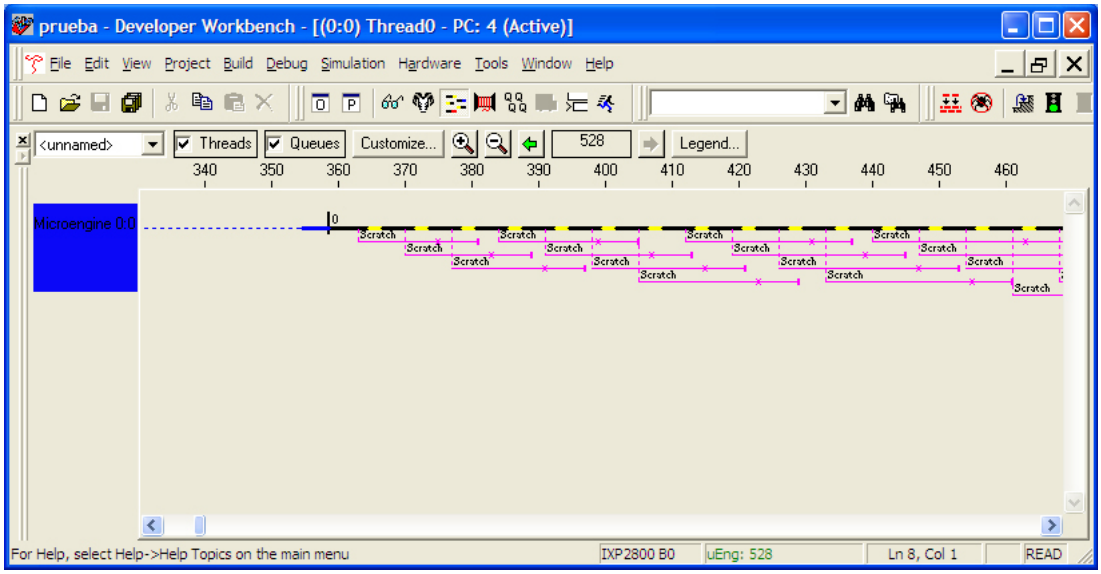


Figura D.27: Run Simulation

know why? The reason is that the thread never swapped out.

You can simulate much more. For example, you can open the Memory Watch View an select Set Break On Change for the scratch address 0x40. Then run simulation. When the scratch location changes in value, simulation stops and then displays the value. Notice that the value of location 0x40 is 0x5a5a5a... This result occurs because that location wasn't initialized at the start of the program. But at least that location's value is incrementing. To get an actual count of hearbeats, you must deposit 0 into that location before the program begins incrementing the value.

Practice:

Objective:
Develop the analytic comprehension of microengine assembler through a simple "Hello_World_uc" code.

Development:

Here you have to compile the "Hello_World_uc" that your tutor gives you and show the complete simulation of it.

Running the microengine assembly code in the simulator is not too different from running the Heartbeat code in the simulator. The principal difference is the assembler

codes that are imported from:
C:\IXA_SDK_3.5\src\library\dataplane_library\microcode like in the figure D.28 include it.

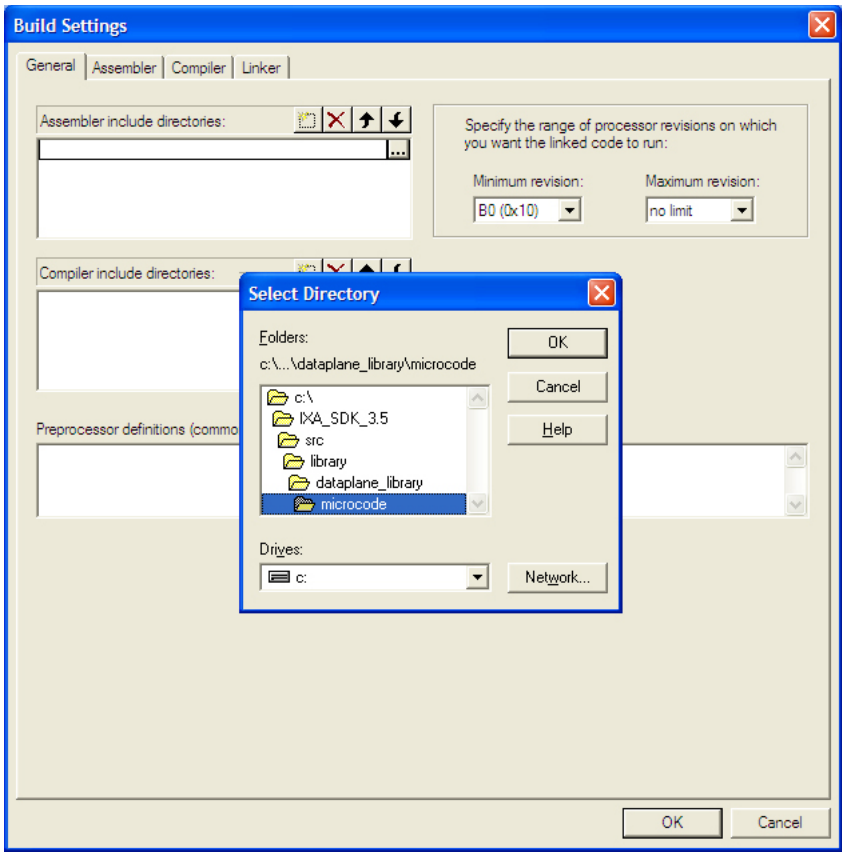


Figura D.28: Assembler include directories

Following the steps from the "Hearbeat Code" section above, you should be able to open the window for thread 0:0, and it should look something like figure D.29. This view shows small green arrows where the microengine assembly macros are. Right clicking a line with a macro brings up a menu with two options that help you see the contents of the macros. The two options are "Expand Macro One Level" and "Expand Macro Fully". The first option expands the macro in the window and leaves any sub-mac-first option expands the macro in the window and leaves any sub-macros unexpanded. The second option expands the macro and all of its sub-macros, recursively. After expanding a macro, right clicking the line with the macro again brings up a menu with the option Collapse Macro. This option collapses the macro in the code viewer. The two buttons show in the figure D.29 can also be used to expand and collapse macros. Clicking the button on the left expands all of the macros in the current view one level.

To see the microengine assembly code in action, expand the reverse_array_simple macro one level, and insert a breakpoint at the end of the first while. Now, run the simulator until the breakpoint is reached. Notice that when you hold your mouse over the old array and new_array pointers, the simulator does not give you information about them. These values are constants, so the simulator does not tell you anything about them. Instead, hold your mouse over the current_old_entry and current_new_entry values and record their contents. The names of these variables get prefixed with a string that might make these register names look weird at first. Don't worry, these prefix identifies the scope in which the registers are declared, keeping the register names unique throughout the code.

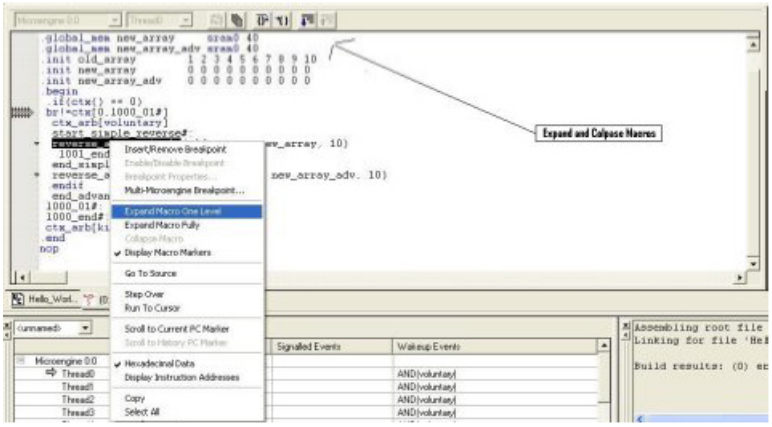


Figura D.29: Simulator Code View for Microengine Assembly Code

At the point where you put the breakpoint, current_old_entry was initialized with the value of old_array and current_new_entry was initialized with the value of new_array. So, you have found the pointers to the original and new arrays. Now you can add memory watches for the arrays in the same way you did for in "Heartbeat Code" and watch the arrays of the code reverse in memory. The SRAM completion signal is sent to the microengine a few cycles before the write is actually performed by the SRAM unit, so don't be surprised if the data watch takes a few cycles to update with the new values.

At the end of the session, all the comments will be revised and the practice will be checked by running the simulation.

Ing. Octaviano Flores Gzz. (Enero-Mayo, 07)

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS MONTERREY

Practice #3: Receive, Process and Transmit

Pre-Practice:

Objective

Learn how the C compiler works simulating a simple program of "Hello world".

Development:

You already have developed a simple "Hello Program" on microengine assembler code and have simulated it. Right now you will compile the same code on C. Running the microengine C code in the simulator is not too diferent from running the Hello_World_uc. The principal diference is the C codes that are imported from: C:\IXA_SDK_3.5\MicroengineC\Include.

You must follow the next steps:

1. Open the Hello_World_C code that your tutor gave you.
2. On the Debug menu, click Simulation.
3. Click the tool bar icon that looks like a bug.
4. On the View menu, click Debug Window.
5. Open the Data Watch, Memory Watch and Thread Status windows.

Explanation:

- The Data Watch window lets you examine variables or CSRs.
- The Memory Watch window lets you examine memory locations.
- The Thread Status window shows the current status of the microengine threads, including the current program counter, condition codes, outstanding events, and events that the thread is waiting for.

Now, run your program and see how it works. Breakpoints are helpful here.

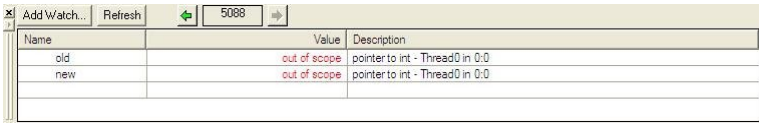


Figura D.30: Data Watch Window

1. In the thread status window, click the plus sign next to microengine 0:0.

2. Double click thread 0 of microengine 0:0. A source file opens in the same area as the current source file. You'll see an arrow pointing into that source file showing where the current program counter is.

3. Scroll down to the reverse_array function and put the cursor on the line with the for statement.

4. On the tool bar click the white-hand icon. A breakpoint is inserted on this line.

5. On the tool bar directly above the source file listing window, you'll see an icon of two files and two arrowsfor called toggle view . When you put your course over it, the tool tip should read "Toggle View". Click this button. Now the thread status window shows the disassembled microengine assembly. You can step or set breakpoints in this window as well. You don't need to do that now, however, so click Tooggle View again to put the view back to the microengine C source.

6. Click the green light icon at the top of the screen to run the program. In pretty short order, the simulator will let you know that you hit your breakpoint. So far, so good!

7. Put your cursor over the old variable and right-click.

8. On the pop-up menu, click Set Data Watch for: old Do the same for the new variable.

9. These variables should show up in the list in the data watch window. The data watch window should look like Figure D.30.

To look the array contents:

1. Click in the first column of the SRAM section of the Memory Watch Window.

2. Type sram[the address of the old array obtained from the data watch window].

Now see if your code actually works. Put together breakpoint at the end of the exit function and clock the green light to continue running the simulator. Again, the simulator should let you know that it hit the breakpoint. The new array in the memory watch should have the original array reversed. Some of the memory writes may not have completed, but you should see at least some of the new array.

Congratulations! Now, fill in the blanks of all the code and explain what the instructions do. Mention which is the main difference from a normal C code.

Practice:

Objective:
Understand the basic receive-process-transmit framework through a Packet Simulation.

Theoretical Framework:

To simplify the code, all of the receive process, and transmit functions execute in a single thread in their respective microengines. This way, synchronization methods on the microengines and complicated data structures and algorithms normally associated with a meaningful processing task are not needed.

Receiving Packets

A packet processing application without packets is fairly boring. So the first step to building an interesting application is to receive packets. Once a packet is received, it can be passed to the processing tasks where you can add all of your great code.

Receiving packets on the IXP2XXX processor consists of reassembling mpackets. Each mpacket is marked, by the hardware, as a start-of-packet (SOP), end-of-packet (EOP), both, or neither. Because mpackets arrive in order, the absence or presence of SOP and EOP marks provides the microengines with enough information to reassemble mpackets back into packets. For SOP mpackets, the microengines allocate a new buffer and the SOP mpacket data is placed into the beginning of the buffer memory [10].

A new buffer is not needed for non-SOP mpackets. Rather, non-SOP mpackets are placed in the same buffer directly after the previous mpacket. Figure D.31 shows the reassembly of mpackets into complete buffers. Each SOP mpacket is placed into a new buffer, and all subsequent mpacket up to, and including, the EOP mpacket are placed directly after each other.

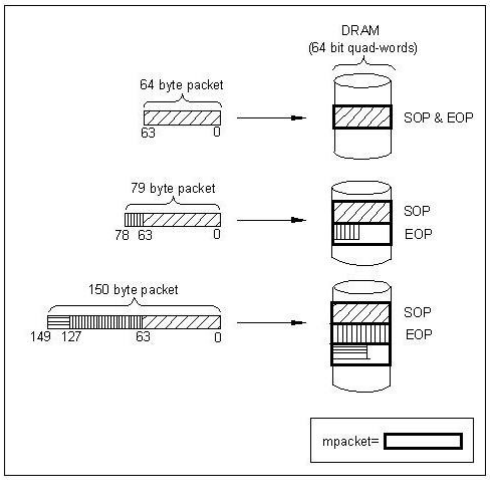


Figura D.31: Reassembling 64 byte Mpacket Contiguosly into Buffer

To put mpackets together we must allocate fixed-size buffers, each one large enough to contain the largest expected physical packet. A more complex solution, which can be found in the Intel IXASDK 3.0 receive driver, would be to allocate multiple buffers for a single packet. During the reassembly process, when the current buffer is full, a new buffer is allocated and linked to the full buffer. This multiple-buffer-per-packet approach can result in better memory utilization, but comes at the cost of extra complexity and reassembly processing requirements.

As already described, for each mpacket a different action is taken based on the SOP and EOP marks. SOP mpackets cause a new buffer to be allocated and then are copied into the new buffer, whereas no-SOP mpackets are simply copied into the existing buffer.

Receiving One Mpacket

The first steps in the reassembly flow are to receive a single mpacket:

1. Add the current thread into the receive thread freelist.
2. Wait for the MSF to signal the arrival of a new mpacket for the current thread.

To add a thread into the receive thread freelist, the RX_THREAD_FREELIST_0 MSF CSR must be written. This CSR requires the following information, formatted as show in Figure D.32: The signal to send when the mpacket has been received The transfer registers where the receive status should be written. The microengine and thread where the signal should be sent and where the transfer registers should be written.

Reassembling Mpackets into Packets

Once a single mpacket is received, you can start assembling these mpacket into whole packets. This process involves checking the mpacket for errors, examining the SOP and EOP flags in the receive status word, and finally moving the mpacket into the appropriate location in DRAM.

To check for errors and the SOP and EOP flags, you must understand receive stats words. The SPI-4 receive status word format is shown in Figure D.32. This format helps you to understand the spi4_rx() routine for reassembling mpackets into packets.

Figura D.32: Format of the RX_THREAD_FREELIST_0 and RSW

Putting the Receive Task Together

Now that we have a packet reassembled, the final step is to put each received packet on a ring for the processing task. After all, almost as boring as not receiving packets, is receiving packets and then not doing anything with them.

Processing Packets

Now things start getting more exciting. We have completed the fist step in our receive-process-transmit pipeline, so you have the code that receives packets and puts them on a ring. So, go ahead and dream about all the possible processing you want to do on these packets, We dreamed and came up with: counting packets.

The basic steps of our packet processing task are:

1. Get the next packet from the receive-to-processing packet ring.
2. Increment the packet counter in scratchpad memory.

3. Put the packet onto the processing-to-transmit ring.

Transmitting Packets

Armed with code to receive and count packets, all that is left to complete the framework is to write the transmit driver . And just think, once you understand this last task, you can expand on the processing task and build any application you want. The transmit drivers reverses the reassembly done in the receive driver by braking packets into one or more mpackets. The size of each mpacket is determined by the size of each TBUF, which can be configured as 64, 128, or 256 bytes.

Transmitting One Mpacket

From the microengines perspective, the process of sending an individual mpacket has four steps:

1. Select a TBUF element in which to write data.
2. Wait for that TBUF element to be clear of valid data.
3. Transfer the mpacket into the TBUF element from DRAM, or possibly microengine transfer registers.
4. Write and validate the control words of the TBUF element with the SOP, EOP, and port information.

Selecting a TBUF Element

The first step of selecting a TBUF element in which to write an mpacket depends, to a great extent, on the ports supported by the device. For example, the code either allocates TBUF elements to particular ports or even to microengines if more than one microengine is transmitting packets. For this chapter and this example, we'll blissfully ignore thee complications and deal with only one port and one thread. Thus, all TBUFs are for use by the single transmit thread for the single port and are allocated in round-robin order.

Waiting for the TBUF Element

The second step, waiting for the TBUF to be clear of valid data, is the most complicated step and involves a global view of the TSM hardware. So lets skip this step for now and return to it last. For now, assume by some miracle, that the microengine

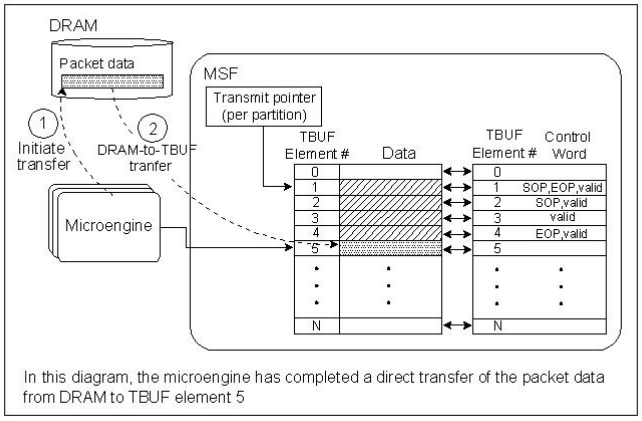


Figura D.33: Transferring Data from DRAM to a TBUF Element

code choose to work on TBUF element 5, which does no contain valid data, as shown in Figure D.33.

Moving DRAM Data to a TBUF

The third step is for the microengines to put data into the TBUF element by initiating a transfer from its SRAM transfer registers or by initiating a transfer from DRAM directly into TBUF element. Figure D.33 shows data from DRAM being transferred directly into the TBUF element from DRAM.

Writing the Transmit Control Words

The fourth step to transmitting a single mpacket is to write to the control words associated with the TBUF element. These control words contain information for the transmit state machine about the port on which to send the data, the amount of data to send, and whether or not the data is the first, middle, or last mpacket of a larger packet, as show in Figure D.34.

Writing to the transmit control words associated with a TBUF element automatically validates the element, as shown in Figure D.34.

Once the transmit control words have been written, the MSF automatically validates the TBUF element, and the microengine code is finished with that mpacket. Transmitting the mpacket is now up to the TSM hardware. It processes the TBUF elements in order and advances the transmit pointer when it completes one. When it

advances the transmit pointer to element 5, the state machine sends the data in the TBUF element to the physical interface and it also marks the TBUF element as invalid.

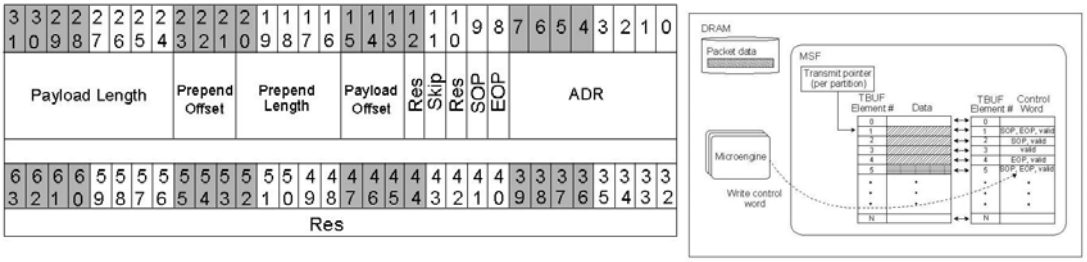


Figura D.34: The Format of TCW and Validating the TBUF Element

Waiting for the TBUF Element: Revisted

A single mpacket has now been successfully transmitted by transferring it from DRAM into a TBUF and the writing the transmit control words for that TBUF. However, a crucial step was skipped, namely ensuring the TBUF did not contain valid data before writing it. This step is necessary to ensure that the transmit microblock does not overwrite previous mpackets before the TSM has had a chance to transmit them. The solution to this problem is to read the current value of the transmit pointer and ensure that it has already passed beyond the TBUF element currently being written. A nave implementation would read the value of the transmit pointer before every TBUF element was written.

A more efficient solution would be to only read this pointer once and then fill all of the TBUF elements possible before reading the pointer again. Whit this solution, reading the transmit pointer is done much less frequently, which means lover latencies for the transmit driver.

Thus, the code which checks that the TBUF element does not contain valid data and the code that segments entire packets into mpackets must be written together. The segmentation tasks can be combined with the solution to checking that TBUF elements contain no valid data before writing them. In each iteration of this loop, the next TBUF element is selected by adding one to the current TBUF element.

The first operation in the loop checks to see whether the code should write into this next TBUF element, which is accomplished by maintaining a count of the number

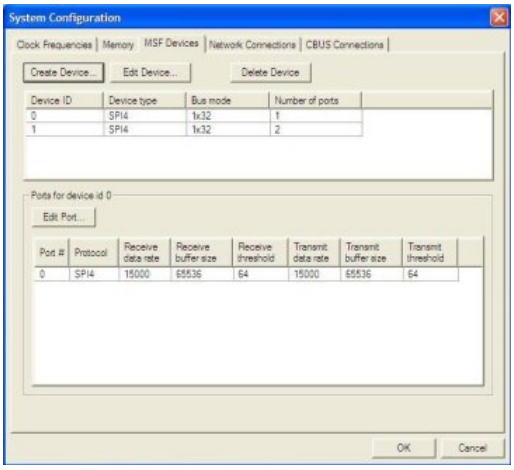


Figura D.35: The MSF Device Simulator's Devices and Network Connections Dialog

of TBUF elements in-flight in a microengine register. The number of TBUF elements in-flight corresponds to the number of TBUF elements the microengines have written data into since the last time the transmit pointer was read.

When the number of in-flight TBUF elements equals the total number of TBUF elements, the transmit microblock must assume that all of the TBUF elements are used contain valid data. While the TSM has most likely advanced the transmit pointer, the microcode cannot assume so.

Rather, when the number of in-flight TBUF elements reaches this threshold, the microengines must read the current transmit pointer and update the actual number of TBUF elements in-flight.

Once the number of in-flight TBUF elements is less than the total number of TBUF elements, the transmit algorithm can proceed to retrieve the segmentation state. The segmentation state is a set of global registers that indicates where the packet being transmitted is in the transmission process. If the code transmitted packets on multiple ports, this algorithm would need segmentation state registers for each port.

The next mpacket, obtained from the segmentation state, is placed into the next TBUF element and the TBUF elements control words are written. When the current number of TBUF elements in-flight is one less than the maximum number of TBUFs, an optimization is made. Specifically, during the latency associated with transferring the mpacket into the current transmit pointer. This optimization enables the code to hide the latency associated with reading the transmit pointer in nearly all situations.

Development:

Simulating the code would be fairly boring. If the simulator did no generate packets for the receive driver to reassemble. So we are going to configure the simulator to inject packets on a single SPI-4 port, we should explain how to do this configuration so you can do the same yourself.

Four steps are needed to configure packet input and output in the simulator:

1. Enable packet generation and transmission in the simulator.
2. Add one or more MSD devices of packets.
3. Create one or more streams of packets.
4. Assign the packet streams to the MSF devices.

Enabling Packets in the Simulator

Before doing anything with the MSF devices, packet streams, or the like, first check the Enable Packet Simulation in the Simulation menu of the Developer Workbench. This option instructs the Workbench to initialize and include the appropriate packet generation libraries. Without enabling this, the rest of your hard work to create packets and media devices will be for not.

Adding an MSF Device and Network Connections

Figure D.35 shows the MSF Devices and Ports dialog (On the Simulation menu, click Simulation - System Configuration), which is used to accomplish the second step. Two SPI-4 devices have been added, one for receiving packets and one for transmitting packets.

Each device has a single port, which can be defined when the device is created. The rate, buffering capabilities, and receive and transmit thresholds can be configured for each port.

The receive threshold indicates the amount of data the device buffers before indicating to the IXP2XXX MSF that an mpacket is available. For optimal performance, this threshold should be set to the size of the RBUF elements. The transmit threshold indicates the amount of data the device buffers before beginning to transmit the packet. The smaller the value the better as a smaller value enables the device to be transmitting

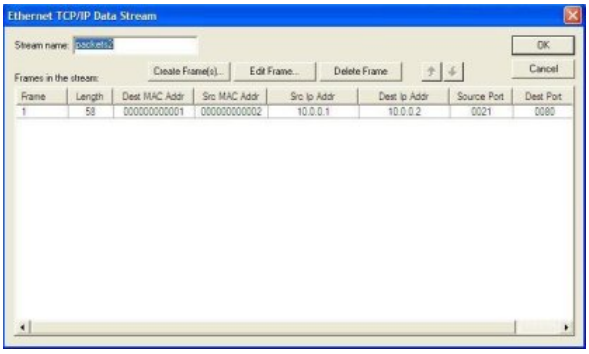


Figura D.36: The SImulator Data Stream Dialog

the beginning of the packet at the same time the transmit microblock is writing the next mpackets into TBUF elements. The only caveat is that the smaller the threshold, the faster the transmit code needs to run. If the transmit microblock does not provide the next mpacket to the device before the device has completed transmitting the data in its internal buffers, then an underflow occurs. The result of an underflow is an invalid transmitted packet. By adding this devices, the simulator includes a simulation of the devices that generate the correct MSF bus signal, including packet data transfers.

For last, change to Network Connections and assign the Media Bus RX and TX devices to their correspond ID number.

Creating Packet Stream

Simply adding devices to the simulator does not mean these devices inject any packet data. Instead, a set of streams that the devices use needs to be configured to simulate packet data arriving at the device. To add streams, on the Simulation menu, click Simulation - Data Streams. In the Data Streams dialog, one or more streams of packets can be created, as shown in Figure D.36

Each stream contains a single type of packet. The Developer Workbench contains templates for several packet types including IP packet encapsulated in Packet Over SONET (POS), IP packets encapsulated in Ethernet, and ATM AAL5 packets.

Within each stream, one or more packets can be defined by clicking the Edit Stream button. The data in each packet can be edited through a dialog box specific to the type of packet in the stream.

Assigning Packet Streams to Devices

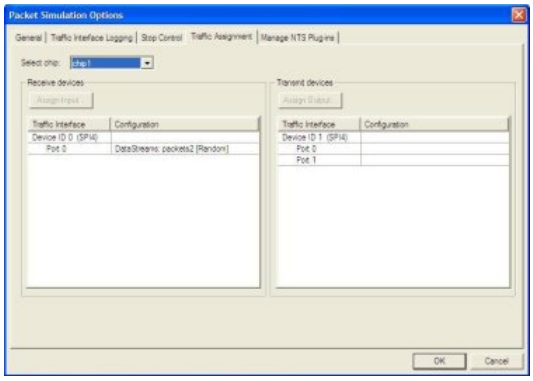


Figura D.37: The Simulator Traffic Assignment

To complete the simulators setup, the packet streams created must be assigned to particular ports. To assign data streams to ports on devices, on the Simulation menu, click Data Streams and then click Port Connections. In the Simulation-Packet Simulation Options- Traffic Assignment, data stream can be assigned to ports, as shown in Figure D.37. Notice we only assign streams to the ports on which we want to receive packets. It does not make sense to inject packets into the ports used for transmitting packets. So how do you determine what packets are transmitted? Have the simulator log packets transmitted out a port! Under the Simulation menu, click Packet Simulation Options. Under the Logging tab of the Packet Simulation Options dialog box, the Enable Logging box is checked and a file into which to log has been specified. After the simulation runs for a while, this file can be examined to see what packets were transmitted.

Congratulations! You have simulated a Receive, count and Transmit packet. Now, hand in the spi4_rx.uc, spi4_tx.uc and count.uc code and explain what the instructions do.

Ing. Octaviano Flores Gzz. (Enero-Mayo, 07)

INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS MONTERREY

Practice #4: Packet Processing in a Single Thread

Pre-Practice:

Objective

Develop a deep knowledge of Rings and Queues through the comprehension of scratchpad rings and SRAM queues on the IXP2400.

Theoretical Framework:

We will cover the hardware support for rings and queues on the IXP2XXX processor. Specifically, you'll learn about the software for initializing and accessing (enqueueing, dequeueing, checking the size, etc.) information in this data structure.

Both rings and queues implement First - In - Firs - Out (FIFO) data structure. In the IXP2XXX literature, rings are fixes-sized, circular FIFO, whereas implemented as rings or queues, are an extremely common concept in networking for several reasons:

- The rates of the tasks producing and consuming on the ring or queue may not be identical. Although when the producing task is faster than the consuming task the ring or queue eventually fills. Rings and queues do insulate the two tasks from temporally burst or stalls present in either task.
- Multiple producing tasks can be coupled with a single consuming task. For example, this coupling might be used to enable multiple processing tasks to all use a single transmit task.
- Using multiple rings or queues, a single producing task can be coupled with multiple consuming tasks. This coupling might b used in a system where some packets require different processing than others.
- represent high-priority traffic reserved for us and a second ring may represent all other, best-effort traffic.

Due to the common use of rings and queues, the IXP2XXX hardware has built-in support for these for these data structures. These data structures can be maintained in scratchpad memory (rings only), SRAM memory (rings and queues), or in next-neighbor

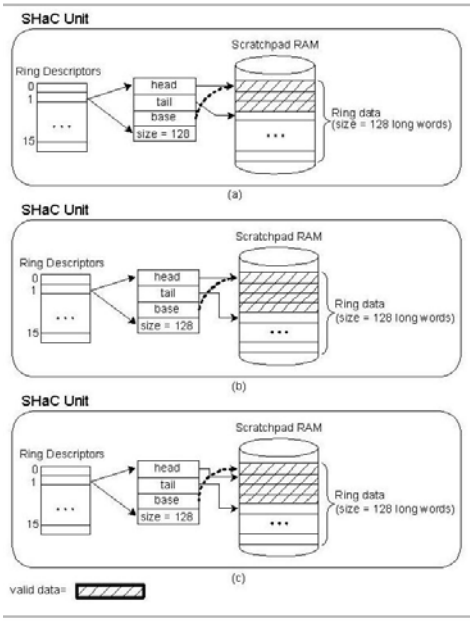


Figura D.38: The implementation of Scratchpad Rings.

registers. The microengines contain instructions, signals, and CSRs for modifying and monitoring these FIFOs [10].

Scratchpad Rings

The IXP2XXX hardware supports 16 Scratchpad rings. As shown in Figure # 1, each ring is implemented as an array (of configurable size) in scratchpad memory, with pointers to the first and last entries on the ring, called the head and tail respectively.

The SHaC unit maintains the head and tail pointers, the base address, and the size (in long-words) of the ring. The head and tail pointers are modified during put and get commands on the ring, whereas the base pointer and size do not change once the ring is created. For example, as shown in Figure D.38 (b), after an entry is put onto the ring, the tail pointer is advanced. Similarly, Figure D.38 (c) shows the head pointer being advanced as the result of a get operation. Using the head and tail pointers, the hardware also implicitly maintains the count of entries currently on the ring. Thus, form a programmer's perspective, the hardware contains enough information to determine the fullness of the ring.

Both the head and tail pointers wrap around the ring so as not to exceed the size of the ring. Each ring can be configured into one of the four sizes: 128, 256, 512, or, 1024 long-words. Since this rings are stored in scratchpad memory, you cannot create 16 rings of 1,024 long-words because that would require 64KB of scratchpad memory.

Scratchpad Ring Size	Threshold When Full is Asserted
128	96
256	192
512	384
1024	768

Cuadro D.1: Fullness Thresholds for Each Scratchpad Ring Size.

Creating a Scratchpad Ring

To create a scratchpad ring, the software must specify the size of the ring, the starting scratchpad address where the ring data should reside, and the ring number to use. The ring number corresponds to one of the 16rings supported by the hardware and is between 0 and 15.

Putting Data on a Scratchpad Ring

After creating a scratchpad ring, the code to put data onto a given scratchpad ring is a single instruction, scratch[put, -]. This instruction takes the ring number and one or more transfer registers and puts them onto the given ring by writing into scratchpad memory.

Getting Data from a Scratchpad Ring

In a similar fashion as putting data on a scratchpad ring, the code to get data from a given scratchpad ring is a single instruction, scratch[get, -]. This instruction takes the ring number and one or more transfer registers and receives the given ring by reading from the scratchpad memory into the transfer registers.

Checking for Scratchpad Ring Fullness

Because rings are of fixed size, before putting data on a ring, the fullness of the ring must be checked. What did mean by fullness? Well, for each of the first 12 scratchpad rings, the hardware provides a fullness bit that is not an exact indication of the ring being full. Rather, this bit indicates that the ring has reached the threshold of the three quarters of the total ring capacity. Figure D.1 shows the thresholds for each of the four scratchpad ring sizes.

The fullness bit is set at these thresholds because multiple microengine threads may simultaneously put data on the same ring. Imagine if the hardware waited to set the fullness bit exactly when a ring was full. Now imagine that particular scratch ring has room for just more entry. The ring is not full, so the fullness bit would not be set. Multiple threads could then check that the ring was not full and issue a putt command.

The result would be that the hardware would receive multiple put commands for a ring that only had room for one more entry. The hardware would be forced to discard some of the requests.

Several conclusions can be drawn:

- When the fullness bit is set, the ring is not necessary full. In our example, we have chosen to interpret the fullness bit as the ring being full, but you don't need to make this same assumption. You can use knowledge of the application to determine how many threads are putting data on the ring and then ignore the fullness indicator for some number of puts before checking again.
- The fullness bit is not sufficient to prevent rings from overflowing. Consider the case where 128 threads are simultaneously putting two long-words of data on a scratchpad ring of size 128 long-words. Even if the ring is initially empty, the fullness bit does not prevent the ring from receiving too much data.

So is there a way to determine exact fullness of a ring? Yes, and luckily a simple solution exists. However, simple does not mean efficient. A counter can be maintained in either scratchpad or SRAM memory that corresponds to the number of long-words currently on the ring.

Any thread that performs a put operation would first automatically test and add to this counter. If the counter was originally less than the size of the ring, the thread could issue the scratch[put, -] instruction. Otherwise, the ring if full and so the thread must automatically subtract from the counter to undo the addition it originally performed.

After a thread performs the scratch[get, -] instruction, if the ring is not empty, the counter is automatically decremented. The cost of this solution is at least one extra memory operation per put (and get) operation. For the case where the ring is full, the put operation must issue two extra memory references.

SRAM Queue Array

For applications requiring a few small FIFOs, scratchpad rings are sufficient. However, scratchpad rings are not sufficient for applications requiring more than 16 FIFOs. For example, quality of service (QoS) applications may need hundreds, perhaps thousands, of FIFOs per port. Furthermore, scratchpad rings are not sufficient for applications requiring very large FIFOs, such as those used for freelists of buffers, which could require many thousands of entries.

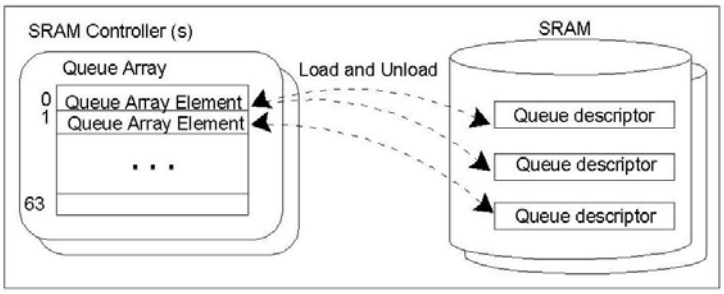


Figura D.39: Using the SRAM Queue Array as Cache

For applications where scratchpad rings are not sufficient, the IXP2XXX processor's solution is to use SRAM-base FIFOs. The IXP2XXX hardware can support as many FIFOs as can fit within SRAM memory and provides access to these FIFOs through a 64-element cache (per SRAM controller) as shown in Figure D.39. Before a new FIFO can be used, its "descriptor" must be loaded into the "cache" (queue array). A queue descriptor contains all of the necessary data to work with the FIFO, such as the head and tail pointers and current number of entries in the FIFO.

When the cache if full and a new FIFO need to be used, one entry from the cache must be unloaded. Thus, the total number of FIFOs supported is not limited by the size of the SRAM queue arrays, but instead by the amount of SRAM dedicated to the FIFOs.

Each queue-array element contains enough information to add or remove an entry from a single SRAM FIFO. For example, each queue array element contains a head pointer; a tail pointer, and a count of the number of entries currently in the FIFO.

If a new FIFO needs to be loaded and no used queue-array element exists, you must unload an existing queue-array element first. The unloading process writes the queue-array element into SRAM. To load a queue descriptor, you specify SRAM controller (called a channel), queue-array element number, and SRAM memory from which the descriptor should be loaded. To unload a queue descriptor, you specify the SRAM controller and queue-array element number. The queue descriptor is written back into the same SRAM location from where it was loaded.

The SRAM controller implements two different types if FIFOs: a linked-list queue, and a circular ring. The usage model for these FIFOs and the format of the queue descriptors in the queue array are different depending on which type of FIFO is being used.

SRAM Rings

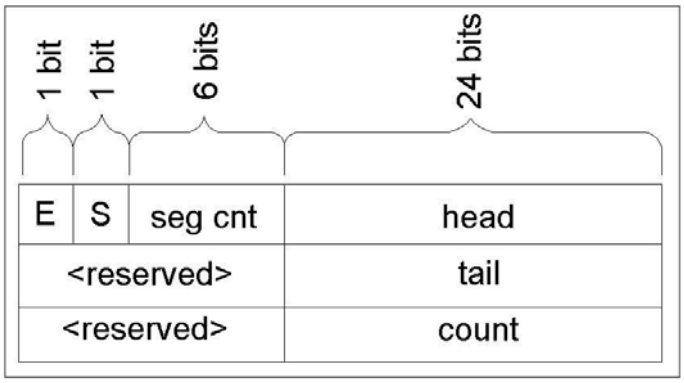


Figura D.40: The Format of a Queue-array Element for an SRAM Ring

A queue-array element can be used to access a queue or a ring, depending on your design. When used as a ring, a SRAM ring is very similar to a scratchpad ring. Of course, instead of the data being stored in scratchpad memory, the data is stored in a contiguous block of SRAM. Also, SRAM rings can be configured into sizes of 512, 1k, 2k, 4k, 8k, 16k, 32k and 64k long-words. Thus one reason for using an SRAM ring instead of a scratchpad ring might be to support a larger FIFO.

When used to implement a ring, a queue-array element contains the head and tail pointers, the size of the ring, and the current number of elements (count) on the ring, as shown in Figure D.40.

SRAM Queues

In addition to being used as a ring, an SRAM queue-array element can also implement a linked-list FIFO, called a queue. Like SRAM rings, the data for the queue is stored in SRAM. However, the size of this linked-list is only constrained by the size of SRAM memory. Unlike a ring, where the data is completely opaque and simple contained within one or more long-words of data, the entries on a queue have a specific format. This format tells the SRAM controller how to find the next entry on the queue (the next link in the list). The format for an entry on an SRAM queue is shown in the Figure D.41.

Each SRAM queue entry contains SOP, EOP, and segment count fields and a pointer to the next entry on the queue. When the SRAM controller dequeues a buffer from a queue-array element, the next queue entry is read from SRAM. The SOP, EOP and segment count field is used to locate any subsequent data on the queue.

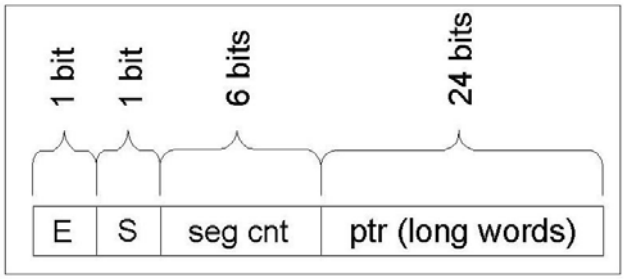


Figura D.41: The Format of an Entry on an SRAM Queue

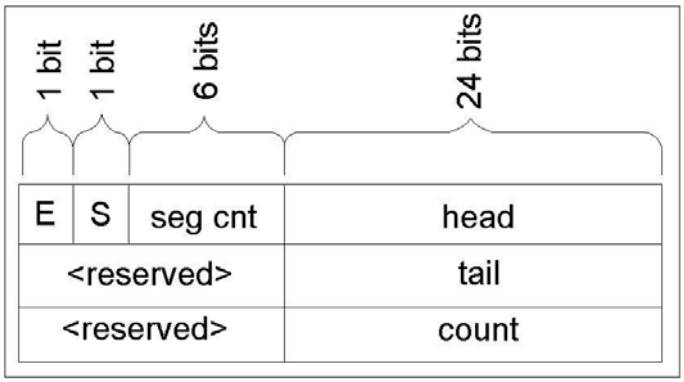


Figura D.42: The Contents of a Queue-array Element for an SRAM Queue.

To properly understand the SOP, EOP segment count and pointer fields of a queue entry, you must know what information is contained in each queue-array element. As shown in Figure D.42, each queue-array element representing a queue contains a head and tail pointers, a count of the number of entries on the queue, start-of-packet (S), end-of-packets (E), and segment count fields. The head and tails pointers, which are long-word addresses, point to the first and last entries in the queue, respectively. The count contains the number of entries currently on the queue. When a dequeue occurs on an SRAM queue, the SOP, EOP, and segment count fields of the next entry (as pointed to by the head pointer of the queue-array-element) are copied into the queue array element, and the value of the pointer is copied into the header pointer in the queue-array element.

The SOP, EOP and segment count fields determine exactly ho enqueue and dequeue operations are performed by the SRAM controller, as shown in Figure D.43. Specifically, you can configure the SRAM controller into three modes:

Mode 0: Use EOP and use segment count.

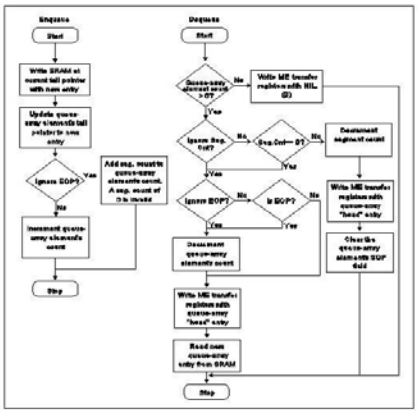


Figura D.43: Flowcharts for the Enqueue and Dequeue Operations in the SRAM Controller

In this mode, the enqueue operation links the new entry to the end of the linked list, updates the tail pointer in the queue-array element, and increments the count in the queue-array element. The dequeue operation decrements the segment count value in the queue-array element. If the value is non-zero the head pointer in the queue-array element is not removed, and the queue count is not modified. The queue count is decremented only when the EOP flag is set.

This mode is used to support both multi-buffer-per packet operations as well as segmented buffers. For example, if a single packet was larger than a single buffer, multiple buffers must be enqueued for a single packet with the SOP and EOP fields used to indicate the beginning and end of the packet.

Within each buffer, multiple segments may exist. This concept is used during the transmit operations to determine how many mpackets are contained within a buffer. For example, the transmit process continues to dequeue a buffer for each mpacket's worth of data in the buffer.

Mode 1: Use EOP but ignore segment count.

In this module, the enqueue operation links the new entry into the end of the linked list, updates the tail pointer in the queue-array element, and increments the count in the queue-array element. The dequeue operation ignores the segment count in the queue array element and always removes the entry from the head of the queue. The count in the queue-array element is decremented only when an entry with the EOP flag set is removed from the queue. You can use the segment count for any purpose. For example, some designs use the segment count as an indication of the relative size of the buffer so that the code dequeueing the buffer immediately knows the approximate size

of the dequeue buffer.

This mode is used to support multi-buffer-per-packet operations. Thus, as described above, this mode is useful for situations where packets are larger than a single buffer.

Mode3: Ignore EOP and ignore segment count.

In this mode, the enqueue operation links the new entry into the end of the linked list, updates the tail pointer in the queue-array element, and adds the segment count of the entry to the count in the queue-array element. Yes, the terminology is confusing in that the segment count is used even though the mode is ignoring 'segment count'. In this mode the segment count must never be zero. Therefore, as you will learn, the default enqueue operation must never be used in this mode since it specifies a segment count of zero. Instead, the enqueue operation should always specify the segment count using and indirect reference.

The dequeue operation ignores the segment count value in the queue-array element and always removes the entry from the head of the queue. The count in the queue-array element is decremented for every dequeue operation performed.

This mode is used to support single buffer per-packet operations, such as buffer freelists.

Note: Mode 2, or "Ignore EOP but use segment coun", is not a valid configuration of the SRAM controller.

Development:

Your instructor will provide: "queue.uc" and "sram_rings.uc" codes. You must hand in the code explanation and Flowchart of each one.

Practice:

Objective

Develop a deep knowledge on the processing of a packet. This package processing includes the comprehension of: Validation of Ethernet Packet, Strip Ethernet Header, IPv4- Five tuple Classify, Add Ethernet Header and RED.

Theoretical Framework:

Instead of just counting packets as the applications of practice #4 did, the application described in this chapter does some more complicated tasks. The application takes IP packets encapsulated in Ethernet II frames, and performs IPv4 five-tuple classification and Random Early Detection (RED) congestion avoidance on them before forwarding the packets out. The example code for this application is much simpler than what you would write in a production system, but will give you an idea of how packet processing works on the IXP2XXX processor. The example application also gives an opportunity to explain some of the cool hardware features available on the IXP2XXX processor that help you process packets, including the hash unit, CRC unit, unaligned access instructions, indeed registers, local memory, multiplication instructions, and random number generator.

Performing these functions involves writing both microblocks on the microengines and core component code on the Intel Xscale core. This chapter focuses on the microblocks because they are used in this practice.

The microblocks in this practice are written to process packets on only one thread. This simplifies the code somewhat, allowing us to focus on some of the features of the IXP2XXX processor that help us process packets. Processing packets on a single thread is, of course, no as fast as processing them on multiple threads.

The Application

The flow of packets on the application is shown in Figure D.44. The packet-processing code takes packets from the scratch ring on which the receive code places packets. The code then processes the packets. When processing is complete, the packets are put on one of the SRAM rings from which the transmit code reads. In between these steps, the Ethernet header of the packet is validated and removed, the packet is classified based on the field in the IP header; a new Ethernet header is added, and RED congestion avoidance is done. Each of the blocks in this figure is implemented as microblock. Writing microblocks implies that the code is written without knowledge of the other microblocks, which allows microblocks to be reused across different applications with different microblocks. So for example, the IPv4 five-tuple classification microblock is used to process packets that arrived as Ethernet frames, but it doesn't care about the encapsulation in which the packet arrived. The classification microblock could also be used to process packets that arrive in other encapsulations, such as Packet-Over-SONET.

The Sample Application Data Flow

The code in this practice is implemented as Intel IXA SDK 3.0 microblocks using dispatch loops. We have also implemented a core component for the microblock that

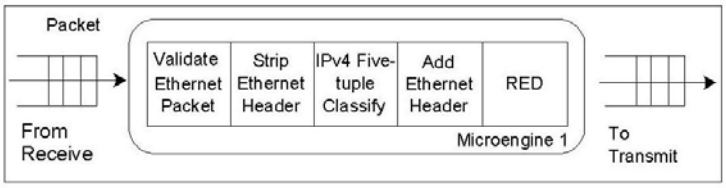


Figura D.44: The Sample Application Data Flow

adds the Ethernet header to the packet.

Ethernet Processing

Three of the microblocks of Figure D.38 are used to either process or generate Ethernet Headers. The microblocks are called ethernet_validate, ethernet_strip_header, and ethernet_add_header. The ethernet_validate microblock ensures that the packet is a valid Ethernet II frame and has different targets for multicast, broadcast, or locally-addressed and other packets that are no multicast, broadcast or locally-addressed. The ethernet_strip_header microblock removes the Ethernet header so that code that processes higher-layer packets- in this case the IPv4 five-tuple classifier- can process the packets encapsulated by the Ethernet frame. The Ethernet_add_header microblock does the opposite, adding Ethernet encapsulation around the IP frame. It determines the Ethernet destination MAC address from the next-hop ID determined by the classifier.

In our design we split up the ethernet_validate and ethernet_strip_header microblocks in an attempt to maximize reuse. We might get better performance by combining the two. But, for example, if we were to write and Ethernet bridge, we would find the combination of the two microblocks to be more work than needed, whereas ethernet_validate by itself might suffice. The code for Ethernet_validate is very straightforward. The packet is validated by checking the actual length to ensure it is grater than or equal to the minimum length and less than or equal to he maximum length of an Ethernet II frame. This microblock also tells the dispatch loop if the frame is locally addressed, multicast or broadcast. To determine the type the type of frame, it looks at the destination MAC address in the Ethernet header.

The implementation of the Ethernet_strip_header microblock is also straightforward. This microblock first gets the Ethernet protocol number from the packet by parsing the packet header. Then, it removes the Ethernet header. Each packet comes with metadata registers that specify the length of the packet and the offset within the buffer at which the packet begins. The ethernet_strip_header microblock adjusts these two metadata values to remove the header. So how does this microblock “pars the heade”? Examining headers in packets is done differently in microengine C than it is in

microengine assembly. In microengine C, the ability to use structures makes it easier. A structure can be defined for most headers. For example, the following structure is used in our code to examine the Ethernet header.

```
    typedef __declspec(packet) struct_ethernet_header
{
Unsigned int destination_addr_hi32;
Unsigned int destination_addr_lo16: 16;
Unsigned int source_addr_hi16: 16;
Unsigned int source_addr_lo32:
Short protocol;
}ethernet_header;
```

With the structure defined, it is very easy to retrieve fields from Ethernet headers. Microengine assembly does not have support for structures, so the code needed to extract fields from headers needs to be written manually. Here is the microengine assembly code used to extract the destination MAC address:

```
dram [read, $eth_header0, eth_header_start, 0, 1],
ctx_swap[dram_sig]
alu[d_mac_hi32, –, B, $eth_header0]
alu[d_mac_lo16, –, B, $eth_header1, 16]
```

To determine whether or not the packet is locally addressed, the destination MAC address of the packet is compared to the device's MAC address. Some MAC devices external to the IXP2XXX processor may do this in hardware, but for fun, let's assume the hardware we are using does not. This address is stored in SRAM and is initialized by the XScale core component. In a production system, the design may involve the ability to assign one or more MAC addresses to each interface. If this is the case, determining if a packet is locally addressed is more complicated.

The final Ethernet packet-processing microblock, ethernet_add_header, is a bit more complicated than the others. The Ethernet add_header microblock needs to determine the Ethernet source address and destination address for the outgoing packet. The source address is just the device's Ethernet address. In our sample application, this address is stored in SRAM and is the same address used earlier to determine if the packet is locally addressed. To determine the destination MAC address, the microblock uses a "next hop ID" that is converted into a destination MAC address. The IPv4 five-tuple classifier microblock sets a packet metadata value called the next-hop ID, which identifies the machine to which the current packet should be forwarded. The

ethernet_add_header microblock maps this next-hop into a destination MAC address. Because the number of next-hop IDs is small, this mapping is done with a simple array, using the next-hop ID as the array index.

The XScale core component associated with the ethernet_add_header microblock maintains the next-hop-ID-to-DMAC array in SRAM. If the ethernet_add_header microblock processes a packet and the table does not have an entry for the packet's destination IP address, the microblock could use a protocol, such as the Address Resolution Protocol (ARP), to resolve the IP address to the proper MAC address. But implementing ARP may take more code store than we can spare. Plus, the majority of packets that go through the device will likely have a destination IP address that is already in the table, so ARP rarely has to be done. These factors lead us to implement ARP on the Intel XScale core component.
With that design decision, the microblock is implemented to send any packets with destination IP addresses that are not in the table to the Intel XScale core for further processing. The microblock portion of this is shown in the code below, while the core component code for handling packets from the microengines is described later in this chapter.

IPv4 Five-tuple Classification

Our sample application performs IPv4 five-tuple exact-match classification on the IP packets extracted from the Ethernet packets. The microblock that implements this classification takes the packet and modifies two pieces of packet metadata based on the contents of the packet: the flow ID, which in our application also identifies the ring number on which to put the packet after the packet has been processed, and a next-hop ID identifying the next hop to which the packet should be forwarded. This classifier is more useful as a programming example than it would be in the real works for a couple of reasons. First, this classifier forwards packets to the next-hops based on an exact match of the five-tuple, which is not normally ho packets are forwarded. Second, this classifier ignores some aspects of IP packets that probably should not be ignored in a production environment, such as the handling of packets with IP options and ICMP packets.

The Ipv4 five-tuple consists of five fields in the packet: IP source and destination addresses, IP protocol number should not be confused with the Ethernet protocol number. The Ethernet protocol number defines the type of packet inside the Ethernet packet, while the IP protocol number defines the type of packet inside the IP packet. The next layer source and destination ports are header fields for the two protocols most commonly placed inside IP packets: TCP and UDP. Our classifier drops packets that are not one of these two protocols.

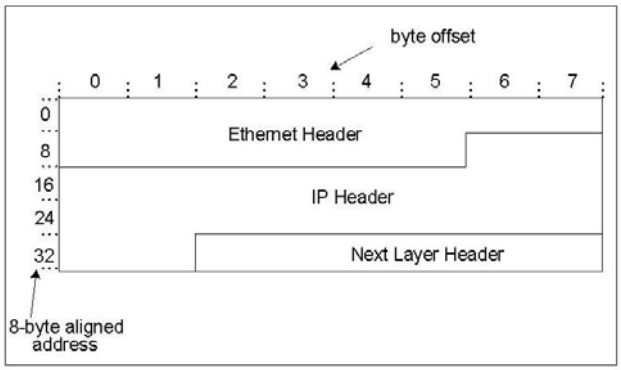


Figura D.45: Unaligned IP Header in an Ethernet Packet

To perform IPv4 five-tuple classification, the microblock needs to extract the five-tuple and search for a matching five-tuple somewhere in it's look up table. Extracting the five-tuple may appear fairly straightforward, but is complicated by the issue of alignment. For maximum reusability, the microblock should be able to handle any alignment of the IP packet within memory, but the memory in which the packet is stored, DRAM, only allows 8 bytes aligned access. Also, the design of the lookup table can greatly effect the ease and speed of the search task. So finding the right data structure for the lookup table is critical.

Unaligned Access

One of our design goals complicates the implementation of the IPv4 five-tuple classifier microblock. We said in the beginning that the microblock should work regardless of the encapsulation in which the IP packet arrived. Without knowledge of this encapsulation, the code also lacks knowledge of the byte offset at which the IP header begins. Because DRAM memory access all happen on 8-byte boundaries, this complicates the code somewhat. For example, if the IP packet arrived in an Ethernet header starts at byte 0. Figure D.45 shows this scenario.

Thankfully, microengine C and microengine assembly provide ways to perform unaligned memory accesses in a way that does not sacrifice much performance. In microengine C, the unaligned access are done using the intrinsics that start with "ua_". Intrinsic exists for signed and unsigned values of 8, 16, 32 and 64 bits. For example, the following intrinsic is used to extract unaligned, signed 32-bit data from memory.

int ua_get_s32(void* ptr, unsigned int offset)

These intrinsic take an aligned pointer and an integer byte offset as parameters. The alignment of the pointer depends on the type of memory being accessed. Remember, all of the memory types are aligned on 4-bytes boundaries except DRAM, which is aligned on 8-byte boundaries.

In microengine assembly, byte alignment instructions helps to provide unaligned access to memory. The byte alignment instructions works on big-endian or little-endian data in any type of registers. These instructions take unaligned data as input and return aligned data as output. The number of bytes of alignment shift must be set beforehand in the BYTE_INDEX microengine CSR. The following sample code takes eight bytes of big-endian data from SRAM byte address 0x2001 and aligns it in two general-purpose registers.

```
sram[read, $xfer0, zero_reg, 0x2000, 3], ctx_swap[my_sig]
local_csr_wr[BYTE_INDEX, 1]
byte_align_be[–, $xfer0]
byte_align_be[reg0, $xfer1]
byte_align_be[reg1, $xfer2]
```

The sram instruction retrieves the data form memory. Notice that the instruction specifies that three long-words be read, even though we only need two. The code does this because the two long-words we need are spread over three long-words of SRAM. The next instruction sets the byte alignment to on byte because the SRAM address in which we are interested is one byte array from four-byte alignment. The next three instructions take the three long-words that contain the data we want and put the aligned data in the destination registers. Notice that the first of the three byte_align_be instruction's implementation helps to explain shit. Figure D.46 shows the operation of the instruction. The diagram shows that the microengine has an internal register in which it stores a temporary value. The output register gets the first byte of the temporary value and the last bytes of the input register. Since the temporary value is undefined when we start this, the first output is useless.

Indexed Transfer Register Access.

Using unaligned access instructions alone is great when you know which register has the first byte of useful data. Because the sample application is reading the IP header from DRAM, which is eight-byte aligned, the IP header could be up to seven bytes out of alignment. So, not knowing the alignment in advance, the first byte of the IP header will be in one of two different registers.

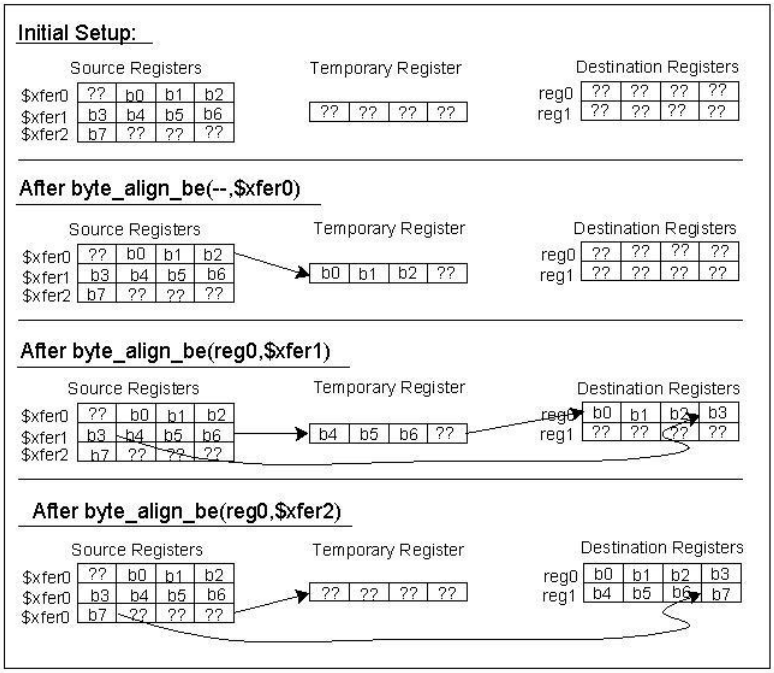


Figura D.46: Unaligned Access Instructions

To resolve this, the microengines provide indexed access to microengine transfer registers. When the code sets the T_INDEX microengine-local CSR, it can than access the transfer registers in an indexed mode. In microengine assembly, this is done by using the *$index keyword and the *$$index keyword for SRAM transfer registers and DRAM transfer registers respectively. If you want to access multiple ordered transfer registers, you can use the post-increment or post-decrement features of indexed transfer register access. For example, using *$index++ in an instruction access the indexed register and increments T_INDEX in one step.

Indexed transfer register usage and byte-alignment instructions are often used at the same time to allow the code to treat transfer registers as an ray of bytes. When you need to do both of these things at the same time, the code can set the T_INDEX_BYTE_INDEX register, which is a one-step way to set both the T_INDEX and BYTE_INDEX at the same time.

Our sample application needs to use both at the same time because we do no know the starting register of the IP header or the starting byte of the IP header within that register.

Hash Tables

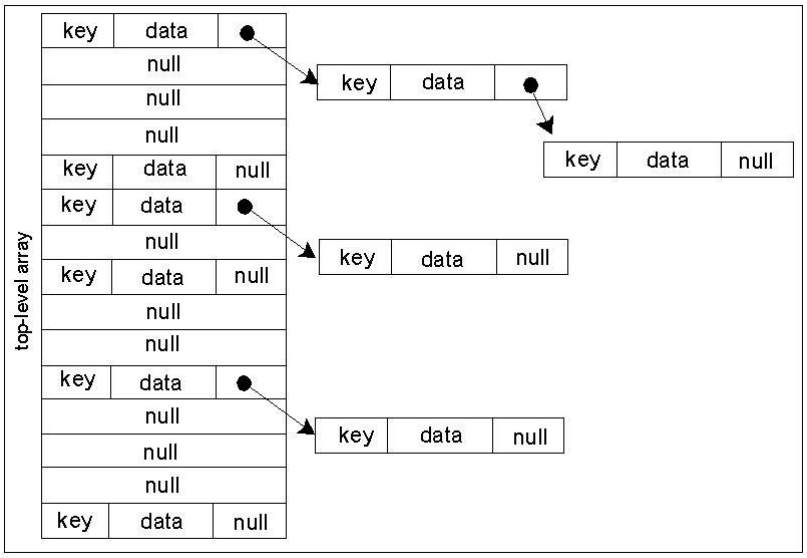


Figura D.47: A Simple Hash table

When application receives a packet, often one or more table lookups, based on the contents of the packet, are used to know how to process the packet. For example, the sample application we are building uses IP five-tuple to determine how to process IP packets. The code has to look for the correct next hop ID and flow ID in some sort of table. The design of the table and algorithm used to perform this lookup has a huge impact on the performance and resource requirements of the system.

A linked-list of five-tuples is easy to search, but is typically slow, especially when you need to support a large number of five-tuple instances in the lookup table. For a faster alternative, you could use an array indexed by the five-tuple. But, since the five-tuple is 104 bits long, using it as an array index would not work because the array would have to have 20*103 entries! Although a table that size would make RAM manufactures happy, it is not practical. The same is true for any other table lookup hey with more than a trivial number of bits.

Here's where hash tables become important. For those of you that may have forgotten, here is a refresher on hash tables. A hash table is like the array approach described above, but first a hash function is used to map a large lookup key to a smaller value that makes for a better table index. Of course, mapping a huge set of lookup keys to a much smaller set of table indexes implies that multiple lookup keys map to a single table index. When two keys map to the same index, the result is called a "collision", and any hash table lookup design must handle such collision property.

To resolve collisions, our hash tables are constructed as an array of linked lists as shown in Figure D.47. Each entry in the linked list contains the full lookup key and the table data we want to acquire. To perform a lookup, the array is indexed by the table index generated from mapping the lookup hey to a smaller value. Then, the linked list at that array index is searched for the correct key. Optimizations to this algorithm exist, but this one is simple enough for illustration purposes and still has reasonable performance.

In the IPv4 five-tuple classifier microblock, the code treats the five-tuple as the key, and the next-hop ID and flow ID as the data. The code maps the five-tuple to an eight-bit quantity that is used as an array index. The array entry is then treated like a linked list. The five-tuple is compared to the five-tuples in the list entries until one matches or until there are no more linked list entries. If one matches, the table lookup was successful and the table data is read. Otherwise, the table lookup fails.

Having a lot of collisions makes a hash table inefficient since the first lookup is O(1) and the linked list search is O(n) in its most basic form. So it is important that the mapping of lookup keys to table indices produce a few collisions as possible. A naïve implementation of this mapping for the classifier microblock would mask off the lower few bits of the five-tuple and use this as the table index. But statistical patterns abound in the fields of the IP five-tuple, so collisions would be common. For example, what if an operator decided to use only even host numbers? In this case, half of the table space could be wasted.

Good hash functions are usually difficult to implement in software, so the IXP2XXX microengines have access to two different hardware hashing mechanisms. The CRC unit in the microengines and hash hardware unit in the SHaC help to remove patterns from lookup keys. The produce results with a uniform statistical distribution, regardless of the input reducing the number of collisions experienced from processing packets. Each bit of the output value is independently one or zero with equal probability. So, to generate good hash-table inputs, the lookup hey can be run through either of these hardware units, and the result can be used to index into the top-level array. Using either of these hash functions certainly produces fewer hash collisions than simply masking off bits of the key.

Both hardware hash approaches have advantages and disadvantages that you must consider when deciding how to generate hash table indexes.

The Hash Unit

The hash functionality within the IXP2XXX SHaC unit is accessed by the same hardware queues that service other SHaC accesses. The hash unit can perform 48-bit, 64-bit, or 128-bit hashes with up to three hashes per instruction.

In microengine assembly, hash instructions are used to perform hashes, while in microengine C, instrinsics are used to perform hashes. The microengine assembly instructions and microengine C instrinsics are of the form hash_n, where n is the number of bits to hash. Both have an input parameter to specify the number of hashes to perform, between on and three.

The data returned by the hash is the same size as the data provided to the intrinsic. So hash_48 returns a 48-bit hash, hash_64 returns a 6bit hash and hash_128 return a 128-bit hash. This is where you are supposed to get suspicious. We told you a 48-bit quantity is too big, but the hash function creates a normal statistical distribution in the lower bits of the hash result as well as the whole result. So it is acceptable to mask off the low order bits of the hash result to fit whatever table size you want. For example, if you have room for a 1024 entry hash table, you can mask off the low-order 10 bits of the hash result and use them as your table index.

In some designs, having extra bits in the hash result is useful. In an application environment with a large number of collisions, the performance of the hash table lookups can sometimes increased by using multiple hash tables. In these designs, the code simultaneously indexes into these hash tables using different sets of bits from the hash result. In some environments, this implementation results in better packet throughput and delay compared to one performing a single hash table lookup per packet.

The CRC Unit

The CRC unit implements two standards-based CRC algorithms often used by network protocols to detect when packets have been corrupted in transit on a network. Both of the CRC algorithms implemented by the CRC unit can also be used to perform hashes of lookup keys as part of ash table lookup.

One advantage of using the CRC unit over the hash unit is that each microengine has its own CRC unit. Sending requests to the CRC unit, therefore, does not require a bus transaction or hardware queuing. So, performing hashes with the CRC with the CRC unit is much faster and consumes fewer resources than performing hashes with the hash unit in the SHaC. The drawback to using the CRC unit is that the result is either 16 or 32, so performing lookups in multiple hash tables as described above may not be an option when using the CRC unit.

The CRC unit takes 32-bits of data input from local memory, general purpose registers, or read transfer registers and performs a computation based on the input data and the contents of a microengine-local CSR called CRC_REMAINDER. The results of this computation are placed back in the CRC_REMAINDER CSR. Before hashing begins, the CRC_REMAINDER must be initialized to some well know value. To hash data larger than 32 bits, the data should be fed to th CRC unit 32 bits at a time. Reading the CRC_REMAINDER register will fetch the result.

Each microengine has only one CRC_REMAINDER register and one CRC unit, so it is important that only one thread attempts to perform a hash at any one time. A good way to ensure this is to simply avoud context swaps while performing the hash. Because the microengine threading is no pre-empitve, as long as the code dos no explicitly give up the context while performing CRC, it is safe.

Hash Table Modificiation

When designing a data structure for the IXP2XXX processor, it is important to keep synchronization in mind. For hash table may be happening in the Intel XScale core at the same time lookups in the hash table are happening in the microengines. Take care to make sure these concurrent operations do not affect one another. In many cases, the fact that the SRAM controller performs 32-bit writes atomically form the Intel XScale core can help avoid the need for synchronization. For example, in the simple hash table show in Figure D.48, adding an entry into a linked list can be done by setting up the entry key and data and latter adding a pointer to the new entry, an multiple SRAM writes would make the data structure invalid for a period of time. To resolve this, we can modify the design of the hash table to have 32-bit pointer in the top-level array instead of actual hash-table entries. This eliminates the need of extra synchronization.

Random Early Detect (RED) Congestion Avoidance.

Before the sample application equeues packets for the transmit code, it performs RED on the queue. RED is a congestion-avoidance algorithm invented by Sally Floyd and Van Jacobson while at the University of California. It allows packet-processing devices in TCP/IP networks to maintain high utilization of output links without including large queuing latencies in the network. In our device, we are only supporting two ports, so congestion is very unlikely. So, the inclusion of RED in our application is somewhat gratuitous. But, many other devices that can be built with IXP2XXX processors may need congestion avoidance, and implementing RED eposes some exciting features of the IXP2XXXX processor, so we included it in our application anyway.

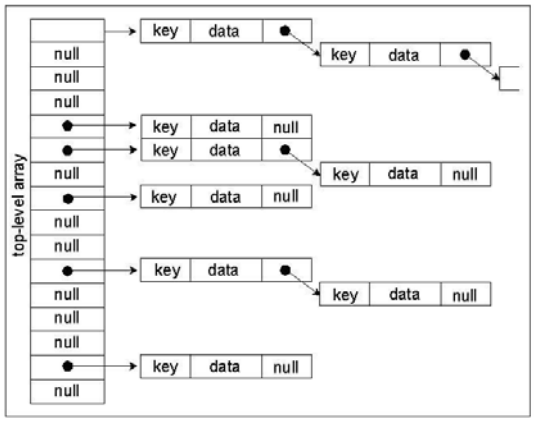


Figura D.48: Modified Hash Table

The RED algorithm is fairly simple. When a packet arrives at the system, the RED algorithm decides to either drop the packet or forward it. This decision is based on minimum and maximum thresholds, defines by the user, for a particular queue. If the number of packets in the queue is less than the minimum and maximum thresholds, the packet is forwarded. If the number of packets in the queue is greater than de maximum threshold, it is dropped. If the number of packets in the queue is between the minimum and maximum thresholds, there is a random probability that the packet will be dropped. This probability increases linearly between the minimum threshold and the maximum threshold. Figure D.49 shows the probability of a packet being dropped as a function of the number of packets in the queue.

TCP/IP traffic can be fairly bursty. So instead of using the instantaneous number of packets in the queue to determine the probability of dropping a packet, RED uses the average number of packet in the queue. Specifically, RED uses the exponential weighted moving average (EWMA) of the number of packets in the queue.

Development:

Simulate and explain the single thread processing code that your instructor provides. At the end of the practice you must hand in the code explanation and flowchart of ipv4_five_tuple_class.uc and ethernet.uc.

At the end of the session, all the comments will be revised and the practice will be checked by running the simulation.

Ing. Octaviano Flores Gzz. (Enero-Mayo, 07)

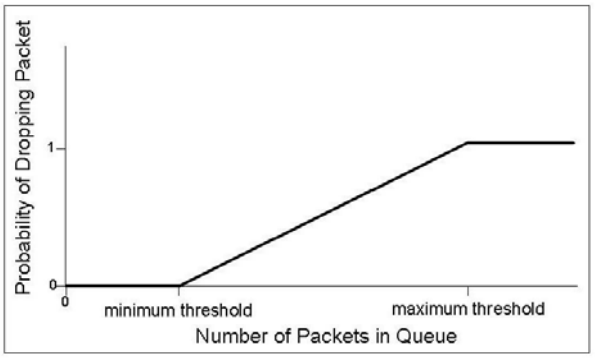


Figura D.49: Probability of Dropping a Packet

INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS MONTERREY

Practice #5: Testing a Real Program

Pre-Practice:

Objective

Develop the skills of a Tester Engineer. Testing IXA SDK 3.5 samples programs: ipv4_diffserv, ipv4forwarder and mpls.

Theoretical Framework:

The board ENP-2611 of Radisys uses the IXP2400 processor that has the capacity to run two operating systems: VxWorks and Hardhat, both of them use the RJ-45 port to load OS and communicate with the IDE (Tornado) for load programs. For example: in our lab we have VxWorks from Windriver and we use the RJ-45 port to load programs.

The VxWorks uses the Target Agent which is a program that manages the communication between the Target and Host, like in Figure D.50. And the Host must use a Target Server that must be configured and launched on the Tornado.

Development:

Making use of the topology lab (Figure 2) you must connect from your host to the VxWorks OS and test the program that you had chosen. Here are the general steps that everyone should use:

1. You must open the program that came with the IXA SDK 3.5 Framework (C:\IXA _SDK_3.5\src\applications). There are like nine different programs that applied for the IP2400 processor.

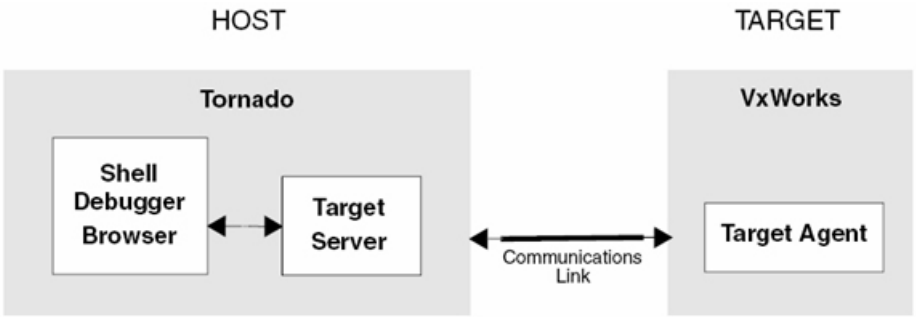


Figura D.50: Communication between Target Server and Target Agent

2. Then you must compile it and change the Preprocessor definitions that came with: "IXP-SIMULATION" and add "USE_IMPORT_VAR".

3. Return to Build menu and select Rebuild. This will create a microengine image file for the Program.

4. Open the Tornado workspace at IXA_SDK_3.5\cource workspace\ixa_sdk_2.2.wsp (Depends of the program it changes).

5. Click on the Build tab in the workspace window. You should look at the A_oc48 _ethernet_process. If you want to run the application in Debug mode right click and select the Build type as "XScalegnube_Debug". Debug mode will print more information to the screen.

6. Build the A_oc48_ethernet_egress project by right clicking and selecting Rebuild All. (Depends of the program it changes).

7. You can download the Egress image. Make sure that the name of the Egress target server is selected in the Tornado window's drop down list.

8. Start the system application on Egress by running ("_ix_sa_entry 1") on Egress shell. (Depends of the program it changes).

9. The system application prints "Started all microengines". (Depends of the program it changes).

Note: In the Folder of each application, there is a README file that helps you compile and explain with detail, specific steps that must be done to load and run successfully the application.

You must hand in a report that must contain:

- **Cover.** With complete names, IDs, Instructor name, Teacher, Project Name and Date.

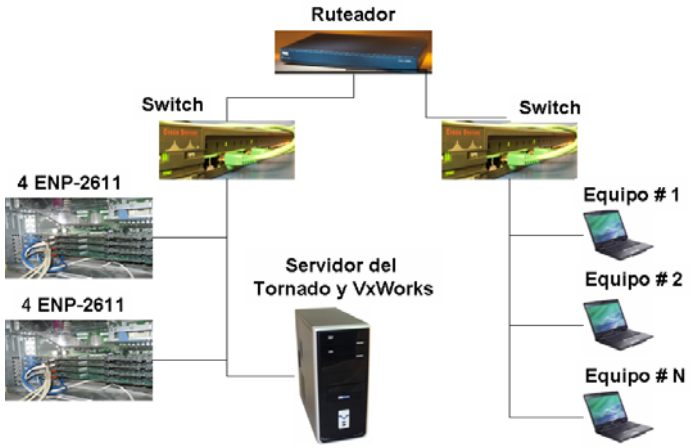


Figura D.51: Network Processor Lab Topology

- **Theoretical framework.** Explanation of code functionality and main purpose of the project.
- **Errors.** Mention each of the errors that are displayed in the shell of Tornado and in the console of NP Board. Explain the meaning these errors.
- **Error correction.** Here you can fix the errors and you will get extra point in other practices.
- **Individual conclusions.** All of the teams must write a paragraph with their conclusion.

Ing. Octaviano Flores Gzz. (Enero-Mayo, 07)

Bibliografía

[1] Douglas E. Comer. *Network System Design using Network Processors (Agere Version)*. Intel Press, Agust 2004.

[2] Douglas E. Comer. *Network System Design using Network Processors*. Pearson Prentice Hall, 2004.

[3] Dougles E. Comer. *Network System Design Using Network Processors (Intel IXP 2XXX Version)*. Pearson Prentice Hall, 2006.

[4] Cisco Company. The next generation in edge aggregation technologynetwork system design using network processors. In *Cisco 10000 Series Performance Routing Engine (PRE-2)*, pages 4–16, 2003.

[5] Windriver Company. In *Tornado users guide windows 2.2*, pages 277–296, 2003.

[6] Windriver Company. In *tornado getting started guide windows 2.2.1*, pages 15–40, 2003.

[7] Ruan Donghua, Lin Chuang, Chen Zhen, Ni Jia, and Peter D. Ungsunan. Handling high speed traffic measurement using network processors. pages 1–5, 2006.

[8] E. P. Ferlin, Jr. Pilla, V., and N. C. S. Filho. The theory-practice partnership. pages F2A/1–F2A/6, 2005.

[9] Donald F. Hooper. *Using IXP2400/2800 Development Tools*. Intel Press, Agust 2004.

[10] Erik J. Johnson and Aaron R. Kunze. *IXP2400/2800 Programming (The complete Microengine Coding Guide)*. Intel Press, April 2003.

[11] B. K. Lee and L. K. John. Npbench: a benchmark suite for control plane and data plane applications for network processors. pages 226–233, 2003.

[12] Leen-Kiat Soh, Ashok Samal, Suzette Person, Gwen Nugent, and Jeff Lang. Closed laboratories with embedded instructional research design for cs1. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pages 297–301, New York, NY, USA, 2005. ACM Press.

[13] Peter Steenkiste. A network project course based on network processors. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 262–266, New York, NY, USA, 2003. ACM Press.

[14] Theo Ungerer, Borut Robič, and Jurij Šilc. A survey of processors with explicit multithreading. *ACM Comput. Surv.*, 35(1):29–63, 2003.

[15] Luiz Filipe M. Vieira, Marcos Augusto M. Vieira, and Newton J. Vieira. Language emulator, a helpful toolkit in the learning process of computer theory. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 135–139, New York, NY, USA, 2004. ACM Press.

[16] J. Yu, W. Wu, X. Chen, H. Hsieh, J. Yang, and F. Balarin. Assertion-based power/performance analysis of network processor architectures. pages 155–160, 2004.