

# Implementación HW/SW de Arquitecturas de Clasificación de Paquetes en Lógica Reconfigurable

Luis Roberto Romano,      Jairo Nicolás Trad

Universidad Nacional de Córdoba

# Índice general

<b>1. Introducción</b>	<b>1</b>
<b>2. Sistema</b>	<b>2</b>
2.1. Sistemas Embebidos y Dispositivos Lógicos Programables . .	2
2.1.1. Sistemas Embebidos . . . . .	2
2.1.2. Dispositivos Lógicos Programables . . . . .	3
2.1.3. Sistemas embebidos en Lógica Programable . . . . .	4
2.2. Redes de Datos . . . . .	6
2.2.1. Modelo OSI . . . . .	6
2.2.2. TCP/IP . . . . .	8
2.2.3. Ethernet . . . . .	9
2.2.4. Protocolo IP . . . . .	11
2.2.5. Búsqueda del prefijo más largo . . . . .	14
2.3. Diseño . . . . .	17
<b>3. Arquitectura</b>	<b>19</b>
3.1. Procesador Embebido . . . . .	19
3.2. Modulo Gestor de Datos . . . . .	21
3.3. Software . . . . .	21
3.3.1. NIOS II SBT . . . . .	21
3.3.2. HAL . . . . .	23
3.3.3. Algoritmos de clasificación . . . . .	26
3.3.4. Cache . . . . .	29
<b>4. Implementación</b>	<b>30</b>
<b>5. Resultados</b>	<b>31</b>
5.1. Caso Algoritmos únicamente . . . . .	31
5.2. Caso loopback . . . . .	32
5.3. Implementación Completa . . . . .	33

<i>ÍNDICE GENERAL</i>	II
5.4. Cache . . . . .	37
<b>6. Conclusiones</b>	<b>38</b>
<b>Bibliografía</b>	<b>39</b>
<b>Apendices</b>	<b>40</b>
6.1. Configuración del Hardware . . . . .	40

# Índice de figuras

2.1. Diseño preliminar . . . . .	3
2.2. Cantidad de Elementos Lógicos/Celdas Lógicas por FPGA . .	5
2.3. Modelo OSI . . . . .	6
2.4. PDUs en las distintas capas del modelo OSI . . . . .	7
2.5. Relación entre los modelos OSI y TCP/IP . . . . .	9
2.6. Formato de trama IEEE 802.3 . . . . .	10
2.7. Formato de datagrama IP . . . . .	12
2.8. Unibit trie . . . . .	17
2.9. Diseño preliminar actualizado. . . . .	18
3.1. Capas de un sistema basado en la HAL. . . . .	24
3.2. Capas de un programa basado en HAL . . . . .	26
5.1. Retardo de Búsqueda LLU vs UTL . . . . .	32
5.2. Caso Loopback para 1 y 15 palabras . . . . .	33
5.3. Retardo mínimo LLU . . . . .	34
5.4. Retardo promedio LLU . . . . .	34
5.5. Retardo máximo LLU . . . . .	35
5.6. Retardo mínimo UTL . . . . .	35
5.7. Retardo promedio UTL . . . . .	36
5.8. Retardo máximo UTL . . . . .	36
5.9. Comparativa UTL - LLU . . . . .	37

# Índice de cuadros

## Capítulo 1

# Introducción

## Capítulo 2

# Sistema

En este capítulo se introduce brevemente una serie de conceptos que fueron utilizados para la realización del presente proyecto integrador. No se pretende que el lector alcance una comprensión exhaustiva de los mismos, sino que tenga las herramientas necesarias para la correcta interpretación de los capítulos posteriores. De manera paralela al desarrollo de dichos conceptos se introducen los bloques básicos que formaran parte del sistema planteado.

### 2.1. Sistemas Embebidos y Dispositivos Lógicos Programables

#### 2.1.1. Sistemas Embebidos

Un Sistema embebido es una combinación de hardware y software diseñado para realizar funciones dedicadas, generalmente en tiempo real. En algunos casos los sistemas embebidos no actúan de manera independiente y se encuentran integrados en un sistema o producto mayor. En la actualidad el 98 % de los microprocesadores fabricados tienen como destino algún sistema embebido mientras que solo el 2 % se destina a microprocesadores de propósito general.

El campo de aplicación para los sistemas embebidos es muy variado, desde dispositivos portátiles como reproductores MP3 o teléfonos celulares, hasta sistemas de control en centrales nucleares. No es posible caracterizar los componentes exactos de un sistema embebido ya que existen una gran cantidad de configuraciones posibles, pero dentro de los componentes fundamentales que se encuentran en la mayoría de ellos están:

- Microprocesador
- Memoria RAM
- Periféricos para Captura de datos y de Comunicación.

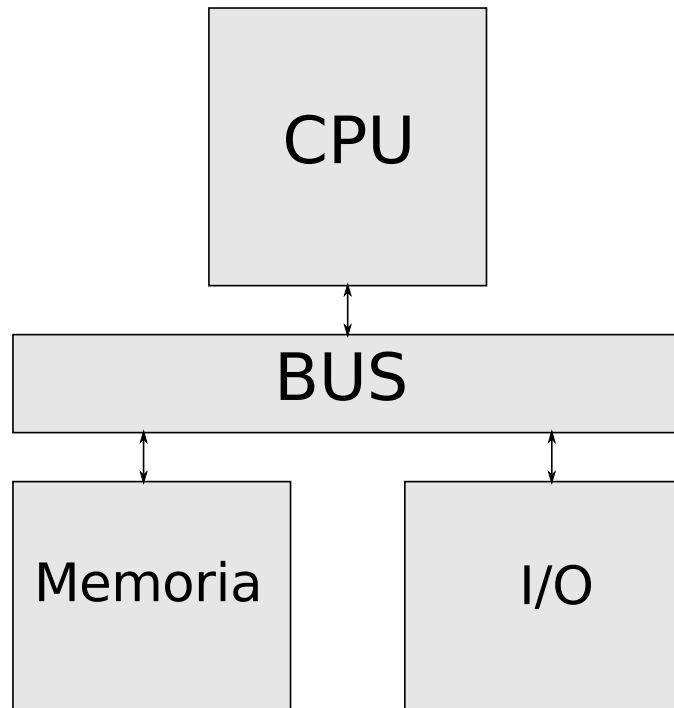


Figura 2.1: Diseño preliminar

El sistema embebido que será generado a partir de este proyecto no será la excepción y contará al menos con los componentes anteriormente nombrados. En la Figura 2.1 es posible ver el diseño preliminar del hardware de nuestro sistema.

### 2.1.2. Dispositivos Lógicos Programables

Los dispositivos lógicos pueden ser clasificados en dos categorías, fijos y programables. Los fijos son permanentes, están preparados para realizar una o una variedad específica de funciones y una vez manufacturados no pueden ser modificados. En cambio, los dispositivos lógicos programables, de ahora en adelante referenciados como PLD, ofrecen la posibilidad de ser



reprogramados en cualquier momento con cualquier función lógica que el usuario requiere y que sea soportado por el dispositivo en cuestión.

Existen dos grandes tipos de PLD, FPGA y CPLD. Los CPLD ofrecen una capacidad lógica pequeña, pero con unas características de temporización predecibles lo que los hace dispositivos ideales para aplicaciones de control y cualquier otra tarea de tiempo real. Además su consumo de potencia reducido y su bajo costo los posicionan muy bien en el campo de los dispositivos portátiles. Por otro lado las FPGA ofrecen una mayor densidad lógica y un mejor rendimiento. Actualmente es posible encontrar hasta 2 millones de celdas lógicas en una FPGA comercial. Además estos dispositivos ofrece módulos embebidos como tranceptores, memoria o microprocesadores. Todo esto le permite a las FPGA ser usadas en una amplia variedad de aplicaciones como procesamiento de datos, instrumentación, telecomunicaciones y procesamiento de señales.

En el presente proyecto se enfocará el desarrollo sobre FPGA ya que se dispone del Hardware necesario para dicha tarea y además se cuenta con el conocimiento previo que requiere el desarrollo en este tipo de dispositivos.

### 2.1.3. Sistemas embebidos en Lógica Programable

La tecnología que forma parte de las FPGA esta disponible desde mediados de los 80. Inicialmente eran usadas en su mayoría como contenedores de la lógica de interconexion que rodea a aplicaciones de microprocesadores. En los últimos años las FPGA se utilizaron para el procesamiento de señales y como una alternativa a los DSP de propósito general.

Actualmente existe un interés particular por el uso de las FPGA como plataforma para sistemas embebidos, especialmente en aplicaciones que no justifican el costo de un ASIC. Esto es posible gracias a dos tendencias observadas en los últimos años, el incremento de la cantidad de elementos lógicos disponibles en cada dispositivo y la evolución de las Herramientas de Software provistas por los fabricantes. En el figura 2.2 es posible ver el aumento de la cantidad de elementos lógicos, o celdas lógicas, disponibles en dispositivos de alta gama de las dos empresas mas importantes del sector.

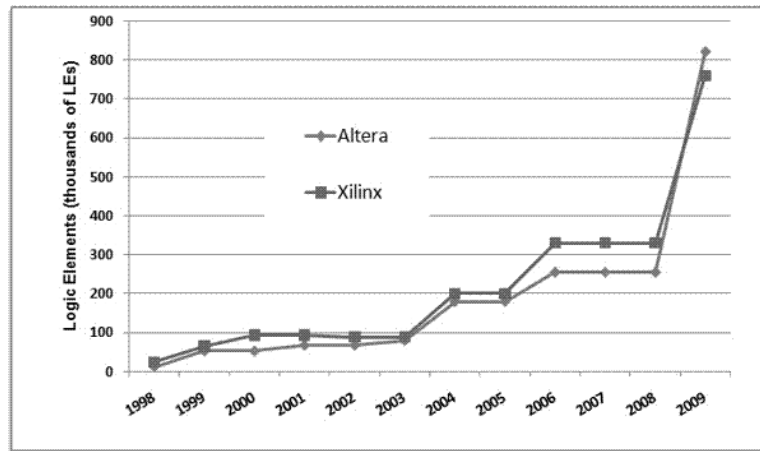


Figura 2.2: Cantidad de Elementos Lógicos/Celdas Lógicas por FPGA

### Procesadores embebidos en Lógica Programable

A la hora de implementar un sistema embebido en una FPGA es importante la selección del procesador que va a formar parte del mismo, aunque se realizará un estudio mas detallado de las opciones disponibles en el próximo capítulo. En este caso tendremos dos tipos de procesadores disponibles, Softcores y Hardcores.

Los Hardcores son procesadores físicos embebidos directamente en el silicio de la FPGA. La evidente ventaja es el mayor rendimiento que se obtiene al contar con un procesador físico, mientras que entre las desventajas están la menor flexibilidad a la hora de diseñar el sistema y el mayor costo de los dispositivos que proveen este tipo de procesador.

Los Softcores son procesadores que se construyen a partir de la lógica de propósito general de la FPGA, por lo cual es necesario que estos se encuentren descritos en algún Lenguaje de Descripción de Hardware(HDL). Aunque el rendimiento de los Softcores es bastante menor que el de los Hardcores, estos poseen un nivel de configuración que permite mas opciones y además son, en general, independientes del dispositivo en el que se los implemente.

En ambos tipos de procesadores la memoria, la interconexión entre periféricos, los controladores de periféricos y los periféricos mismos deben ser construidos con la lógica de propósito general de la FPGA.

## 2.2. Redes de Datos

### 2.2.1. Modelo OSI

Es el modelo de red descriptivo creado por la Organización Internacional para la Estandarización en el año 1984. Sirve como marco de referencia para la definición de arquitecturas de interconexión de sistemas de comunicaciones.

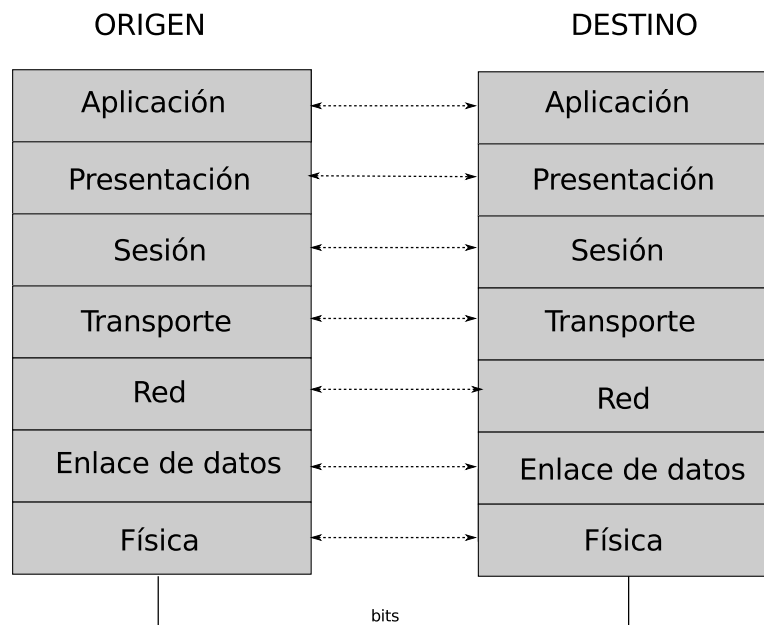


Figura 2.3: Modelo OSI

Se basa en el concepto de *capa*: un conjunto de rutinas (y en algunos casos, hardware) que cumplen una serie de funciones determinadas. Cada capa ofrece servicios a las q se ubican por encima y hace uso de servicios ofrecidos por las de abajo.

El modelo está conformado por las siguientes capas:

- **Aplicación**: representa el punto de acceso de las aplicaciones que hacen uso del esquema de transmisión de datos.
- **Presentación**: relacionada con la representación de los datos.
- **Sesión**: se encarga de mantener y controlar el enlace establecido entre dos computadores que están transmitiendo datos.

- Transporte: encargada de transportar los datos (que se encuentran dentro del paquete) de la máquina origen a la de destino, independizándolo del tipo de red física que se esté utilizando.
- Red: se encarga de identificar el enrutamiento existente entre una o más redes.
- Enlace de datos: se ocupa del direccionamiento físico, de la topología de la red, del acceso al medio, de la detección de errores, de la distribución ordenada de tramas y del control del flujo.
- Física: se encarga de las conexiones físicas de la computadora hacia la red, tanto en lo que se refiere al medio físico como a la forma en la que se transmite la información.

### Unidad de datos del protocolo

Es un bloque de datos que cada capa utiliza para el intercambio de información con su par en el sistema destino. Suele denominarse PDU, por las siglas en inglés de *Protocol Data Unit*.

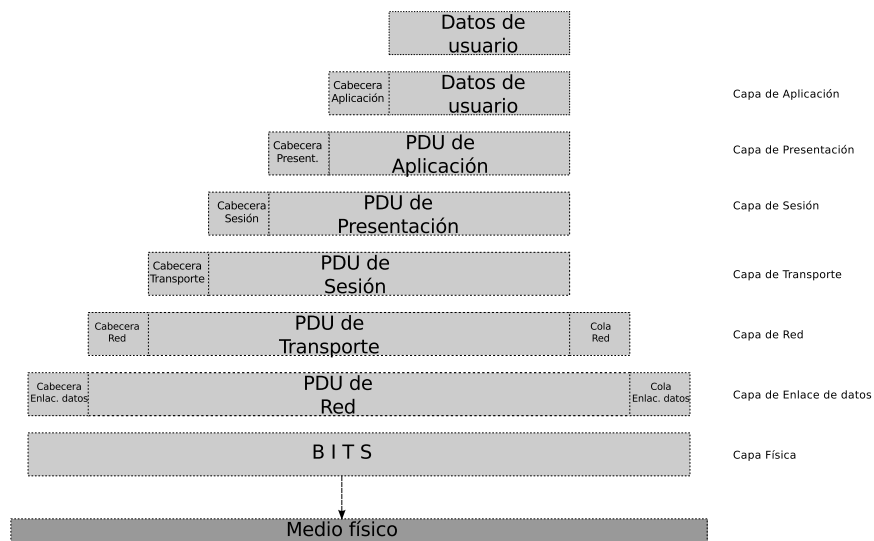


Figura 2.4: PDUs en las distintas capas del modelo OSI

Cuando llega la información por parte del usuario, la capa de aplicación le agrega información de control y se conforma la PDU de dicha capa, la cual es transferida a la siguiente, que va a hacer lo mismo. De esta manera,

cada capa agrega información a la PDU recibida de la anterior y transfiere la PDU resultante hacia la de abajo. Esto se repite hasta llegar al nivel físico, en el cual la información se transmite desde el sistema origen hacia el sistema destino en forma de bits.

Una vez que los datos han llegado a destino, comienza el proceso inverso. Los bits en capa física son transferidos a la capa de enlace de datos, la cual los delimita conformando tramas (PDUs de dicha capa). En base a la información de control, cada capa va transfiriendo su PDU hacia arriba, hasta que los datos llegan a la aplicación de destino. Se dice que la comunicación entre capas pares es transparente, ya que cada una de ellas no es consciente de los detalles de la implementación de las capas inferiores, tanto en el sistema origen como en el sistema destino. La figura 2.3 muestra este concepto.

### 2.2.2. TCP/IP

Es un conjunto de protocolos de red en los que se basa Internet y que permiten la transmisión de datos entre computadoras. Se le denomina simplemente TCP/IP, en referencia a los dos protocolos más importantes que la componen: Protocolo de Control de Transmisión (TCP) y Protocolo de Internet (IP).

#### Relación con el modelo OSI

TCP/IP puede describirse como un modelo de capas al igual que OSI. Sin embargo, éste último es más bien una abstracción teórica, que sirve como referencia, mientras que TCP/IP es una implementación práctica que cuenta con 5 capas en vez de 8.

- Aplicación: contiene toda la lógica necesaria para posibilitar las distintas aplicaciones de usuario.
- Transporte: encargada de garantizar que todos los datos se entreguen en forma fiable, esto es, que los datos lleguen a la aplicación de destino en el mismo orden en que fueron enviados. El protocolo más utilizado a tal fin es TCP (Transfer Control Protocol).
- Internet: provee una serie de procedimientos que permite que los datos atraviesen las distintas redes interconectadas. El protocolo IP (Internet protocol) se utiliza en esta capa para ofrecer el servicio de encaminamiento a través de varias redes.

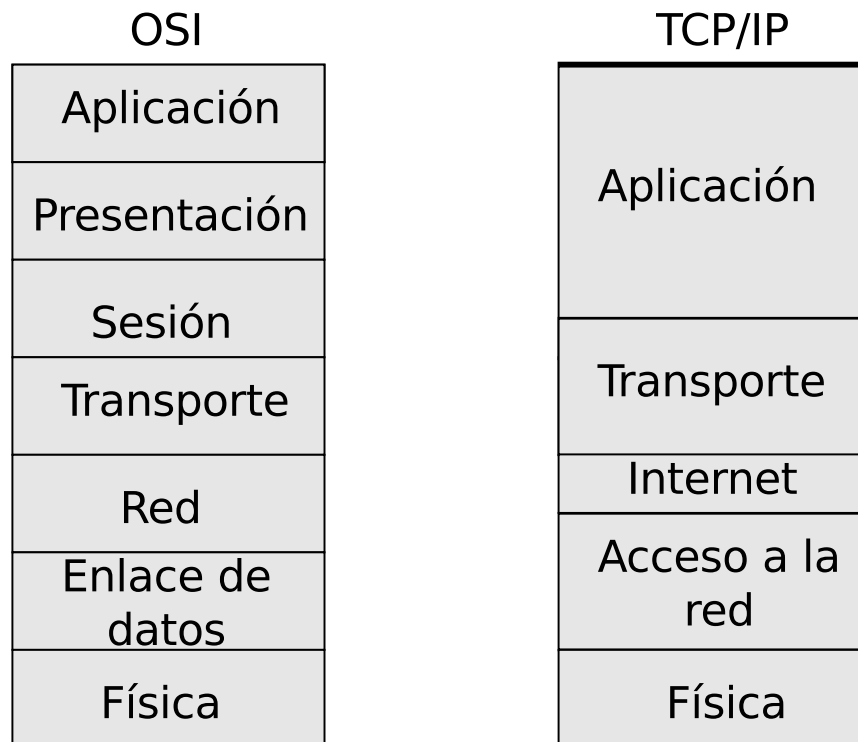


Figura 2.5: Relación entre los modelos OSI y TCP/IP

- Acceso a la red: es responsable del intercambio de datos entre el dispositivo y la red a la cual está conectado.
- Física: define la interfaz física entre el dispositivo de transmisión y la red. (Análoga a la capa física en el modelo OSI).

### 2.2.3. Ethernet

Ethernet es un estándar de transmisión de datos para redes de área local. El mismo define las características del cableado y señalización en capa física, así como también el formato de trama en capa de enlace de datos.

#### Formato de trama

La trama Ethernet definida en el estándar original IEEE 802.3 contiene los siguientes campos

Preámbulo	Comienzo de trama	Dirección MAC de destino	Dirección MAC de origen	Longitud / Tipo	Payload	Relleno	Sec. comprob. trama
-----------	-------------------	--------------------------	-------------------------	-----------------	---------	---------	---------------------

Figura 2.6: Formato de trama IEEE 802.3

- **Preámbulo:** es una secuencia de 56 bits que tiene 1 y 0 alternados que son utilizados para sincronización. Estos sirven para detectar la presencia de señal y para ir leyendo la señal antes que la trama de datos arribe.
- **Delimitador de comienzo de trama:** secuencia de 8 bits que tiene la configuración 10101011 que indica el comienzo de una trama.
- **Direcciones MAC de origen y destino:** la dirección de destino identifica la estación o las estaciones que reciben el cuadro. La dirección de fuente identifica la estación que ha originado la trama. Una dirección de destino que tiene todos sus bits igual a 1 se refiere a que la trama es enviado a todas las estaciones de la LAN. Ocupan 6 bytes.
- **Longitud / tipo:** si el valor de este campo es menor o igual que 1500 indica el número de bytes que contiene el campo *Payload*. Si el valor de este campo es mayor o igual a 1536, indica el tipo de protocolo de la capa MAC. Es un campo de 2 bytes.
- **Payload:** este campo contiene los datos transferidos. El largo máximo de este campo es 1500 bytes. Si la medida del campo es menor de 46 bytes es necesario usar el campo *relleno* para llevar el tamaño de la trama hasta el largo mínimo.
- **Relleno:** en este campo se agregan los bytes de datos extra para llevar el largo del cuadro hasta su tamaño mínimo. El mínimo largo de trama de Ethernet es de 64 bytes desde el campo de la dirección de destino MAC hasta el campo Frame Check Sequence.
- **Secuencia de comprobación de trama:** este campo contiene el valor de 4 bytes del chequeo cíclico redundante (CRC) usado para el chequeo de errores. Cuando una estación arma un cuadro MAC esta realiza el calculo del CRC en todos los bits desde la dirección de destino MAC hasta el campo *Relleno* (esto es, todos los campos excepto el Preámbulo, el delimitador de comienzo de trama y secuencia de comprobación de trama). El valor se guarda en este campo y se transmite como parte de la trama. Cuando el cuadro es recibido por la estación de destino,

ésta realiza un chequeo idéntico. Si el valor calculado no es igual al valor en este campo, se asume que un error a ocurrido durante la transmisión y se descarta la trama.

#### 2.2.4. Protocolo IP

Es un protocolo no orientado a conexión (esto es, establece una comunicación entre dos puntos finales de una red en los que un mensaje puede ser enviado desde un punto final a otro sin acuerdo previo), para la comunicación de datos, a través de una red de paquetes conmutados no fiable y de mejor entrega posible sin garantías.

IP no provee ningún mecanismo para determinar si un paquete alcanza o no su destino y únicamente proporciona seguridad (mediante checksums o sumas de comprobación) de sus cabeceras y no de los datos transmitidos. Por lo tanto, la seguridad de éstos queda a cargo de algún mecanismo de capa superior.

#### Fragmentación

La unidad máxima de transferencia (Maximum Transfer Unit - MTU) se define como el tamaño en bytes de la unidad de datos más grande que puede enviarse usando un protocolo de comunicaciones. Para el caso del protocolo IP este valor es de 64 KBytes. Muy pocos protocolos o tecnologías a nivel de enlace admiten enviar tramas de semejante tamaño. Normalmente el nivel de enlace no fragmenta<sup>3</sup>, por lo que tendrá que ser IP el que adapte el tamaño de los datagramas para que quepan en las tramas del nivel de enlace; por tanto en la práctica el tamaño máximo del datagrama viene determinado por el tamaño máximo de trama característico de la red utilizada.

En este contexto, pueden distinguirse 2 tipos de fragmentación:

- Fragmentación en ruta: se produce cuando un datagrama es creado por un host en una red con un valor determinado de MTU y en su camino hacia el host de destino ha de pasar por otra red con una MTU menor. En estos casos el router que hace la transición a la red de MTU menor ha de fragmentar los datagramas para que no excedan el tamaño de la nueva red.
- Fragmentación en origen: se produce como consecuencia del diseño de la aplicación (es decir, cuando ésta trabaja con bloques de datos mayores al MTU del protocolo de enlace de datos).



**Datagrama IP**

1	4	8	16	19	32
Version	Long. Cabecera	Tipo de servicio	Longitud total		
Identificación			Flags	Desplazamiento del fragmento	
Tiempo de vida		Protocolo	Suma de comprobación de la cabecera		
Dirección origen					
Dirección destino					
Opciones + Relleno					

Figura 2.7: Formato de datagrama IP

La PDU del protocolo IP se denomina *datagrama IP* y se compone de los siguientes campos:

- Versión: indica la versión del protocolo.
- Longitud de la cabecera: longitud de la cabecera expresada en palabras de 32 bits.
- Tipo de servicio: especifica los parámetros de fiabilidad, prioridad, retardo y rendimiento.
- Longitud total: longitud total del datagrama en Bytes.
- Identificador: un número de secuencia que, junto a la dirección origen y destino y el protocolo usuario, se utiliza para identificar de forma unívoca al datagrama.
- Flags: son 3 bits de los cuales solo 2 están definidos. El bit de *no fragmentación* prohíbe la fragmentación cuando está puesto a 1. El bit de *Más datos* se utiliza para fragmentación y reensamblado.

- Desplazamiento del fragmento: indica el lugar donde se sitúa el fragmento dentro del datagrama original, medido en unidades de 64 bits.
- Tiempo de vida: en cada router se decrementa en 1 unidad. Tiene como fin evitar que un datagrama se quede dando vueltas para siempre en la red.
- Protocolo: especifica a que protocolo del nivel de transporte corresponde el datagrama.
- Suma de comprobación de la cabecera: Es el complemento a uno de la suma en complemento a uno de todas las palabras de 16 bits de la cabecera.
- Dirección origen: codificada para permitir una asignación variable de bits para especificar la red y el sistema final conectado a la red especificada.
- Dirección destino: igual que el campo anterior.
- Opciones: contiene las opciones solicitadas por el usuario que envía los datos.
- Relleno: se usa para asegurar que la cabecera del datagrama tenga una longitud múltiplo de 32 bits.
- Datos: debe tener una longitud múltiplo de 8 bits.

### Dirección IP

Es un número de 32 bit que identifica un dispositivo dentro de una red que utilice el protocolo IP. Las direcciones IP se suelen representar por cuatro números decimales separados por puntos, que equivalen al valor de cada uno de los cuatro bytes que componen la dirección.

Como ocurre en la mayoría de las redes las direcciones IP tienen una estructura jerárquica. Una parte de la dirección corresponde a la red, y la otra al host dentro de la red. Cuando un dispositivo de enrutamiento recibe un datagrama por una de sus interfaces compara la parte de red de la dirección con las entradas contenidas en sus tablas (que normalmente sólo contienen direcciones de red, no de host) y envía el datagrama por la interfaz correspondiente.

En el diseño inicial de Internet se reservaron los ocho primeros bits para la red, dejando los 24 restantes para el host; se creía que con 254 redes

habría suficiente para la red experimental de un proyecto de investigación del Departamento de Defensa americano. Pronto se vio que esto resultaba insuficiente, por lo que se reorganizó el espacio de direcciones reservando unos rangos para definir redes más pequeñas. El resultado de esa reorganización es lo que hoy se conoce como las redes clase A, B y C.

- Una red de clase A se caracteriza por tener a 0 el primer bit de dirección; el campo red ocupa los 7 bits siguientes y el campo host los últimos 24. Puede haber hasta 128 redes de clase A con 16777216 direcciones cada una.
- Una red de clase B tiene el primer bit a 1 y el segundo a 0; el campo red ocupa los 14 bits siguientes, y el campo host los 16 últimos. Puede haber 16384 redes clase B con 65536 direcciones cada una.
- Una red clase C tiene los primeros tres bits a 110; el campo red ocupa los siguientes 21 bits, y el campo host los 8 últimos. Puede haber hasta 2097152 redes clase C con 256 direcciones cada una.

Los bits que corresponden a la parte de red conforman lo que se denomina *prefijo de red*.

La forma más común de representar una dirección IP es en lo que se denomina *notación punto-decimal*. Esto es, cada uno de los cuatro octetos que conforman la dirección se colocan en base decimal separados por puntos.

Las máscaras permiten extraer de forma sencilla la parte de red o de host de una dirección. Por ejemplo un router que ha de enviar un datagrama puede realizar un AND entre la dirección de destino y la máscara correspondiente, con lo que extraerá la parte de red de la dirección.

Existen además direcciones (no redes) clase D cuyos primeros cuatro bits valen 1110. Las direcciones clase D se utilizan para definir grupos multicast. El grupo queda definido por los 28 bits siguientes. Puede haber hasta 268435456 direcciones multicast en Internet. Las direcciones clase D nunca puede aparecer como direcciones de origen de un datagrama.

Por último, la clase E, que corresponde al valor 1111 en los primeros cuatro bits, no se utiliza de momento y está reservada para usos futuros.

### 2.2.5. Búsqueda del prefijo más largo

#### Formas de representar un prefijo

Existen 3 maneras de representar un prefijo de red:

- Binario con asterisco: por ejemplo, el prefijo 132.239 se denotaría 1000010011101111\* (dado que 132 es en binario 10000100 y 239 es 11101111). El asterisco al final denota que los bits restantes pueden ser de cualquier valor.
- Notación A/L, donde A es una dirección IP y L es la longitud del prefijo. Siguiendo el ejemplo anterior, la notación sería 132.239.0.0/16.
- Notación máscara: se utiliza una dirección de red y una máscara en vez de un prefijo explícito. De esta manera, volviendo al ejemplo anteriormente mencionado, éste puede expresarse como 132.239.0.0 con máscara 255.255.0.0

### Concepto de Lookup

El procedimiento que se lleva a cabo en un dispositivo de enrutamiento podría describirse de la siguiente manera:

Un paquete llega por una interfaz de entrada. Éste porta una dirección IP determinada. El dispositivo consulta una tabla de forwardo para determinar la interfaz de salida para el paquete en cuestión. Dicha tabla contiene un conjunto de prefijos con sus correspondientes interfaces de salida. El paquete es correspondido con el prefijo más largo que esté contenido en la dirección de destino y luego es redirigido a la correspondiente interfaz de salida. Esta tarea de determinar el enlace de salida es denominada *Búsqueda de dirección (address lookup)*.

### Búsqueda lineal

El esquema más simple de lookup consiste en una tabla, donde cada entrada contiene un prefijo (expresado en notación máscara) más un identificador de enlace de salida. Cuando llega un paquete se examina su dirección IP y se va comparando con cada uno de los prefijos almacenados. Esto es, se efectúa una operación AND entre la dirección IP del paquete y la máscara, que debe dar igual a la dirección de red de la entrada de tabla. Si esto se cumple para varios prefijos, se opta por el más largo de ellos y el paquete se expide por el enlace asociado a dicho prefijo. Para optimizar este esquema, los prefijos se almacenan en orden decreciente de longitud, de manera que al encontrar una coincidencia no sea necesario seguir buscando en el resto de la tabla.

### Unibit tries

Un unibit trie es un árbol en el cual cada nodo contiene un *puntero-cero* y un *puntero-uno*. Partiendo del nodo raíz, todos los prefijos que comienzan con 0 son almacenados en el subárbol apuntado por el puntero-cero y aquellos que comienzan con 1 se almacenan en el subárbol apuntado por el puntero-uno.

Cada subárbol es construido recursivamente de manera similar usando los bits restantes de cada uno de los prefijos.

Considerar la siguiente tabla de ruteo

Prefijo	Enlace de salida
101*	S1
111*	S2
11001*	S3
1*	S4
0*	S5
1000*	S6
100000*	S7
100*	S8
110*	S9

A la misma le corresponde representación en unibit trie de la figura 2.8

Considerar una dirección de destino  $D$ . Para efectuar la búsqueda del prefijo más largo los bits de  $D$  son usados para trazar un camino a lo largo del trie. Dicho camino comienza en el nodo raíz y continúa hasta toparse con un puntero vacío o un nodo vacío. Durante el recorrido a través del trie, el algoritmo mantiene un registro del último prefijo encontrado en un nodo en el camino. Cuando la búsqueda falla, ese es el prefijo retornado.

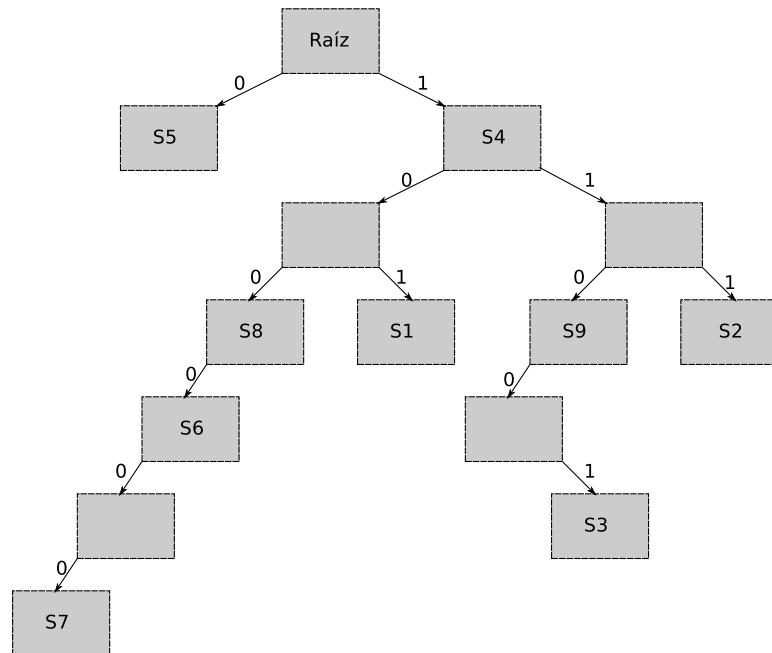


Figura 2.8: Unibit trie

### 2.3. Diseño

A lo que es posible ver en la figura 2.1 es necesario agregarle algún tipo de interfaz de red que permita el ingreso de datos al sistema para luego procesarla. Además, según lo planteado en los objetivos de este trabajo, se desea implementar un sistema operativo o algún tipo de capa de abstracción que permita la ejecución de un Software en el que se realizará dicho procesamiento de estos paquetes. Para vincular estos dos módulos también es necesario un módulo que transporte los datos recibidos por la interfaz de red hacia el software y luego sobrescriba los resultados obtenidos en una etiqueta adjunta a los datos procesados. En la figura 2.9 se puede ver como resulta el diseño propuesto.

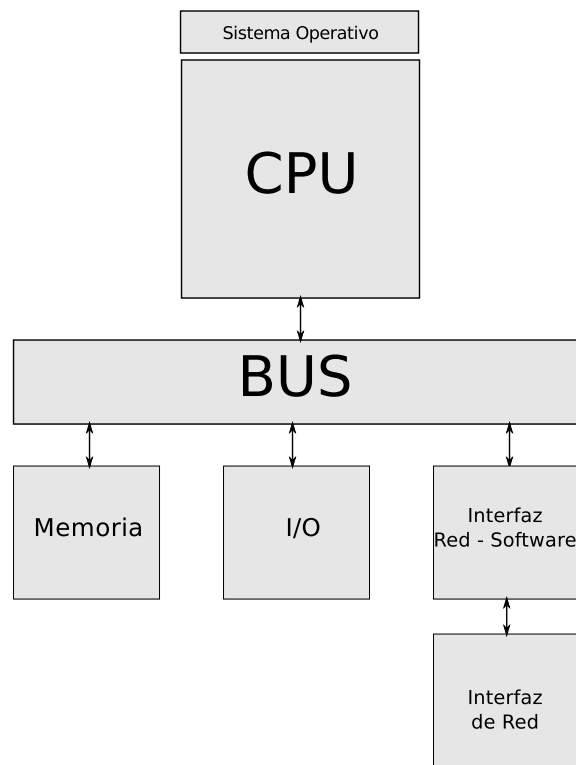


Figura 2.9: Diseño preliminar actualizado.

## Capítulo 3

# Arquitectura

En este capitulo se detallara la arquitectura del proyecto en general, poniendo foco en la seleccion adecuada de las tecnologias a utilizar y completando mas detalladamente el diseño obtenida en el capitulo anterior.

### 3.1. Procesador Embebido

Como se dijo en el capitulo 2, los microprocesadores embebidos en logica reprogramable se clasifican, basicamente, en dos tipos Hardcore y Softcore. Aunque los Hardcores ofrecen unas mejores prestaciones existen muy pocos dispositivos en el mercado que ofrecen este tipo de modulos y, en general, el costo de este tipo de dispositivos es varias veces mayor que el de uno que no cuenta con un procesador embebido. Por esta razones de ahora en adelante se limitara el estudio a Softcores, siempre buscando que lo implementado siga los principios del diseño modular a los fines de facilitar la portabilidad.

#### 3.1.1. Estudio de los Softcores disponibles en el mercado

Los SoftCores pueden ser clasificados en tres tipos segun su origen

- Provistos por los fabricantes de las FPGA
- Provistos por empresas distribuidoras de bloques IP(intelectual Property)
- Provistos por la comunidad Open Source



**Provistos por los fabricantes de las FPGA**

Xilinx y Altera, los dos mayores fabricantes de FPGA, proveen un Soft-core para integrar en sus dispositivos. La ventaja natural de este tipo de core es que se encuentran altamente integrados en las herramientas de desarrollo de dichos fabricantes, permitiendo montar un sistema embebido en minutos de manera grafica e intuitiva. Ademas estan a disposición del usuario una buena cantidad de modulos IP para anexar al procesador y aumentar asi sus funcionalidades.

Dentro de las desventajas esta el hecho de que son dependientes de la plataforma y no es posible implementar el core propietario de Altera en un dispositivo Xilinx, y viseversa. Ademas se debe pagar una licencia por el uso de estos cores y no se cuenta con la posibilidad de observar o modificar el codigo de los mismos.

Xilinx ofrece a los usuarios de sus dispositivos el MicroBlaze, un procesador RISC de 32 bits con una arquitectura muy similar al MIPS DLX. A nivel herramientas de desarrollo la empresa provee el "Xilinx's EDK(Embedded Development Kit)" que permite generar un sistema embebido completo con las interconexiones necesarias. Es posible correr el sistema operativo GNU/Linux en este tipo de procesador.

Altera por su parte ofrece la linea de Cores NIOS II, tambien RISC de 32 bits, que cuenta en tres variantes principales Nios II Fast, Economy y Standard y dos variantes especializadas Nios II SC core, para aplicaciones militares y aeroespaciales, y DesignWare IP, para implementar en ASIC. Altera provee ademas una interfaz de usuario para la construccion del sistema embebido llamado SOPC que se encuentra integrado a Quartus y ademas el Nios II Embedded Design Suite (EDS) que permite diseñar el software requerido para cada aplicacio. Permite la implementacion del sistema operativo GNU/Linux.

**Provistos por empresas distribuidoras de bloques IP**

Existe una buena cantidad de empresas que se dedican a generar y comercializar bloques de propiedad intelectual (IP Cores o IP Blocks), unidades logicas reusables que son propiedad intelectual de un individuo o empresa. Varias de estas empresas cuentan con Softcores con distintas capacidades y tecnologias. La ventaja principal es la portabilidad de estos cores entre FPGA de distintos fabricantes. Entre las desventajas podemos nombrar la calidad de la documentacion y la complejidad intrinseca que implica comprender y utilizar una herramienta de desarrollo de un tercero.

Dentro de los mas conocidos se puede nombrar a la linea Leon de la empresa Aeroflex, LatticeMico32 de Lattice semiconductor o ARM cortex de Freescale. Los IP core pueden estar licenciados de diferentes formas incluidas entre ellas GPL.

### **Provistos por la comunidad Open Source**

La comunidad open source se ha encargado de producir una buena cantidad de softcores, las capacidades y perfomance de los mismos cubre casi todos los rangos. En la pagina web que lleva una lista de este tipo de proyectos, opencores, se pueden encontrar alrededor de 130 cores. El problema con este tipo de core esta en la poco uniforme calidad de la documentacion y en la calidad dudosa de varios de los cores. Sin embargo la flexibilidad para poder modificar la arquitectura a gusto y el costo cero de implementacion son ventajas nada despreciables de estos productos.

Se destaca el proyecto Openrisc 1200, que tiene una gran comunidad que lo soporta y algun prestigio ganado en este terreno.

### **Tabla comparativa**

## **3.2. Modulo Gestor de Datos**

## **3.3. Software**

### **3.3.1. NIOS II SBT**

El NIOS II Software Building Tools (o SBT) es un conjunto de utilidades y scripts que sirve para crear y construir aplicaciones embebidas basadas en C/C++, librerías de usuario y paquetes de soporte de placa (board support packages o BSP).

Puede invocarse desde la IDE Eclipse o desde el intérprete de comandos del NIOS II.

El NIOS II SBT puede crear los siguientes tipos de proyecto:

- Aplicación NIOS II: un programa que implementa alguna funcion deseada.
- NIOS II BSP: una librería que provee acceso al hardware en el sistema. Brinda un entorno de rutinas a medida para un procesador y, eventualmente, un sistema operativo.
- Librería de usuario: un conjunto de funciones reutilizables.

### Aplicaciones y librerías de usuario

Para el caso de aplicaciones y librerías de usuario, el SBT genera un makefile privado (denominado **Makefile**) el cual es usado para construir el proyecto. Al hacer esto se genera un archivo .elf para una aplicación, o .a para una librería. En este último caso, también se produce un makefile público (denominado **public.mk**), que se incluye en el privado para cualquier aplicación que use la librería de usuario.

Cuando se crea un makefile se provee al SBT con una lista de archivos de código fuente y una referencia al directorio donde se almacena el BSP. Luego, la herramienta examina la extensión de cada archivo fuente para determinar el lenguaje de programación.

Actualmente, los soportados son:

- C (extensión .c).
- C++ (extensiones .cpp, .cxx, .cc).
- NIOS II Assembler (extensiones .s, .S).

### Board Support Packages

Un BSP es una librería especializada que contiene código de soporte específico del sistema. Aisla la aplicación de los detalles del sistema, tales como mapeo de memoria, dispositivos disponibles y configuración del procesador.

Se compone de:

- Capa de abstracción de Hardware (Hardware Abstraction Layer o HAL): Permite al software interactuar con el hardware del sistema. Se describirá en detalle más adelante en este capítulo.
- Librería C estándar newlib: ANSI C estándar diseñada para sistemas embebidos.
- Drivers de dispositivos: para manejar cada uno de los componentes del sistema.
- Paquetes de software opcionales: permiten proveer funcionalidad adicional.
- Sistema operativo de tiempo real opcional: implementación de MicroC/OS-II RTOS.

### Proceso de construcción del software

Para crear un proyecto de software se llevan a cabo una serie de pasos:

1. Se obtiene el diseño de hardware sobre el cual va a correr el software. Dicha información está almacenada en un archivo de extensión **.sopcinfo**, el cual es generado por la herramienta SOPC builder.
2. Se genera el BSP con las características necesarias según las funcionalidades requeridas. También se genera un makefile para dicho paquete.
3. Opcionalmente, se crea una librería de usuario (junto a su correspondiente makefile).
4. Se escribe el software de aplicación. Se colecta todo el código fuente, y luego se genera el makefile correspondiente.
5. Se construye el proyecto.

#### 3.3.2. HAL

La Capa de Abstracción de Hardware (*Hardware Abstraction Layer o HAL*) provee una interfaz simple de drivers de dispositivos para conectar los programas con el hardware subyacente. La API está integrada con la librería estándar ANSI C, lo cual permite al software acceder a los dispositivos mediante el uso de funciones C ampliamente conocidas, tal como printf(), fopen(), fwrite(), etc.

### Servicios

La HAL provee los siguientes servicios:

- Integración con la librería estándar newlib: provee funciones estándar ANSI C de amplio uso.
- Drivers de dispositivo: brinda acceso a cada dispositivo en el sistema.
- API: proporciona una interfaz estándar consistente a los servicios de la HAL, tales como acceso a dispositivos y manejo de interrupciones.
- Inicialización de sistema: Lleva a cabo la inicialización para el procesador y las rutinas antes de ejecutar la función principal (main).
- Inicialización de dispositivos: Instancia e inicializa cada uno de los dispositivos del sistema antes de que se ejecute la función main.

La figura 3.1 muestra las capas de un sistema basado en la HAL, desde el nivel de hardware hasta el programa de usuario.

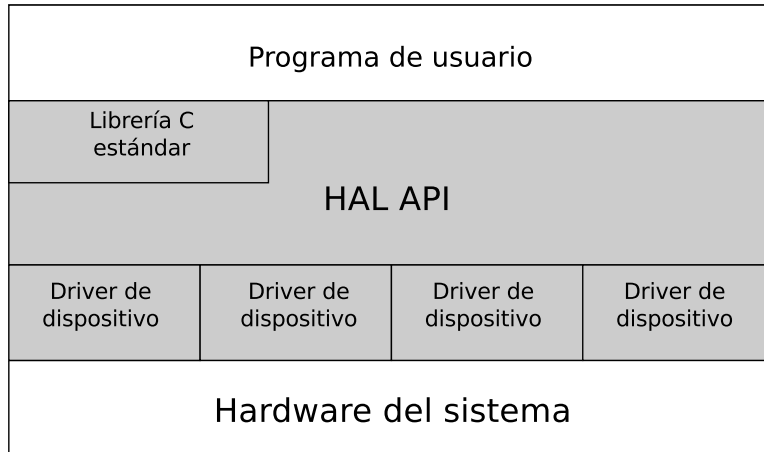


Figura 3.1: Capas de un sistema basado en la HAL.

### Modelos de dispositivos genéricos

La HAL provee modelos de dispositivos genéricos para diversos tipos de periféricos que se encuentran en sistemas embebidos, tales como timers, interfaces Ethernet y dispositivos de I/O que transmiten datos de carácter. Estos modelos permiten escribir programas usando una API consistente sin preocuparse por el hardware subyacente.

Los tipos de periféricos cubiertos son:

- Dispositivos de carácter: envían y/o reciben caracteres en forma serial, como ser una UART.
- Timers: dispositivos que llevan la cuenta de los tics de un clock y pueden generar interrupciones periódicas.
- Subsistemas de archivos: un mecanismo para acceder a archivos almacenados en dispositivos físicos. Dependiendo de la implementación interna, el driver del subsistema de archivos podría acceder a los dispositivos subyacentes en forma directa o usar un driver aparte.
- Dispositivos Ethernet: proveen acceso a una conexión Ethernet para una pila de red, tal como la NicheStack® TCP/IP Stack, provista por Altera.

- Dispositivos DMA: periféricos que llevan a cabo una gran cantidad de transacciones de datos. El origen y destino pueden ser la propia memoria o algún otro dispositivo.
- Dispositivos con memoria flash: periféricos con memoria no volátil que utilizan un protocolo especial de programación para almacenar datos.

Todos los periféricos, ya sean de Altera o de terceros, deben proveer un archivo de cabecera que defina la interfaz de bajo nivel del dispositivo con el hardware.

Ciertos dispositivos tienen características específicas de hardware con requerimientos de uso que no se adaptan bien a una API de propósito general. La HAL maneja dichos requerimientos mediante la función `ioctl()`, cuyas opciones dependerán del periférico en cuestión.

Algunos periféricos proveen funciones de acceso dedicadas que no están basadas en el modelo genérico de la HAL. En ese caso, se debe proveer un archivo de cabecera con dichas funciones.

### **Librería C estándar : newlib**

La HAL integra su entorno de rutinas con una implementación open-source de la librería C estándar: **newlib**. La misma está hecha para ser utilizada en sistemas embebidos.

### **La HAL dentro de un proyecto de software**

La creación y administración de proyectos basados en la HAL están altamente ligadas al NIOS II SBT

La figura 3.2 muestra el diagrama en bloques de un software basado en la HAL. Todo programa de este tipo consta de dos proyectos. El código de aplicación específico se encuentra en uno de ellos, el cual depende a su vez de otro proyecto BSP separado.

El primero contiene todo el código que el programador desarrolla. El segundo, toda la información necesaria para la interacción hardware-software. Este último a su vez depende del hardware del sistema, cuya información se encuentra en un archivo generado por la herramienta SOPC Builder.

### **El archivo de descripción del sistema (system.h)**

El archivo `system.h` provee una descripción completa del software del sistema basado en NIOS II. Describe cada periférico e incluye:

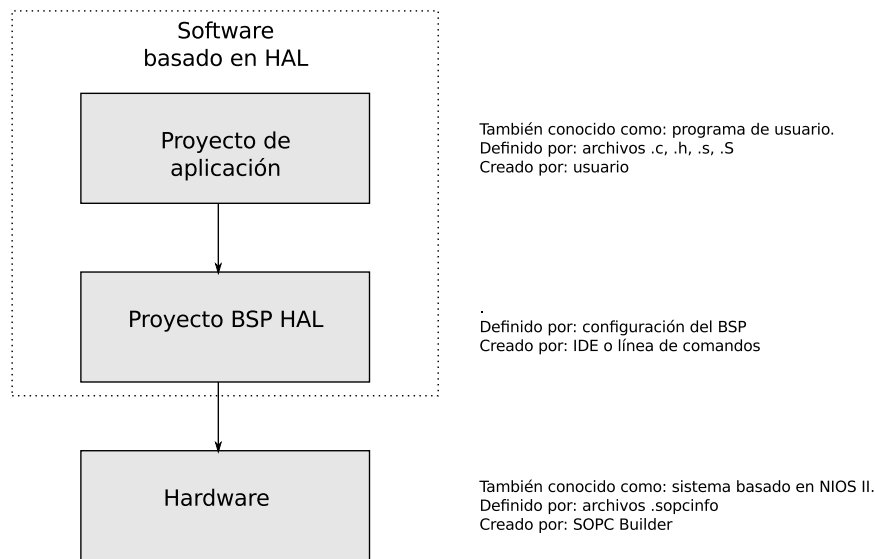


Figura 3.2: Capas de un programa basado en HAL

- La configuración de hardware del periférico.
- La dirección base.
- Información IRQ (si es necesario).
- Un nombre simbólico para el periférico.

El SBT genera un archivo **system.h** para cada proyecto BSP, cuyo contenido dependerá del archivo .sopcinfo mencionado anteriormente en este capítulo.

La razón de utilizar la HAL en vez de un sistema operativo, yace principalmente en la mayor simplicidad de uso que ésta presenta. En el caso de haber optado por el segundo enfoque hubiese sido necesario el desarrollo de drivers para el manejo de los dispositivos, con la complejidad que ello implica. La HAL, por otra parte, ofrece un conjunto de funciones de la librería estándar ANSI C para la interacción con el hardware del diseño, lo cual hace que dicha tarea sea significativamente más simple.

### 3.3.3. Algoritmos de clasificación

Se implementaron 2 algoritmos de clasificación: Búsqueda lineal y Búsqueda en árbol unibit.

El software utilizado para realizar las pruebas consistió en 2 proyectos por separado. Uno para cada tipo de búsqueda en la tabla.

El mismo fue desarrollado en lenguaje c++, por presentar éste ciertas facilidades para las implementaciones llevadas a cabo. Puntualmente se sacó ventaja de un STL container (list) para implementar la búsqueda lineal. La característica utilizada en este caso fue el ordenamiento de la lista con sólo una llamada a función.

Para efectuar el intercambio de datos con el hardware se hizo uso de las macros IOWR e IORD, las cuales escriben y leen respectivamente los datos hacia/desde un componente conectado al bus Avalon MM. La razón de haber usado dichas macros yace en el hecho de que las mismas no son puestas en caché. Esta característica se torna indispensable en este diseño, ya que en el mismo no se puede leer un dato sin saber si está verdaderamente disponible en el bus.

### Búsqueda lineal

Se implementó en una lista enlazada, creada a partir del template list de c++. Los nodos de la lista contienen 3 campos:

- Dirección de red (entero de 32 bit sin signo)
- Máscara de red (entero de 32 bit sin signo)
- Identificador de decisión (entero de 32 bit con signo)

Como se le dió prioridad a los prefijos de red más largos, se debió sobrecargar el operador de comparación (<) para que la función sort pudiese ordenar en base a la longitud de máscara. De esa manera, los nodos que contenían valores de máscara más grandes quedaban en las primeras posiciones de la lista.

Cuando la función encargada del lookup recibe una dirección IP de destino, realiza los siguientes pasos:

- Coloca un iterador al comienzo de la lista.
- Realiza un AND con el valor de máscara del nodo que está siendo apuntado. Si el resultado de la operación es igual al valor de dirección de red de dicho nodo, entonces se retorna con el valor identificador de decisión. En otro caso, continúa la búsqueda en el siguiente nodo.



**Busqueda en Arbol unibit**

Se implementó una clase en la cual se definieron las características de los nodos del arbol, como así tambien las operaciones de inserción y búsqueda.

En este contexto, pueden existir 2 tipos de nodo:

- Común: no está asociado a una decisión.
- Decisión: contiene un valor que identifica a la decisión a tomar.

Cada nodo cuenta con los siguientes campos:

- gw: es un identificador de la decisión a tomar. En los nodos no asociados a una decision, tiene el valor estipulado en la macro NONE.
- zero / one: Son punteros a nodo, asociados a los bits 0/1 del prefijo que se esté leyendo.

El algoritmo de búsqueda toma como entrada la dirección IP de destino del paquete a clasificar. Luego de ello, va haciendo un testeo bit a bit de la misma, partiendo con un puntero de recorrido desde el nodo raíz. Si el bit de la dirección es 0 y el puntero zero está apuntando hacia algun nodo, el puntero de recorrido se mueve al nodo apuntado por el puntero zero. En caso contrario, se mueve al nodo apuntado por el puntero one (En caso de que exista alguno). Esto se repite nodo a nodo, hasta que ocurre alguna de las siguientes situaciones:

- El puntero de recorrido queda varado en un nodo decision, con lo cual se retorna el valor de gw.
- El puntero de recorrido queda varado en un nodo común.

Contemplando esta última posibilidad, el algoritmo hace que en cada nodo se chequee si se trata de un nodo decisión. En dicho caso, se almacena el campo gw en una variable y se continua el recorrido. Si se da un caso en el cual el nodo de recorrido queda apuntando a un nodo comun y luego de testear un bit se determina que el mismo no tiene un nodo asociado (es decir, que alguno de los punteros zero / one esté en NULL) la funcion retorna la variable anteriormente mencionada.

### 3.3.4. Cache

Se implementó una cache directa. La misma consta de una tabla hash de 16 entradas. Las colisiones se resuelven por reemplazo directo. La misma fue testeada con ambos algoritmos mencionados anteriormente. Para ello, se agregó una lógica adicional que consistió en:

- Al tomar una direccion IP, chequear primero si el valor de decisión se encuentra en caché.
- Si está, retornar dicho valor.
- En otro caso, efectuar el lookup y almacenar el valor de decisión en caché.

Para evitar el overhead introducido por el uso de clases, se optó por el uso de estructuras para la implementación de este último enfoque.

## Capítulo 4

# Implementación

## Capítulo 5

# Resultados

En este capítulo se presentan los datos obtenidos de la ejecución del proyecto bajo ciertas condiciones representativas, con la intención de validar la funcionalidad y también de encontrar los alcances y límites del mismo. Primero se estudiara el tiempo de respuesta de los algoritmos de manera individual, más tarde el rendimiento en la configuración mas simple, luego el sistema completo bajo condiciones varias y por ultimo el estudio de la mejora introducida por el uso de la cache. Para la obtencion de los datos que corresponden al sistema completo se utilizo el Script Python/Bash que se encuentra en el Apendice XX.

### 5.1. Caso Algoritmos únicamente

Con la intención de obtener un gráfico que represente el rendimiento de los dos algoritmos implementados, se medirá el retardo de búsqueda en función de la posición en la tabla de enrutamiento, se realizaran las pruebas de manera independiente al modulo extracto de cabeceras. En el eje de las abscisas se expresa la ubicación en una tabla de 100 elementos del prefijo mas largo y en las ordenadas se puede observar el tiempo de búsqueda en ciclos de reloj.

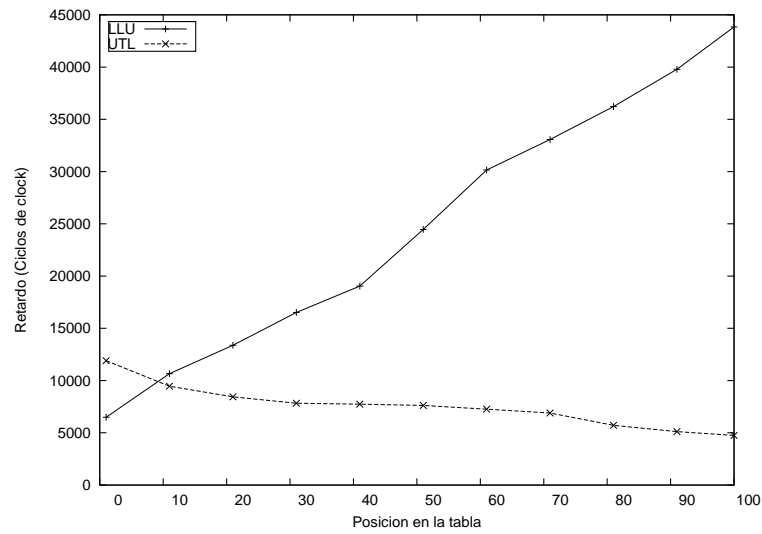


Figura 5.1: Retardo de Búsqueda LLU vs UTL

## 5.2. Caso loopback

Se estudia el caso loopback a los fines de encontrar los límites superiores del sistema. En este, el software solo se limita a recibir los datos e inmediatamente después confirma el procesamiento y envía los resultados de regreso al hardware. Se realizan las pruebas correspondientes para las dos versiones de Uplink. En el eje de las abscisas es posible ver la cantidad de paquetes por segundo, el origen corresponde a la mayor velocidad a la que es posible transmitir sin pérdidas. En las Ordenadas se puede observar la cantidad de paquetes perdidos en valores porcentuales, para obtener esta métrica se procesa una cantidad constante de paquetes, 9000, y luego se contrasta este valor con un contador global que el Generador imprime en la última palabra de cada paquete. Así se calcula la cantidad de paquetes perdidos, sobre la cantidad total de paquetes generados. Este mismo sistema es el usado en todos los gráficos posteriores.

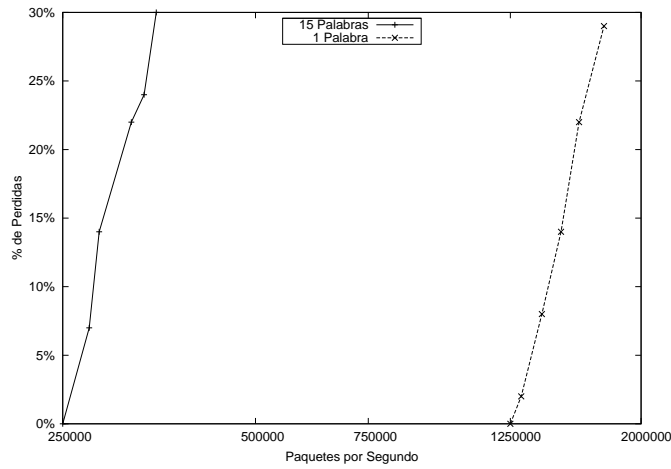


Figura 5.2: Caso Loopback para 1 y 15 palabras

### 5.3. Implementación Completa

Se verificara el rendimiento del sistema implementado de manera completa. Se consideraran tres puntos en las curvas que indican los tiempos de accesos del algoritmos, un punto mínimo que corresponde al menor tiempo de acceso, un punto promedio que ejercita 10 entradas equidistantes a lo largo de la tabla y un punto máximo que indica el peor tiempo de acceso posible para un algoritmo dado.

#### Linear Lookup

En la figura 5.3 se puede observar que la diferencia en la cantidad de paquetes que pueden ser transmitidos sin errores en el mejor caso entre el modulo que envía una palabra al procesador y el que envía toda la cabecera, es considerable, mientras que a medida que aumenta la profundidad de búsqueda en la tabla estos valores convergen, como es posible ver en la figuras 5.4 y 5.5, lo que era esperable ya que para accesos muy lentos a la tabla el retardo introducido por el Hardware se vuelve despreciable.

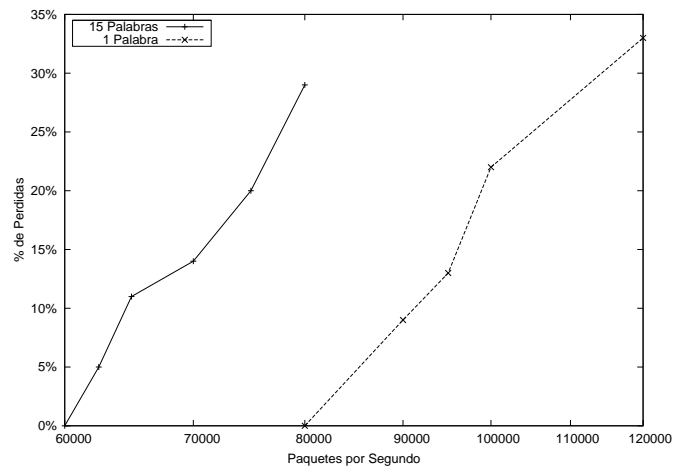


Figura 5.3: Retardo mínimo LLU

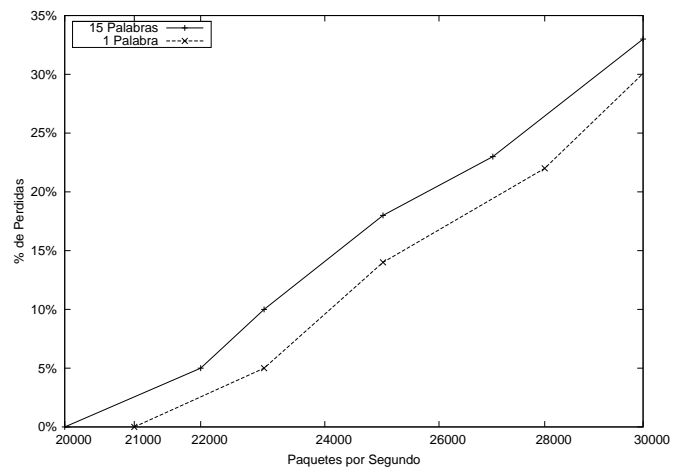


Figura 5.4: Retardo promedio LLU

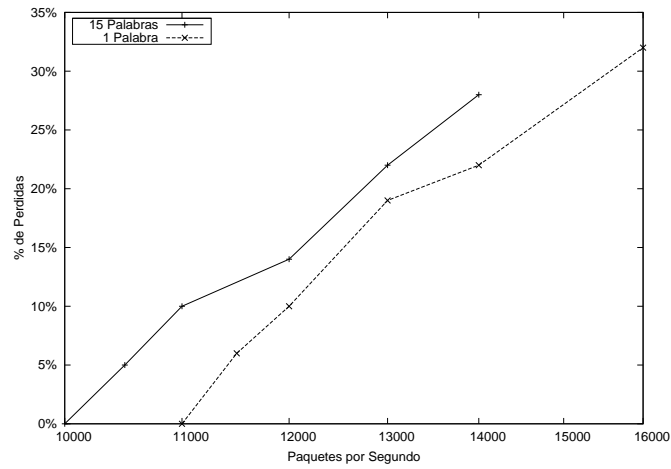


Figura 5.5: Retardo máximo LLU

### Unibit Trie Lookup

En los gráficos que corresponden al Unibit Trie Lookup es posible observar que existe una menor diferencia entre la máxima cantidad de paquetes que pueden ser transmitidos sin error en cada uno de los 3 puntos elegidos. Así como también la diferencia entre enviar el paquete entero y solo la IP destino se reduce, lo que da la pauta de que cuando el tiempo de acceso es uniforme el impacto de la mejoras del Hardware tiende a ser menor.

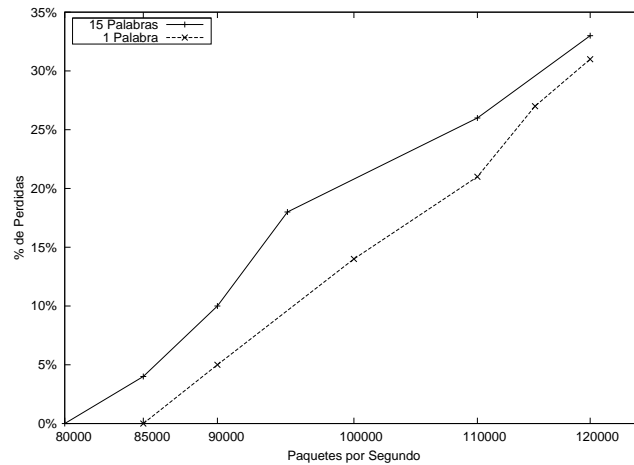


Figura 5.6: Retardo mínimo UTL



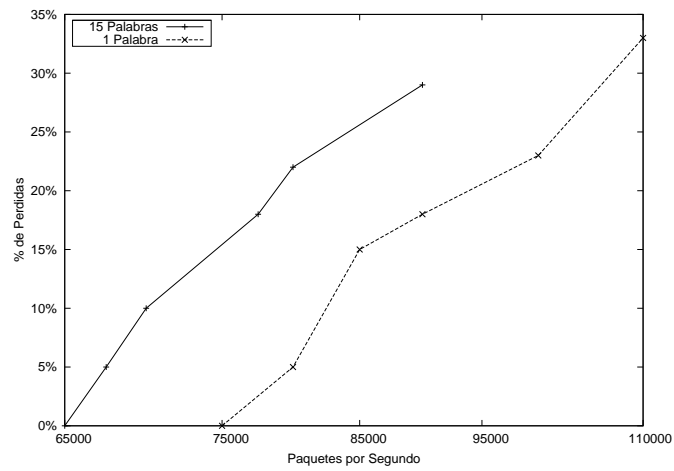


Figura 5.7: Retardo promedio UTL

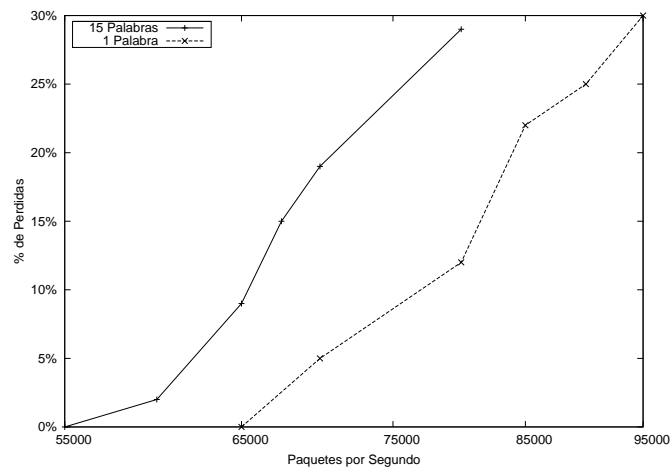


Figura 5.8: Retardo máximo UTL

### Comparativa Inter-Algoritmos

Se presenta a modo de comparacion un grafico que contiene el minimo, el maximo y el promedio para el caso de una palabra en los dos algoritmos implementados.

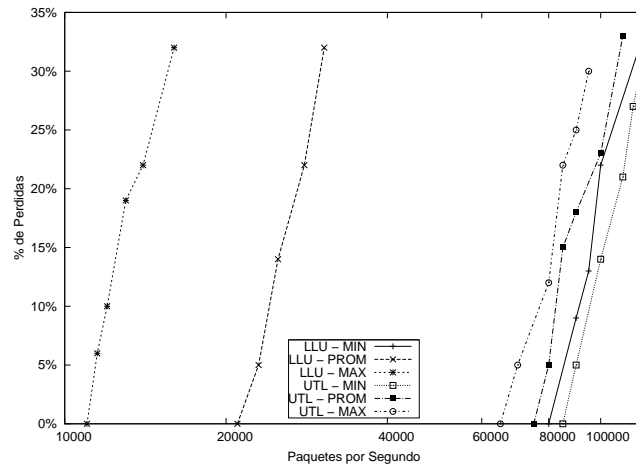


Figura 5.9: Comparativa UTL - LLU

## 5.4. Cache

(Falta Mejorar y agregar el grafico) Se probara el funcionamiento del sistema incluyendo la cache implementada en software. Para ello, se efectuara una toma de datos en la cual se compararan 2 sucesiones: la primera de ella resultara en fallos al no estar presente ninguno de los valores.

Puede notarse que, en el caso de que la búsqueda termine en un fallo, se introduce un retardo adicional (si se compara con el escenario en el cual no existe la cache). Ello se debe a la búsqueda que se efectúa en la cache antes de hacerlo propiamente en la tabla de ruteo, tanto en la implementación linked-list como en la implementación unibit-trie.

## Capítulo 6

# Conclusiones

# Bibliografía

# Appendices

## 6.1. Configuración del Hardware

Feature	Description
FPGA	<ul style="list-style-type: none"><li>■ Cyclone II EP2C35F672C6 with EPCS16 16-Mbit serial configuration device.</li></ul>
I/O Interfaces	<ul style="list-style-type: none"><li>■ Built-in USB-Blaster for FPGA configuration</li><li>■ Line In/Out, Microphone In (24-bit Audio CODEC)</li><li>■ Video Out (VGA 10-bit DAC)</li><li>■ Video In (NTSC/PAL/Multi-format)</li><li>■ RS232</li><li>■ Infrared port</li><li>■ PS/2 mouse or keyboard port</li><li>■ 10/100 Ethernet</li><li>■ USB 2.0 (type A and type B)</li><li>■ Expansion headers (two 40-pin headers)</li></ul>

Memory	<ul style="list-style-type: none"> <li>■ 8 MB SDRAM, 512 KB SRAM, 4 MB Flash</li> <li>■ SD memory card slot</li> </ul>
Displays	<ul style="list-style-type: none"> <li>■ Eight 7-segment displays</li> <li>■ 16 x 2 LCD display</li> </ul>
Switches and LEDs	<ul style="list-style-type: none"> <li>■ 18 toggle switches</li> <li>■ 18 red LEDs</li> <li>■ 9 green LEDs</li> <li>■ Four debounced pushbutton switches</li> </ul>
Clocks	<ul style="list-style-type: none"> <li>■ 50 MHz clock</li> <li>■ 27 MHz clock</li> <li>■ External SMA clock input</li> </ul>

La FPGA incluida en la placa es una Cyclone II EP2C35 cuyas especificaciones son:

Feature	Description
LEs	33216
Total RAM bits	483840
Embedded multipliers	35
PLLs	4
Maximum user I/O pins	475