Capítulo 3 Implementación en VHDL

A continuación se va a explicar brevemente el funcionamiento de VHDL y las componentes de programación para poder entender mejor el programa. Una vez explicado esto, se interpretan los programas creados para esta tesis y el funcionamiento que se espera tener. Después se explican con más detenimiento cada uno de los programas y las diferencias entre cada uno de ellos.

3.1 Introducción a VHDL

A mediados de los años ochenta el Departament de Defensa de los Estados Unidos y la IEEE (*Institute of Electrical and Electronics Engineers*) patrocinaron el desarrollo de este lenguaje de descripción de hardware con el propósito de desarrollar circuitos integrados de muy alta velocidad. Hoy en día VHDL se ha convertido en uno de los estándares en la industria más usado para describir sistemas digitales. Este tipo de lenguajes es usado para implementarlos en un PLD (*Programmable Logic Device*).

El PLD más simple esta formado por un vector de compuertas de tipo AND que es programable y otro de compuertas de tipo OR que no es programable. Estos pueden interconectarse utlilizando tecnología EPROM, EEPROM o FLASH de la manera que se desee, así teniendo diferentes funcionamientos. Las ventajas de utilizar PLDs es que un solo PLD puede sustituir 10 o más circuitos TTL. Otra ventaja es que es mucho más económico que un circuito TTL.

El lenguaje VHDL es un lenguaje de descripción de hardware que provee un medio para especificar un sistema digital en una amplia gama de niveles de abstracción. El lenguaje soporta los primeros estados conceptuales de un diseño con su nivel de

abstracción de comportamiento. Este lenguaje también provee construcciones jerárquicas que permiten al diseñador controlar la complejidad de la descripción.

La programación en VHDL esta formada por entidades que tienen una arquitectura y esta a su vez tiene procesos.

En la declaración de entidades se definen las entradas y salidas del circuito especificando cuántas son, el tamaño, si son de entrada o salida y el tipo. Se puede decir que una entidad es el equivalente al esquemático de lo que queremos representar que describe las conexiones de un componente al resto del proyecto. La forma general de una entidad se muestra en la Figura 3.1.

```
entity NOMBRE is [ generic declaraciones genéricas);]
     port (señal_1 : mode type ;
           señal_2 : mode type ;
           señal_3 : mode type) ;
end NOMBRE ;
```

Figura 3.1 : Entidad

El modo de una entidad puede ser de cuatro tipos, entrada (*in*), salida(*out*), buffer o bidireccional (*inout*). El tipo puede ser de la siguiente forma:

- *Bit* - Puede tener el valor de 0 ó 1.
- *Bit_vector* – Es un vector de bits del tamaño que se necesite
- *Std_logic, std_ulogic, std_logic_vector, std_ulogic_vector* – Puede tener nueve valores para indicar la intensidad de la señal.
- *Boolean* – Puede tener los valores de verdadero y falso.
- *Integer* – Puede tener un rango de valores de números enteros.
- *Real* – Puede tener un rango de números reales
- *Character* – Puede tener el valor de cualquier carácter.

- *Time* – Utilizado para indicar tiempo.

La arquitectura describe lo que se puede hacer con los puertos de entrada para que los puertos de salida puedan tener un funcionamiento. En la arquitectura se dice que se debe de hacer con los puertos de entrada para obtener la salida. Si la entidad se ve como una caja negra, la arquitectura es el conjunto de detalles dentro de la caja negra. La representación básica de una arquitectura se muestra en la Figura 3.2.

```
architecture nombre_arquitectura of NOM_ENTIDAD is
-- Declaraciones
      -- declaración de componentes
      -- declaración de señales
      -- declaración de constantes
      -- declaración de funciones
      -- declaración de procedimientos
      -- declaración de tipos

      :

begin
-- Núcleo del Programa

      :

end nombre_arquitectura;
```

Figura 3.2: Arquitectura

Para describir una arquitectura hay estilos con diferentes niveles de abstracción:

- Estilo de comportamiento (*Behavioral*) – Este estilo utiliza las condiciones o comandos comúnmente usados en un lenguaje de programación tales como condiciones IF, THEN o CASE sin que el programador se preocupe de la distribución de las compuertas del PLD. Es necesario incluir un proceso ya que es una estructura secuencial.
- Estilo de flujo de datos (*Dataflow*) - En este estilo se describe como la información es transimitida de señal a señal y de la entrada a la salida sin el uso de asignaciones secuenciales. En este estilo no se utilizan los procesos.

- Estilo estructural (*Structural*) – En este estilo se describe un "netlist" de VHDL en donde se interconectan los componentes y se evaluan inmediatamente con señales. En el estilo estructural se especifican cuales componentes son usadas y de que manera están interconectadas.
- Estilo mixto – Este es la combinación de dos o más de los estilos anteriores.

Un proceso es un evento concurrente de una arquitectura. Una arquitectura puede tener varios procesos para describir la operación concurrente de varias piezas de la arquitectura como una máquina de estados, un contador o una ALU (*Arithmetic Logic Unit*). El proceso es el evento concurrente más usado porque provee un amplio rango de flexibilidad como resultado de su naturaleza secuencial. También permite la sincronización con otros eventos concurrentes.

3.2 Programa Base Para Redes IP y ATM

En esta tesis se creo un programa base con el propósito de facilitar la programación y la simulación. A partir de ese programa se crearon otros dos programas, uno para su funcionamiento en redes ATM y otro para su funcionamiento en redes IP. Se hizo de esa manera ya que la diferencia no es mucha y utilizando valores genéricos se puede cambiar fácilmente de un programa a otro.

El programa cuenta con dos interfases de entrada, A y B, y tres interfases de salida, X, Y y Z. Tiene dos entidades, Switch y Fifo, cada una con sus arquitecturas, Mem y Fifo_v1, respectivamente. La Figura 3.1 muestra un diagrama con los pines de entrada y salida de cada una de las entidades.

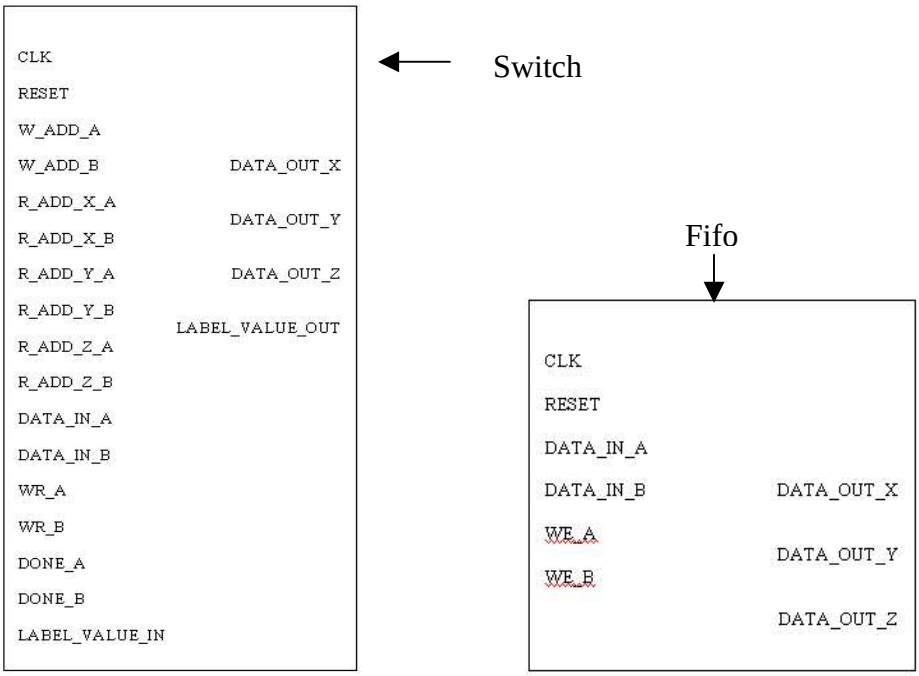

Figura 3.3 : Entidades del Programa

Dentro de la entidad Switch, hay cuatro memorias: Data_1, Data_2, lut_data_in y lut_data_out. Las primeras dos es en donde se van almacenando los datos, el tamaño varía dependiendo de un genérico para la longitud de las direcciones. En Data_1 se van almacenando los datos que llegan por la entrada A y en Data_2 los que llegan por la entrada B. Las otras dos memorias tienen una longitud de 16 localidades pero cada una con diferentes longitudes de direcciones. En lut_data_in se almacenan datos de 20 bits, que es la longitud de las etiquetas, en lut_data_out se almacenan datos de 22 bits. Esos dos bits de más son para decidir por cuál puerto los datos deben de salir.

Esta entidad tiene dos procesos importantes (Ver Apéndice A para mayor información). Uno es en donde se lleva a cabo la lectura y la escritura de los datos en la memoria y la inicialización de las memorias. El otro, llamado MPLS, hace tres cosas muy importantes:

- Determina cuál es el valor de la etiqueta de entrada

- Determina cuál es la interfase de salida para el paquete
- Coloca el nuevo valor de la etiqueta de salida

La entidad FIFO tiene como propósito proporcionar las direcciones de lectura y escritura que corresponden. Estas direcciones las va proporcionando de manera FIFO, el primer dato que entra es el primero que sale. También manda las señales de done_a y done_b que indican cuando se termina de mandar el paquete que entró al conmutador.

3.3 Adaptación del Programa Base para Redes ATM

Como se había mencionado antes, el programa base no es difícil de cambiar gracias a que se utilizaron genéricos en su elaboración. Sin embargo, primero que nada hay que entender como funcionan los conmutadores ATM como LSRs.

La forma de direccionar paquetes en MPLS no difiere mucho a la del direccionamiento de los conmutadores ATM. Los conmutadores ATM usan el puerto de entrada y el valor del VPI/VCI (*Virtual Path Identifier/Virtual Channel Identifier*) como un índice en una tabla de la cual obtienen un puerto de salida y un valor para el VPI/VCI. De igual manera, si una o más etiquetas pueden ser codificadas directamente en los campos que los conmutadores ATM acceden, estos conmutadores, con simples actualizaciones de programa, pueden ser usados como LSRs. A estos conmutadores se les conoce como ATM-LSRs [RVC01].

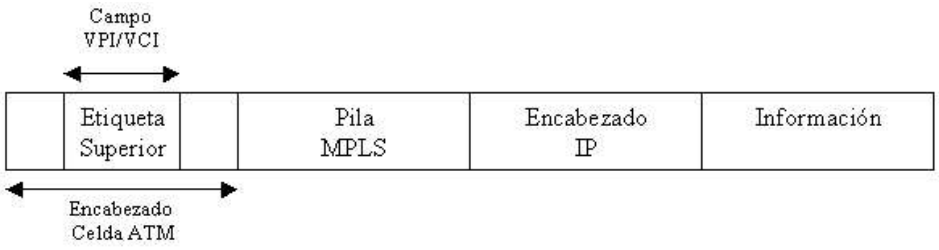

Figura 3.4 : Ubicación de la etiqueta

Para poder simular la entrada de datos, tenemos que saber de qué forma los vamos a recibir. El encabezado de una red ATM se demuestra en la Figura 3.3.

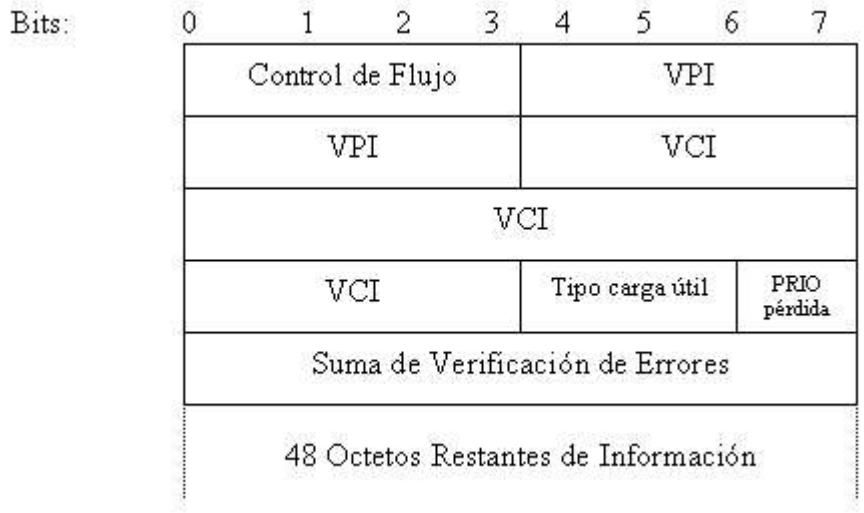

Figura 3.5 : Encabezado ATM

Como se puede observar los primeros cuatro octetos es en donde se debe de localizar la etiqueta de 32 bits. Haciendo las modificaciones necesarias al programa base, se llegó a un programa que puede simular el intercambio de etiquetas para una red ATM.

3.4 Adaptación del Programa Base para Redes IP

Para implementar el programa base para redes IP se tiene que entender el funcionamiento de las redes IP y su funcionamiento con MPLS. En cada frame se carga exactamente un solo paquete etiquetado. La ubicación de la pila de etiquetas está antes del encabezado de capa de red y después del encabezado de capa de enlace de red incluyendo cualquier encabezado IP que pueda existir.

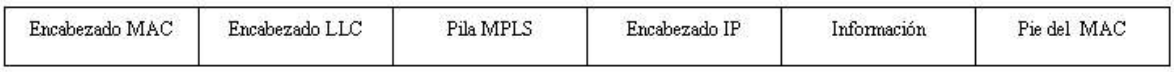

Figura 3.6 : Ubicación de la etiqueta

Para ubicar en qué octeto se empieza a leer la etiqueta, se tiene que analizar el encabezado de una red IP (Ver Figura 3.5).

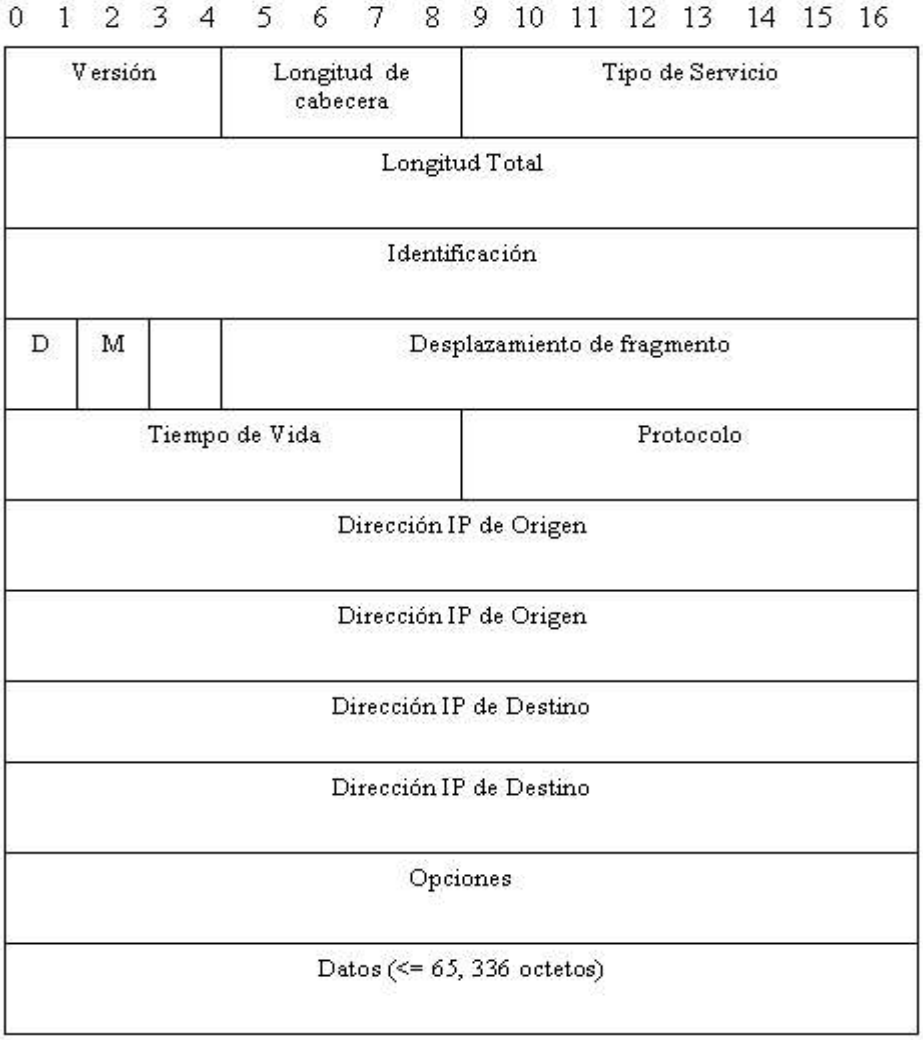

Figura 3.7 : Datagrama IP

De esta manera se puede ver que se empieza a leer en el octeto número 13, que es donde se localizan las direcciones IP. Otra modificación que se le tuvo que hacer al programa base fue ampliar la memoria Data_1 y Data_2 ya que el tamaño del datagrama IP es mucho más grande que el de ATM, esto se hace fácilmente modificando el genérico del ancho de las direcciones.